Camilla Kjølstad

Uncertainty in Calibration of Reference Standards for Dimensional Measurements

Master's thesis in Mechanical Engineering
Supervisor: Knut Sørby
June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Camilla Kjølstad

Uncertainty in Calibration of Reference Standards for Dimensional Measurements

Master's thesis in Mechanical Engineering
Supervisor: Knut Sørby
June 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This is a master thesis in Manufacturing Technology at the Norwegian University of Science and Technology (NTNU) as a part of the Master's degree program Mechanical Engineering. The work has been carried out during the spring semester of 2019.

The theme of the thesis is uncertainty in calibration of reference standards for dimensional measurements. It was suggested by Knut Sørby, based on the collaboration between NTNU and Justervesenet (JV) regarding traceable reference objects. The results from the thesis can simplify the workload of the department.

The report is written for anyone interested in reading it. It is assumed that the reader has some knowledge of uncertainty analysis and calibration procedures in manufacturing technology.

Trondheim, 07.06.2019

Camilla Kjølstad

Acknowledgment

I would like to express my gratitude toward Professor Knut Sørby. His willingness to give his time so generously and provide guidance throughout the project has been very much appreciated.

I would also like to thank Justervesenet for their kindness of lending out reference objects used during this period.

C.K.

Abstract

The metrology laboratory at NTNU is used to calibrate reference standards, used by Justervesenet to verify other measuring devices. Reference standards, such as rectangular parallelepipeds and cylinders, are calibrated by a dimensional measuring device. An uncertainty analysis follows after the calibration process.

The current calibration method used by NTNU is time consuming, as it has a high measurement density to ensure low uncertainty. When the object has an underlying geometry of a rectangular parallelepiped, the current minimum bounding box procedure is suboptimal.

A minimum bounding box algorithm is developed. HYBBRID algorithm turned out to be exact in practice, possible to implement in Python and have low computation time. The developed algorithm uses data from the coordinate measuring machine or the laser tracker and calculates the dimensional measurements.

An estimation of the uncertainty in calibration is conducted for three rectangular parallelepipeds. While a high measurement density is desired to keep the uncertainty low, it will lead to an increase in computational time. As it is desirable with a sensible balance between low uncertainty and time consumption in a calibration process, the significance of the measurement density was explored.

The results strongly indicate that measurement density currently used in calibration processes by NTNU can be reduced. The time consumption would significantly decrease while keeping the associated uncertainty sufficiently low. However, the results only apply for rectangular parallelepipeds similar to the three used in the uncertainty analysis, with similar dimensions, material and manufacturing technique.

NTNU wants to develop new working methods and enhancement when it comes to the validation of its calibration process. The development of the HYBBRID algorithm renews the working methods of deciding dimensional measurements of a rectangular parallelepiped in a calibration process. The results of the uncertainty analysis indicate that the measurement density can be reduced to improve time usage while keeping the uncertainty low.

Sammendrag

Målelaboratoriet ved NTNU brukes til å kalibrere normaler (eng: reference standards) som brukes av Justervesenet for å verifisere andre måleinstrumenter. Normaler, som rektangulære parallellepiped og sylindere, blir kalibrert av en koordinatmålemaskin. Etter kalibreringsprosessen utføres en usikkerhetsanalyse.

Den nåværende kalibreringsmetoden som brukes av NTNU er tidkrevende da den utføres med høy måletetthet for å sikre lav måleusikkerhet. Dagens metode for å finne minste omskrevne boks (eng: minimum bounding box) av en normal med geometri som et rektangulært parallellepiped er suboptimal.

En minimum bounding box algoritme er utviklet. HYBBRID-algoritmen viste seg å være eksakt i praksis, mulig å implementere i Python og å ha lav beregningstid. Den utviklede algoritmen bruker data fra koordinatmåler eller laser tracker og beregner dimensjonene.

En estimering av måleusikkerheten ved kalibrering er utført for tre rektangulære parallellepiped. Høy måletetthet er ønskelig for å holde måleusikkerheten lav, men dette fører til en økning i beregningstiden. Ettersom det er ønskelig med en fornuftig balanse mellom lav usikkerhet og tidsforbruk i en kalibreringsprosess, er betydningen av måletetthet undersøkt.

Resultatene gir gode indikasjoner på at måletettheten som blir benyttet i NTNUs kalibreringsprosesser kan reduseres. Tidsforbruket vil minske betydelig, mens måleusikkerheten fortsatt vil være tilstrekkelig lav. Resultatene gjelder imidlertid kun for rektangulære parallellepiped med tilsvarende dimensjoner, material og produksjonsmetode som de som ble brukt i usikkerhetsanalysen.

