

Master's Programme in Department of Mechanical Engineering

Python Workflow for Point Cloud Registration and 3D Inspection

Lauri Novio

Master's thesis
2024

Copyright ©2024 Lauri Novio

Author	Lauri Novio				
Title of thesis	Python Workflow for Point Cloud Registration and 3D Inspection				
Programme	Mechanical Engineering				
Major	Production Engineering				
Thesis supervisor	Prof. Jouni Partanen				
Thesis advisor(s)	Prof. Jan Akmal				
Date	28.09.2024	Number of pages	60 + 5 + 35	Language	English

Abstract

Quality is often associated with expensive products and top brands. However, in manufacturing the meaning of quality is more about manufacturing products that are in given tolerances. Inspection of tolerances is usually implemented using a Coordinate Measuring Machine (CMM) or portable scanner. Those measurement devices give 3D point clouds as results and enable many different applications for dimensional and geometrical 3D Inspection.

3D Inspection needs data processing and calculation to get desired information from the measured point clouds. There are numerous software available for 3D Inspection including paid and open-source software. Widely used methods in those software are point cloud registration and different form-fitting methods. In addition, the open-source software Python has lots of libraries for point cloud registration and form-fitting.

This thesis aims to investigate the possibilities of Python for 3D Inspection. The thesis provides comprehensive background information in the form of a literature review and a case study of 3D Inspection using Python with 3D scanned point clouds. The case study includes 3D Inspection of cylindrical part diameters with two other Python workflows. One of those workflows uses point cloud registration and the nominal CAD data and the other uses cylinder-fitting without the nominal CAD data. In addition, the case study compares state-of-the-art point cloud registration methods available for Python.

The investigation proved Python to be suitable for 3D Inspection. The measurement results of Python workflows were extremely close to the industrial-aimed GOM Inspect software with a maximum difference of 0,006 mm. In addition, there were more accurate cylinder fitting algorithms available for Python than for the open-source software CloudCompare. The algorithm used for Python was Least Squares (LS) and for CloudCompare Ransac. The investigation was also implemented for the point cloud registration methods available for Python. Noteworthy was that the commonly used combination Random Sample Consensus (Ransac) + Iterative Closest Point (ICP) gave more accurate results than the state-of-the-art neural networks-based PointNetLK.

Keywords 3D Inspection, Point Cloud Registration, 3D Scanning, Cylinder Fitting, Surface Comparison

Tekijä	Lauri Novio				
Työn nimi	Python-työnkulku Pistepilvien Rekisteröintiin ja 3D-tarkastukseen				
Koulutusohjelma	Konetekniikka				
Pääaine	Tuotantotekniikka				
Vastuuopettaja/valvoja	Prof. Jouni Partanen				
Työn ohjaaja(t)	Prof. Jan Akmal				
Päivämäärä	28.09.2024	Sivumäärä	60 + 5 + 35	Kieli	Englanti

Tiivistelmä

Laatu yhdistetään usein kalliisiin tuotteisiin ja huippumerkkeihin. Valmistusteollisuudessa laadun merkitys painottuu kuitenkin tuotteiden valmistamiseen annettujen toleranssien mukaisesti. Toleranssien tarkastus toteutetaan yleensä koordinaattimittauskoneella tai kannettavalla skannerilla. Nämä mittauslaitteet tuottavat tulokseksi 3D-pistepilviä, ja ne mahdollistavat monia erilaisia sovelluksia dimensionaaliseen ja geometriseen 3D-tarkastukseen.

3D-tarkastuksessa tarvitaan tietojen käsittelyä ja laskentaa, jotta mitatuista pistepilvistä saadaan haluttua tietoa. 3D-tarkastukseen on saatavilla lukuisia ohjelmistoja, mukaan lukien maksullisia ja avoimen lähdekoodin ohjelmistoja. Näissä ohjelmistoissa käytetään yleisesti pistepilvien rekisteröintiä ja erilaisia muotojensovituskäytännöitä. Kyseisten menetelmien käyttöön on saatavilla lukuisia Pythonille suunnattuja avoimen lähdekoodin kirjastoja.

Tämän opinnäytetyön tarkoituksena on tutkia Pythonin mahdollisuuksia 3D-tarkastuksessa. Opinnäytetyössä annetaan kattavat taustatiedot kirjallisuuskatsauksen muodossa ja suoritetaan tapaustutkimus, jossa Python-ohjelmistoa käytetään 3D-tarkastukseen skannattujen pistepilvien kanssa. Tapaustutkimus sisältää sylinterimäisen kappaleen halkaisijoiden tarkastuksen kahdella Python-työnkullulla. Toisessa näistä työnkuluista käytetään pistepilven rekisteröintiä ja CAD-referenssimallia ja toisessa sylinterin sovitusta ilman CAD-referenssimallia. Lisäksi tapaustutkimuksessa suoritetaan vertailua Pythonille saatavilla oleville pistepilvien rekisteröintimenetelmille.

Tutkimus osoitti Pythonin soveltuvan 3D-tarkastukseen. Python-työnkulkujen mittaustulokset olivat erittäin lähellä teolliseen käyttöön tarkoitettua Gom Inspect-ohjelmistoa suurimmalla erolla 0,006 mm. Lisäksi Pythonille oli saatavilla tarkempia sylinterinsovitusalgoritmeja kuin avoimen lähdekoodin CloudCompare-ohjelmistolle. Pythonissa käytetty algoritmi oli pienimmän neliösumman menetelmä (LS) ja CloudComparessa satunnaisotosmenetelmä (Ransac). Tutkimusta suunnattiin myös Pythonille saatavilla oleville pistepilvien rekisteröintimenetelmille. Huomionarvoista oli, että yleisesti käytetty yhdistelmä (Ransac) + iteratiivinen lähimmän pisteen menetelmä (ICP) antoi tarkempia tuloksia kuin viimeisimpään tekniikkaan (neuroverkkoihin) perustuva PointNetLK.

Avainsanat 3D-tarkastus, Pistepilvien Rekisteröinti, 3D-skannaus, Sylinterinsovitus, Pintojen Vertailu

Table of Contents

Acknowledgements	7
Abbreviations	8
1 Introduction	9
1.1 Goals.....	9
1.2 Research Questions.....	10
1.3 Constraints	10
1.4 Structure.....	10
2 Background Information.....	10
2.1 Quality	11
2.2 Quality in Manufacturing	11
2.3 Inspection.....	12
2.3.1 Coordinate Measuring Machine (CMM)	13
2.3.2 Portable Scanners	15
2.3.3 Point Cloud Processing.....	17
2.4 Point Cloud Registration Algorithms	17
2.4.1 Random Sample Consensus (RANSAC).....	17
2.4.2 Normal Distribution Transform (NDT)	18
2.4.3 Iterative Closest Point (ICP).....	19
2.4.4 Coherent Point Drift (CPD)	19
2.4.5 Neural Networks.....	20
2.5 Point Cloud Cylinder Fitting Algorithms.....	20
2.5.1 Least Squares (LS).....	21
2.5.2 Minimum Zone (MZ).....	21
2.5.3 Maximum Inscribed Element (MIE).....	22
2.5.4 Minimum Circumscribed Element (MCE).....	22
2.5.5 Ransac.....	23
2.6 Python Libraries for Point Cloud Registration.....	23
2.6.1 Open3D.....	23
2.6.2 Learning3D	23
2.7 Python Libraries for Cylinder Fitting	24
2.7.1 PyRANSAC-3D	24

2.7.2	Py-cylinder-fitting	24
3	Python Workflow for 3D Inspection of Machined Tolerance Bar	24
3.1	Design.....	25
3.2	Manufacturing.....	26
3.3	Scanning.....	32
3.4	Python Workflow for 3D Inspection.....	36
3.4.1	Workflow 1. Surface Comparison	36
3.4.2	Selection of Registration Algorithm.....	40
3.4.3	Workflow 2. Cylinder Fitting.....	44
3.4.4	Validation Methods	45
3.4.5	Sensitivity Analysis.....	49
3.5	3D Inspection Results.....	52
4	Conclusions/Discussion	58
	References.....	61
	Appendix A (Workflow 1, Ransac + ICP)	66
	Appendix B (Ransac)	72
	Appendix C (PointNetLK).....	77
	Appendix D (Ransac + PointNetLK)	83
	Appendix E (Ransac + PointNetLK + ICP)	90
	Appendix F (Workflow 2, LS Cylinder Fitting)	97
	Appendix G	100

Acknowledgements

I thank Professors Jouni Partanen and Jan Akmal for making this thesis work possible. In addition, I want to thank Jan Akmal for the precious meetings where I got useful advice for the thesis.

I also want to thank the staff of ENG-workshop Janne Peuraniemi and Seppo Nurmi for their help with machining and measuring.

Helsinki, 28 September 2024
Lauri Novio

Abbreviations

CMM	Coordinate Measuring Machine
CAD	Computer Aided Design
CNC	Computerised Numerical Control
RANSAC	Random Sample Consensus
NDT	Normal Distribution Transform
ICP	Iterative Closest Point
CPD	Coherent Point Drift
LS	Least Squares
MZ	Minimum Zone
MIE	Maximum Inscribed Element
MCE	Minimum Circumscribed Element
MPE	Maximum Permissible Error

1 Introduction

Quality is often associated with expensive products and top brands. However, in manufacturing the meaning of quality is more about manufacturing products that are in given tolerances [1]. Tolerances are extremely important in manufacturing due to the effects on manufacturing speed, customer satisfaction, and manufacturing costs [1, 2].

Inspection of tolerances requires a lot from measurement devices. For example, geometrical tolerances can be impossible to inspect without 3D model-based measuring. The most common 3D inspection device is a coordinate measuring machine (CMM) with contact or non-contact-based measuring [3]. However mobile scanners are also used due to the advantages over CMM which are speed, price, and suitability for large products [3].

The result from a 3D inspection device such as a CMM machine or mobile scanner is just a series of point clouds [4]. Before getting the desired metrics out, a lot of different data processing is required. Data processing includes the registration of point clouds into the same coordinate systems, removing noise and gaps from the model [4]. After that, the calculation of desired feature metrics can be done by comparing measured point clouds to the nominal CAD data or using different form-fitting methods without the CAD data.

There are numerous additional 3D inspection software for point cloud data processing and 3D inspection but these cost a lot of money from users. However, there is also an open-source software called CloudCompare that provides many different features for point cloud data processing and 3D inspection. In addition, Python has many other open-source libraries for point cloud data processing and 3D inspection. One significant advantage of Python is that users can select the most suitable data processing methods for specific applications.

1.1 Goals

This thesis aims to find out how accurate 3D inspection workflow can be created using Python compared to the available 3D inspection open-source and paid software.

This thesis also aims to explore state-of-the-art algorithms in 3D inspection and find out Python libraries that use those algorithms. One objective is also to present a repeatable investigation procedure for the 3D inspection of a simple machined part. This process offers an efficient way to compare different inspection software.

1.2 Research Questions

1. What are the state-of-the-art algorithms in data processing for 3D inspection?
2. Are there open-source libraries for Python that use the algorithms?
3. How accurate Python workflow can be created for 3D inspection compared to available open-source and paid software?

1.3 Constraints

The thesis is limited to exploring the role of quality in production from a perspective of conformance only. In addition, the case study is limited to handling only quality assurance of one machining part where separate point clouds of scanned data are already registered to each other using the software of the utilized 3D scanner.

3D inspection can be applied for inspecting a wide range of different forms and features. However, this thesis focuses on the 3D inspection of the diameters of a cylindrical part.

1.4 Structure

This thesis consists of a literature review and a case study. The literature review provides background information about the importance of 3D inspection for manufacturing and explores the most used methods for it. In addition, it delves into computational algorithms used in 3D inspection and finds out Python open-source libraries for using those algorithms.

The case study provides an easily repeatable investigation process for comparing different 3D inspection software. The investigation compares Python workflow to other 3D inspection software including open-source software CloudCompare and paid software GOM Inspect. In addition, the case study includes a comparison between contact and noncontact-based 3D inspection providing measurements taken by Zeiss CMM workflow.

2 Background Information

This section provides background information about the importance of 3D inspection in production and about the methods used in it. In addition, the section delves into the point cloud data processing behind 3D inspection and explores the state-of-the-art calculation methods of it. The outcome of the

section is to find out the available Python open-source libraries for using those methods.

2.1 Quality

David A. Garvin's article *Competing on the Eight Dimensions of Quality* 1987 illustrated the importance of quality in meeting customer needs [5]. The article also illustrated the importance of quality in inter-firm competition with the example of the United States and Japan competing in the production of Random-Access Memory chips. The US was the underdog in terms of product quality and needed new strategies to improve it.

According to Garvin, product design needed to focus more on life-cycle costs, and new methods of production management were needed to identify quality from the point of view of customer needs. As a solution, Garvin proposed eight dimensions to assess quality more effectively. The eight dimensions of quality can be found in the following Figure 1. The highlighted dimension number 8 is the most significant dimension for manufacturing and inspection, and the next section talks more about it.

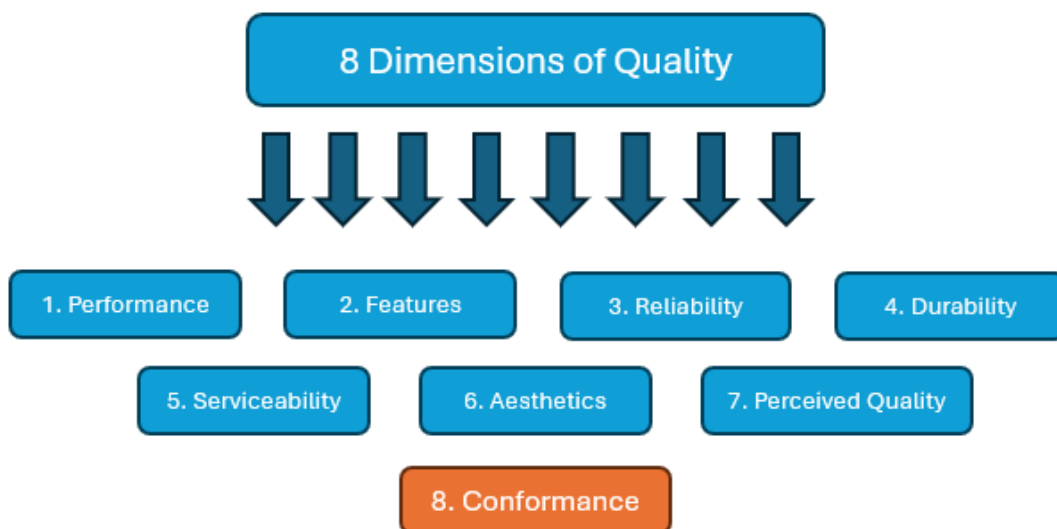


Figure 1. Garvin's Eight Dimensions of Quality

2.2 Quality in Manufacturing

Dimension number 8 (Conformance) plays a major role in manufacturing. It means that manufactured products must meet specified requirements, including material specifications and tolerances [1]. Both are extremely

important in production, as material requirements directly affect the cost of the product in terms of material and the appropriate manufacturing process [6]. In addition, the material has an impact on the cost-effectiveness of manufacturing. For example, in machining, stiff and strong materials can be slow to operate [7].

Tolerances also play a crucial role in manufacturing and set the limits for manufacturing accuracy. Tolerances are not intended to ensure that parts are produced in the same dimensions in the same way all the time, but within a range of precision that meets the customer needs. Tighter tolerances demand more from production and the basic assumption is that as the tolerance range is reduced, production costs increase. [2]

2.3 Inspection

Tolerances require always inspection. Inspection slows down the throughput time of products and therefore only a small proportion of products are usually inspected. Depending on the product, inspection measurements can be carried out with different types of measuring devices. Measurement devices can operate based on contact or non-contact measuring. For all, calibration needs to be done at regular intervals. [1]

Common measurement devices used for inspection are CMM, 3D scanner, micrometer, caliper, and oscilloscope. 3D-based measurement devices CMM and 3D scanners are usually used according to the possibility of estimating different geometrical features using mathematical calculation. Inspection increases additional costs and takes time, but it aims to generate long-term savings by reducing the number of defective products and increasing customer satisfaction [3].

Inspection can be carried out during manufacturing or after it [3]. For example, in additive manufacturing, automatic quality control during the manufacturing process has been found useful for optimizing printing settings for the remaining layers [8]. In many cases, inspection during manufacturing can be also implemented by operator [9]. For example, during manual machining operators usually inspect dimensions using micrometers or calipers.

The basic rule of selection of inspection device is not to use too accurate one. Measuring devices are expensive and increase considerable costs for production. In addition, the use of more accurate measuring equipment is slower. It is therefore advisable to start with a less accurate method of precision measurement and to move to a more accurate method only if the measurement results are close to the tolerance limits set for the product. However, it should

be noted that there are standard measurement methods suitable for certain tolerances and features [9].

2.3.1 Coordinate Measuring Machine (CMM)

Coordinate measuring machines are widely used for inspection in industry. They find a set of points on the surface of a piece and establish a connection between these points. Coordinate measuring machines are a fast and accurate means of verifying dimensions and can be based on contact or non-contact measurements [3]. Different types of coordinate measuring machines are better or worse suited to specialized measuring applications, but in general, any coordinate measuring machine can measure any geometric feature, as long as there are no obstacles in front of the surface to be measured and the object to be measured is not too large in relation to the measuring device [4].

Nowadays, coordinate measuring machines are typically equipped with numeric control and scanning probes. Both touch and non-contact machines can automatically measure a very large number of points in a short time. A contact-based measurement can measure up to 200 p/s at a speed of 150 mm/s. Non-contact measurement achieves even faster measurement speeds by relying on optical principles such as autofocus, triangulation and, others. [3]

A coordinate measuring machine can contain different measuring sensors and consist of slightly different components. Different configurations of measuring devices have different advantages and disadvantages and are therefore suitable for different measurement applications. In general, a coordinate measuring machine configuration includes a mechanical setup of machine axes and transducers, sensors, a control unit, and a computer with software for data processing. [4] Figure 2 below shows a CMM setup which was also used in the case study section.

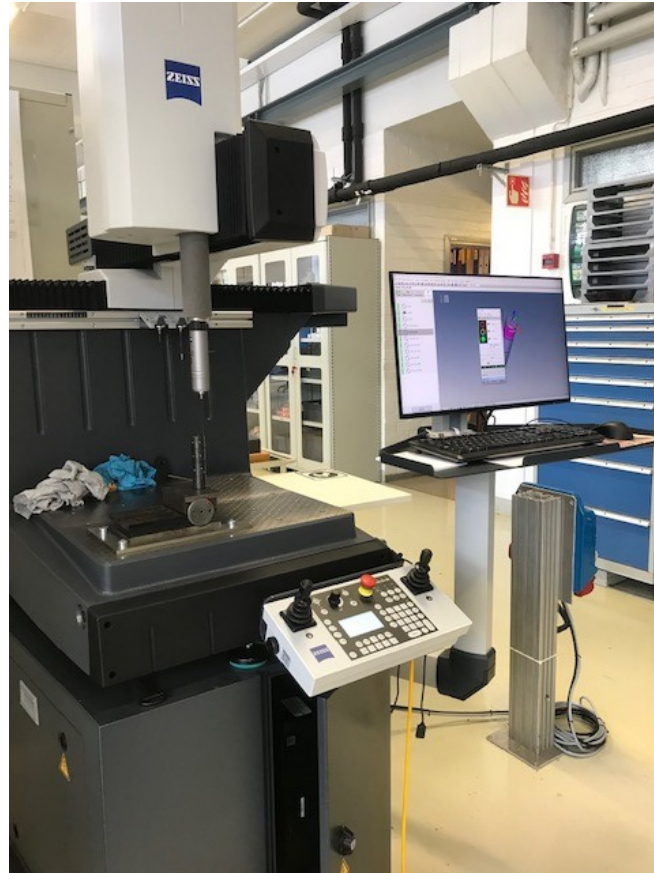


Figure 2. Zeiss CMM Setup.

The sensor of the CMM machine plays the most important role in inspection accuracy. As mentioned earlier, sensors come with different operating principles. Some of them physically touch the object to be inspected and others can perform the measurement without physical contact. Sensors that physically touch the object are widely used because they achieve the highest possible accuracy. However, compared to non-contact sensors, they are not only slow but also unsuitable for measuring objects that cannot be touched, for example, because of their soft structure. [4]

For non-contact sensors, on the other hand, there are many different measurement techniques. These measurement techniques include for example laser triangulation, various focalization technologies, confocal holography, and vision systems. With non-contact sensors, there are more situations where measurement fails. For example, deep holes can be impossible to measure. In addition, the accuracy of these sensors is not as good as that of touch sensors. In general, non-contact sensors are much faster and there are also situations where they are the only option, for example in the measurement of soft objects and printed circuits. [4]

The book Metrology 2006 presents non-contact measurement techniques used in CMM machines. According to the book, the most common metrology techniques used in those machines are the methods based on optical triangulation which includes for example laser triangulation, photogrammetry, and fringe projection. The great popularity of these methods is based on their cost-effectiveness, high accuracy, and high measurement speed. [10]

The working principle of triangular measurement methods is that two sensors are working together. One sensor is an active sensor that transmits light to the surface of the workpiece. The second sensor is a passive sensor, which does not emit any energy to the surface of the workpiece but records the position of the light transmitted to the surface of the workpiece. Once the distance between the sensors and the triangulation angle is known, the distance to the workpiece can be calculated using geometric formulas. [3]

2.3.2 Portable Scanners

3D scanning can be used for inspection in addition to the CMM machine with portable 3D scanners. The scanning process is quick, and the size of the product is not so limiting factor. Portable 3D scanning devices can be also integrated into CNC-controlled machine tools or robots to achieve an automated inspection solution [3]. They can also be used to scan products at different locations, which makes the process flexible, for example for maintenance applications [10].

The advantages of quality assurance with a mobile scanner are the same as with a non-contact CMM device, but without the constraints of the CMM machine's trajectories, making it even faster and more suitable for complex shapes. In addition, it is also significantly less expensive than CMM. However, a significant disadvantage is that the measurement accuracy is lower than the accuracy of CMM. In addition, it is more sensitive to ambient light than scanning-based CMM. [3, 10]

A wide range of scanners are available for many different industrial applications. Table 1 below shows the accuracies, resolutions, and scanning areas of some Artec 3D and Zeiss scanners. The difference between resolution and accuracy is important to understand. Accuracy refers to how close the measured points are to the actual surface of the measured part. Resolution refers to how detailed measurements can be taken which means the distances between measured points.

Table 1 shows that the properties of the scanners differ considerably. The Artec Micro II and the Metronom 1 scanners seem to be designed for precise scanning of small parts achieving a very high accuracy of 0,005 mm and

resolutions of 0,04 mm and 0,03 mm. [11, 12]. The Atos Q 12M can be moved by the user and allows a larger measuring area but the larger measuring area degrades the resolution to 0,12 mm [12]. The Artec Leo can be also moved by the user and has the largest measuring area but the accuracy of 0,10 mm and resolution of 0,20 are worse than the accuracies and resolutions of Artec Micro II and Metronom 1 scanners [11].

Table 1. Properties of Industrial 3D Scanners.

Scanner	Measuring Area (mm ²)	Accuracy (mm)	Resolution (mm)
Artec 3D: Artec Micro II	200 x 200	0,005	0,04
Zeiss: Atos Q 12M	100 x 70 – 500 x 370	-	0,03 – 0,12
Zeiss: Metronom 1	165 x 140	0,005	0,03
Artec 3D: Artec Leo	244 x 142 – 838 x 488	0,10	0,20

The scanners presented in table 1 can be found in Figure 3 below. On the left is the Atos Q 12 M, on the top right is the Artec Leo, on the lower left is the Zeiss Metronom 1, and on the lower right is the Artec Micro II.



Figure 3. The Industrial Scanners Presented in Table 1.

2.3.3 Point Cloud Processing

3D measurement device gives the measurements as point clouds [4]. Usually, more than one measurement is needed to pick up all desired features, and then occurs a problem where each measured point cloud is in a different coordinate system [13, 14]. For achieving a coherent point cloud, each point cloud needs to be connected to the same coordinate system. This process is called point cloud registration. The basic idea of point cloud registration is to pick up equal points or features from point clouds and calculate the transformation matrix to minimize the distances between them [14].