NTNU ønsker å utvikle nye arbeidsmetoder og forbedringer ved validering av deres kalibreringsprosess. Utviklingen av HYBBRID-algoritmen fornyer arbeidsmetodene for å bestemme de dimensjonale målene til rektangulære parallellepiped i en kalibreringsprosess. Resultatene av usikkerhetsanalysen indikerer at måletettheten kan reduseres for å minske tidsbruk, samtidig som måleusikkerheten forblir lav.

Table of Contents

List of Figures

List of Tables

List of Equations

Acronyms

AABB Axis Aligned Minimum Bounding Box

ANSI The American National Standards Institute

ASME The American Society of Mechanical Engineers

BIPM Bureau International des Poids et Mesures

CIPM Comité International des Poids et Mesures

CMM Coordinate Measurement Machine

csv comma separated values

CTE Coefficient of Thermal Expansion

GNU General Public License

HYBBRID Hybrid Bounding Box Rotation Identification

ISO International Organization for Standardization

JCGM Joint Committee for Guides in Metrology

JV Justervesenet

LSC Least Squares Circle

MIC Maximum Inscribed Circle

MMC Minimum Circumscribed Circle

MPE Maximum Permissible Error

NumPy Numerical Python

OBB Oriented Bounding Box

PCA Principal Component Analysis

PTC Power Test Codes

SVD Singular Value Decomposition

txt text

Chapter 1

Introduction

The background of the problems associated with the theme of this master thesis is presented in this chapter. The objectives are introduced and motivated.

1.1 Background

Metrology is a critical factor within research, science and innovation. While the society has evolved to what it is today, the science of measurement has evolved to a foundation in the society. The importance of not only the ability to measure but a universal agreement about measurement standards is essential in both industry and trade. Measurement technologies are essential when scientists explore their theories, as well as when measures are used as a basis for purchases.

Justervesenet (JV), the Norwegian Metrology Service, ensures the national and international acceptance of the Norwegian metrology infrastructure (Justervesenet, 2019). The supervisory department of JV controls the measurements and measurement systems in the business sector in Norway. The control is subject to instruments with a traceable chain to national reference standards and SI units. All equipment that is subject to control needs a report to confirm that the measuring equipment yield results within approved limits.

The metrology laboratory at NTNU is used in calibration processes of reference standards, used by the JV. Reference standards, such as rectangular parallelepipeds and cylinders, are calibrated by a dimensional measuring device. The device could be a coordinate measuring machine (CMM) or a laser tracker, that senses points on the objects with a probe or a laser, respectively, measuring the dimensions of the physical object. An uncertainty analysis, estimating the

uncertainty of the dimensional measurements of the object, follows after the calibration process. The objects are further used as reference standards for verification of other measuring devices by the JV.

To enable documentation of dimensional measurement, the measurement results need to be traceable. A traceable chain assures that the measurement results can be related to a reference through an unbroken chain of comparisons, all with stated uncertainties. The calibrated reference standards from the metrology laboratory at NTNU are a part of such a traceable chain.

The current calibration method used by NTNU is time consuming due to the need for low uncertainty. The calibration method is highly dependant on the number of measuring points that are used to measure the dimensions of the object. The current method measures points closely to ensure the low uncertainty of the results. A measurement session of a reference standard with the current method can use up to four hours in the CMM. This results in a high cost and creates practical challenges. The measurement session occupies the CMM for the same amount of time, and the associated computer in the laboratory faces the challenge of going into sleep mode, disrupting the measurement procedure.

The uncertainty of the measurement results is assumed to decrease with an increase in the number of measuring points. However, a certain number of points that provide sufficiently low uncertainty of the calibration process is assumed to exist. There is a lack of information about this specific number. Another assumption is that the current procedure to decide the minimum object that encloses all the measuring points, when the object has an underlying geometry of a rectangular parallelepiped, is not optimal. A new method could provide a minimum object both smaller and with lower uncertainty, than with the current method. The uncertainty associated with the minimum bounding rectangular parallelepiped is larger than the uncertainty associated with the minimum bounding cylinder. A new method of finding the minimum bounding cylinder is not developed. Experience from the metrology laboratory with reference standards with an underlying geometry of a cylinder has shown that it is easy to implement a solution in the software of the CMM. The solution is efficient and has sufficiently low uncertainty.

NTNU wants to develop new working methods and enhancement when it comes to the validation of its calibration process. This master thesis is a part of this work.

1.2 Objectives

The main objectives of this master thesis are

- A literature review on uncertainty analysis. An overview of the differences between ANSI/ASME and ISO standards when used in uncertainty analysis should be included. Scientific literature and other sources are used in the review.
- A literature review on finding minimal enclosing objects, both cylinder and rectangular parallelepiped. Scientific literature and other sources are used in the review.
- Develop an algorithm in Python to find the minimal enclosing object to a measured reference object from the metrology laboratory. The input in the algorithm will be the measured point data, the output should be the minimum enclosing object.
- Verify the developed algorithm on simple physical parts in the metrology laboratory. Both Leitz PMM-C 600 CMM and Leica Absolute Tracker AT960 laser tracker should be used. The physical parts should be similar to parts used by Justervesenet.
- Conduct an estimation of the uncertainty in a calibration.

1.3 Approach

In order to cover updated and relevant information of uncertainty analysis and minimum enclosing object algorithms, an extensive literature study limited to appropriate databases, such as Google Scholar and Science Direct, has been conducted.

An algorithm is developed in Python during the project. I have not programmed in Python earlier. However, Python is chosen as the programming language. The master thesis is an excellent opportunity to learn a new programming language. Several introduction tutorials were conducted in Python to learn the syntax of the programming language. At the beginning of the semester, coding in Matlab® was performed to compare the results and the syntax to Python.

Measurement procedures with both the CMM and the laser tracker was conducted in the metrology laboratory. The created data sets were used regularly in the development of the algorithm to verify the results, as the dimensional measurements of the parts were known.

An uncertainty analysis was performed on three rectangular parallelepipeds. The rectangular parallelepipeds were manufactured to function as reference standards in calibration processes.

The developed algorithm in Python is used to conduct an estimation of the uncertainty.

1.4 HSE considerations

At several occasions, I have been in the metrology laboratory at Valgrinda, NTNU. A measurement machine, Leitz PMM-C 600, and a laser tracker, Leica Absolute Tracker AT960, was used to measure physical parts. Both are secure devices, with no pinch hazard, fire hazard or any sharp objects that can hurt operators or observers of the machines.

The measurement machine is extremely robust. It is not possible for the operator of the machine to damage the machine. In the event of a collision, embedded sensors ensure that the safety and collision system stops the measuring sequence before the collision. It will also stop if there is short-circuiting between the cords in the machine.

The metrology laboratory is in the basement of Valgrinda, NTNU. A review of the escape routes in case of a fire has been completed.

1.5 Outline

The report is structured as follows. In the first chapter, the theme of the master thesis is introduced and motivated. The second chapter introduces the reader to the necessary theoretical background, covers the basic theory and the state of the art of uncertainty analysis. From chapter three and on, work done during the project is presented. Chapter three introduces the reader to the development of the minimum enclosing object algorithm used in the project. In the fourth chapter, the uncertainty of three parallelepipeds that can be used as reference standards in calibration processes is analyzed. An uncertainty analysis is performed, based on previous experience in the metrology laboratory and the results from the developed minimum enclosing object algorithm. Chapter five presents conclusions based on the main objectives. A discussion and reflection on the work done are given, and ideas or further work are proposed.

The following table, Table 1, gives an overview of the structure in the report.

Table 1: Report outline

Chapter	Objective
1	Introduction chapter
2	A literature review on uncertainty analysis
	A literature review on finding minimal enclosing objects
3	Develop an algorithm in Python to find the minimal enclosing object to a measured
	reference object from the metrology laboratory
4	Verify the developed algorithm on simple physical parts in the metrology laboratory
	Conduct an estimation of the uncertainty in a calibration
5	Final chapter

Chapter 2

Theoretical background

This chapter presents the essential theoretical background about uncertainty analysis, calibration process, minimum bounding objects and bounding box algorithms. Theory about dimensional measuring devices used in the project, computational geometry and rotation matrices are also included.

2.1 State of the art

Engineers are bound to make decisions in their engineering tasks, and a measurement result can be the information needed to make a decision. A measurement result consists of both a measured value of a measurand and an uncertainty associated with the measured value. The procedure of an uncertainty analysis is conducted to estimate the uncertainty associated with the measured value. Thus, an uncertainty analysis gives information about the possible error of the measurement value that a decision is based on. Stated by Dieck (2007), the uncertainty of the measurement can be as significant as the measurement value, as a measurement value with no knowledge of the uncertainty give little knowledge of the state or the performance of a measurement process.