When all measurements are taken and registered with each other, the coherent point cloud needs to be processed before evaluating its dimensions and features. This processing means removing unneeded features and outliers. After that, the point cloud or the mesh made from the point cloud can be used for 3D inspection. For the 3D inspection procedure, software is used that calculates dimensions from the point cloud using form-fitting methods or comparing measured data to nominal CAD data using point cloud registration algorithms. [4, 15]

2.4 Point Cloud Registration Algorithms

Point cloud registration includes lots of different algorithms which can be roughly divided into global and local registration. Global registration methods give rough initial alignment as a result and they do not need any pre-alignment [16]. Local registration aims to optimize the registration as accurately as possible using initial alignment from global registration algorithms [16]. Usually, a combination of global and local registration algorithms is used to achieve the most accurate registration result [17]. In addition, neural networks can be used for end-to-end point cloud registration where the whole process is implemented at once [18].

This section provides information about registration algorithms commonly found in literature and explains their advantages and disadvantages. The chosen algorithms include global and local registration. In addition, end-to-end neural network-based algorithms are presented.

2.4.1 Random Sample Consensus (RANSAC)

Random sample consensus (RANSAC) is the most widely used robust matching method in computer vision [19]. RANSAC is based on a hypothesis and test tactic, where a minimum number of points are selected from the reference data and matched with corresponding points from the target data [19, 20]. A rigid transformation is then performed on the target data and the fit

of all points to the resulting model is checked based on their distances [21]. The fitting is repeated numerous times, and the best-fitting model is selected as the solution.

RANSAC is suitable for a wide range of applications, as it can process data that contains errors such as noise [16]. The RANSAC method distinguishes between inliers and outliers in scoring results [19]. Inliers points are points located at the required distance from the fit and are used to evaluate the results. Outliers are points that are far from the fit and are discarded in the evaluation of the results. Based on this, the RANSAC algorithm can reject the incorrect part of the data.

RANSAC is an efficient method for pre-registering point cloud data, but it needs a very large number of iterations to achieve accurate results [20]. Therefore, RANSAC is usually used for pre-registration, and optimization is performed by another method. In addition, the RANSAC method is challenging to register a large amount of data due to the amount of computation it requires [16]. However, several versions of the RANSAC method are available to reduce the need for counts [16].

2.4.2 Normal Distribution Transform (NDT)

Normal distribution transform (NDT) was originally developed for 2D data registration in 2003 [22]. In 2008, the method was extended to also work for 3D data [22]. The idea of the method is that instead of single points, the method uses a combination of normal distributions that describe the probability of finding a given point at a given location [23]. In 3D registration, the algorithm divides the point cloud model into cubes [24]. The algorithm then computes a mean vector and a covariance matrix for the reference point data in each cube, which are used to generate a separate normal distribution for each cube [24]. The target and the reference data are combined using numerical solution methods [24].

NDT is presented in the article provided by Gu et al. 2020 as a global registration method [17]. In many other articles, it is also considered suitable for local registration. For example, in the article provided by Magnusson et al. 2007, the NDT method was compared to the ICP method in a 3D mapping test [23]. The article recommended the use of a modified version of the NDT method due to its accurate results with a shorter registration time than ICP. However, the application discussed in the article was aimed at the registration of large point clouds and may give a misleading view of smaller-scale applications.

In conclusion, NDT is a potential method for a variety of point cloud registration applications. It is useful for handling large point clouds since recording normal distributions requires less memory than point clouds [23]. The previously mentioned study provided by Magnusson et al. 2007 also reported faster registration results compared to the ICP method [23]. However, the registration accuracy and speed of the NDT method are greatly affected by the size of the cubes [25]. Larger cubes give a faster process, but less accurate results, and smaller cubes do the opposite.

2.4.3 Iterative Closest Point (ICP)

Iterative closest point (ICP) is a widely used method for local point cloud registration [26]. It is an iterative method that aims to minimize the distances between the closest reference and target points until the given threshold is achieved [27]. According to the literature, the advantages of ICP are its accuracy and simple utilization [27, 28]. However, it also has some disadvantages. The main disadvantages of the algorithm are slowness and the need for good initial alignment [29].

Without good initial alignment, ICP can stuck into the local minimum when it tries to minimize the distances between closest points [29]. In addition, it needs lots of calculations because typically it uses all points of the point clouds [29, 30]. However, lots of different ICP variants are developed to speed up the calculation time. These variants exploit larger features instead of points. Two common variants of ICP are point-to-line and point-to-plane ICP algorithms [17, 26]. However, those variants can reduce the accuracy of the registration [17].

2.4.4 Coherent Point Drift (CPD)

Andriv Myrenko et al. 2006 presented a probabilistic Coherent Point Drift algorithm for registering non-rigid point clouds. Andriv Myrenko and Xubo Song 2010 presented their follow-up to the previous paper. This time, the CPD algorithm was brought to be used also for the registration of rigid point clouds. The idea of the CPD method is that the Gaussian Mixture Model (GMM) is fitted to the first point cloud, whose Gaussian centroids are fitted to the second point cloud data. The transformation uses a coherence constraint that forces the GMM centroids to move as a group to preserve the topological structure of the point data. [31, 32]

In both papers, experiments were conducted comparing the CPD algorithm with commonly used registration methods such as ICP. The results of the papers showed that CPD was more accurate than ICP when noise and missing points were included in the conditions.

2.4.5 Neural Networks

Neural networks are important to consider regarding point cloud registration. There are many other ways to use neural networks. End-to-end-based registration algorithms can implement the whole registration procedure at once. Feature-based learning methods can be used for accurate preregistration when the optimization can be done for example using RANSAC with just a few iterations. That kind of combination is useful for large point clouds where Ransac alone is too slow. One way is also to use neural networks for preregistration and implement the fine registration by commonly used fine registration methods such as ICP. [18]

Akiyoshi et al. 2020 presented an efficient registration method based on neural networks [33]. The method used PointNet for point cloud detection and implemented fine registration itself. PointNet is a widely used object detection method due to its high accuracy. However, the development of the methods is fast and there are more accurate methods available. ModelNet40 and ModelNet40-C keep the lists of the most accurate point cloud detection methods including their academic papers [34, 35]. On these lists, neural networks are widely used in the most accurate methods.

The article compared ICP, and several registration methods based on neural networks. Results of the article showed that neural networks-based methods had more accurate results than ICP. Wen-Chung Chang et al. 2019 have also compared the neural networks- and ICP [36]. Their investigation work showed that convolutional neural networks can reduce the time of registration keeping the accuracy at the same level as ICP.

Neural networks seem to be state-of-the-art methods for point cloud registration because of their robustness, accuracy, speed, and possibility to implement the registration process at once [18, 33, 36]. However, there are some limitations regarding the use of neural networks which are the fact that they need training and registration results can decrease in unknown conditions [18]. In addition, considering local structure information can be difficult [18, 33].

2.5 Point Cloud Cylinder Fitting Algorithms

Cylinder fitting is a widely used method in point cloud processing and there are lots of different algorithms for it. From a perspective of 3D inspection, the most used algorithms are Least Squares (LS) also known as Gaussian fit and Minimum Zone (MZ) also known as Chebyshev fit [4]. In addition, Ransac is commonly used in applications where outliers are faced [37].

3D inspection has also cases where filtering of suitable points is essential. For example, when the shaft and hole are fitted, measuring can aim to investigate the maximum or minimum values of the diameters. Therefore, the Maximum Inscribed Element (MIE) and Minimum Circumscribed Element (MCE) are important to consider regarding 3D inspection and cylinder fitting.

2.5.1 Least Squares (LS)

LS is a widely used cylinder fitting method in different applications in 3D inspection [4, 38]. The idea of LS is to minimize the sum of square error between measured points and fitted form [4]. One major advantage of LS is that it gives a good overall picture of the inspected dimension due to the consideration of all measurement points. In addition, LS is more robust for dimensional error of measurement points than MZ and therefore it is preferred for dimensional inspection [4]. Figure 4 below shows the weighting of measurement points of the LS algorithm.

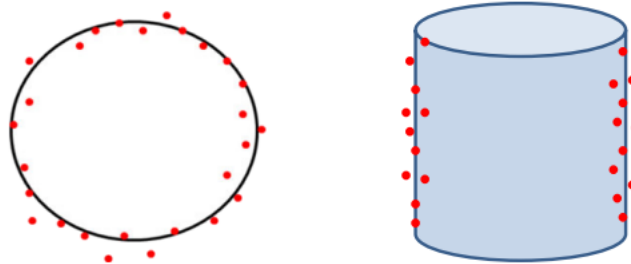


Figure 4. LS Cylinder Fitting [39].

2.5.2 Minimum Zone (MZ)

MZ fits two cylinders with the same centre points into the point cloud. One cylinder is fitted inside the point cloud with maximum diameter and the other outside the point cloud with minimum diameter. Using the cylinders with the maximum and minimum diameters the algorithm enables also a cylinder to be fitted in the middle of them. In addition, MZ is found to be more suitable for inspection of geometrical tolerances than the least squares method but one significant disadvantage of MZ is sensitivity for outliers. Figure 5 below illustrates the working principle of the algorithm. [4, 39]

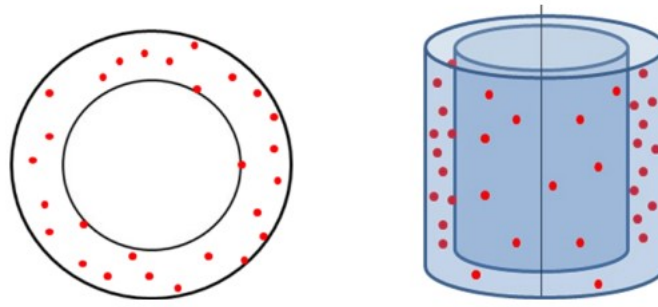


Figure 5. MZ Cylinder Fitting [39].

2.5.3 Maximum Inscribed Element (MIE)

The idea of MIE cylinder fitting is to fit a cylinder inside a point cloud with a maximal diameter [40]. The difference between the MIE cylinder and the smaller cylinder of MZ is the different centre points due to the equality of the centre points of MZ cylinders [4]. MIE cylinder fitting is needed in different functional fitting applications, but the disadvantages are sensitivity for outliers and an inaccurate overall picture of the measured diameter [4, 39, 40]. Figure 6 shows the working principle of MIE cylinder fitting.

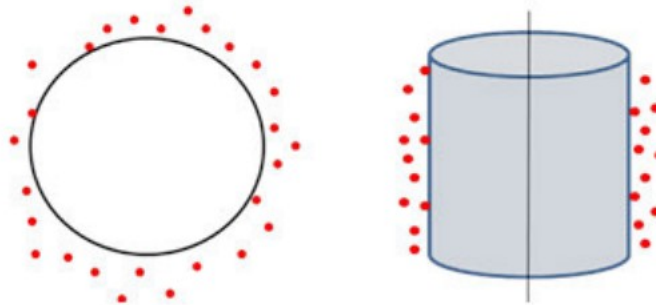


Figure 6. MIE Cylinder Fitting [39].

2.5.4 Minimum Circumscribed Element (MCE)

MCE is another useful method for different functional fitting applications. The method gives the smallest cylinder that covers all measurement points [40]. The difference between the MCE cylinder and the MZ larger cylinder is the different positioning of centre points [4]. The disadvantage of the method is the inaccurate overall picture of the measured diameter and sensitivity for outliers [4, 39, 40]. Figure 7 below shows the working principle of MCE cylinder fitting.

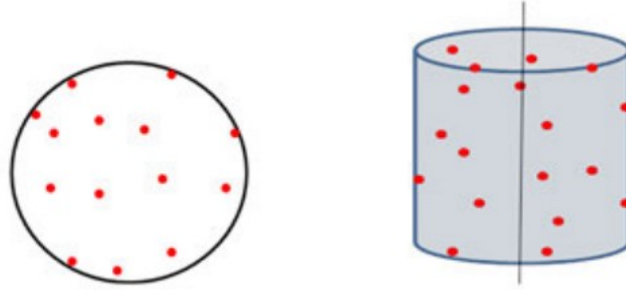


Figure 7. MCE Cylinder Fitting [39].

2.5.5 Ransac

Previously presented Ransac regarding point cloud registration is also used in different point cloud cylinder fitting applications. The advantages of Ransac are its robustness for outliers and calculation efficiency. However, in cylinder fitting the strengths of Ransac are in applications where noisy point clouds are handled without high accuracy requirements. [37, 41]

2.6 Python Libraries for Point Cloud Registration

This section provides two Python open-source libraries Open3D and Learning3D for point cloud registration. Open3D provides Ransac for global registration and ICP for local registration and Learning3d focuses on neural networks-based registration methods. According to the literature research, these algorithms cover the most used algorithms and the state-of-the-art algorithms to be investigated.

2.6.1 Open3D

Open3d is an open-source library that can be used in Python and C++. The library includes many features for handling 3D data such as point clouds and mesh structures. For point cloud registration Open3D provides Ransac for global registration and ICP for local registration. Open3D has also visualization tools that can be used for both point clouds and mesh structures. [42]

2.6.2 Learning3D

Learning3D is an open-source library for Python that provides neural network algorithms for different 3D data processing applications. The library provided PointNetLK, PCRNet, DCP, PRNet, RPM-Net, and DeepGMR

algorithms for point cloud registration. In addition, it had two options for shape detection which were PointNET and DGCNN algorithms. [43]

Learning3D provided also pre-trained models of registration algorithms but allowed also the implementation of training by user. The library had comprehensive instructions on its websites and there were also ready-made codes for different applications including point cloud registration. [43]

2.7 Python Libraries for Cylinder Fitting

This section provides available cylinder fitting libraries for Python. It was noteworthy that there were not many libraries available for that purpose and only two other libraries were found. One of the libraries used the Ransac algorithm and the other used the LS algorithm. The algorithms were in line with the literature about the most used algorithms for cylinder fitting.

2.7.1 PyRANSAC-3D

PyRANSAC-3D is an open-source library that uses the Ransac algorithm for fitting different forms into point clouds. Provided fitting shapes of the library were plane, cylinder, cuboid, sphere, line, circle, and point. However, there was a warning on the library website for the cylinder fitting. The warning informed that the development process of the cylinder fitting algorithm was still in progress and the algorithm did not provide accurate results. [44]

2.7.2 Py-cylinder-fitting

Py-cylinder-fitting is an open-source library for Python, and it provides a cylinder fitting method based on the LS algorithm. On the website of the library readymade codes for cylinder fitting can be found. The method needs coordinate points of the point cloud as input and it returns the radius of the fitted cylinder. [45]

3 Python Workflow for 3D Inspection of Machined Tolerance Bar

This section provides a case study for 3D inspection of a machined cylindrical tolerance bar using Python and its libraries for point cloud registration and cylinder fitting. The 3D inspection of the tolerance bar was focused on four different diameters with the same nominal dimension but different tolerances. In addition, the case study provides comprehensive instructions for

any parts of the process including designing, manufacturing, scanning, and validation of results.

3.1 Design

The design process was implemented using Creo software for 3D modelling. The design was a simple machining part that was designed to be manufactured by turning and milling. The design included four diameters with the same nominal dimension 25 mm but different tolerances. The diameters were separated by 2 mm wide grooves. In addition, a keyway was designed at the other end of the tolerance bar.

The most important features were the different diameters inspected using the Python workflows. The grooves were just designed to make the turning process easier and the keyway for making it easier to visualize the tolerance bar during 3D inspection. Later, this thesis provides the inspection results using numbers 1-4 of the inspected diameters. The numbers and the design can be seen in Figure 8 below.

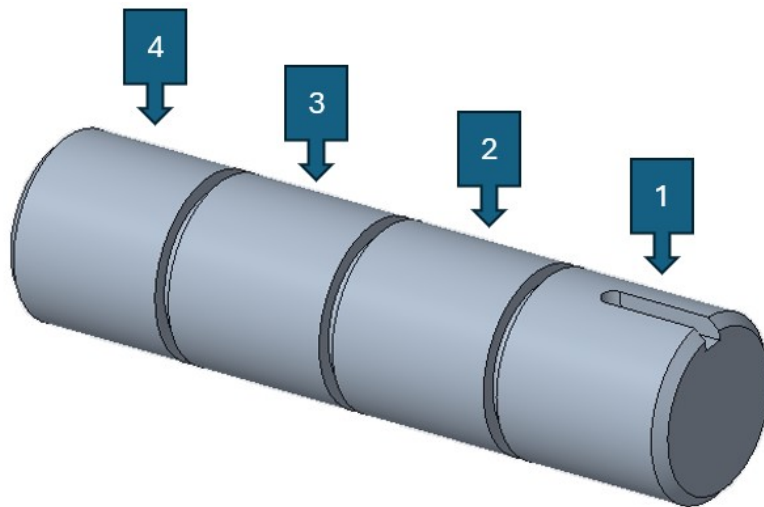


Figure 8. Diameter Numbers of the Tolerance Bar.

When designing for manufacturing, it needs to be considered that all dimensions are tolerated, and all features are possible to manufacture by the desired manufacturing method. In the case of the tolerance bar the width and radius of the keyway needed to be dimensioned in such a way that the keyway was able to be milled using an end mill with a diameter of 3 mm. Figure 9 below shows the drawing of the tolerance bar. The dimensions of the keyway and all other features can be seen in Figure 9. In addition, it can be seen that there are specific tolerances given for the diameters and all other dimensions are tolerated using the general tolerances.

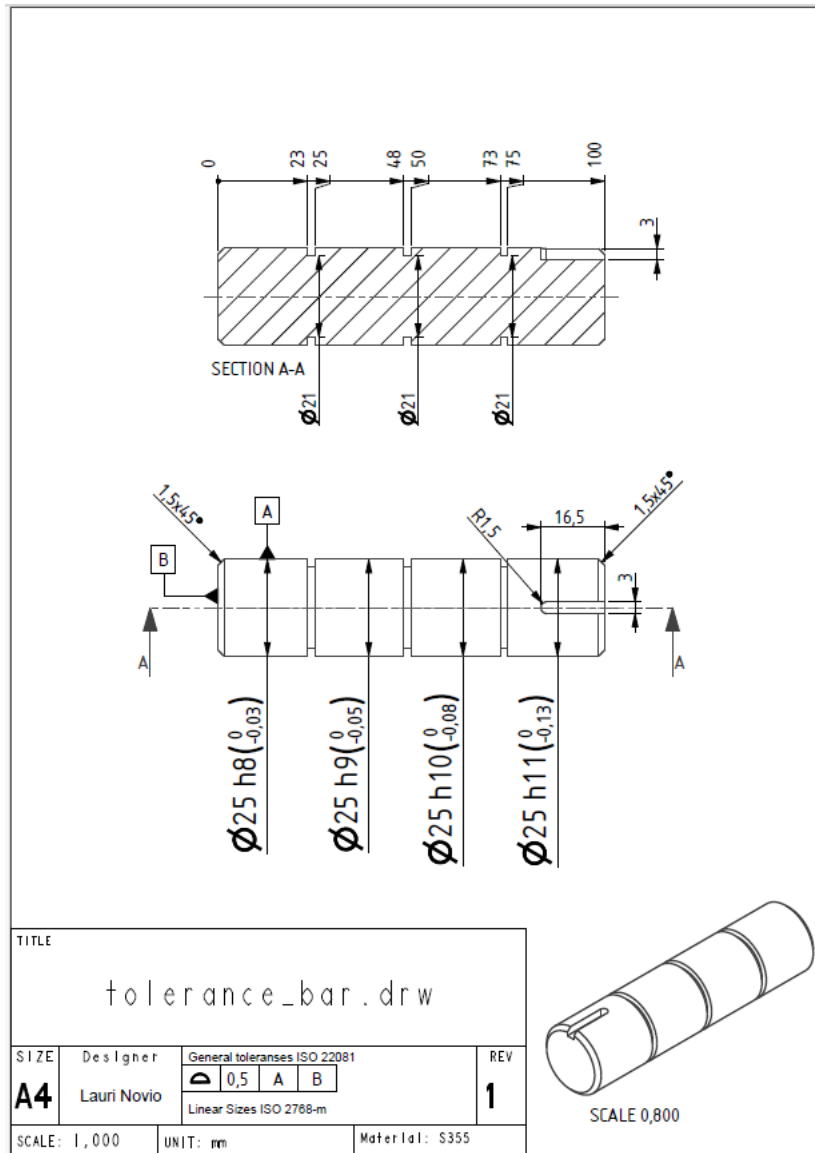


Figure 9. Drawing of the Tolerance Bar.

3.2 Manufacturing

Manufacturing was implemented by turning and milling using S355 as a raw material. In addition, coating was conducted using blackening. The purpose of the coating was to make scanning process easier regarding less of lightning from the coated surface.

The first stage in the manufacturing process was turning. Turning was implemented using a manual lathe. The steps of the process were roughing, single cut grooving, finishing, and chamfering. In addition, at the beginning of the process, a small hole was drilled at the end of the bar for fixturing the

workpiece as shown in Figure 10. The purpose of the fixturing from both ends was to reduce vibration and deflection of the workpiece to maximize the straightness of the final product



Figure 10. Turning of the Tolerance Bar.

Turning parameters were essential for a successful turning process. There were formulas available to calculate those parameters using estimation values for certain parameters provided by tool insert manufacturers. The following calculations provide turning parameters for the tolerance bar using tools provided by Sandvik Coromant. However, the used formulas are intended for advanced CNC lathes and may give too high values for old manual lathes. In the manufacturing process, a manual lathe was used and a machinist familiar with the lathe adjusted the machining parameters to be suitable. In addition, the turning inserts presented in this thesis were not used in the turning process but offer a good alternative.

The required turning parameters were the depth of cut, feed, and spindle speed. Estimation values for cutting speed, feed, and depth of cut were found from the information of the tool inserts. In addition, the spindle speed was able to be calculated using the value of the cutting speed. The calculation was implemented using the following equation where n was spindle speed, v_c was cutting speed, and D_m was the diameter of the workpiece [46].

$$n = \frac{v_c \times 1000}{\pi \times D_m} \quad (1)$$

Turning includes four steps which are roughing, single cut grooving, finishing, and chamfering. The inserts proposed for the turning steps can be seen in figure 11 below. In addition, Sandvik Coromant provides an estimation of cutting speed, depth of cut, and feed for the inserts. These values and the insert codes can also be found in figure 11 below.

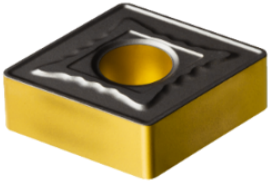

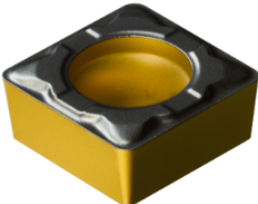
Roughing	Single cut Grooving	Finishing and Chamfering
		
Depth of cut: 4mm Feed: 0,5 mm/r Cutting speed: 280 m/min	Feed: 0,08 mm/r Cutting speed: 220 m/min	Depth of cut: 0,3 mm Feed: 0,11 mm/r Cutting speed: 295 m/min Insert shape code: Square
CNMG 12 04 08-MR 4425	QD-NE-0200-0002-CM 4425	SCMT 09 T3 04-PF 1515

Figure 11. Turning Inserts Provided by Sandvik Coromant.

The first step of the turning process is roughing. It can be implemented at the same time for all diameters. A suitable initial diameter of the workpiece is 30 mm. Roughing can be done by one pass with a 2 mm depth of cut when the diameter decreases to 26 mm. The next step is single cut grooving where the initial diameter is 26 mm, and the final diameter is 23 mm. Table 2 below shows the turning parameters for roughing and single cut grooving. The parameters include the parameters provided by Sandvik Coromant and the spindle speeds calculated using equation 1.

Table 2. Turning Parameters for Roughing and Single Cut Grooving.

	Roughing	Single cut grooving
Depth of cut	2 mm	-
Feed	0,5 mm/r	0,08 mm/r
Cutting speed	280 mm/min	220 m/min
Spindle speed	2971 r/min	2693 r/min

The final steps of turning are finishing and chamfering which can be done using the same insert. The square shape of the insert enables chamfering at 45 degrees and therefore saves time regarding no need to change the insert. Due to the tolerances of the diameters, turning parameters for finishing need to be calculated carefully. For chamfering the parameters are not so precise and the finishing parameters can be used.