The methodical approach of uncertainty analysis is objective and standardized. However, detailed knowledge of the measurand's nature and the measurement procedure is crucial to achieving a satisfying measurement result, along with critical thinking, integrity and professional skills of those performing the uncertainty analysis (ISO/IEC 98-3:2008, 2008). As of today, two professional documents on uncertainty analysis are accepted: The American National Standards Institute/American Society of Mechanical Engineers (ANSI/ASME) Power

Test Codes (PTC) and "Guide to the Expression of Uncertainty in Measurement" by The International Organization for Standardization (ISO). The standard by ANSI/ASME is the standard followed by the United States, while the guide by ISO is an international standard. The guide by ISO is referred to as *the guide* and the standard by ANSI/ASME is referred to as *the standard*.

Defined by Kline (1953), the uncertainty of a measurement is "A possible value the error might have". This definition of uncertainty is still relevant. However, how to determine and perform the methods and procedures to establish the uncertainty has not always been clear. Stated by Abernethy and Ringhiser (1985), confusion, argument and controversy has affected several decades of scientists, engineers and practitioners arguing about uncertainty analysis.

In the 1950s, ASME started their work on a professional document on uncertainty in measurement for America. It was based on the work of Kline (1953). However, the methods presented by ASME were debated and disagreed of until 1986, when ASME reached a consensus with their developed standard of uncertainty analysis. In 1994, Steele et al. (1994) published an article with further details of the terminology, definitions and methods in the standard.

As different methodologies to evaluate measurements affected across boarders, the highest authority in metrology in the world, Comité International des Poids et Mesures (CIPM), asked the Bureau International des Poids et Mesures (BIPM) in 1977, to make a recommendation of how to evaluate a measurement. A guide providing international consensus on the expression of uncertainty in measurement was wanted. An internationally recognized guide, "Evaluation of measurement data - Guide for the expression of uncertainty in measurement", was presented in 1993 after collaboration and extensive effort made by the member organization of the Joint Committee for Guides in Metrology (JCGM): BIPM, ISO, International Electrotechnical Commission (IEC), International Federation of Clinical Chemistry (IFCC), International Union of Pure and Applied Chemistry (IUPAC), International Union of Pure and Applied Physics (IUPAP), International Organization of Legal Metrology (OIML). Later, the International Laboratory Accreditation Cooperation (ILAC) has also become a member organization of JCGM.

After several years of different methodologies to evaluate measurements, the expression of uncertainty was provided international consensus by the standard and the guide. The terminology of the standard and the guide differ from each other. Both updated periodically and from 2005, the standard adopted the methodology from the guide. As of now, proper guidance on uncertainty analysis is available in the standard (ASME PTC 19.1-2013, 2013) and the guide (ISO/IEC 98-3:2008, 2008).

2.2 Uncertainty

A measurement is a procedure where the objective is to find the value of the measurand. Before a measurement procedure of the measurand, a precise specification of the measurand, the measurement method and the measurement procedure is defined. The measurement result is an estimate of the value of the measurand and only complete when stated with the associated uncertainty.

When the result of a measurement is analyzed, it is important to distinguish between the different definitions of error, uncertainty, accuracy and precision. As written in the guide, the definitions are as follows (ISO/IEC 98-3:2008, 2008):

- *The measurement error* defines the difference between the true value and the result of the measurement.
- *The measurement uncertainty* defines an estimate of the probable error in the measurement result, presented as a standard deviation.
- *The measurement accuracy* defines the closeness of agreement between the result of a measurement and the true value.
- *The measurement precision* is the closeness of agreement between repeated measurement results.

Figure 1 visualizes the difference between the accuracy and the precision of a measurement. The two definitions are independent of each other. The desired quality of a measurement result requires both high accuracy and high precision, giving closeness of agreement between a measured value and the true value, repeatedly.