The tolerance bar includes four different diameters. Each diameter has its tolerance and finishing needs to be done separately for each. The idea is to aim for the middle of the tolerance zone with each diameter. Table 3 below shows the finishing parameters of the turning process. Noteworthy is that the depth of cut has been calculated to achieve the middle dimension of the tolerance area by two passes. In addition, the initial workpiece diameter (26 mm) has been used for calculating the depth of cut.

Table 3. Turning Parameters for Finishing and Chamfering.

	Diameter 1	Diameter 2	Diameter 3	Diameter 4
Depth of cut	0,266 mm	0,260 mm	0,256 mm	0,254 mm
Feed	0,11 mm/r	0,11 mm/r	0,11 mm/r	0,11 mm/r
Cutting speed	295 m/min	295 m/min	295 m/min	295 m/min
Spindle speed	3612 r/min	3612 r/min	3612 r/min	3612 r/min

After turning, the keyway was manufactured using a manual milling machine. As in the turning process, a machinist familiar with the milling machine adjusted the machining parameters to be suitable. However, this section provides also the milling parameters for milling the keyway with a CNC milling machine. The parameters have been calculated with the Sandvik Coromant machining parameter calculator, by giving the calculator the desired tool. The tool chosen was a Sandvik Coromant end mill with a 3 mm diameter. However, the same tool was not used in the manual milling process of the tolerance bar. Figure 12 below shows the milling process using the manual milling machine.



Figure 12. Keyway Milling Process.

The end mill selected for the parameter calculation can be seen in figure 13 below. In addition, the code of the tool can be found from the figure 13.

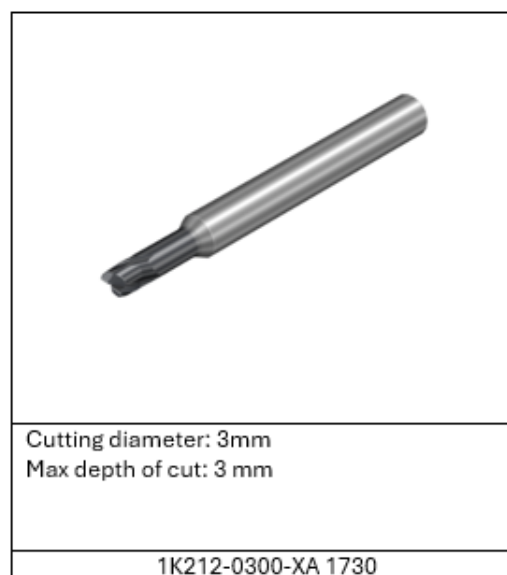


Figure 13. End Mill Provided by Sandvik Coromant.

The tool did not have any estimated milling parameters which made the parameter calculation with general equations impossible. The reason for the missing estimated parameters is that the tool can be used for numerous milling processes for different features using many other materials. However, Sandvik Coromant provided a parameter calculator where information about the feature to be milled and used material was able to be set. The following table 4 shows the milling parameters calculated using the calculator.

Table 4. Keyway Milling Parameters.

Keyway Milling	
Table feed	493 mm/min
Cutting speed	180 m/min
Number of passes	1
Depth of cut	3 mm
Spindle speed	19100 r/min

The last step of manufacturing was coating, and it was conducted using the blackfast method. Blackfast is a chemical process where a metal part is immersed in a chemical liquid, creating a black oxide layer on the surface of the part. The process is possible to implement at room temperature and the layer creates protection against corrosion and humidity. However, in this case study, the layer was desired for decreasing the lightning from the surface during 3D scanning. In addition, blackfast seemed to be a suitable coating method regarding the thin surface thickness of only 0,0002 – 0,003 mm. The blackened tolerance bar can be seen in Figure 14 below. [47]



Figure 14. The Coated Tolerance Bar.

3.3 Scanning

Scanning was implemented using a Gom ATOS Core 200 3D scanner. The first step of the process was to let the scanner warm up and take calibration measurements. The calibration included scanning a calibration template from different positions and the scanning software gave comprehensive instructions to the process. Figure 15 below shows the scanner and the calibration plate.

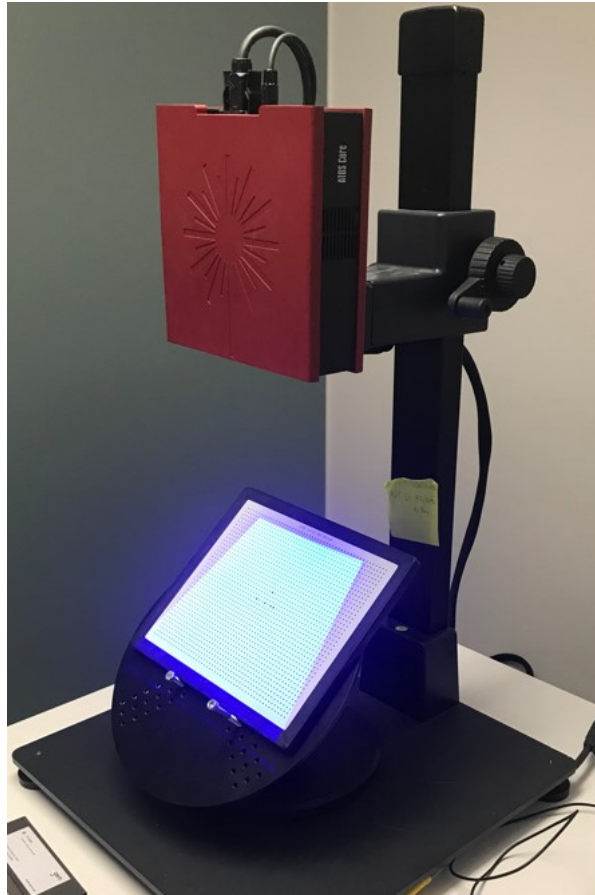


Figure 15. Calibration of the Gom Scanner.

When enough scans were taken the program informed that the calibration was completed and gave calibration results. The calibration results showed that the calibration deviation was 0,052 pixels which was lower than the limit value of 0,100 pixels. The whole calibration results can be found in Figure 16 below.

Current Calibration Info	
General	
Calibration date	Fri May 10 10:26:40 2024
Calibrated sensor	ATOS Core 200
Calibration object	
Object type	Panel (Triple Scan)
Name	CP40-200-100369
Calibration points	3657 points
Certified lengths	360.000 / 360.000 mm
Certification temperature	22.0 °C
Expansion coefficient	3.25 x 10 ⁻⁶ 1/K
Measurement temperature	23.0 °C
Calibration settings	
Camera lenses	12.50 mm
Focal length (projector)	8.00 mm
Light intensity	100%
Snap mode	Double snap
Ellipse quality	0.4
Calibration Result	
Calibration deviation	0.052 Pixels
Calibration deviation (optimized)	0.015 Pixels
Calibration deviation (check)	OK (limit value: 0.100 Pixels)
Projector calibration	0.122 Pixels
Projector calibration (optimized)	0.015 Pixels
Projector calibration (check)	OK (limit value: 0.250 Pixels)
Camera angle	24.2°
Height variance	148 mm
Measuring volume	195 / 150 / 155 mm
Sensor status	
Remaining sensor warm-up time	0:00 min

Figure 16. Calibration Results.

After calibration scanning was implemented for the tolerance bar which was prepared by cleaning its surfaces using ethanol. The position of the tolerance bar proved to be important for scanning results. First scans were taken by holding the scanning plate horizontally. The results of those sets were not proficient regarding the reflection of ambient light. Figure 17 below shows the camera picture of the scanner and scanned points when the scanning plate was placed horizontally. The red areas in the camera picture mean a reflection of ambient light. It was noteworthy that there were holes in the scanned point data at the same places where the reflection of ambient light occurred.

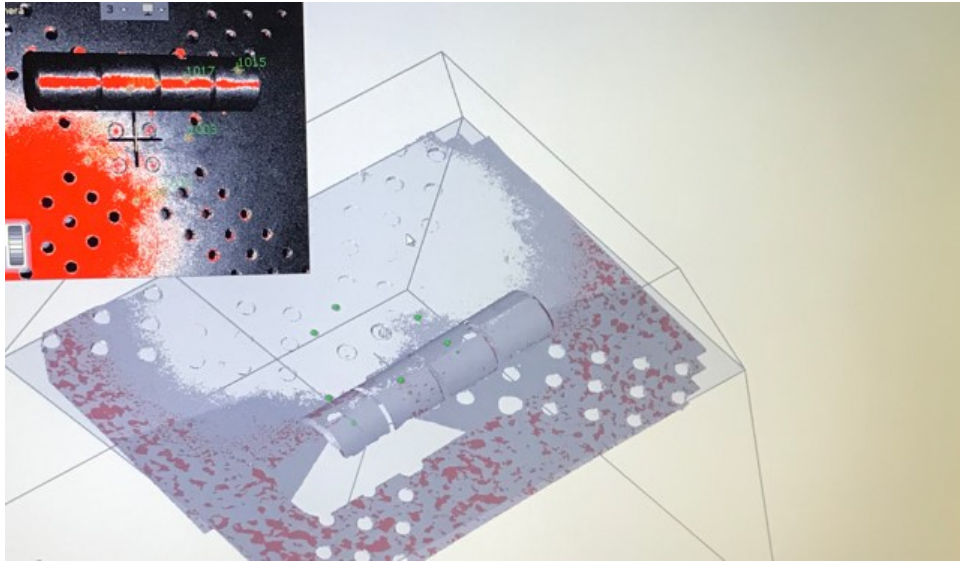


Figure 17. Scanner's Camera Picture and Scanned Points.

Changing the position of the tolerance bar to be more vertical improved the scanning efficiency a lot. It decreased the effect of ambient light and enabled capturing lots of more points in each scan. The better scanning position can be seen in the Figure 18 below.

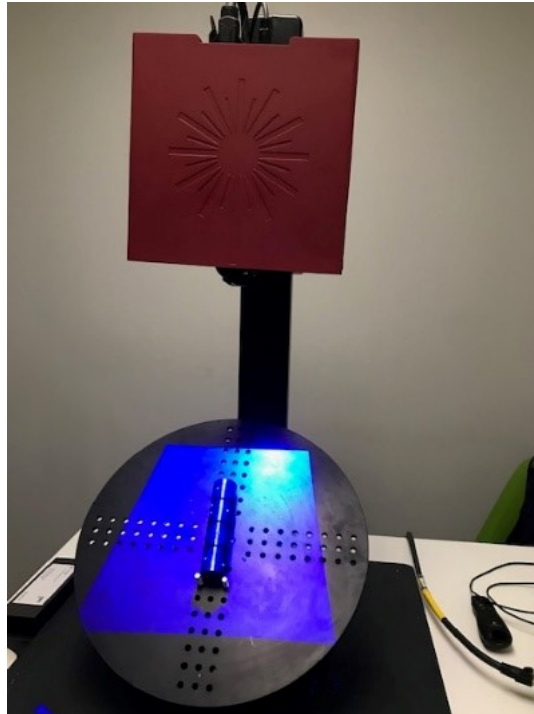


Figure 18. An Efficient Scanning Position of the Tolerance Bar.

The GOM scanning software had a function for the registration of scans together. The requirement for the registration process was that reference markers with a white centre and black background were placed on the surface of the tolerance bar. The idea of those reference markers was to help the scanner identify the position and location of the tolerance bar. It was also important to put enough reference markers to ensure that the scanner sees some same reference markers during different scanning sets. The reference markers on the surface of the tolerance bar can be seen in the previous Figure 18 above.

After achieving enough scanned surfaces, all unnecessary features from the point cloud needed to be removed and mesh generation done before saving the final STL file. Removing the unnecessary features was done using a tool provided by the GOM scanning software. The working principle of that tool was to delimit an area with the drawing tool and choose whether to delete the delimited area or the area outside it. With the possibility to change the scale of the point cloud to extremely large, the tool was efficient and simple. Finally, the mesh file was generated from the cleaned point cloud using the mesh generation feature of the GOM scanning software. The final mesh file can be seen in Figure 19 below.

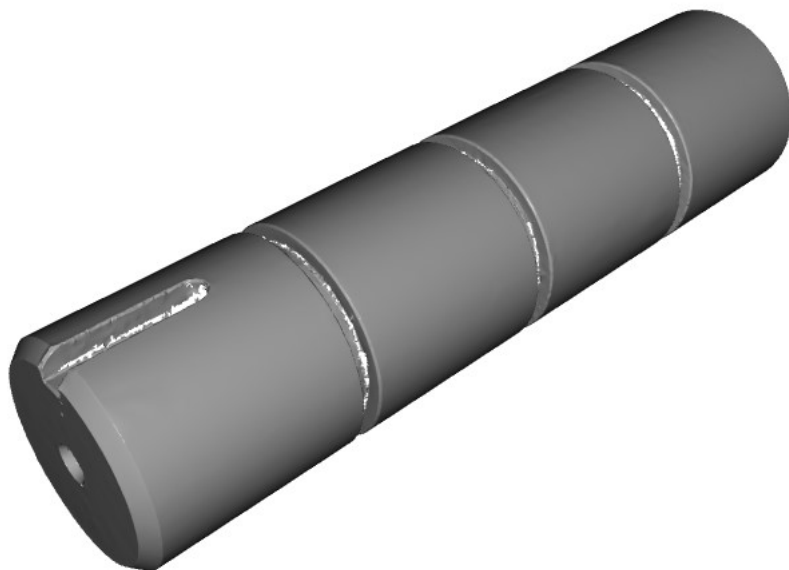


Figure 19. The Final STL File.

3.4 Python Workflow for 3D Inspection

This section provides two Python workflows for 3D inspection of the tolerance bar diameters. Workflow 1 was created using point cloud registration and surface comparison and workflow 2 using cylinder fitting. The workflows are explained in this section and the codes of them can be found in the appendixes.

Workflow 1 used the Ransac + ICP combination of registration algorithms. That combination was selected according to the literature review and the comparison of the Ransac, ICP, and PointNetLK algorithms with different combinations. Workflow 2 used the LS algorithm. It was selected according to the literature review and sensitivity analysis where it was compared to Ransac. The sensitivity analysis and the comparison of the registration algorithms are included in this section.

3.4.1 Workflow 1. Surface Comparison

The first step of the workflow was to import STL files of the CAD data and the scanned data. Figure 20 below shows the STL files which are in the different coordinate systems.

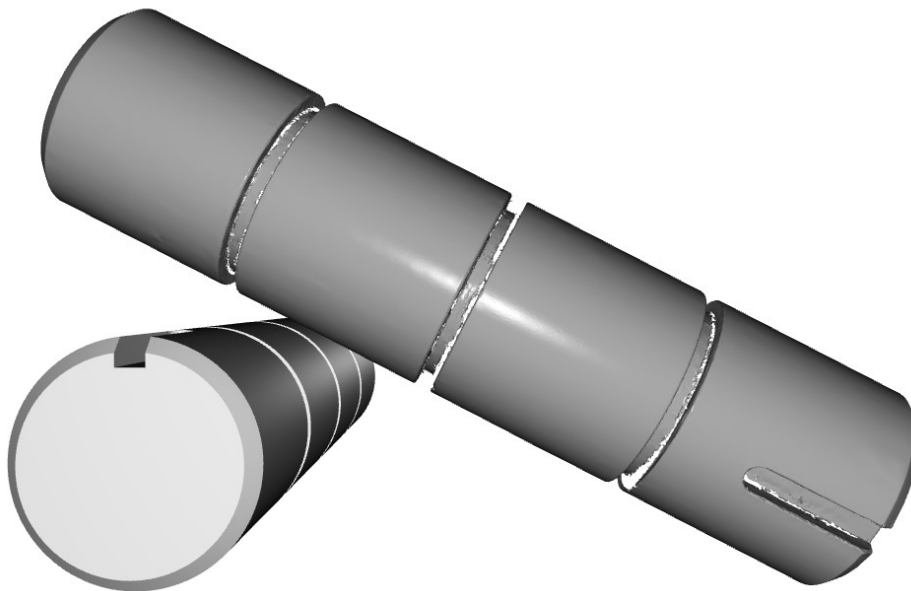


Figure 20. The Initial Position of the STL files.

The next step was to convert the STL files into point clouds and it was done using the Open3D library. The library offered the possibility to choose the number of points in the point clouds. The number of points played an important role in point cloud registration and visualization of results. A high number of points increased the accuracy of the registration and gave better visualization results, but it also increased the time needed for the calculation. Figure 21 shows the downsampled point clouds with 100 000 points.

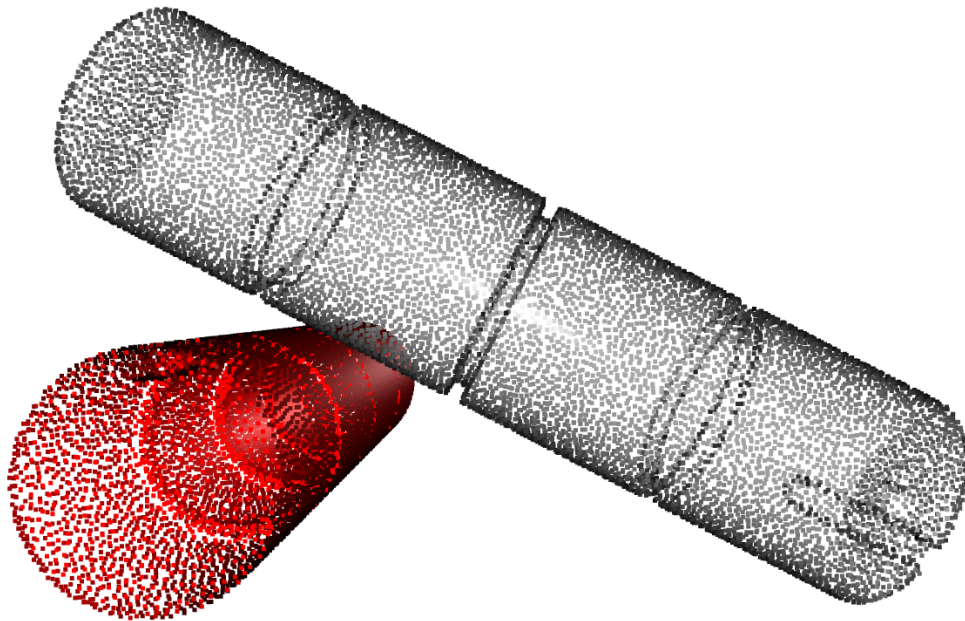


Figure 21. Downsampled Point Clouds.

After downsampling, the point cloud registration was implemented, resulting a transformation matrix for transforming the source point cloud on the template point cloud. The registration was implemented using Ransac for global registration and ICP for finishing. Figure 22 below shows point clouds registered by Ransac + ICP algorithms. The red point cloud is the CAD data (template) and the grey point cloud is the scanned data (source).

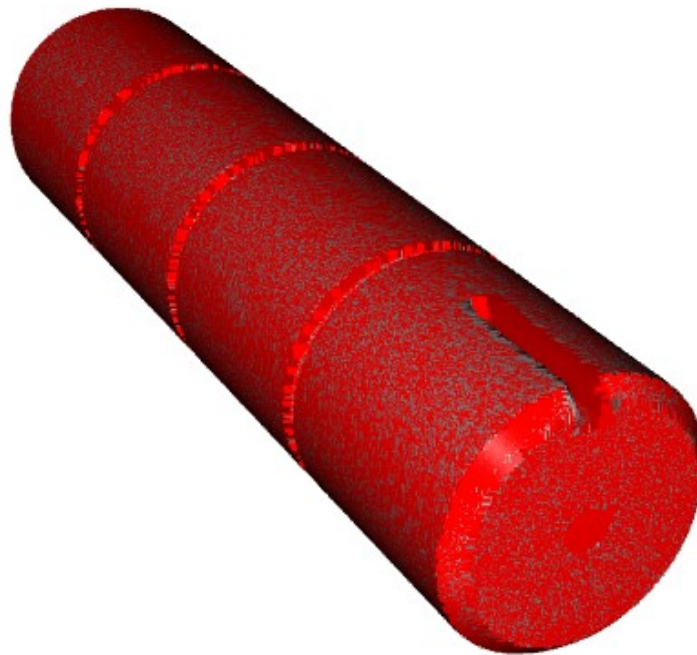


Figure 22. Ransac + ICP Point Cloud Registration Result.

After registration, a surface comparison of the whole template and source models was done to show how well the registration was successful. First, the transformation matrix obtained from the registration was used to translate the source point cloud onto the template STL file. A different number of points was possible to be selected from the source point cloud than in the registration processes.

Next, the distance from each point in the source point cloud to the template surface was calculated. For calculating the distances, a Pyvista library with Euclidean distance algorithm was used and it allowed to identify the position of the points inside or outside the template model. Points on the inside of the template were assigned negative values for the distances. Figure 23 shows the results of surface comparison of the whole models.

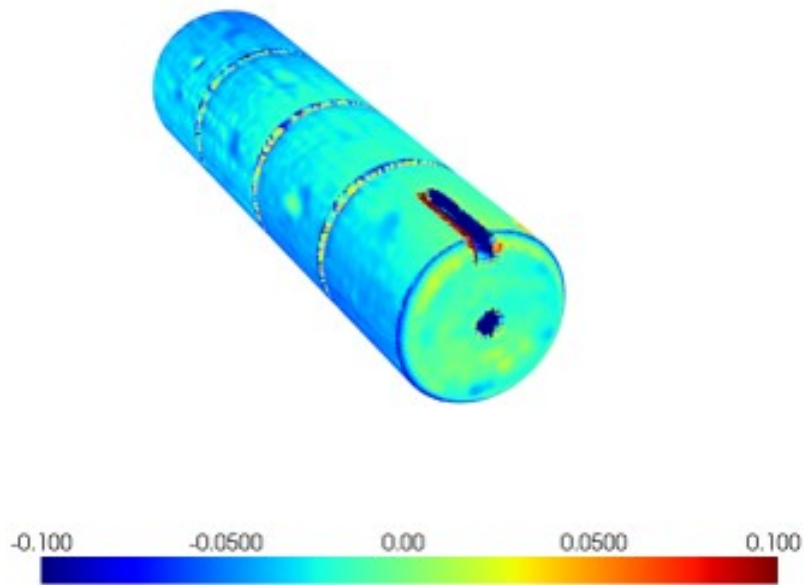


Figure 23. Surface Comparison Between Source Point Cloud and Template Mesh.

The next step was to select the desired diameter from the scanned point cloud. It was conducted by a box widget clipping tool provided by the Pyvista library. Figure 24 below shows the Pyvista box widget clipping tool.

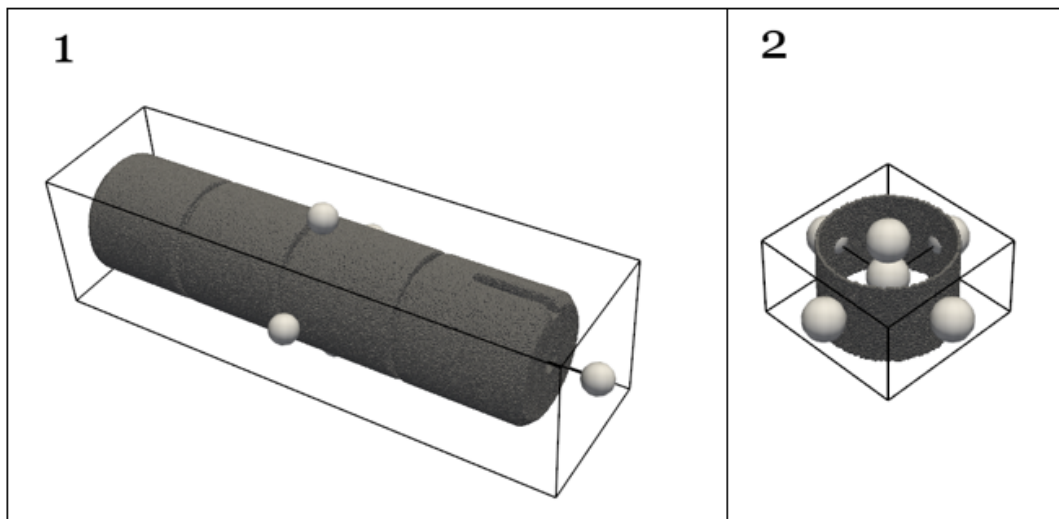


Figure 24. Selection of Desired Area of the Tolerance Bar.

Then a surface comparison between the clipped diameter and the template STL file was implemented. The surface comparison gave average, min, and max distances which could be used to calculate the diameter value using equation 1. In addition, outlier removal provided by Open3D was performed before the surface comparison. The idea of the outlier removal was to delete all erroneous measurement points that could distort the diameter value. Figure 25 below shows the outlier removal on the left and the surface comparison on the right. In the outlier removal figure, the red points are the outliers to be removed.

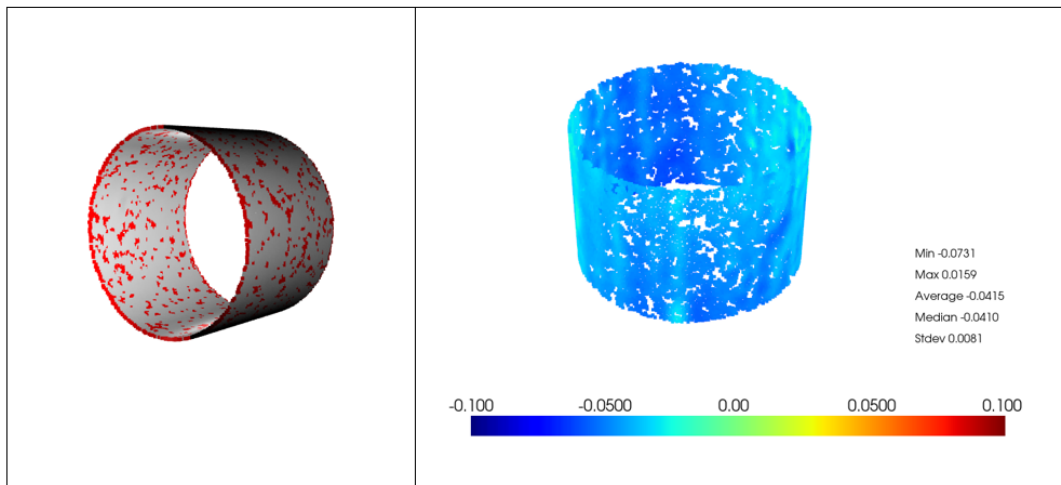


Figure 25. Outlier Removal and Surface Comparison of the Diameter 4.

The last step of the workflow was to calculate the diameter using the following equation 2. In the calculation, it was possible to calculate the minimum, maximum, or average diameter of the point cloud. However, the average diameter was used in comparison to other software due to its robustness and better overall picture of the diameter. The Python code of the workflow 1 can be found in Appendix A.

$$\text{Diameter} = \text{Nominal Diameter} + 2 \times \text{Average or Min or Max Distance} \quad (2)$$

3.4.2 Selection of Registration Algorithm

This section provides a selection process of the point cloud registration algorithm for the previously described workflow 1. The investigated algorithms were Ransac, ICP, and PointNetLK which were selected based on the literature review. The comparison of the algorithms was implemented using a surface comparison of the whole CAD and scanned models and of the whole CAD model and clipped diameter 4.

The first test was performed on the Ransac algorithm. Ransac managed to extract also local features such as keyway for the registration. However, the registration accuracy was not sufficient for the Inspection purpose. In Figure 26 below on the left are the point clouds registered to each other, where the template point cloud is red, and the source point cloud is grey. On the right are the surface comparison results where the left one is performed on the whole source point cloud and the right one on the diameter 4. The figures show that the registration was successful regarding shape detection but overall, the point clouds were skewed with each other. The Ransac workflow can be found in Appendix B.

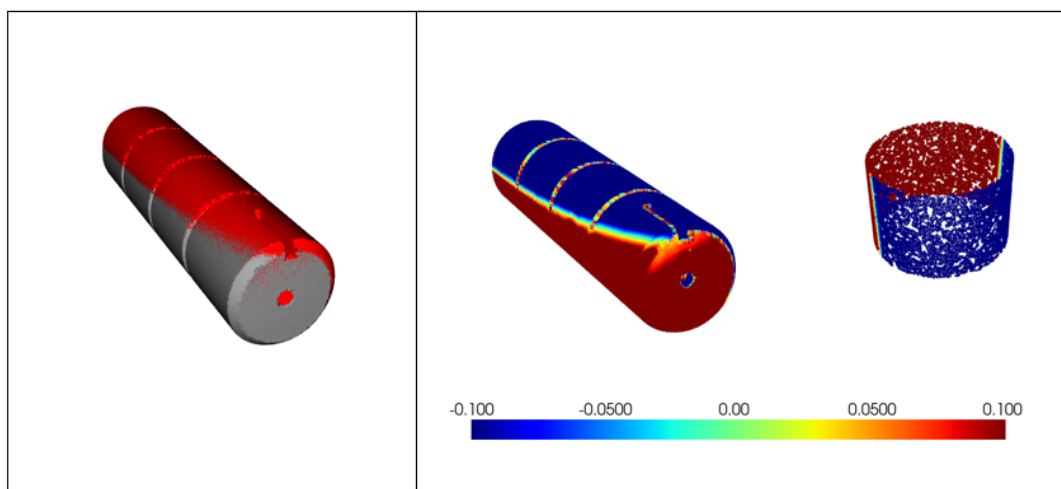


Figure 26. Surface Comparison in Millimetres Using Ransac.

The following test was performed on the PointNetLk algorithm. Figure 27 below shows that regarding global features the point clouds were registered more accurately to each other than by Ransac. However, there were some serious errors with the registration results. The algorithm did not consider the keyway in the registration and from Figure 27 below it can be seen that the keyways are on different sides in the registration results. The PointNetLK workflow can be found in Appendix C.

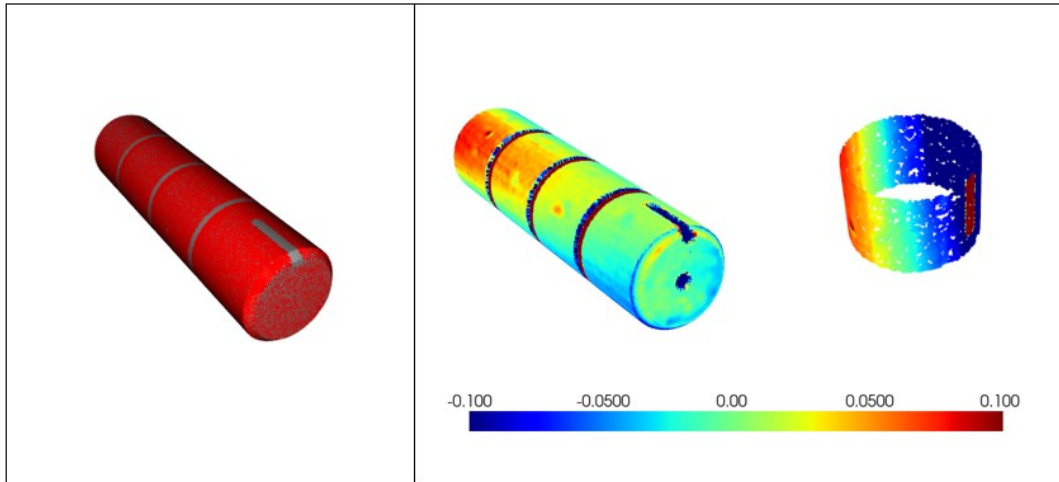


Figure 27. Surface Comparison in Millimetres Using PointNetLK.

Then the Ransac + ICP combination was investigated. In Figure 28 below, the registration results were significantly better than the previous results. The registration succeeded in considering local features such as the keyway and globally the point clouds were registered accurately and in a straight line. This combination was selected for workflow 1 and can be found in Appendix A.

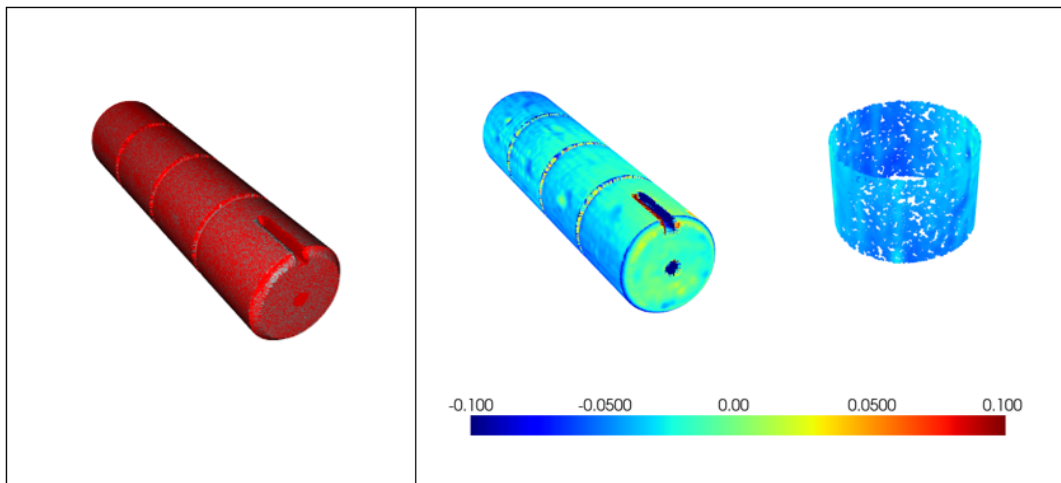


Figure 28. Surface Comparison in Millimetres Using Ransac + ICP.

The next registration procedure was implemented for a combination of Ransac and PointNetLK. Figure 29 below shows the registration results. The results were significantly better than with Ransac or PointNetLK alone and the registration was successful in extracting local and global features. However, the registration results were much more accurate with Ransac + ICP combination. The Ransac + PointNetLk workflow can be found in Appendix D.

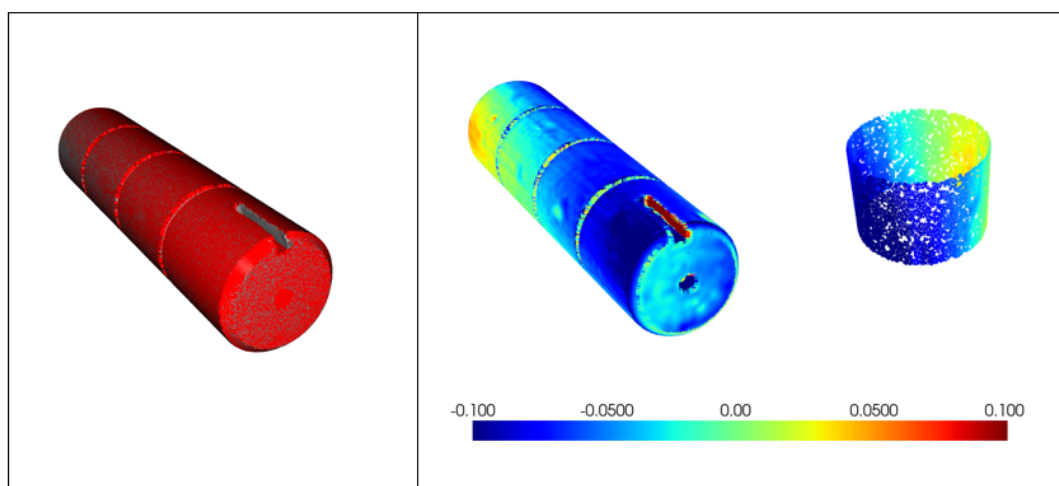


Figure 29. Surface Comparison in Millimetres Using Ransac + PointNetLK.

The last registration test was implemented for the combination of all algorithms Ransac + PointNetLK + ICP. Figure 30 below shows that the registration was successful in extracting local and global features. The Ransac + PointNetLk + ICP workflow can be found in Appendix E.

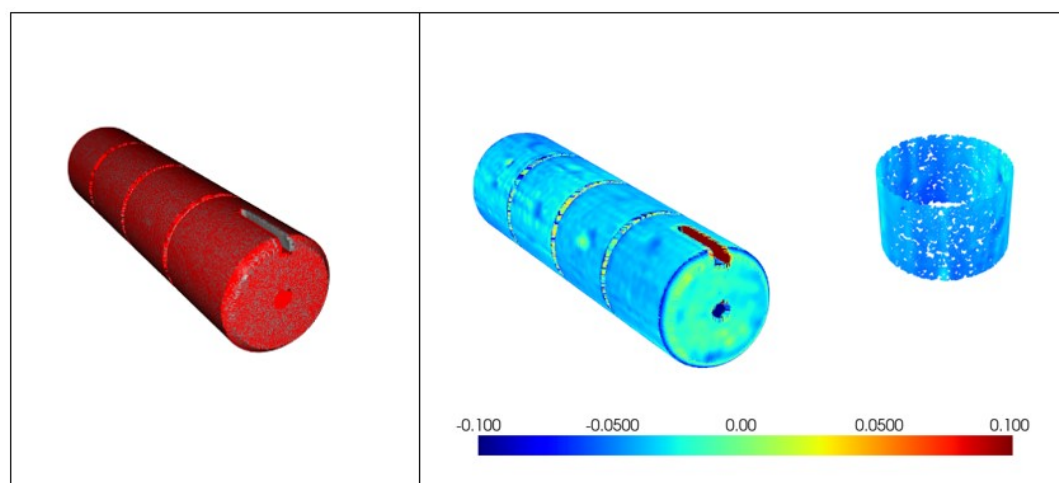


Figure 30. Surface Comparison in Millimetres Using Ransac + PointNetLK + ICP.

The results of Ransac + ICP and Ransac + PointNetLK + ICP were so close that they could not be compared based on visual presentation alone. The following figure shows the already presented cuts of the diameter 4 for both combinations but with the metrics information added to the presentation. The results show that the minimum, maximum, average, median, and standard deviation values of the distances were extremely close to each other. The values are so close to each other that they are lost in the deviation of registration results between different registration runs.

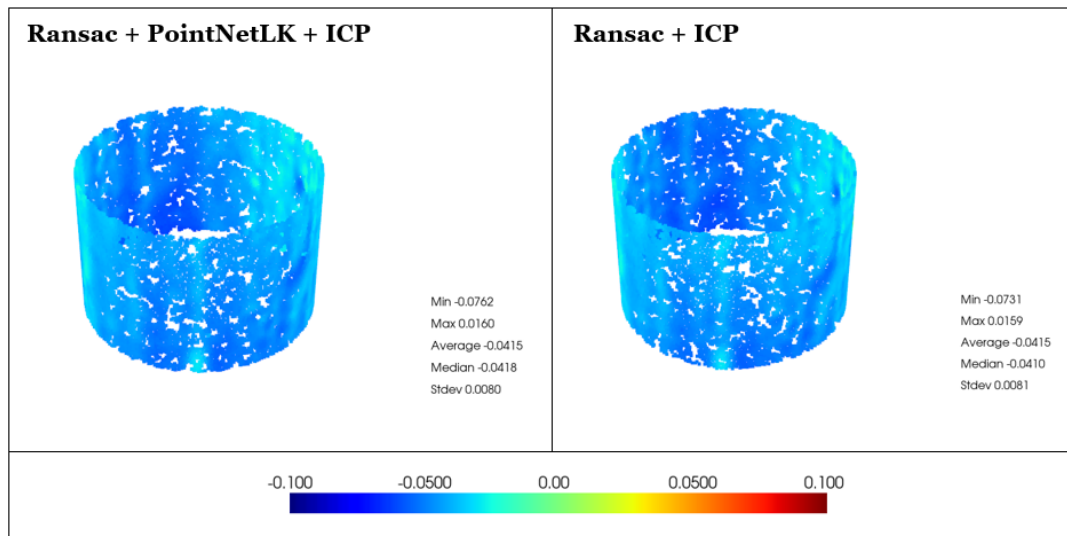


Figure 31. Comparison of the Two Best Registration Methods.

To conclude on the choice of registration algorithms, Ransac was the best for local feature detection, but for 3D inspection, it needs a fine registration method to improve accuracy. PointNetLK, on the other hand, can perform the whole registration process quite accurately, but it may not take local features into account. The combination of Ransac + ICP proved to be an accurate method, but Ransac + PointnetLK + ICP was still slightly more accurate. However, the differences were so small, that they were lost in the deviation in the registration results and the computational inaccuracy in calculating the diameter. Therefore, Ransac + ICP was chosen as the registration method for performing the 3D inspection of the tolerance bar.

3.4.3 Workflow 2. Cylinder Fitting

Measuring the diameters was also implemented in Python using cylinder fitting that did not require the original CAD model. The used algorithm in the workflow was LS cylinder fitting. In addition, the selection of the desired diameter was implemented using the Pyvista box widget clipping tool in the same way as in the previous workflow 1.

There was also another opinion for the cylinder fitting algorithm which was Ransac. However, the sensitivity analysis presented in section 3.5.1 showed that the robustness and accuracy of the algorithm were not sufficient. Workflow 2. (cylinder fitting) can be found in Appendix F.

3.4.4 Validation Methods

Validation of Python results was implemented using CloudCompare, Gom Inspect, and Zeiss CMM. CloudCompare and Gom Inspect gave a picture of how well the results match with the results of available software. In addition, it was important to use other software for the validation process due to the measurement error of scanning. On the other hand, Zeiss CMM provided results without inaccuracy of scanning.

There were lots of different methods available for Zeiss CMM and Gom Inspect for cylinder fitting. For CloudCompare the only available cylinder fitting method was Ransac which did not include to the most used methods in cylinder fitting for 3D Inspection. In addition, CloudCompare with Ransac proved to give inaccurate and unrobust results according to the sensitivity analysis which can be found in the next section. Therefore, CloudCompare was also used with a circle fitting method but there was no information about its algorithm on the user manual of CloudCompare. The following Table 5 shows the available cylinder fitting methods for all validation programs.

Table 5. Cylinder and Circle Fitting Methods of the Validation Programs.

Zeiss CMM Circle Fitting Methods	Gom Inspect Cylinder Fitting Methods	CloudCompare Cylinder Fitting Methods
Least squares	Least squares	Ransac
L1 Feature	Minimum zone	
Outer tangential feature	Maximum inscribed element	
Inner tangential feature	Minimum circumscribed element	
Maximum inscribed element		
Minimum circumscribed element		
Minimum feature		

In this investigation, the best estimation of the overall diameter of a certain area of the tolerance bar was desired. Therefore, LS was selected for all Python, Gom Inspect, and Zeiss CMM. As mentioned earlier, there were no choices for CloudCompare.

According to the literature research, contact-based CMM's are more accurate than portable scanners. Therefore, Zeiss CMM measurement results were used as a reference for all other methods. Zeiss CMM measurements were implemented using one circle path for each four diameters of the tolerance bar. In addition, as mentioned before, the filtering of points was implemented using the LS algorithm.

Figure 32 below shows the Zeiss CMM and the tolerance bar to be measured. There can be seen reference markers on the surface of the tolerance bar. Those reference markers were needed for the scanning process, and they set limitations for creating the measurement paths for the CMM. The measurement paths had to be made in such a way that the measuring stick did not hit the reference markers. Without the reference markers, cylindrical tracks could have been created for the measurement to obtain even more accurate results.



Figure 32. Zeiss CMM Inspection of The Tolerance Bar.

When dealing with measurement devices, it is extremely important to ensure that they are calibrated. With CMM, the Maximum Permissible Error (MPE) gives the limit value for the measurement error. According to the standard ISO 10360-1, the MPE can be calculated using the following equation 3 with measurement results of gauge blocks [48]. A and K represent constant values provided by the manufacturer of the measurement device and L represents the measured distance in mm. For the utilized Zeiss CMM, A was 2.4 μm and K was 300 mm/ μm .

$$MPE = A + L/K \quad (3)$$

The following figures 33-35 are extracts from the calibration certificate of the utilized Zeiss CMM. The figures show the calibration results for the x,y, and z axes. The results are taken with three measurements of each gauge block. The graph of the results shows the MPE values with each measurement. It can be seen from the graph that all measurements are in the MPE values. However, it should be noted that the calibration results are old because the normal interval of calibration is about one year, and that calibration is done in 2022.

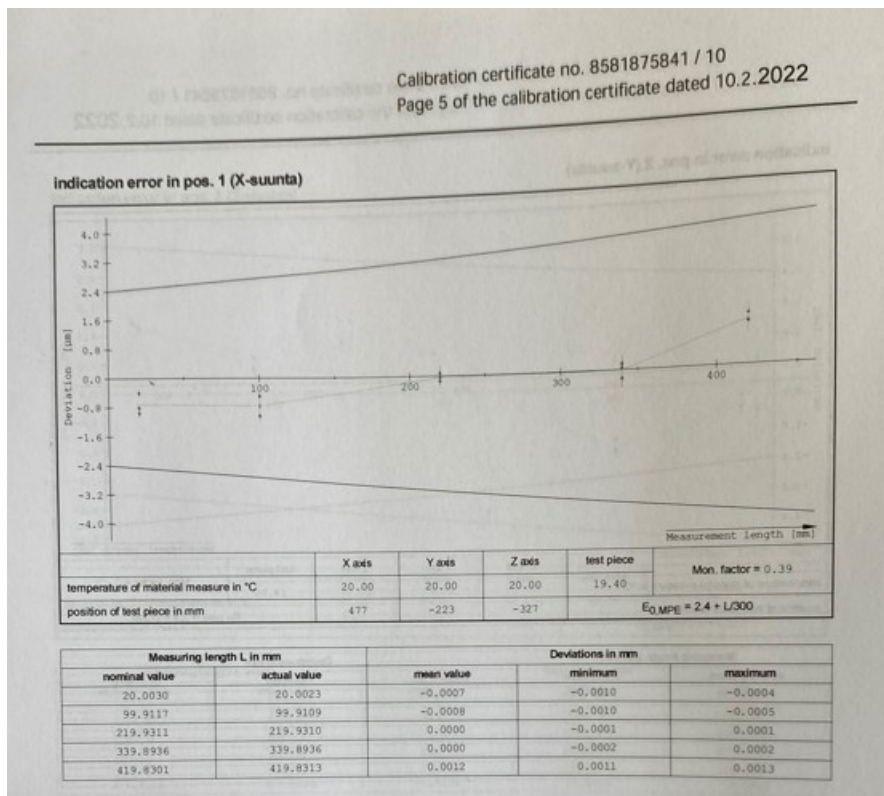


Figure 33. Zeiss CMM X-axis Calibration Results.

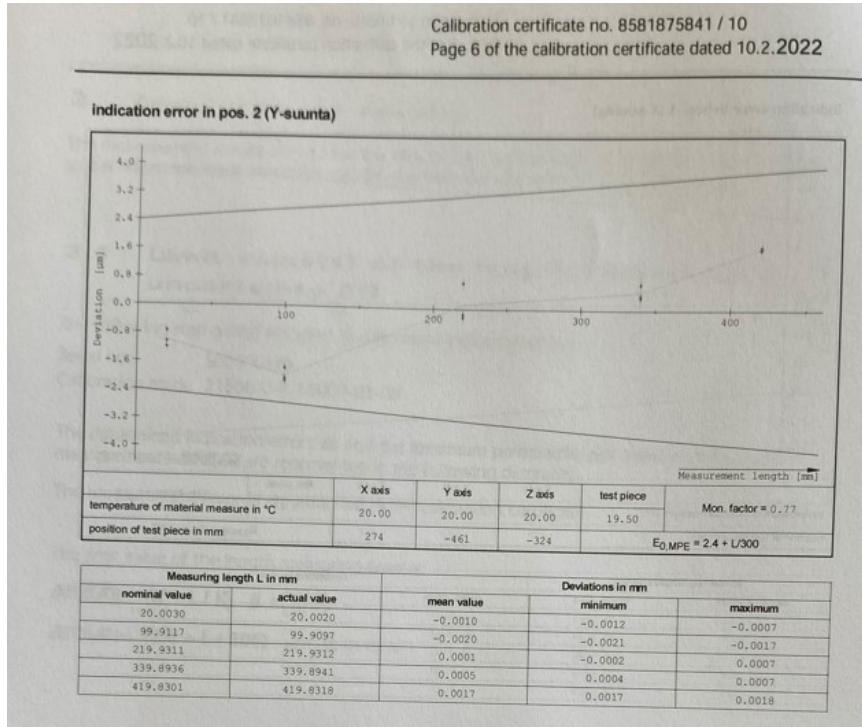


Figure 34. Zeiss CMM Y-axis Calibration Results.