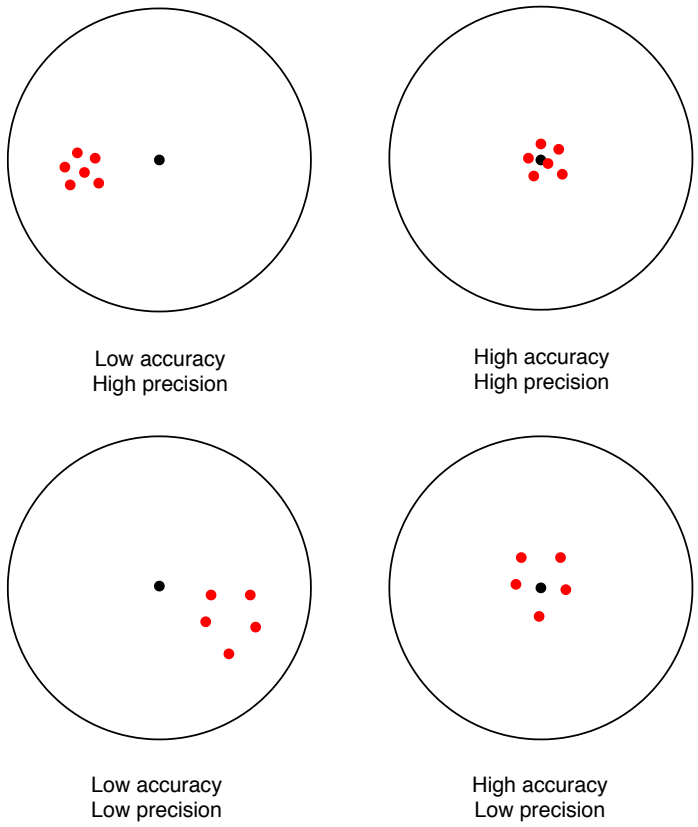


Figure 1: Difference between accuracy and precision

The four definitions, especially the uncertainty of the measurement, can indicate the quality of the measurement results (Figliola and Beasley, 2015). An uncertainty analysis is conducted to determine the uncertainty. The analysis is a numeric methodical approach, defining the potential error that is present in all data. The uncertainty of the measurement is estimated in an objective and standardized way.

When a measurement is conducted, several sources of uncertainty can be present. The following list is presented by the guide to identify possible sources (ISO/IEC 98-3:2008, 2008).

(a) incomplete definition of the measurand;

(b) imperfect realization of the definition of the measurand;

(c) nonrepresentative sampling — the sample measured may not represent the defined measurand;

(d) inadequate knowledge of the effects of environmental conditions on the measurement or imperfect measurement of environmental conditions;

(e) personal bias in reading analogue instruments;

(f) finite instrument resolution or discrimination threshold;

(g) inexact values of measurement standards and reference materials;

(h) inexact values of constants and other parameters obtained from external sources and used in the data-reduction algorithm;

(i) approximations and assumptions incorporated in the measurement method and procedure;

(j) variations in repeated observations of the measurand under apparently identical conditions.

Some of the possible sources can be considered as independent. It is assumed that the effects of (a)-(i) can contribute to the effect of (j).

The measurand is often determined from other quantities, instead of being measured directly. N quantities, $X_1, X_2, ..., X_N$, determine the measurand, Y, through a functional relationship, f, defined in Equation 1.

$$Y = f(X_1, X_2, ..., X_N) \tag{1}$$

y denotes the estimate of the measurand Y and the input estimates, $x_1, x_2, ..., x_N$, denotes the input quantities, $X_1, X_2, ..., X_N$. The result of the measurement, being the output estimate is defined in Equation 2.

$$y = f(x_1, x_2, ..., x_N) \tag{2}$$

The combined standard uncertainty, $u_c(y)$, is determined by the standard uncertainty of each input estimate, $u(x_i)$. It is the estimated standard deviation of the estimate of the measurand, y. When a statistical model is used in the estimations, the estimated variance of the uncertainty, u^2, is the statistical variance, s^2, and the estimated uncertainty, u, is equal to the standard deviation, s.

If all input quantities are independent, they are uncorrelated and the combined standard uncertainty, $u_c(y)$, is estimated from Equation 3.

$$u_c^2 = \sum_{i=1}^{N} \left(\frac{\partial f}{\partial x_i} \right)^2 u^2(x_i) \tag{3}$$

If two or more input quantities are correlated, the standard uncertainty, $u_c(y)$, is estimated from Equation 4. The estimated covariance between x_i and x_j, $u(x_i, x_j) = u(x_j, x_i)$, is estimated from a correlation coefficient between minus one and one.

$$u_c^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} u(x_i, u_j) \tag{4}$$

When needed, the expanded uncertainty is estimated. An interval around the measurement result is defined, based on the required level of confidence. The expanded uncertainty of the measurement, U, is defined as the standard uncertainty of the output estimate, $u(y)$, multiplied with a coverage factor, k, as in Equation 5. The coverage factor decides the level of confidence of the uncertainty if a normal distribution is used in the calculations and the reliability of the standard uncertainty is sufficient. It is often between two and three, as $k = 2$ provides a confidence level of approximately 95 % and $k = 3$ provides a confidence level of approximately 99 %probability.

$$U = ku(y) \tag{5}$$

2.2.1 Comparing ANSI/ASME and ISO

The standard and the guide are similar in many ways, though they differ in some of the terminologies. As the error is an idealized concept and can not be known precisely, it opens for several categorizations. The main difference in the standard and the guide is how the errors are categorized.

As stated in the standard, measurement error is present in all measurements (ASME PTC 19.1-2013, 2013). The final error of the measurement, δ_k, is divided into two components, a systematic error, β, and a random error, ϵ_k, as in Equation 6.

$$\delta_k = \beta + \epsilon_k \tag{6}$$

Two other names for the systematic error and the random error are biased error and precision error, respectively. There is a third component to the error, blunders. This component is neglected, as the blunders are assumed to be absent with good engineering practice.

Figure 2 illustrates the relationship between the true value of the measurand and the measured values. The illustration includes the effect of systematic and random errors. The systematic error is a fixed value, causing the mean of the sample to shift from the mean of the true value. When several measurements are conducted, the random errors cause the measured values to range within a distribution around the mean of the sample.

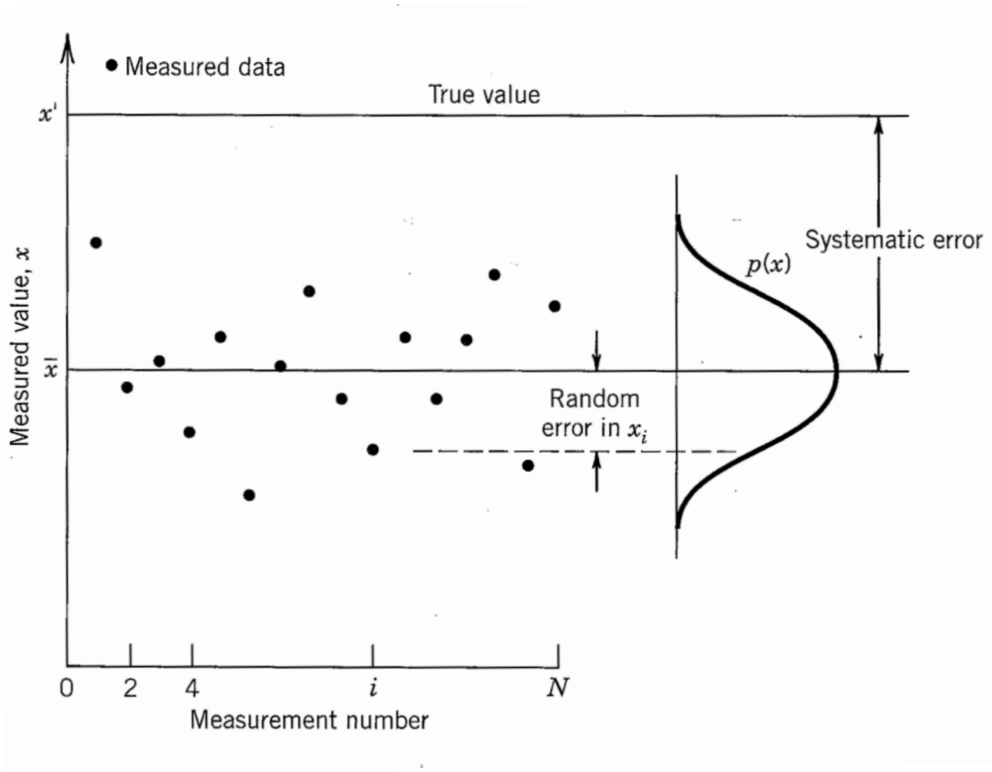
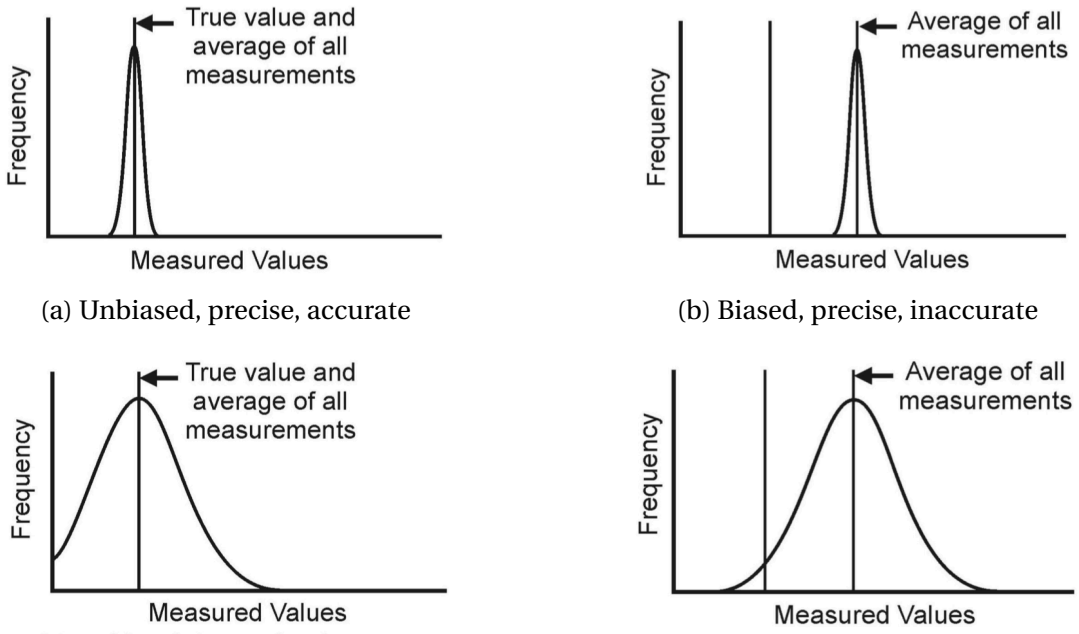