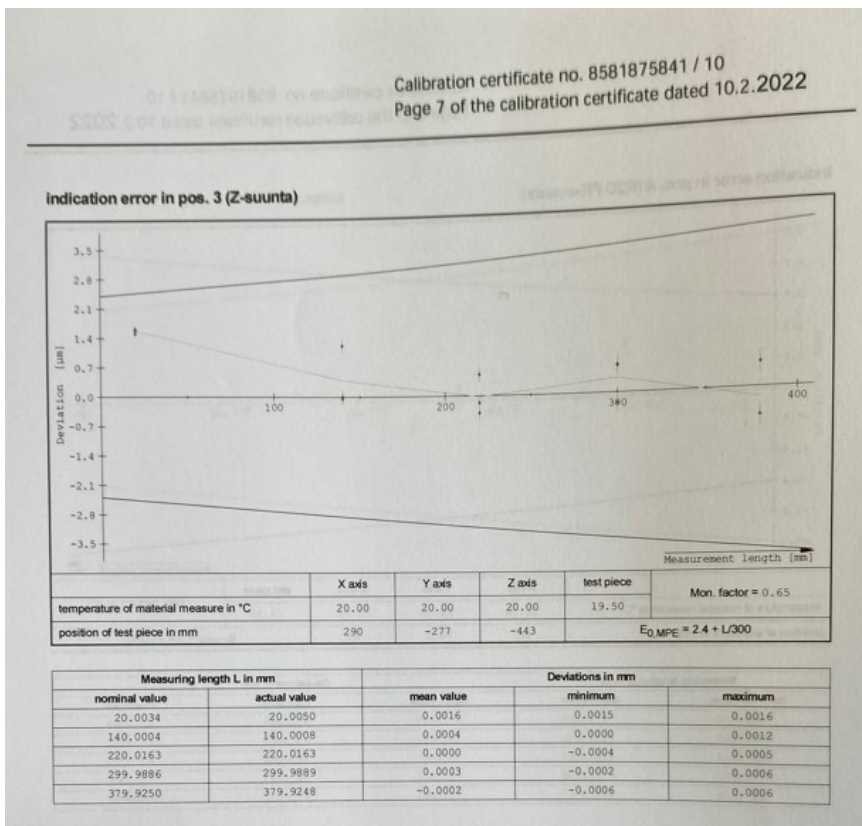


Figure 35. Zeiss CMM Z-axis Calibration Results.

3.4.5 Sensitivity Analysis

The registration and measurement results are greatly influenced by the number of points in the point clouds. To maximize the accuracy of the results and to allow a fair comparison of the results, a sensitivity analysis was performed on the number of points for both Python workflow and CloudCompare. The sensitivity analysis compared the mean values of the three measurements as the number of points increased. The measurements were taken from diameter 4 of the tolerance bar.

A sensitivity analysis for CloudCompare was performed on the cylinder and circle fitting inspection methods, where the number of points varied between 100 000 points and 2 000 000 points. The first measurement was taken with 100 000 points and thereafter the measurements were repeated every 100 000 points. The aim of the sensitivity analysis was to smooth the standard deviation of the measurement results within a certain number of points.

In the cylinder fitting measurement based on the Ransac algorithm, the results varied widely with a standard deviation of 0.014 mm. The results did not stabilize during the sensitivity analysis. Figure 36 below shows the sensitivity analysis results for Ransac cylinder fitting. The average of the measurement results was 24,917 mm.

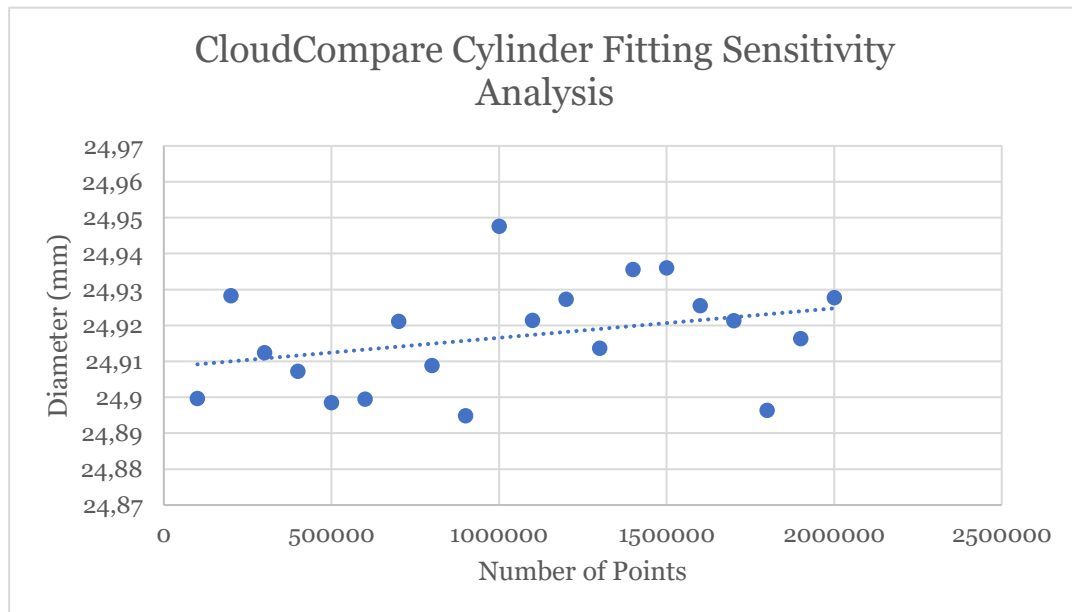


Figure 36. CloudCompare Cylinder Fitting Sensitivity Analysis.

The sensitivity analysis of the circle fitting showed that the variation in results was smaller than with the Ransac cylinder fitting. The average of the

results was 24,921 mm and the standard deviation was 0,002 mm. Figure 37 below shows the sensitivity analysis of CloudCompare circle fitting.

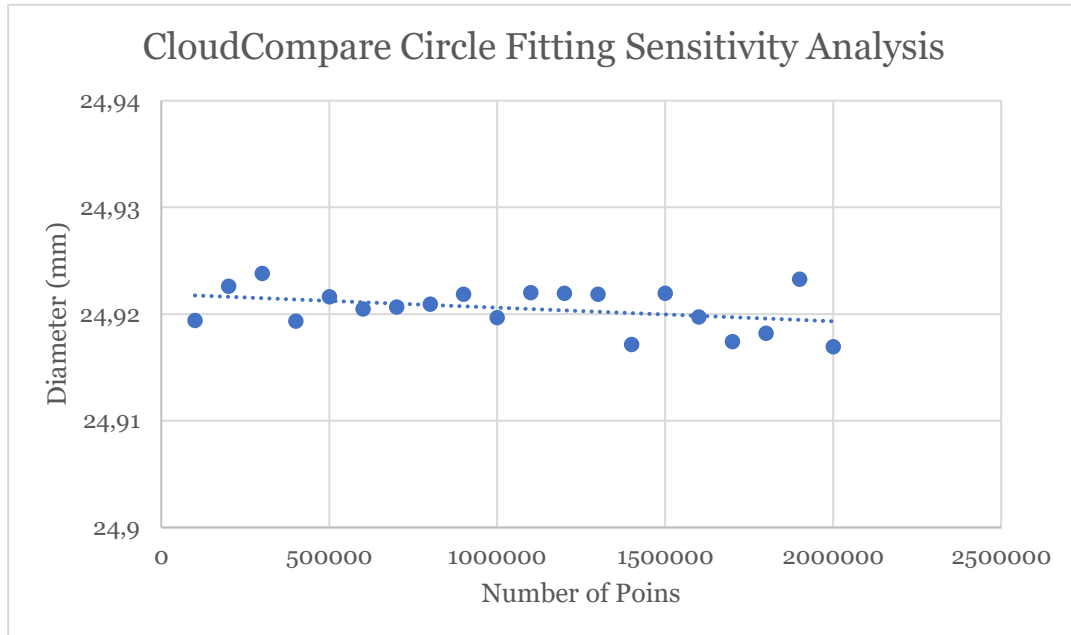


Figure 37. CloudCompare Circle Fitting Sensitivity Analysis.

Sensitivity analysis was also implemented for the Python workflows. These were performed between 5000 and 100000 points, as the computation time in Python increased significantly at a higher number of points. Analyses were performed for the surface comparison method using a CAD model and for the LS and Ransac cylinder fitting methods without a CAD model.

Cylinder fitting based on Ransac results showed the same as CloudCompare Ransac cylinder fitting results. The variability of the results was large with a standard deviation of 0,118 mm and the variability did not smooth out as the number of points increased. Figure 38 below shows the sensitivity analysis results of the Ransac cylinder fitting. The average of the measurement points in the figure was 24,963 mm.

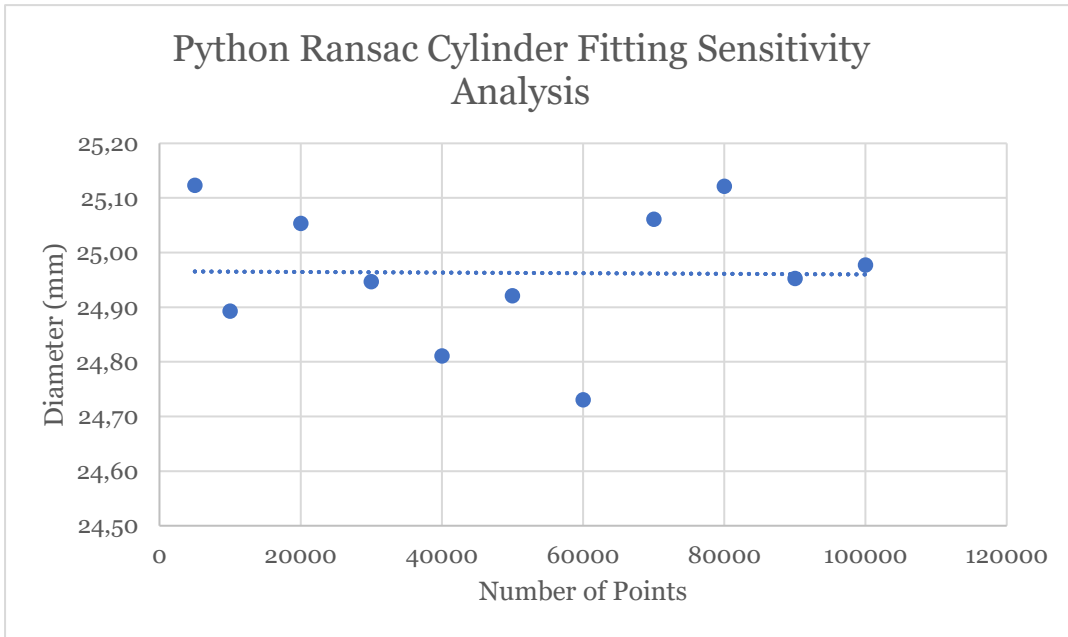


Figure 38. Python Ransac Cylinder Fitting Sensitivity Analysis.

The following analysis was performed for the Python LS cylinder fitting. Its results were robust with a standard deviation of 0,0004 mm. The variability of the results did not change as the number of points increased. The average of the results shown in the figure was 24,916 mm.

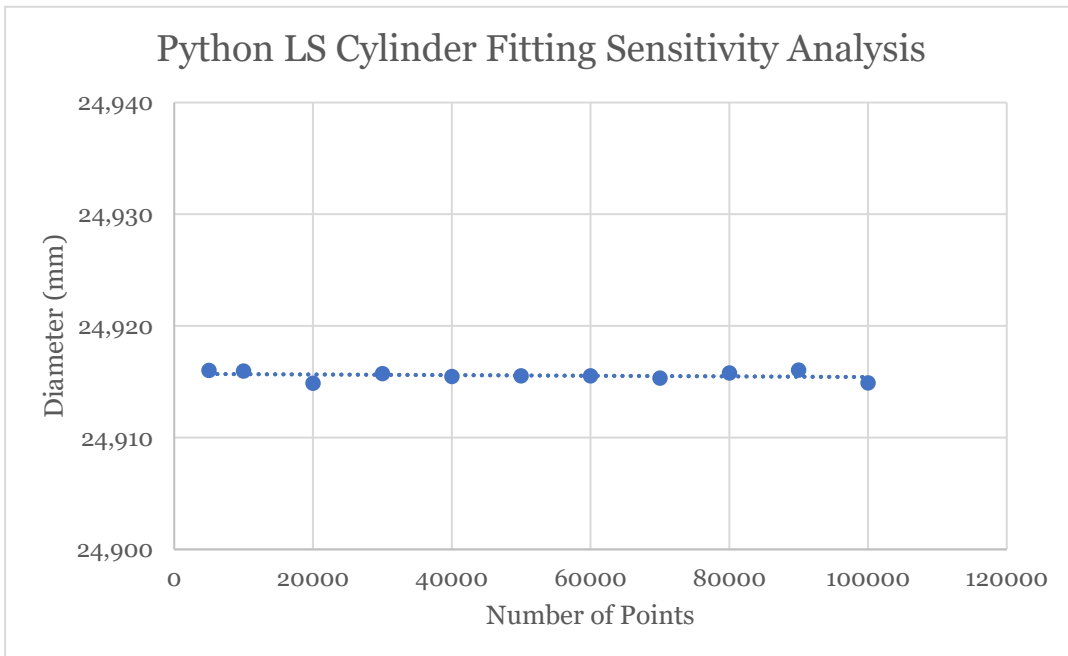


Figure 39. Python LS Cylinder Fitting Sensitivity Analysis.

The last sensitivity analysis for Python was performed on a surface comparison workflow using a CAD model. Its results were robust with a standard deviation of 0,0003 mm and an average of 24,917 mm. There was no significant change in the variability of the results as the number of points increased. Figure 40 below shows the results of the sensitivity analysis for Python surface comparison.

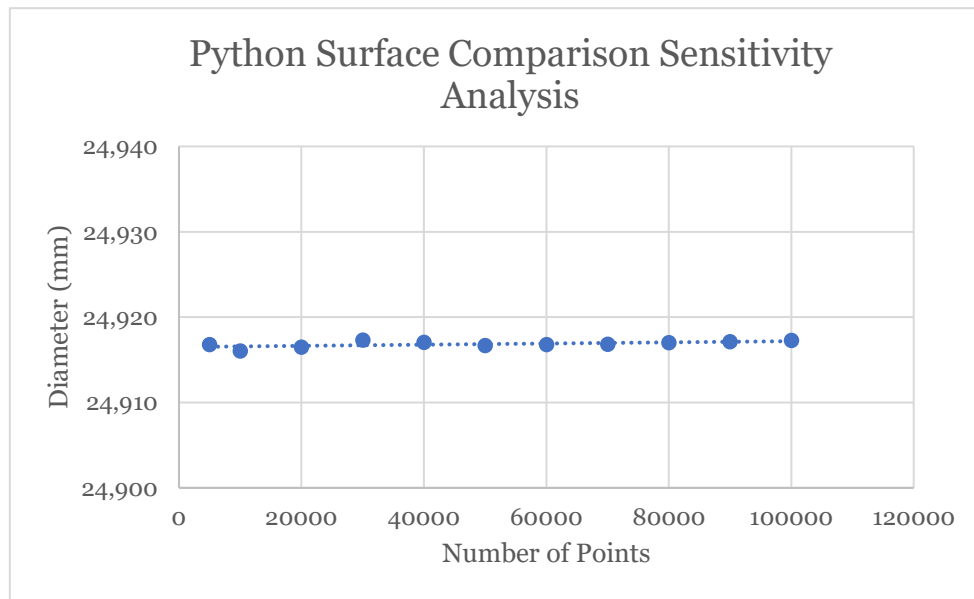


Figure 40. Python Surface Comparison Sensitivity Analysis.

To conclude the sensitivity analyses of CloudCompare and Python based on the Ransac algorithm, the results were not reliable due to the high standard deviation. However, it should be noted that for inspection purposes the standard deviation requirements are high, and these methods may be suitable for other purposes.

Python LS cylinder fitting, Python surface comparison, and CloudCompare circle fitting gave robust results with a small standard deviation. In this investigation, a deeper comparison of these methods will be explored in the next section.

3.5 3D Inspection Results

This section includes measurement results of the tolerance bar diameters using Python workflows 1 and 2. In addition, the reference results are taken by CloudCompare, Gom Inspect, and Zeiss CMM. The measurements are taken by weighting the average value of the measured diameters. In addition, the results are compared together and to the given tolerances of each diameter.

According to the literature review, in certain cases, minimum or maximum values of diameters can be needed. This section provides measurements of the minimum and maximum values of the tolerance bar diameters using the Python workflow 1 and the Gom Inspect.

Figure 41 below shows the measurement results of four different diameters of the tolerance bar. The results show that the measurements based on scanned data are close to each other and the reference results from Zeiss CMM differ a lot from other measurements in any diameters. Noteworthy is that the diameters 3 and 4 were out of the tolerances with each method that used the 3D scanned point cloud. On the other hand, the same diameters measured by Zeiss CMM seem to be in tolerances. The tolerance zones are marked in red in Figure 41.

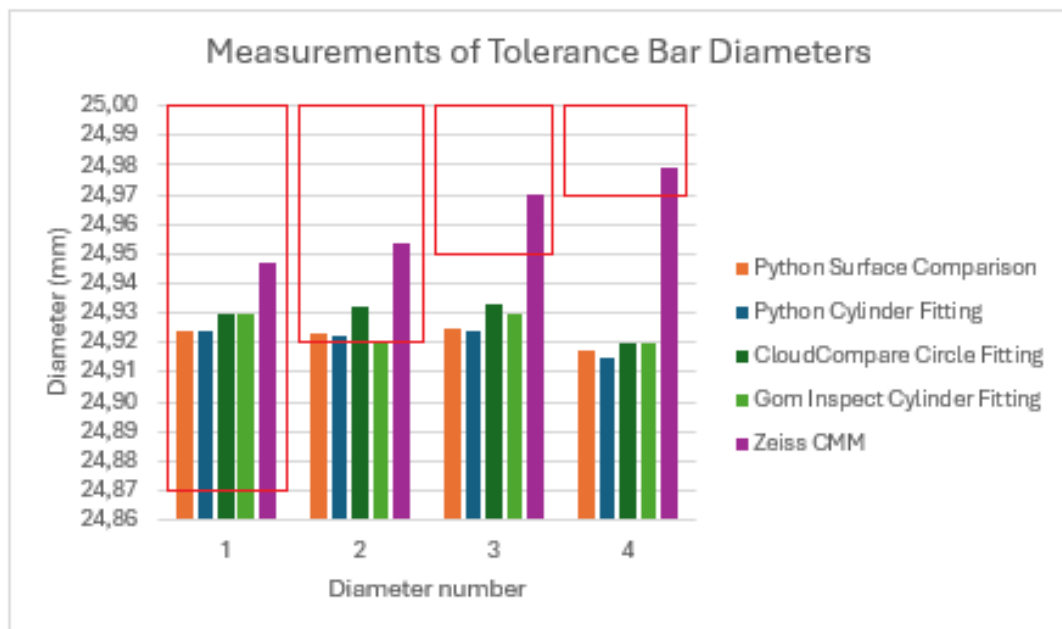


Figure 41. Tolerance Bar Diameters Measured with All Methods.

The following Table 6 shows the averages and standard deviations of the measurements. CloudCompare and Python had 3 measurements of each diameter. Gom Inspect and Zeiss CMM had only one measurement of each diameter. In addition, the table shows the percentual difference between each measurement and the most accurate Zeiss CMM measurement. An extended version of the table can be found in Appendix G. The extended version shows every single measurement for Python and CloudCompare.

Table 6. Measurement Results and Comparison Between Zeiss CMM and Other Methods.

Diameter	Measurement	100 000 points			Gom Inspect	Zeiss CMM
		Python Surface comparison	Python Cylinder fitting	CloudCompare Circle fitting		
1	Avg	24,924	24,924	24,930	24,930	24,947
	Difference %	0,093	0,095	0,071	0,070	-
	Stdev	0,0001	0,0006	0,0012	-	-
2	Avg	24,923	24,922	24,932	24,920	24,954
	Difference %	0,123	0,127	0,087	0,134	-
	Stdev	0,0001	0,0002	0,0027	-	-
3	Avg	24,925	24,924	24,933	24,930	24,970
	Difference %	0,180	0,185	0,149	0,159	-
	Stdev	0,0002	0,00003	0,0029	-	-
4	Avg	24,917	24,915	24,919	24,920	24,979
	Difference %	0,248	0,258	0,240	0,237	-
	Stdev	0,0002	0,00005	0,0031	-	-

The standard deviations show that both Python workflows provided robust results. The standard deviation of Python workflow 1 based on surface comparison was between 0,0001 mm and 0,0002 mm. Standard deviations of the Python workflow 2 based on cylinder fitting were between 0,00003 mm and 0,0006 mm. These results were better than the results of CloudCompare. Standard deviations of CloudCompare results were between 0,0012 mm and 0,0031 mm.

The results show that the Python workflows provided more robust results than the open-source software CloudCompare. However, the only cylinder fitting algorithm for CloudCompare was based on Ransac. According to the sensitivity analysis, it did not provide robust results. Therefore, the circle fitting algorithm was used for CloudCompare measurements. It affects the results because the circle fitting algorithm does not consider point clouds in 3D.

From Table 6 it can be also seen that the measurements of Python workflows were extremely close to each other. The results of Gom Inspect differed a little bit and were also slightly closer to Zeiss CMM results. However, the differences between the results were so small that they can be considered insignificant.

The following Table 7 provides a comparison of the previous average measurements and the given tolerances of each diameter. The difference in the table describes the difference between the measurement and the nominal dimension of the diameter. The red crossed areas mean that the measurements

are out of the tolerances. The pink crossed areas mean that the measurements are in the given tolerances.

Table 7. Comparison of Measured Diameters to the Given Tolerances.

in tolerance			100 000 points						
out of tolerance					Python	Python	CloudCompare		
Diameter	Nominal (mm)	Tolerance	Limits	Measurement	Surface comparison	Cylinder fitting	Circle fitting	Gom Inspect	Zeiss CMM
1	25	h11	+ 0	Avg	24,924	24,924	24,930	24,930	24,947
			-0,13	Difference	-0,076	-0,076	-0,070	-0,070	-0,053
2	25	h10	+ 0	Avg	24,923	24,922	24,932	24,920	24,954
			-0,08	Difference	-0,077	-0,078	-0,068	-0,080	-0,047
3	25	h9	+ 0	Avg	24,925	24,924	24,933	24,930	24,970
			-0,05	Difference	-0,075	-0,076	-0,067	-0,070	-0,030
4	25	h8	+ 0	Avg	24,917	24,915	24,919	24,920	24,979
			-0,03	Difference	-0,083	-0,085	-0,081	-0,080	-0,021

Table 7 shows that all measurement methods noticed the diameters 1 and 2 to be in the given tolerances. However, all methods that used 3D scanning identified the diameters 3 and 4 to be out of the given tolerances. On the other hand, the contact-based Zeiss CMM identified the same diameters to be in the given tolerances. From the results, it can be concluded that the accuracy of the used 3D scanner was not enough for dimensional inspection of h9 and h8 tolerances. In addition, the results showed that the Python workflows were as good as Gom Inspect and CloudCompare for the dimensional inspection of the tolerances.

According to the literature review, the different cylinder fitting methods have significant effects on the measurement results. The previous tolerance comparison table included measurements taken using the LS method. The following Figure 42 shows the measurements of each tolerance bar diameter using Least Squares (LS), Minimum Zone (MZ), Maximum Inscribed Element (MIE), and Minimum Circumscribed Element (MCE). Tolerance zones are marked in red in the figure. The measurements are taken using the 3D scanned point cloud data and Gom Inspect.

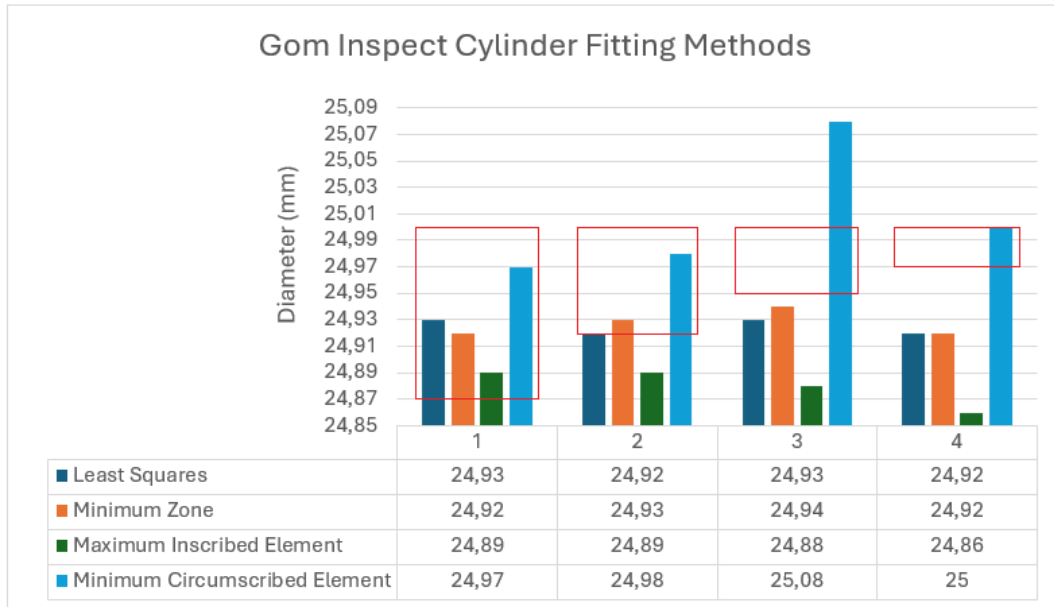


Figure 42. Gom Inspect Cylinder Fitting Methods.

The results showed that the different cylinder fitting algorithms had big effects on the measurement results. The results of the LS and MZ were close to each other, and they weighed the best overall picture of the diameters. The results of the MIE were the lowest, and the results of the MCE were the highest. For the case of dimensional inspection of shaft diameters, the interest was aimed at the MCE algorithm. However, even it was not able to correctly inspect h8 and h9 tolerances.

There were no available MCE or MIE cylinder fitting libraries for Python. However, the Python workflow 1 based on surface comparison was able to use average, minimum, or maximum distance between the nominal and the measured data. In the previous measurements, the average distance was used because it gave the best indication of the actual diameter of the tolerance bar and was comparable to the LS cylinder fitting algorithm. Figure 43 below shows the measurement results of Python surface comparison workflow 1 for minimum, maximum, and average values of the tolerance bar diameters. The tolerances of the diameters are marked in red in the figure.

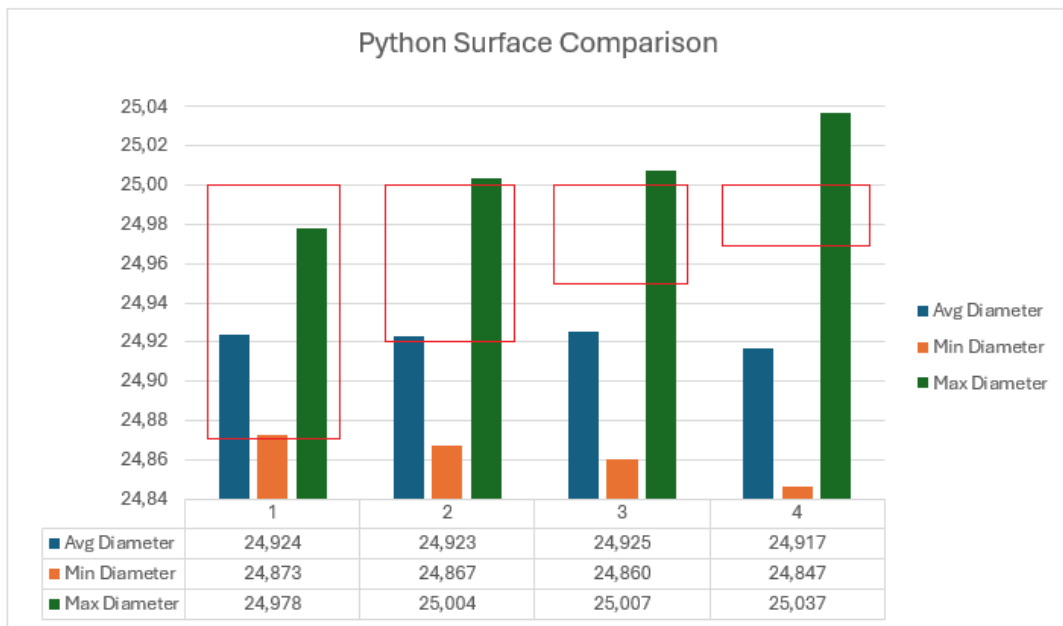


Figure 43. The Minimum, Maximum and Average Diameters Measured Using Python Workflow 1.

Python results showed also a large difference between the different measurement methods. This was of great importance for the verification of tolerances, but in this case, the calculation based on the average distance was the most successful. It was noteworthy that the measurements behaved very similarly to the Gom Inspect software. The value of the largest diameter increased between diameters 1-4 and the value of the smallest diameter decreased. However, the exception was diameter 3, for which Gom Inspect gave a significantly higher value for the largest diameter than Python. However, this can be explained by the fact that Python workflow made use of outlier removal, where single points far away from the others were ignored.

The measurement results shown in Figure 43 above were average values of three measurements such as all previously presented Python results. The standard deviations for minimum and maximum diameter measurements were significantly higher than for diameters measured using average distance or LS cylinder fitting. The table below shows the standard deviations of all Python measurements.

Table 8. Standard Deviations of Python Measurements.

	LS cylinder fitting	Surface comparison (avg)	Surface comparison (min)	Surface comparison (max)
Diameter 1	0,00057	0,00014	0,01512	0,01799
Diameter 2	0,00022	0,00007	0,01626	0,01391
Diameter 3	0,00003	0,00021	0,01038	0,00777
Diameter 4	0,00005	0,00021	0,00731	0,01339

Standard deviations were very small in the LS cylinder fitting and in the surface comparison with the average distance. However, the standard deviation increased significantly when measuring the smallest and largest diameters. This was because the minimum and maximum distances came from a single point. When the average distance was used, all points affected the calculations.

4 Conclusions/Discussion

Garvin's 8 dimensions of quality include conformance which plays a crucial role in manufacturing. In manufacturing, conformance means that products need to be manufactured within required tolerances and it demands inspection. Usually, inspection is implemented using a non-contact or contact-based CMM machine or portable scanner. Those measurement devices give point clouds as measurement results and lots of data processing is needed for achieving the desired dimensional or geometrical information.

Point cloud registration was an important method for 3D inspection. Point cloud registration connects different point clouds into the same coordinate system before they can be compared. Therefore, the accuracy of point cloud registration was extremely important for 3D inspection results. In addition, cylinder fitting was widely used for 3D inspection and one significant advantage of it was the possibility to inspect without a nominal CAD model.

The most used algorithm for point cloud registration was Ransac + ICP due to the efficient combination of global and local registration. However neural networks-based registration methods proved to be developing fast and there were already investigation results where they were faster, more accurate, and more robust than Ransac + ICP combination. For cylinder fitting, the most used algorithms were Least Squares (LS) and Minimum Zone (MZ). In addition, the Maximum Inscribed Element (MIE) and the Minimum Circumscribed Element (MCE) were used in cases where the minimum or maximum diameter of a certain cylinder needs to be measured.

Python had many other libraries for point cloud registration. However, according to the literature review, the main interests were the neural networks-based methods and the Ransac + ICP combination. Therefore, Open3D and Learning3D libraries were selected to be investigated. For cylinder fitting only two libraries were found. The libraries were py-cylinder-fitting and py-RANSAC which used LS and Ransac algorithms. Py-cylinder-fitting gave more robust results and was selected for Python workflow 2.

For investigating the properties of Python workflows, the case study provided a 3D inspection of tolerance bar diameters. The case study also compared Python results to open-source software CloudCompare and industrial software Gom Inspect results. All of them used the same scanned point cloud data for the 3D inspection. In addition, contact-based Zeiss CMM with Zeiss software was used for taking the most accurate reference measurements.

The comparison and testing of the point cloud registration algorithms showed that Ransac + ICP seemed to be the most efficient and accurate method for tolerance bar cases. It connected the Ransac ability to detect small features for global registration and ICP finishing accuracy. Alone Ransac did not give accurate enough results for 3D inspection use and ICP was not able to operate without an initial alignment. Neural networks-based PointnetLK was able to implement the whole registration once with accurate results. However, it used PointNet for shape detection and it had issues with detecting small features of the part. Therefore, it was not able to detect the keyway of the tolerance bar. In addition, the global accuracy of the registration was not as high as by Ransac + ICP.

Ransac + PointNetLK and Ransac + PointNetLK + ICP were also tested but they did not give better results than Ransac + ICP. Ransac + PointNetLK detected the keyway of the tolerance bar, but the global accuracy of the registration result was not as high as with Ransac + ICP. In addition, Ransac + PointNetLK + ICP gave equal results with Ransac + ICP.

The 3D inspection results of the case study showed that both Python workflows were accurate for the inspection of cylindrical part diameters. The results were extremely close to the results of CloudCompare and Gom Inspect. The maximum difference between Python workflows and Gom Inspect was 0,006 mm and between Python workflows and CloudCompare was 0,01 mm. However, CMM results differed a lot from all Python, CloudCompare, and Gom Inspect results.

Zeiss CMM showed that all diameters of the manufactured tolerance bar were in the given tolerances. However, Python, CloudCompare, and Gom Inspect showed that the diameters 3 and 4 with tolerances h8 and h9 were out of the

tolerances. The resolution of the utilized scanner was 0,08 mm, which explains the uncertainty of the measurement results. In addition, the probe head of the coordinate measuring machine also filtered out the variation in the diameter of the tolerance bar due to variations in surface roughness. It needs to be also considered that the measurements using Zeiss CMM were taken by circular measurement path which does not consider the whole area of the investigated diameter of the tolerance bar.

Python workflow 1 was also tested for measuring the maximum and minimum diameters and the results were compared to Gom Inspect MIE and MCE results. The results showed that the point cloud filtering method had large effects on the measurement results and can affect significantly for example inspecting tolerances. Noteworthy was also that the results of Python and Gom Inspect behaved similarly and that the standard deviations were larger for minimum or maximum measurements than for average measurements with Python.

From the 3D inspection results, it can be concluded that the Python workflows are competitive with the available software. The results were different from the Zeiss CMM results, but the best picture was obtained by comparing the results with CloudCompare and Gom Inspect software, which used the same 3D scanned point cloud for the measurements. The case study focused on cylindrical part, but Python has a lot of libraries that enable numerous other 3D inspection tasks. However, Python had also one major shortcoming. There were no libraries for cylinder fitting that used MIE or MCE algorithms, which were provided algorithms in Gom Inspect and Zeiss CMM. However, surface comparison enabled similar measurements for Python.

Regarding the comparison between different registration algorithms, the results were partly in line with the literature review and partly contradictory. Ransac + ICP seemed to be an efficient and accurate combination, which was in line. On the other hand, Neural networks did not deserve competitive results with Ransac + ICP and it was contradictory to the literature research. In addition, in cylinder fitting, LS gave more accurate results than Ransac which was in line with the literature.

This investigation covered only 3D inspection of a small cylindrical part. The next way to the investigation could be an investigation of the suitability of Python workflow for different formed and sized parts. The registration of separate point clouds during 3D scanning was excluded from the investigation. Next, the Python workflow should be extended to include this as well. In addition, 3D inspection using point clouds measured by CMM could be also tested with Python.

References

- [1] J. S. Oakland, *Statistical Process Control*, 5th ed. Hoboken: Taylor and Francis, 2003.
- [2] M. Kutz, *Mechanical engineers' handbook. Volume 3 : Manufacturing and management*, Fourth edition. ed. (Manufacturing and management). Hoboken, New Jersey: John Wiley & Sons, Inc., 2015.
- [3] W. Gao, *Metrology (Precision manufacturing)*. Singapore: Springer, 2019.
- [4] B. M. Colosimo and N. Senin, *Geometric Tolerances Impact on Product Design, Quality Inspection and Statistical Process Monitoring*, 1st 2011. ed. London: Springer London, 2011.
- [5] D. A. Garvin. (1987) Competing on the Eight Dimensions of Quality. *Harvard Business Review*. 9.
- [6] M. Zairi, *Total quality management for engineers*. Cambridge, England: Woodhead Pub., 1991.
- [7] M. Ghazali, A. Mazlan, L. Wei, C. Tying, T. Sze, and N. Jamil, "Effect of machining parameters on the surface roughness for different type of materials," in *IOP Conference Series: Materials Science and Engineering*, 2019, vol. 530, no. 1: IOP Publishing, p. 012008.
- [8] S. Di Cataldo *et al.*, "Optimizing quality inspection and control in powder bed metal additive manufacturing: Challenges and research directions," *Proceedings of the IEEE*, vol. 109, no. 4, pp. 326-346, 2021.
- [9] G. Henzold, *Geometrical dimensioning and tolerancing for design, manufacturing and inspection : a handbook for geometrical product specifications using ISO and ASME standards*, 2nd ed. Oxford ;: Butterworth-Heinemann, 2006.
- [10] S. Gerbino, D. M. Del Giudice, G. Staiano, A. Lanzotti, and M. Martorelli, "On the influence of scanning factors on the laser scanner-based 3D inspection process," *The International Journal of Advanced Manufacturing Technology*, vol. 84, pp. 1787-1799, 2016.
- [11] "Artec 3D." <https://www.artec3d.com/> (accessed 27.6.2024, 2024).

- [12] "HandsOnMetrology."
<https://www.handsonmetrology.com/products/metrotom-1/>
 (accessed 27.6.2024, 2024).
- [13] N. Saba and K. M. Abdel Aziz, "3D data registration evaluation of indoor laser scanner based on various techniques," *Journal of Al-Azhar University Engineering Sector*, vol. 18, no. 67, pp. 396-412, 2023.
- [14] J.-Y. Lai, W.-D. Ueng, and C.-Y. Yao, "Registration and data merging for multiple sets of scan data," *The International Journal of Advanced Manufacturing Technology*, vol. 15, pp. 54-63, 1999.
- [15] C. H. P. Nguyen and Y. Choi, "Comparison of point cloud data and 3D CAD data for on-site dimensional inspection of industrial plant piping systems," *Automation in Construction*, vol. 91, pp. 44-52, 2018.
- [16] Y. Díez, F. Roure, X. Lladó, and J. Salvi, "A Qualitative Review on 3D Coarse Registration Methods," *ACM computing surveys /*, vol. 47, no. 3, pp. 1-36, 2015, doi: 10.1145/2692160.
- [17] X. Gu, X. Wang, and Y. Guo, "A Review of Research on Point Cloud Registration Methods," *IOP conference series.*, vol. 782, no. 2, p. 022070, 2020, doi: 10.1088/1757-899X/782/2/022070.
- [18] X. Huang, G. Mei, J. Zhang, and R. Abbas, "A comprehensive survey on point cloud registration," *arXiv preprint arXiv:2103.02690*, 2021.
- [19] J. Li, Q. Hu, and M. Ai, "Point cloud registration based on one-point ransac and scale-annealing biweight estimation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 11, pp. 9716-9729, 2021.
- [20] R. Schnabel, R. Wahl, and R. Klein, "Efficient RANSAC for point-cloud shape detection," in *Computer graphics forum*, 2007, vol. 26, no. 2: Wiley Online Library, pp. 214-226.
- [21] S. Xue, Z. Zhang, X. Meng, Q. Lv, and X. Tu, "Point cloud registration method for pipeline workpieces based on RANSAC and improved ICP algorithms," in *IOP Conference Series: Materials Science and Engineering*, 2019, vol. 612, no. 3: IOP Publishing, p. 032190.
- [22] T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal, "Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations," *The*

- International Journal of Robotics Research*, vol. 31, no. 12, pp. 1377-1393, 2012.
- [23] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3D-NDT," *Journal of Field Robotics*, vol. 24, no. 10, pp. 803-827, 2007.
- [24] A. Das and S. L. Waslander, "Scan registration using segmented region growing NDT," *The International Journal of Robotics Research*, vol. 33, no. 13, pp. 1645-1663, 2014.
- [25] E. Takeuchi and T. Tsubouchi, "A 3-D scan matching using improved 3-D normal distributions transform for mobile robotic mapping," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006: IEEE, pp. 3068-3073.
- [26] N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas, "Registration of point cloud data from a geometric optimization perspective," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 2004, pp. 22-31.
- [27] L. Li, R. Wang, and X. Zhang, "A tutorial review on point cloud registrations: principle, classification, comparison, and technology challenges," *Mathematical Problems in Engineering*, vol. 2021, pp. 1-32, 2021.
- [28] Y. He, B. Liang, J. Yang, S. Li, and J. He, "An iterative closest points algorithm for registration of 3D laser scanner point clouds with geometric features," *Sensors*, vol. 17, no. 8, p. 1862, 2017.
- [29] M. Weinmann, *Reconstruction and Analysis of 3D Scenes From Irregularly Distributed 3D Points to Object Classes*, 1st 2016. ed. Cham: Springer International Publishing, 2016.
- [30] C. Chinese Control and Decision Conference Shenyang, "Point cloud registration based on improved ICP algorithm," *Proceedings of the 30th Chinese Control and Decision Conference (2018 CCDC) :*, pp. 1511-1465, 2018, doi: 10.1109/CCDC.2018.8407357.
- [31] A. Myronenko, X. Song, and M. Carreira-Perpinan, "Non-rigid point set registration: Coherent point drift," *Advances in neural information processing systems*, vol. 19, 2006.
- [32] A. Myronenko and X. Song, "Point set registration: Coherent point drift," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 12, pp. 2262-2275, 2010.

- [33] A. Kurobe, Y. Sekikawa, K. Ishikawa, and H. Saito, "CorsNet: 3D point cloud registration by deep neural network," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 3960-3966, 2020.
- [34] "3D Point Cloud Classification on ModelNet40." <https://paperswithcode.com/sota/3d-point-cloud-classification-on-modelnet40> (accessed 9.8.2024, 2024).
- [35] "3D Point Cloud Classification on ModelNet40-C." <https://paperswithcode.com/sota/3d-point-cloud-classification-on-modelnet40-c> (accessed 9.8.2024, 2024).
- [36] W.-C. Chang and V.-T. Pham, "3-d point cloud registration using convolutional neural networks," *Applied Sciences*, vol. 9, no. 16, p. 3273, 2019.
- [37] A. Nurunnabi, Y. Sadahiro, R. Lindenbergh, and D. Belton, "Robust cylinder fitting in laser scanning point cloud data," *Measurement*, vol. 138, pp. 632-651, 2019.
- [38] G. Moroni, W. P. Syam, and S. Petró, "Performance improvement for optimization of the non-linear geometric fitting problem in manufacturing metrology," *Measurement Science and Technology*, vol. 25, no. 8, p. 085008, 2014.
- [39] P. Mohan, J. Shah, and J. K. Davidson, "A library of feature fitting algorithms for GD&T verification of planar and cylindrical features," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2013, vol. 55850: American Society of Mechanical Engineers, p. V02AT02A005.
- [40] M. Shunmugam and N. Venkaiah, "Establishing circle and circular-cylinder references using computational geometric techniques," *The International Journal of Advanced Manufacturing Technology*, vol. 51, pp. 261-275, 2010.
- [41] Y.-H. Jin and W.-H. Lee, "Fast cylinder shape matching using random sample consensus in large scale point cloud," *Applied Sciences*, vol. 9, no. 5, p. 974, 2019.
- [42] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv preprint arXiv:1801.09847*, 2018.
- [43] "Learning3D." <https://github.com/vinits5/learning3d?tab=readme-ov-file#documentation> (accessed 25.7.2024, 2024).
- [44] "pyransac3d." <https://pypi.org/project/pyransac3d/> (accessed 25.7.2024, 2024).

- [45] "py-cylinder-fitting." <https://pypi.org/project/py-cylinder-fitting/> (accessed 25.7.2024, 2024).
- [46] "Turning formulas and definitions." Sandvik Coromant. <https://www.sandvik.coromant.com/en-gb/knowledge/machining-formulas-definitions/general-turning-formulas-definitions> (accessed 29.7.2024, 2024).
- [47] "ALL PURPOSE METAL BLACKING SOLUTIONS." <https://blackfast.com/blacking-chemicals/> (accessed 14.8.2024, 2024).
- [48] *SFS-EN ISO 10360-1 + AC*, S. S. SFS, 2001.

Appendix A (Workflow 1, Ransac + ICP)

```
import numpy as np
import open3d as o3d
from learning3d.losses import ChamferDistanceLoss
import torch
import copy
import pyvista as pv

def display_inlier_outlier(cloud, ind):
    inlier_cloud = cloud.select_by_index(ind)
    outlier_cloud = cloud.select_by_index(ind, invert=True)

    print("Showing outliers (red) and inliers (gray): ")
    outlier_cloud.paint_uniform_color([1, 0, 0])
    inlier_cloud.paint_uniform_color([0.8, 0.8, 0.8])
    o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

#Point cloud import
mesh = o3d.io.read_triangle_mesh('tolerance_bar_black_v3.stl')
mesh.compute_vertex_normals()
mesh.paint_uniform_color([0.5, 0.5, 0.5])
source12 = mesh.sample_points_poisson_disk(100000)
source12.paint_uniform_color([0.5, 0.5, 0.5])
mesh1 = o3d.io.read_triangle_mesh('tolerance_bar_template.stl')
mesh1.compute_vertex_normals()
mesh1.paint_uniform_color([1, 0, 0])
template12 = mesh1.sample_points_poisson_disk(100000)
template12.paint_uniform_color([1, 0, 0])

#Converting to coordinates
source22 = np.asarray(source12.points, dtype=np.float32)
template2 = np.asarray(template12.points, dtype=np.float32)
source1 = torch.tensor(source22, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)

#Visualization of initial positions
o3d.visualization.draw_geometries([mesh, mesh1])
o3d.visualization.draw_geometries([source12, template12])

#Ransac registration functions
def preprocess_point_cloud(pcd, voxel_size):
    pcd_down = pcd.voxel_down_sample(voxel_size)
    radius_normal = voxel_size * 2
    pcd_down.estimate_normals(
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_normal, max_nn=100))
    radius_feature = voxel_size * 5
    pcd_fpfh = o3d.pipelines.registration.compute_fpfh_feature(
        pcd_down,
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_feature, max_nn=100))
```

```

return pcd_down, pcd_fpfh

def execute_global_registration(source_down, target_down, source_fpfh,
                               target_fpfh, voxel_size):
    distance_threshold = voxel_size * 1.5
    result = o3d.pipelines.registration.registration_ransac_based_on_feature_matching(
        source_down, target_down, source_fpfh, target_fpfh, True,
        distance_threshold,
        o3d.pipelines.registration.TransformationEstimationPointToPoint(False),
        3, [
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnEdgeLength(
                0.9),
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnDistance(
                distance_threshold)
        ], o3d.pipelines.registration.RANSACConvergenceCriteria(100000, 0.999))
    return result

#Visualization of registration results for Ransac and ICP
def draw_registration_result(source, target, transformation):
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([0.5, 0.5, 0.5])
    target_temp.paint_uniform_color([0, 1, 0])
    source_temp.transform(transformation)
    o3d.visualization.draw_geometries([source_temp, target_temp])

#Ransac registration main
source_down, source_fpfh = preprocess_point_cloud(source12, voxel_size=1.1)
target_down, target_fpfh = preprocess_point_cloud(template12, voxel_size=1.1)
result_ransac = execute_global_registration(source_down, target_down,
                                           source_fpfh, target_fpfh,
                                           voxel_size=1.1)
#print(result_ransac.transformation)
draw_registration_result(source_down, target_down, result_ransac.transformation)

#ICP registration
trans_init = result_ransac.transformation
threshold = 0.3
reg_p2p = o3d.pipelines.registration.registration_icp(
    source_down, target_down, threshold, trans_init,
    o3d.pipelines.registration.TransformationEstimationPointToPoint(),
    o3d.pipelines.registration.ICPConvergenceCriteria(max_iteration=30000))
draw_registration_result(source12, template12, reg_p2p.transformation)

#Evaluation of registration using Chamfer's distance
loss = ChamferDistanceLoss()
source3 = source_down.transform(reg_p2p.transformation)
source2 = np.asarray(source3.points, dtype=np.float32)
template2 = np.asarray(target_down.points, dtype=np.float32)
source1 = torch.tensor(source2, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)