Figure 2: The distribution of error in a repeated measurement (Figliola and Beasley, 2015)

During fixed measurement conditions, the systematic error is fixed. As the value is constant and could be both high and low, it can be difficult to estimate it. The systematic error can be adjusted and reduced but not eliminated. A calibration procedure of the measurement equipment can uncover the systematic error in the equipment and indicate an associated uncertainty. The calibration must follow a standard and method, such as presented by ANSI/ASME or ISO and be performed by a quality instrument with an associated calibration certificate. Other methods to

estimate the systematic error can be concomitant methodology, interlaboratory comparisons or judgment/experience (Figliola and Beasley, 2015). When a systematic error is quantified, a correction factor in the measurement procedure can compensate for the effect that causes the systematic error. After an adjustment is introduced, the systematic error is assumed to be zero.

During fixed measurement conditions and repeated measurements, the random error is noticeable as the measurement values are scattered. Causes of the random errors can be calibration of the equipment, repeatability and resolution of the components in the measurement system, the technique of the measurement procedure, variations in the measured object (measurand) and environmental conditions (Figliola and Beasley, 2015). By increasing the number of measurements, the random error can be reduced, but not eliminated. The deviation of the mean of the measured values indicated the uncertainty of the mean, due to the effects that cause the random error. However, the value of the error in the mean due to the random error cannot be estimated.



(a) Unbiased, precise, accurate

(b) Biased, precise, inaccurate

(c) Unbiased, imprecise, inaccurate

(d) Biased, imprecise, inaccurate

Figure 3: Visualization of the error effects of a measurement (AIAA, 1995)

All measurements have some quantity of both systematic and random errors. The idealized graphic illustrations in Figure 3, shows how the random and systematic effects influence the distribution of the measurement values when repeating a measurement. There are no random effects in Figure 3a and Figure 3c. The distribution is high and thin in (a), as the measurements

are precise and accurate. In (c), the measurements are both imprecise and inaccurate, causing a large range in the distribution around the mean. There are random effects in both Figure 3b and Figure 3d. The random effects, combined with the inaccuracy, shifts the mean away from the mean of the true value. As the measurements in (b) are precise, the distribution is high and thin. As the measurements in (d) are imprecise, the measurements are more scattered, creating a larger distribution.

Stated by the guide of ISO, the errors are categorized by how the estimation of their uncertainties is defined (ISO/IEC 98-3:2008, 2008). The classification indicates how the evaluation of uncertainty components should be. It does not imply that the nature of the components is different.

Probability distributions are the basis of both type A and type B evaluation, where variances and standard deviations specify the uncertainty components.