```

```

loss_registration = loss(source, template)
print(f"Registration Loss: {loss_registration}")

#Surface comparison
source_result = source12.transform(reg_p2p.transformation)
mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)
source_pyvista = pv.PolyData(source22)
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

# 3D Inspection

#Gui point selection
source_inspection_pcd = source_result
o3d.visualization.draw_geometries([source_inspection_pcd, template12])
source_inspection_array = np.asarray(source_inspection_pcd.points, dtype=np.float32)
source_inspection_pyvista = pv.PolyData(source_inspection_array)
plotter = pv.Plotter()
plotter.add_mesh_clip_box(source_inspection_pyvista, style="points", color = 'gray',
point_size=3.5, render_points_as_spheres=True)

```

```

result_point_selection = plotter.box_clipped_meshes
plotter.show()
for row in result_point_selection:
    selected_points = row.points
    arrays = row.cell_connectivity
List1 = []
i = 0
for point_coordinates in selected_points:
    if i in arrays:
        List1.append(point_coordinates)
    i = i + 1

# Outlier removal
Outlier_removal_pcd = o3d.geometry.PointCloud()
Outlier_removal_pcd.points = o3d.utility.Vector3dVector(List1)
Outlier_removal_pcd.estimate_normals()
o3d.visualization.draw_geometries([Outlier_removal_pcd])
cl, ind = Outlier_removal_pcd.remove_statistical_outlier(nb_neighbors=10,
                                                         std_ratio=1)
display_inlier_outlier(Outlier_removal_pcd, ind)
result_outlier_removal = np.array(cl.points)
o3d.visualization.draw_geometries([cl, mesh1])

#Calculation of distances between selected points and template model
source_result1 = cl
source_pyvista1 = np.asarray(cl.points, dtype=np.float32)

mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result1.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)

#Visualization of the distances
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista1, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
                shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
                shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,

```

```

        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

```

```
#Diameter calculation
```

```

print("Diameter (average):", signed_distance_average*2 + 25)
print("Diameter (min):", signed_distance_min*2 + 25 )
print("Diameter (max):", signed_distance_max*2 + 25)

```

```
#References
```

```
"""
```

```

@article{Zhou2018,
  author = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
  title = {{Open3D}: {A} Modern Library for {3D} Data Processing},
  journal = {arXiv:1801.09847},
  year = {2018},
}

```

```

@article{sullivan2019pyvista,
  doi = {10.21105/joss.01450},
  url = {https://doi.org/10.21105/joss.01450},
  year = {2019},
  month = {may},
  publisher = {The Open Journal},
  volume = {4},
  number = {37},
  pages = {1450},
  author = {C. Bane Sullivan and Alexander Kaszynski},
  title = {{PyVista}: 3D plotting and mesh analysis through a streamlined interface for the
Visualization Toolkit ({VTK})},
  journal = {Journal of Open Source Software}
}

```

```

@Article{ harris2020array,
  title = {Array programming with {NumPy}},
  author = {Charles R. Harris and K. Jarrod Millman and St{\e}fan J.
van der Walt and Ralf Gommers and Pauli Virtanen and David
Cournapeau and Eric Wieser and Julian Taylor and Sebastian
Berg and Nathaniel J. Smith and Robert Kern and Matti Picus
and Stephan Hoyer and Marten H. van Kerkwijk and Matthew
Brett and Allan Haldane and Jaime Fernandez del
Rio and Mark Wiebe and Pearu Peterson and Pierre
G{e}rard-Marchant and Kevin Sheppard and Tyler Reddy and

```

```

Warren Weckesser and Hameer Abbasi and Christoph Gohlke and
Travis E. Oliphant},
year      = {2020},
month     = sep,
journal   = {Nature},
volume    = {585},
number    = {7825},
pages     = {357--362},
doi       = {10.1038/s41586-020-2649-2},
publisher = {Springer Science and Business Media {LLC}},
url       = {https://doi.org/10.1038/s41586-020-2649-2}
}

@Website{Learning3D,
  url = https://github.com/vinit5/learning3d
}

```

Appendix B (Ransac)

```
import numpy as np
import open3d as o3d
import torch
import copy
import pyvista as pv

def display_inlier_outlier(cloud, ind):
    inlier_cloud = cloud.select_by_index(ind)
    outlier_cloud = cloud.select_by_index(ind, invert=True)

    print("Showing outliers (red) and inliers (gray): ")
    outlier_cloud.paint_uniform_color([1, 0, 0])
    inlier_cloud.paint_uniform_color([0.8, 0.8, 0.8])
    o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

#Point cloud import
mesh = o3d.io.read_triangle_mesh('tolerance_bar_black_v3.stl')
mesh.compute_vertex_normals()
mesh.paint_uniform_color([0.5, 0.5, 0.5])
source12 = mesh.sample_points_poisson_disk(100000)
source12.paint_uniform_color([0.5, 0.5, 0.5])

mesh1 = o3d.io.read_triangle_mesh('tolerance_bar_template.stl')
mesh1.compute_vertex_normals()
mesh1.paint_uniform_color([1, 0, 0])
template12 = mesh1.sample_points_poisson_disk(100000)
template12.paint_uniform_color([1, 0, 0])

#Converting to coordinates
source22 = np.asarray(source12.points, dtype=np.float32)
template2 = np.asarray(template12.points, dtype=np.float32)
source1 = torch.tensor(source22, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)

#visualization of initial positions
o3d.visualization.draw_geometries([mesh, mesh1])
o3d.visualization.draw_geometries([source12, template12])

#Ransac registration functions
def preprocess_point_cloud(pcd, voxel_size):
    pcd_down = pcd.voxel_down_sample(voxel_size)
    radius_normal = voxel_size * 2
    pcd_down.estimate_normals(
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_normal, max_nn=100))
    radius_feature = voxel_size * 5
    pcd_fpfh = o3d.pipelines.registration.compute_fpfh_feature(
        pcd_down,
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_feature, max_nn=100))
```

```

return pcd_down, pcd_fpfh

def execute_global_registration(source_down, target_down, source_fpfh,
                               target_fpfh, voxel_size):
    distance_threshold = voxel_size * 1.5
    result = o3d.pipelines.registration.registration_ransac_based_on_feature_matching(
        source_down, target_down, source_fpfh, target_fpfh, True,
        distance_threshold,
        o3d.pipelines.registration.TransformationEstimationPointToPoint(False),
        3, [
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnEdgeLength(
                0.9),
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnDistance(
                distance_threshold)
        ], o3d.pipelines.registration.RANSACConvergenceCriteria(100000, 0.999))
    return result

#Visualization of registration results for Ransac and ICP
def draw_registration_result(source, target, transformation):
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([0.5, 0.5, 0.5])
    target_temp.paint_uniform_color([1, 0.5, 0])
    source_temp.transform(transformation)
    o3d.visualization.draw_geometries([source_temp, target_temp])

#Ransac registration main
source_down, source_fpfh = preprocess_point_cloud(source12, voxel_size=1.2)
target_down, target_fpfh = preprocess_point_cloud(template12, voxel_size=1.2)
result_ransac = execute_global_registration(source_down, target_down,
                                           source_fpfh, target_fpfh,
                                           voxel_size=1.2)
draw_registration_result(source_down, target_down, result_ransac.transformation)

#Surface comparison of whole models
source_result = source12.transform(result_ransac.transformation)
mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)
source_pyvista = pv.PolyData(source22)
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,

```

```

        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

```

```
# 3D Inspection
```

```
#Gui point selection
```

```

source_inspection_pcd = source_result
o3d.visualization.draw_geometries([source_inspection_pcd, template12])
source_inspection_array = np.asarray(source_inspection_pcd.points, dtype=np.float32)
source_inspection_pyvista = pv.PolyData(source_inspection_array)
plotter = pv.Plotter()
plotter.add_mesh_clip_box(source_inspection_pyvista, style="points", color = 'gray',
point_size=3.5, render_points_as_spheres=True)
result_point_selection = plotter.box_clipped_meshes
plotter.show()
for row in result_point_selection:
    selected_points = row.points
    arrays = row.cell_connectivity
List1 = []
i = 0
for point_coordinates in selected_points:
    if i in arrays:
        List1.append(point_coordinates)
    i = i + 1

```

```
# Outlier removal
```

```

Outlier_removal_pcd = o3d.geometry.PointCloud()
Outlier_removal_pcd.points = o3d.utility.Vector3dVector(List1)
Outlier_removal_pcd.estimate_normals()
o3d.visualization.draw_geometries([Outlier_removal_pcd])
cl, ind = Outlier_removal_pcd.remove_statistical_outlier(nb_neighbors=10,
std_ratio=1)
display_inlier_outlier(Outlier_removal_pcd, ind)
result_outlier_removal = np.array(cl.points)

```

```

o3d.visualization.draw_geometries([cl, mesh1])

#Calculation of distances between the selected points and the template model
source_result1 = cl
source_pyvista1 = np.asarray(cl.points, dtype=np.float32)

mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result1.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)

#Visualization of the distances
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista1, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

#Diameter calculation
print("Diameter (average):", signed_distance_average*2 + 25)
print("Diameter (min):", signed_distance_min*2 + 25 )
print("Diameter (max):", signed_distance_max*2 + 25)

#References
"""
@article{Zhou2018,

```

```

author = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
title = {{Open3D}: {A} Modern Library for {3D} Data Processing},
journal = {arXiv:1801.09847},
year = {2018},
}

```

```

@article{sullivan2019pyvista,
doi = {10.21105/joss.01450},
url = {https://doi.org/10.21105/joss.01450},
year = {2019},
month = {may},
publisher = {The Open Journal},
volume = {4},
number = {37},
pages = {1450},
author = {C. Bane Sullivan and Alexander Kaszynski},
title = {{PyVista}: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit ({VTK})},
journal = {Journal of Open Source Software}
}

```

```

@Article{ harris2020array,
title = {Array programming with {NumPy}},
author = {Charles R. Harris and K. Jarrod Millman and St{\e}fan J. van der Walt and Ralf Gommers and Pauli Virtanen and David Cournapeau and Eric Wieser and Julian Taylor and Sebastian Berg and Nathaniel J. Smith and Robert Kern and Matti Picus and Stephan Hoyer and Marten H. van Kerkwijk and Matthew Brett and Allan Haldane and Jaime Fernandez del Rio and Mark Wiebe and Pearu Peterson and Pierre G{e}rard-Marchant and Kevin Sheppard and Tyler Reddy and Warren Weckesser and Hameer Abbasi and Christoph Gohlke and Travis E. Oliphant},
year = {2020},
month = sep,
journal = {Nature},
volume = {585},
number = {7825},
pages = {357--362},
doi = {10.1038/s41586-020-2649-2},
publisher = {Springer Science and Business Media {LLC}},
url = {https://doi.org/10.1038/s41586-020-2649-2}
}

```

.....

Appendix C (PointNetLK)

```
import numpy as np
import open3d as o3d
from learning3d.models import PointNetLK, PointNet
from learning3d.losses import ChamferDistanceLoss
import torch
import pyvista as pv

#Outlier visualization
def display_inlier_outlier(cloud, ind):
    inlier_cloud = cloud.select_by_index(ind)
    outlier_cloud = cloud.select_by_index(ind, invert=True)

    print("Showing outliers (red) and inliers (gray): ")
    outlier_cloud.paint_uniform_color([1, 0, 0])
    inlier_cloud.paint_uniform_color([0.8, 0.8, 0.8])
    o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

#Point cloud import
mesh = o3d.io.read_triangle_mesh('tolerance_bar_black_v3.stl')
mesh.compute_vertex_normals()
mesh.paint_uniform_color([0.5, 0.5, 0.5])
source12 = mesh.sample_points_poisson_disk(70000)
source12.paint_uniform_color([0.5, 0.5, 0.5])
mesh1 = o3d.io.read_triangle_mesh('tolerance_bar_template.stl')
mesh1.compute_vertex_normals()
mesh1.paint_uniform_color([1, 0, 0])
template12 = mesh1.sample_points_poisson_disk(70000)
template12.paint_uniform_color([1, 0, 0])

#Converting to coordinates
source22 = np.asarray(source12.points, dtype=np.float32)
template2 = np.asarray(template12.points, dtype=np.float32)
source1 = torch.tensor(source22, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)

#visualization of the initial positions
o3d.visualization.draw_geometries([mesh, mesh1])
o3d.visualization.draw_geometries([source12, template12])

#Downsampling
downsource = source12.voxel_down_sample(voxel_size=1.5)
downtemplate = template12.voxel_down_sample(voxel_size=1.5)
o3d.visualization.draw_geometries([downsource, downtemplate])
dsource22 = np.asarray(downsource.points, dtype=np.float32)
dtemplate2 = np.asarray(downtemplate.points, dtype=np.float32)
dsource1 = torch.tensor(dsource22, dtype=torch.float32)
dtemplate1 = torch.tensor(dtemplate2, dtype=torch.float32)
dsource = dsource1.unsqueeze(0)
```

```

dtemplate = dtemplate1.unsqueeze(0)

#Settings for PointNet and PointNetLK algorithms
pn = PointNet(emb_dims=1024, input_shape='bnc', use_bn=True, global_feat=True)
model = PointNetLK(feature_model=pn, delta=1e-02, learn_delta = True, xtol=1.0e-07,
po_zero_mean=False, p1_zero_mean=False, pooling='max')

# Training of PointNet and PointNetLK algorithms
loss_pn = ChamferDistanceLoss()
Parameters_pn = pn.parameters()
Parameters_pnlk = model.parameters()
optimizer = torch.optim.Adam(Parameters_pn, lr=0.001)
optimizer1 = torch.optim.Adam(Parameters_pnlk, lr=0.001)
model.train()
pn.train()
#model.load_state_dict(torch.load('model.pth'))
#pn.load_state_dict(torch.load('pn.pth'))
num_train_set_pn = 5
for train_set in range(num_train_set_pn):
    net = model(dtemplate, dsource)
    transformed_source1 = net['transformed_source']
    loss = loss_pn(dtemplate, transformed_source1)
    trs1 = torch.squeeze(transformed_source1)
    trs2 = trs1.detach().numpy()
    pcd6 = o3d.geometry.PointCloud()
    pcd6.points = o3d.utility.Vector3dVector(trs2)
    optimizer.zero_grad()
    optimizer1.zero_grad()
    loss.backward()
    optimizer.step()
    optimizer1.step()
    print(f"train_set {train_set+1}: Loss = {loss}")
#torch.save(model.state_dict(), 'model.pth')
#torch.save(pn.state_dict(), 'pn.pth')

#Switching practise mode off
model.eval()
pn.eval()

#PointNetLK registration
transformation_matrix = model(dtemplate, dsource)
result = transformation_matrix['transformed_source']
result1 = torch.squeeze(result)
taulukko = result1.detach().numpy()
pcd5 = o3d.geometry.PointCloud()
pcd5.points = o3d.utility.Vector3dVector(taulukko)
cd = ChamferDistanceLoss()

#Visualization of registration results
o3d.visualization.draw_geometries([pcd5, template12])

#Surface comparison of whole models
estimated_transform = transformation_matrix['est_T']

```

```

estimated_transform2 = torch.squeeze(estimated_transform)
estimated_transform3 = estimated_transform2.detach().numpy()
source_result = source12.transform(estimated_transform3)
o3d.visualization.draw_geometries([source_result, template12])
mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)
source_pyvista = pv.PolyData(source22)
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

# 3d Inspection

#Gui point selection
source_inspection_pcd = source_result
o3d.visualization.draw_geometries([source_inspection_pcd, template12])
source_inspection_array = np.asarray(source_inspection_pcd.points, dtype=np.float32)
source_inspection_pyvista = pv.PolyData(source_inspection_array)
plotter = pv.Plotter()
plotter.add_mesh_clip_box(source_inspection_pyvista, color= [1, 0.5, 0], opacity=0.5 )
result_point_selection = plotter.box_clipped_meshes
plotter.show()

```

```

for row in result_point_selection:
    selected_points = row.points
    arrays = row.cell_connectivity
List1 = []
i = 0
for point_coordinates in selected_points:
    if i in arrays:
        List1.append(point_coordinates)
    i = i + 1

# Outlier removal
Outlier_removal_pcd = o3d.geometry.PointCloud()
Outlier_removal_pcd.points = o3d.utility.Vector3dVector(List1)
Outlier_removal_pcd.estimate_normals()
o3d.visualization.draw_geometries([Outlier_removal_pcd])
cl, ind = Outlier_removal_pcd.remove_statistical_outlier(nb_neighbors=10,
    std_ratio=1)
display_inlier_outlier(Outlier_removal_pcd, ind)
result_outlier_removal = np.array(cl.points)
o3d.visualization.draw_geometries([cl, mesh1])

#Calculation of distances between the selected points and the template model
source_result1 = cl
source_pyvista1 = np.asarray(cl.points, dtype=np.float32)
mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result1.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista1, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
    shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
    shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
    shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
    shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)

```

```

plotter.add_text(text=f'Stdev {signed_distance_stddev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

```

```

#Diameter calculation

```

```

print("Diameter (average):", signed_distance_average*2 + 25)
print("Diameter (min):", signed_distance_min*2 + 25 )
print("Diameter (max):", signed_distance_max*2 + 25)

```

```

#References

```

```

"""

```

```

@article{Zhou2018,
author = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
title = {{Open3D}: {A} Modern Library for {3D} Data Processing},
journal = {arXiv:1801.09847},
year = {2018},
}

```

```

@article{sullivan2019pyvista,
doi = {10.21105/joss.01450},
url = {https://doi.org/10.21105/joss.01450},
year = {2019},
month = {may},
publisher = {The Open Journal},
volume = {4},
number = {37},
pages = {1450},
author = {C. Bane Sullivan and Alexander Kaszynski},
title = {{PyVista}: 3D plotting and mesh analysis through a streamlined interface for the
Visualization Toolkit ({VTK})},
journal = {Journal of Open Source Software}
}

```

```

@Article{ harris2020array,
title = {Array programming with {NumPy}},
author = {Charles R. Harris and K. Jarrod Millman and Stfan J.
van der Walt and Ralf Gommers and Pauli Virtanen and David
Cournapeau and Eric Wieser and Julian Taylor and Sebastian
Berg and Nathaniel J. Smith and Robert Kern and Matti Picus
and Stephan Hoyer and Marten H. van Kerkwijk and Matthew
Brett and Allan Haldane and Jaime Fernandez del
Rio and Mark Wiebe and Pearu Peterson and Pierre
Garrard-Marchant and Kevin Sheppard and Tyler Reddy and
Warren Weckesser and Hameer Abbasi and Christoph Gohlke and
Travis E. Oliphant},
year = {2020},
month = sep,
journal = {Nature},
volume = {585},
}

```

```
number    = {7825},
pages     = {357--362},
doi       = {10.1038/s41586-020-2649-2},
publisher = {Springer Science and Business Media {LLC}},
url       = {https://doi.org/10.1038/s41586-020-2649-2}
}
```

```
@Website{Learning3D,
  url = https://github.com/vinits5/learning3d
}

```

Appendix D (Ransac + PointNetLK)

```
import numpy as np
import open3d as o3d
from learning3d.models import PointNetLK, PointNet
from learning3d.losses import ChamferDistanceLoss
import torch
import copy
import pyvista as pv

#Visualization of outliers
def display_inlier_outlier(cloud, ind):
    inlier_cloud = cloud.select_by_index(ind)
    outlier_cloud = cloud.select_by_index(ind, invert=True)
    print("Showing outliers (red) and inliers (gray): ")
    outlier_cloud.paint_uniform_color([1, 0, 0])
    inlier_cloud.paint_uniform_color([0.8, 0.8, 0.8])
    o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

#File import
mesh = o3d.io.read_triangle_mesh('tolerance_bar_black_v3.stl')
mesh.compute_vertex_normals()
mesh.paint_uniform_color([0.5, 0.5, 0.5])
source12 = mesh.sample_points_poisson_disk(100000)
mesh1 = o3d.io.read_triangle_mesh('tolerance_bar_template.stl')
mesh1.compute_vertex_normals()
template12 = mesh1.sample_points_poisson_disk(100000)

#Covertng to coordinates
source22 = np.asarray(source12.points, dtype=np.float32)
template2 = np.asarray(template12.points, dtype=np.float32)
source11 = torch.tensor(source22, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source11.unsqueeze(0)
template = template1.unsqueeze(0)

#Visualization of the initial positions
o3d.visualization.draw_geometries([mesh, mesh1])
o3d.visualization.draw_geometries([source12, template12])

def preprocess_point_cloud(pcd, voxel_size):
    pcd_down = pcd.voxel_down_sample(voxel_size)
    radius_normal = voxel_size * 2
    pcd_down.estimate_normals(
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_normal, max_nn=30))

    radius_feature = voxel_size * 5
    pcd_fpfh = o3d.pipelines.registration.compute_fpfh_feature(
        pcd_down,
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_feature, max_nn=100))
    return pcd_down, pcd_fpfh
```

```

def execute_global_registration(source_down, target_down, source_fpfh,
                              target_fpfh, voxel_size):
    distance_threshold = voxel_size * 1.2
    result = o3d.pipelines.registration.registration_ransac_based_on_feature_matching(
        source_down, target_down, source_fpfh, target_fpfh, True,
        distance_threshold,
        o3d.pipelines.registration.TransformationEstimationPointToPoint(False),
        3, [
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnEdgeLength(
                0.9),
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnDistance(
                distance_threshold)
        ], o3d.pipelines.registration.RANSACConvergenceCriteria(100000, 0.999))
    return result

def draw_registration_result(source, target, transformation):
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([1, 0.706, 0])
    target_temp.paint_uniform_color([0, 0.651, 0.929])
    source_temp.transform(transformation)
    o3d.visualization.draw_geometries([source_temp, target_temp])

#Ransac registration
source_down, source_fpfh = preprocess_point_cloud(source12, voxel_size=1.2)
target_down, target_fpfh = preprocess_point_cloud(template12, voxel_size=1.2)
result_ransac = execute_global_registration(source_down, target_down,
                                          source_fpfh, target_fpfh,
                                          voxel_size=1.2)

#Visualization of Ransac registration
draw_registration_result(source_down, target_down, result_ransac.transformation)

#Pointnetlk registration
#Converting data to suitable format
source3 = source_down.transform(result_ransac.transformation)
source2 = np.asarray(source3.points, dtype=np.float32)
template2 = np.asarray(target_down.points, dtype=np.float32)
source1 = torch.tensor(source2, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)

#Settings for PointNet and PointNetLK algorithms
pn = PointNet(emb_dims=1024, input_shape='bnc', use_bn=True, global_feat=True)
model = PointNetLK(feature_model=pn, delta=1e-02, learn_delta = True, xtol=1.0e-07,
po_zero_mean=False, p1_zero_mean=False, pooling='max')

# Training of PointNet and PointNetLK algorithms
loss_pn = ChamferDistanceLoss()
Parameters_pn = pn.parameters()
Parameters_pnlk = model.parameters()
optimizer = torch.optim.Adam(Parameters_pn, lr=0.001)
optimizer1 = torch.optim.Adam(Parameters_pnlk, lr=0.001)

```

```

model.train()
pn.train()
#model.load_state_dict(torch.load('model.pth'))
#pn.load_state_dict(torch.load('pn.pth'))
num_train_set_pn = 5
for train_set in range(num_train_set_pn):
    net = model(template, source)
    transformed_source1 = net['transformed_source']
    loss = loss_pn(template, transformed_source1)
    trs1 = torch.squeeze(transformed_source1)
    trs2 = trs1.detach().numpy()
    pcd6 = o3d.geometry.PointCloud()
    pcd6.points = o3d.utility.Vector3dVector(trs2)
    #o3d.visualization.draw_geometries([pcd6, template12])
    optimizer.zero_grad()
    optimizer1.zero_grad()
    loss.backward()
    optimizer.step()
    optimizer1.step()
    print(f"train_set {train_set+1}: Loss = {loss}")
#torch.save(model.state_dict(), 'model.pth')
#torch.save(pn.state_dict(), 'pn.pth')

#Switching practise mode of
model.eval()
pn.eval()

# PointNetLK registration
transformation_matrix = model(template, source)
result = transformation_matrix['transformed_source']
result1 = torch.squeeze(result)
array1 = result1.detach().numpy()
pcd5 = o3d.geometry.PointCloud()
pcd5.points = o3d.utility.Vector3dVector(array1)
cd = ChamferDistanceLoss()

#Visualization of registration results
o3d.visualization.draw_geometries([pcd5, template12])

#Evaluation of registration using Chamfer's distance
loss = ChamferDistanceLoss()
loss_registration = loss(source, template)
print(f"Registration Loss Ransac: {loss_registration}")

#Surface comparison
estimated_transform = transformation_matrix['est_T']
estimated_transform2 = torch.squeeze(estimated_transform)
estimated_transform3 = estimated_transform2.detach().numpy()
source_from_Ransac = source12.transform(result_ransac.transformation)
source_result = source_from_Ransac.transform(estimated_transform3)
mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)

```

```

query_points = np.asarray(source_result.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stddev = np.std(signed_distance2)
source_pyvista = pv.PolyData(source22)

#Visualization of surface comparison
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.02])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stddev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

# 3d inspection using cad model

#Gui point selection
source_inspection_pcd = source_result
o3d.visualization.draw_geometries([source_inspection_pcd, template12])
source_inspection_array = np.asarray(source_inspection_pcd.points, dtype=np.float32)
source_inspection_pyvista = pv.PolyData(source_inspection_array)
plotter = pv.Plotter()
plotter.add_mesh_clip_box(source_inspection_pyvista, style="points", color = 'gray',
point_size=3.5, render_points_as_spheres=True)
result_point_selection = plotter.box_clipped_meshes
plotter.show()
for row in result_point_selection:
    selected_points = row.points
    arrays = row.cell_connectivity
List1 = []

```

```

i = 0
for point_coordinates in selected_points:
    if i in arrays:
        List1.append(point_coordinates)
    i = i + 1

# Outlier removal
Outlier_removal_pcd = o3d.geometry.PointCloud()
Outlier_removal_pcd.points = o3d.utility.Vector3dVector(List1)
Outlier_removal_pcd.estimate_normals()
o3d.visualization.draw_geometries([Outlier_removal_pcd])
cl, ind = Outlier_removal_pcd.remove_statistical_outlier(nb_neighbors=10,
                                                         std_ratio=1)
display_inlier_outlier(Outlier_removal_pcd, ind)
result_outlier_removal = np.array(cl.points)
o3d.visualization.draw_geometries([cl, mesh1])

#Calculation of distances between the selected points and the template model
source_result1 = cl
source_pyvista1 = np.asarray(cl.points, dtype=np.float32)
mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result1.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)

#Visualization of the distances
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista1, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
                 shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
                 shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
                 shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
                 shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,

```

```

        shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

```

```

print("Diameter (average):", signed_distance_average*2 + 25)
print("Diameter (min):", signed_distance_min*2 + 25 )
print("Diameter (max):", signed_distance_max*2 + 25)

```

```

#References

```

```

"""

```

```

@article{Zhou2018,
  author = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
  title = {{Open3D}: {A} Modern Library for {3D} Data Processing},
  journal = {arXiv:1801.09847},
  year = {2018},
}

```

```

@article{sullivan2019pyvista,
  doi = {10.21105/joss.01450},
  url = {https://doi.org/10.21105/joss.01450},
  year = {2019},
  month = {may},
  publisher = {The Open Journal},
  volume = {4},
  number = {37},
  pages = {1450},
  author = {C. Bane Sullivan and Alexander Kaszynski},
  title = {{PyVista}: 3D plotting and mesh analysis through a streamlined interface for the
Visualization Toolkit ({VTK})},
  journal = {Journal of Open Source Software}
}

```

```

@Article{    harris2020array,
  title      = {Array programming with {NumPy}},
  author      = {Charles R. Harris and K. Jarrod Millman and St{\e}fan J.
van der Walt and Ralf Gommers and Pauli Virtanen and David
Cournapeau and Eric Wieser and Julian Taylor and Sebastian
Berg and Nathaniel J. Smith and Robert Kern and Matti Picus
and Stephan Hoyer and Marten H. van Kerkwijk and Matthew
Brett and Allan Haldane and Jaime Fernandez del
Rio and Mark Wiebe and Pearu Peterson and Pierre
G{e}rard-Marchant and Kevin Sheppard and Tyler Reddy and
Warren Weckesser and Hameer Abbasi and Christoph Gohlke and
Travis E. Oliphant},
  year       = {2020},
  month      = sep,
  journal     = {Nature},
  volume     = {585},
  number     = {7825},
  pages      = {357--362},
  doi        = {10.1038/s41586-020-2649-2},

```

```
publisher = {Springer Science and Business Media {LLC}},  
url       = {https://doi.org/10.1038/s41586-020-2649-2}  
}
```

```
@Website{Learning3D,  
  url = https://github.com/vinits5/learning3d  
}  
""
```

Appendix E (Ransac + PointNetLK + ICP)

```
import numpy as np
import open3d as o3d
from learning3d.models import PointNetLK, PointNet
from learning3d.losses import ChamferDistanceLoss
import torch
import copy
import pyvista as pv

#Visualization of outliers
def display_inlier_outlier(cloud, ind):
    inlier_cloud = cloud.select_by_index(ind)
    outlier_cloud = cloud.select_by_index(ind, invert=True)
    print("Showing outliers (red) and inliers (gray): ")
    outlier_cloud.paint_uniform_color([1, 0, 0])
    inlier_cloud.paint_uniform_color([0.8, 0.8, 0.8])
    o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

#File import
mesh = o3d.io.read_triangle_mesh('tolerance_bar_black_v3.stl')
mesh.compute_vertex_normals()
mesh.paint_uniform_color([0.5, 0.5, 0.5])
source12 = mesh.sample_points_poisson_disk(70000)
source12.paint_uniform_color([0.5, 0.5, 0.5])
mesh1 = o3d.io.read_triangle_mesh('tolerance_bar_template.stl')
mesh1.compute_vertex_normals()
mesh1.paint_uniform_color([1, 0.5, 0])
template12 = mesh1.sample_points_poisson_disk(70000)
template12.paint_uniform_color([1, 0.5, 0])

#Covertng to coordinates
source22 = np.asarray(source12.points, dtype=np.float32)
template2 = np.asarray(template12.points, dtype=np.float32)
source1 = torch.tensor(source22, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)

#Visualization of the initial positions
o3d.visualization.draw_geometries([mesh, mesh1])
o3d.visualization.draw_geometries([source12, template12])

#Ransac registration functions
def preprocess_point_cloud(pcd, voxel_size):
    pcd_down = pcd.voxel_down_sample(voxel_size)
    radius_normal = voxel_size * 2
    pcd_down.estimate_normals(
        o3d.geometry.KDTreeSearchParamHybrid(radius=radius_normal, max_nn=100))
    radius_feature = voxel_size * 5
    pcd_fpfh = o3d.pipelines.registration.compute_fpfh_feature(
        pcd_down,
```

```

    o3d.geometry.KDTreeSearchParamHybrid(radius=radius_feature, max_nn=100))
return pcd_down, pcd_fpfh

def execute_global_registration(source_down, target_down, source_fpfh,
                               target_fpfh, voxel_size):
    distance_threshold = voxel_size * 1.5
    result = o3d.pipelines.registration.registration_ransac_based_on_feature_matching(
        source_down, target_down, source_fpfh, target_fpfh, True,
        distance_threshold,
        o3d.pipelines.registration.TransformationEstimationPointToPoint(False),
        3, [
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnEdgeLength(
                0.9),
            o3d.pipelines.registration.CorrespondenceCheckerBasedOnDistance(
                distance_threshold)
        ], o3d.pipelines.registration.RANSACConvergenceCriteria(100000, 0.999))
    return result

#Visualization of registration results for Ransac and ICP
def draw_registration_result(source, target, transformation):
    source_temp = copy.deepcopy(source)
    target_temp = copy.deepcopy(target)
    source_temp.paint_uniform_color([0.5, 0.5, 0.5])
    target_temp.paint_uniform_color([1, 0, 0])
    source_temp.transform(transformation)
    o3d.visualization.draw_geometries([source_temp, target_temp])

#Ransac registration main
source_down, source_fpfh = preprocess_point_cloud(source12, voxel_size=1.1)
target_down, target_fpfh = preprocess_point_cloud(template12, voxel_size=1.1)
result_ransac = execute_global_registration(source_down, target_down,
                                           source_fpfh, target_fpfh,
                                           voxel_size=1.1)

#Ransac registration results visualization
draw_registration_result(source_down, target_down, result_ransac.transformation)

#Pointnetlk registration
#Converting data to suitable format
source3 = source_down.transform(result_ransac.transformation)
source2 = np.asarray(source3.points, dtype=np.float32)
template2 = np.asarray(target_down.points, dtype=np.float32)
source1 = torch.tensor(source2, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)

#Settings for PointNet and PointNetLK algorithms
pn = PointNet(emb_dims=1024, input_shape='bnc', use_bn=True, global_feat=True)
model = PointNetLK(feature_model=pn, delta=1e-02, learn_delta = True, xtol=1.0e-07,
po_zero_mean=False, p1_zero_mean=False, pooling='max')

# Training of PointNet and PointNetLK algorithms
loss_pn = ChamferDistanceLoss()

```

```

Parameters_pn = pn.parameters()
Parameters_pnlk = model.parameters()
optimizer = torch.optim.Adam(Parameters_pn, lr=0.001)
optimizer1 = torch.optim.Adam(Parameters_pnlk, lr=0.001)
model.train()
pn.train()
#model.load_state_dict(torch.load('model.pth'))
#pn.load_state_dict(torch.load('pn.pth'))
num_train_set_pn = 5
for train_set in range(num_train_set_pn):
    net = model(template, source)
    transformed_source1 = net['transformed_source']
    loss = loss_pn(template, transformed_source1)
    trs1 = torch.squeeze(transformed_source1)
    trs2 = trs1.detach().numpy()
    pcd6 = o3d.geometry.PointCloud()
    pcd6.points = o3d.utility.Vector3dVector(trs2)
    #o3d.visualization.draw_geometries([pcd6, template12])
    optimizer.zero_grad()
    optimizer1.zero_grad()
    loss.backward()
    optimizer.step()
    optimizer1.step()
    print(f"train_set {train_set+1}: Loss = {loss}")
#torch.save(model.state_dict(), 'model.pth')
#torch.save(pn.state_dict(), 'pn.pth')

#Switching practise mode off
model.eval()
pn.eval()

# PointNetLK registration
transformation_matrix = model(template, source)
result = transformation_matrix['transformed_source']
estimated_transform_pointnetlk = transformation_matrix['est_T']
result1 = torch.squeeze(result)
taulukko = result1.detach().numpy()
pcd5 = o3d.geometry.PointCloud()
pcd5.points = o3d.utility.Vector3dVector(taulukko)
cd = ChamferDistanceLoss()

#Visualization of registration results
o3d.visualization.draw_geometries([pcd5, template12])

#ICP registration
trans_init = estimated_transform_pointnetlk
trans_init2 = torch.squeeze(trans_init)
trans_init3 = trans_init2.detach().numpy()
source_ransac = source12.transform(result_ransac.transformation)
threshold = 0.3
reg_p2p = o3d.pipelines.registration.registration_icp(
    source_ransac, template12, threshold, trans_init3,
    o3d.pipelines.registration.TransformationEstimationPointToPoint(),

```

```

o3d.pipelines.registration.ICPConvergenceCriteria(max_iteration=30000))
draw_registration_result(source12, template12, reg_p2p.transformation)

#Evaluation of registration using Chamfer's distance
loss = ChamferDistanceLoss()
source3 = source_down.transform(reg_p2p.transformation)
source2 = np.asarray(source3.points, dtype=np.float32)
template2 = np.asarray(target_down.points, dtype=np.float32)
source1 = torch.tensor(source2, dtype=torch.float32)
template1 = torch.tensor(template2, dtype=torch.float32)
source = source1.unsqueeze(0)
template = template1.unsqueeze(0)
loss_registration = loss(source, template)
print(f"Registration Loss: {loss_registration}")

#Surface comparison
source_result = source12.transform(reg_p2p.transformation)
mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stdev = np.std(signed_distance2)
source_pyvista = pv.PolyData(source22)
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])
plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)

```

```

plotter.show()

#Gui point selection
source_inspection_pcd = source_result
o3d.visualization.draw_geometries([source_inspection_pcd, template12])
source_inspection_array = np.asarray(source_inspection_pcd.points, dtype=np.float32)
source_inspection_pyvista = pv.PolyData(source_inspection_array)
plotter = pv.Plotter()
plotter.add_mesh_clip_box(source_inspection_pyvista, style="points", color = 'gray',
point_size=3.5, render_points_as_spheres=True)
result_point_selection = plotter.box_clipped_meshes
plotter.show()
for row in result_point_selection:
    selected_points = row.points
    arrays = row.cell_connectivity
List1 = []
i = 0
for point_coordinates in selected_points:
    if i in arrays:
        List1.append(point_coordinates)
    i = i + 1

# Outlier removal
Outlier_removal_pcd = o3d.geometry.PointCloud()
Outlier_removal_pcd.points = o3d.utility.Vector3dVector(List1)
Outlier_removal_pcd.estimate_normals()
o3d.visualization.draw_geometries([Outlier_removal_pcd])
cl, ind = Outlier_removal_pcd.remove_statistical_outlier(nb_neighbors=10,
std_ratio=1)
display_inlier_outlier(Outlier_removal_pcd, ind)
result_outlier_removal = np.array(cl.points)
o3d.visualization.draw_geometries([cl, mesh1])

#Calculation of distances between the selected points and the template model
source_result1 = cl
source_pyvista1 = np.asarray(cl.points, dtype=np.float32)

mesh3 = o3d.t.geometry.TriangleMesh.from_legacy(mesh1)
scene = o3d.cpu.pybind.t.geometry.RaycastingScene()
scene.add_triangles(mesh3)
query_points = np.asarray(source_result1.points, dtype=np.float32)
signed_distance = scene.compute_signed_distance(query_points)
signed_distance2 = signed_distance.numpy()
signed_distance_min = min(signed_distance2)
signed_distance_max = max(signed_distance2)
signed_distance_average = np.mean(signed_distance2)
signed_distance_median = np.median(signed_distance2)
signed_distance_stddev = np.std(signed_distance2)

#Visualization of the distances
plotter = pv.Plotter()
plotter.add_mesh(source_pyvista1, scalars=signed_distance2, cmap='jet', clim=[-0.1, 0.1])

```

```

plotter.add_text(text=f'Min {signed_distance_min:.4f}', position=(830, 280, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Max {signed_distance_max:.4f}', position=(830, 250, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Average {signed_distance_average:.4f}', position=(830, 220, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Median {signed_distance_median:.4f}', position=(830, 190, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_text(text=f'Stdev {signed_distance_stdev:.4f}', position=(830, 160, 0),
font_size=8, color=None, font=None,
shadow=False, name=None, viewport=False, orientation=0.0, font_file=None,
render=True)
plotter.add_title(title='Distance to template model', font_size=14, color=None, font=None,
shadow=False)
plotter.show()

```

```
#Diameter calculation
```

```
print("Diameter (average):", signed_distance_average*2 + 25)
```

```
print("Diameter (min):", signed_distance_min*2 + 25 )
```

```
print("Diameter (max):", signed_distance_max*2 + 25)
```

```
#References
```

```
"""
```

```
@article{Zhou2018,
```

```
author = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
```

```
title = {{Open3D}: {A} Modern Library for {3D} Data Processing},
```

```
journal = {arXiv:1801.09847},
```

```
year = {2018},
```

```
}
```

```
@article{sullivan2019pyvista,
```

```
doi = {10.21105/joss.01450},
```

```
url = {https://doi.org/10.21105/joss.01450},
```

```
year = {2019},
```

```
month = {may},
```

```
publisher = {The Open Journal},
```

```
volume = {4},
```

```
number = {37},
```

```
pages = {1450},
```

```
author = {C. Bane Sullivan and Alexander Kaszynski},
```

```
title = {{PyVista}: 3D plotting and mesh analysis through a streamlined interface for the  
Visualization Toolkit ({VTK})},
```

```
journal = {Journal of Open Source Software}
```

```
}
```

```

@Article{harris2020array,
  title = {Array programming with {NumPy}},
  author = {Charles R. Harris and K. Jarrod Millman and Stefan J.
    van der Walt and Ralf Gommers and Pauli Virtanen and David
    Cournapeau and Eric Wieser and Julian Taylor and Sebastian
    Berg and Nathaniel J. Smith and Robert Kern and Matti Picus
    and Stephan Hoyer and Marten H. van Kerkwijk and Matthew
    Brett and Allan Haldane and Jaime Fernandez del Rio and Mark
    Wiebe and Pearu Peterson and Pierre G{e}rard-Marchant and Kevin
    Sheppard and Tyler Reddy and Warren Weckesser and Hameer Abbasi
    and Christoph Gohlke and Travis E. Oliphant},
  year = {2020},
  month = sep,
  journal = {Nature},
  volume = {585},
  number = {7825},
  pages = {357--362},
  doi = {10.1038/s41586-020-2649-2},
  publisher = {Springer Science and Business Media {LLC}},
  url = {https://doi.org/10.1038/s41586-020-2649-2}
}

```

```

@Website{Learning3D,
  url = https://github.com/vinits5/learning3d
}

```

Appendix F (Workflow 2, LS Cylinder Fitting)

```
import open3d as o3d
import pyvista as pv
import numpy as np

#Visualization of outlier removal
def display_inlier_outlier(cloud, ind):
    inlier_cloud = cloud.select_by_index(ind)
    outlier_cloud = cloud.select_by_index(ind, invert=True)
    print("Showing outliers (red) and inliers (gray): ")
    outlier_cloud.paint_uniform_color([1, 0, 0])
    inlier_cloud.paint_uniform_color([0.8, 0.8, 0.8])
    o3d.visualization.draw_geometries([inlier_cloud, outlier_cloud])

def main():
    #Importing point cloud
    mesh = o3d.io.read_triangle_mesh('tolerance_bar_black_v3.stl')
    mesh.compute_vertex_normals()
    mesh.paint_uniform_color([1, 0.5, 0])
    sampled_pcd = mesh.sample_points_poisson_disk(100000)
    array_pcd = np.asarray(sampled_pcd.points, dtype=np.float32)
    pyvista_pcd = pv.PolyData(array_pcd)

    #Selection of points
    plotter = pv.Plotter()
    plotter.add_mesh_clip_box(pyvista_pcd, style="points", color = 'gray', point_size=3.5,
render_points_as_spheres=True )
    result_point_selection = plotter.box_clipped_meshes
    plotter.show()
    for row in result_point_selection:
        selected_points = row.points
        arrays = row.cell_connectivity
    list1 = []
    i = 0
    for point_coordinates in selected_points:
        if i in arrays:
            list1.append(point_coordinates)
        i = i + 1

    # Outlier removal
    Outlier_removal_pcd = o3d.geometry.PointCloud()
    Outlier_removal_pcd.points = o3d.utility.Vector3dVector(list1)
    Outlier_removal_pcd.estimate_normals()
    o3d.visualization.draw_geometries([Outlier_removal_pcd])
    cl, ind = Outlier_removal_pcd.remove_statistical_outlier(nb_neighbors=20,
std_ratio=1)
    display_inlier_outlier(Outlier_removal_pcd, ind)
    result_outlier_removal = np.array(cl.points)
    o3d.visualization.draw_geometries([cl])
    source_cylinder_fitting = result_outlier_removal
```

```

#py-cylinder-fitting
from py_cylinder_fitting import BestFitCylinder
from skspatial.objects import Points
best_fit_cylinder = BestFitCylinder(Points(source_cylinder_fitting))
radius = best_fit_cylinder.radius
best_fit_cylinder.vector
best_fit_cylinder.point
print("Diameter: ", radius * 2)

"""
#pyransac cylinder fitting
import pyransac3d as pyrsc
points = source_cylinder_fitting # Load your point cloud as a numpy array (N, 3)
cyl = pyrsc.Cylinder()
center, axis, radius, inliers = cyl.fit(points, thresh=0.3)
sph = pyrsc.Sphere()
center1, radius1, inliers1 = sph.fit(points, thresh=0.3)
print("pyransac cylinder fitting diameter:", radius * 2)
#print("pyransac sphere fitting diameter:", radius1 * 2)
"""

main()

#References
"""
@article{Zhou2018,
  author = {Qian-Yi Zhou and Jaesik Park and Vladlen Koltun},
  title = {{Open3D}: {A} Modern Library for {3D} Data Processing},
  journal = {arXiv:1801.09847},
  year = {2018},
}

@article{sullivan2019pyvista,
  doi = {10.21105/joss.01450},
  url = {https://doi.org/10.21105/joss.01450},
  year = {2019},
  month = {may},
  publisher = {The Open Journal},
  volume = {4},
  number = {37},
  pages = {1450},
  author = {C. Bane Sullivan and Alexander Kaszynski},
  title = {{PyVista}: 3D plotting and mesh analysis through a streamlined interface for the
Visualization Toolkit ({VTK})},
  journal = {Journal of Open Source Software}
}

@article{harris2020array,
  title = {Array programming with {NumPy}},
  author = {Charles R. Harris and K. Jarrod Millman and Stfan J.
van der Walt and Ralf Gommers and Pauli Virtanen and David
Cournapeau and Eric Wieser and Julian Taylor and Sebastian

```

Berg and Nathaniel J. Smith and Robert Kern and Matti Picus
and Stephan Hoyer and Marten H. van Kerkwijk and Matthew
Brett and Allan Haldane and Jaime Fernandez del
Rio and Mark Wiebe and Pearu Peterson and Pierre
Gérard-Marchant and Kevin Sheppard and Tyler Reddy and
Warren Weckesser and Hameer Abbasi and Christoph Gohlke and
Travis E. Oliphant},

```
year      = {2020},  
month     = sep,  
journal   = {Nature},  
volume    = {585},  
number    = {7825},  
pages     = {357--362},  
doi       = {10.1038/s41586-020-2649-2},  
publisher = {Springer Science and Business Media {LLC}},  
url       = {https://doi.org/10.1038/s41586-020-2649-2}  
}
```

```
@website{pyRansac-3D,  
  url = https://leomariga.github.io/pyRANSAC-3D/  
}
```

```
@website{py-cylinder-fitting,  
  url = https://pypi.org/project/py-cylinder-fitting/  
}  
"""
```

Appendix G

Table 9. The Extended Version of Table 6.

Diameter	Measurement	100 000 points			Gom Inspect Cylinder fitting	Zeiss CMM
		Python Surface comparison	Python Cylinder fitting	CloudCompare Circle fitting		
1	1	24,9244	24,9240	24,9310	24,9300	24,9474
	2	24,9241	24,9241	24,9280	-	-
	3	24,9242	24,9228	24,9300	-	-
	Avg	24,9242	24,9236	24,9297	24,9300	24,9474
	Difference % Stdev	0,093 0,00014	0,095 0,00057	0,071 0,00125	0,070	- -
2	1	24,9229	24,9217	24,9346	24,9200	24,9535
	2	24,9230	24,9222	24,9328	-	-
	3	24,9229	24,9218	24,9282	-	-
	Avg	24,9229	24,9219	24,9319	24,9200	24,9535
	Difference % Stdev	0,123 0,00007	0,127 0,00022	0,087 0,00269	0,134	- -
3	1	24,9251	24,9236	24,9362	24,9300	24,9698
	2	24,9246	24,9236	24,9324	-	-
	3	24,9250	24,9236	24,9292	-	-
	Avg	24,9249	24,9236	24,9326	24,9300	24,9698
	Difference % Stdev	0,180 0,00021	0,185 0,00003	0,149 0,00286	0,159	- -
4	1	24,9175	24,9150	24,9234	24,9200	24,9793
	2	24,9173	24,9149	24,9190	-	-
	3	24,9170	24,9149	24,9158	-	-
	Avg	24,9173	24,9149	24,9194	24,9200	24,9793
	Difference % Stdev	0,248 0,00021	0,258 0,00005	0,240 0,00312	0,237	- -