Type A evaluation of uncertainty is based on a series of observations, resulting in a statistical analysis (ISO/IEC 98-3:2008, 2008). During fixed measurement conditions, independent measurements q_k are conducted n times. The arithmetic mean is defined as in Equation 7.

$$\bar{q} = \frac{1}{n}\sum_{k=1}^{n} q_k \tag{7}$$

The experimental variance of the measurements, due to random effects, is defined as in Equation 8. The experimental standard deviation is defined as the positive square root, $s(q_k)$.

$$s^2(q_k) = \frac{1}{n-1}\sum_{j=1}^{n}(q_j - \bar{q})^2 \tag{8}$$

The experimental variance of the arithmetic mean is as defined in Equation 9.

$$s^2(\bar{q}) = \frac{s^2(q_k)}{n} \tag{9}$$

Type B evaluation of uncertainty is based on other methods than statistical analysis from a series of observations (ISO/IEC 98-3:2008, 2008). The variance is estimated, based on available

information. ISO establishes possible information sources such as "previous measured data; experience with or general knowledge of the behaviour and properties of relevant materials and instruments; manufacturer's specifications; data provided in calibration and other certificates; uncertainties assigned to reference data taken from handbooks" (ISO/IEC 98-3:2008, 2008).

The uncertainties associated with errors due to systematic or random effects are evaluated by type A in some cases and type B in other cases. ISO/IEC 98-3:2008 (2008) argues that the categorization of uncertainty components based on the origin of the effect, such as systematic and random errors, may be ambiguous. The uncertainty components in one category may be estimated with different methods. Categorization by the standard of ISO maintains a precise evaluation and discussion of the uncertainty components, as the evaluation method categorizes them (ISO/IEC 98-3:2008, 2008).

If the standard of ANSI/ASME or the guide of ISO is utilized to perform an uncertainty analysis, the resulting uncertainty to the measurement should be similar. Where ANSI/ASME categorizes the error into systematic and random errors, ISO categorizes errors into type A and type B. The effects that cause the errors classify the systematic and random errors. Type A and type B errors are classified by how the uncertainties of the effects causing the errors are estimated.

The article, "Comparison of ANSI/ASME and ISO models for calculation of uncertainty", by Steele et al. (1994) explored how the use of either method of the two standards affected the outcome of an uncertainty analysis. The robust statistical tool, Monte Carlo simulations, was used to simulate several experiments to obtain an uncertainty interval, with a confidence interval defined as either 95 % or 99 %. The methodology in the two standards was different, resulting in different outcomes depending on which model was used to estimate the uncertainty. Both the calculated uncertainty intervals and the confidence level provided by the intervals were different. The study of Steele et al. (1994) concluded with ISO providing the most accurate results. The models presented by ISO were more appropriate for uncertainty analysis than those presented by ANSI/ASME. Later, after the revision of the ANSI/ASME standard, the models of ISO were incorporated in the standard of ANSI/ASME. In the current versions of the standard and the guide, the main difference is how the errors are categorized. However, the outcome of an uncertainty analysis following either of the standards is similar (Figliola and Beasley, 2015). Both standards define that the categorization of each error has no impact on the methods used in the uncertainty analysis. The categorization of error is for convenience. As all uncertainty components are treated the same in the methods, the combined standard uncertainty or expanded uncertainty of a measurement result is unaffected of the categorization of errors.

2.2.2 Uncertainty analysis

The true value of the measurand cannot be known. As the uncertainty is a property of the measurement result, valuable knowledge of how close the measured value possibly is to the true value is gained through an uncertainty analysis. While a number defines the uncertainty, errors are defined as effects (Figliola and Beasley, 2015), causing the measured value to be different from the true value. The uncertainty analysis identifies, quantifies and combines the different errors of the measured value, resulting in an interval around the measured value. It is expected, with a stated probability, that the true value of the measured object lies within the interval around the measurement result. The measured value can be very close to the true value, even though the measured value has a large uncertainty.

The following assumptions, following the ANSI/ASME categorization of errors, are made when an uncertainty analysis is conducted (Figliola and Beasley, 2015):

1. The measurement process is clearly defined and the objectives of the test are known.
2. When possible, a correction factor is applied in the measurement procedure to compensate for the effect causing a systematic error. The uncertainty of the correction is assumed to be the uncertainty of the systematic error.
3. A normal distribution of reporting of uncertainties and errors is assumed unless otherwise stated.
4. Independent, thus uncorrelated of each other, errors are assumed, unless otherwise stated.
5. The engineer has some "experience" with the system components.

All steps of an uncertainty analysis require the experience and judgment of an engineer. The flow chart in Figure 4 shows clear guidance of the order in the performance of the analysis, as presented in the guide.

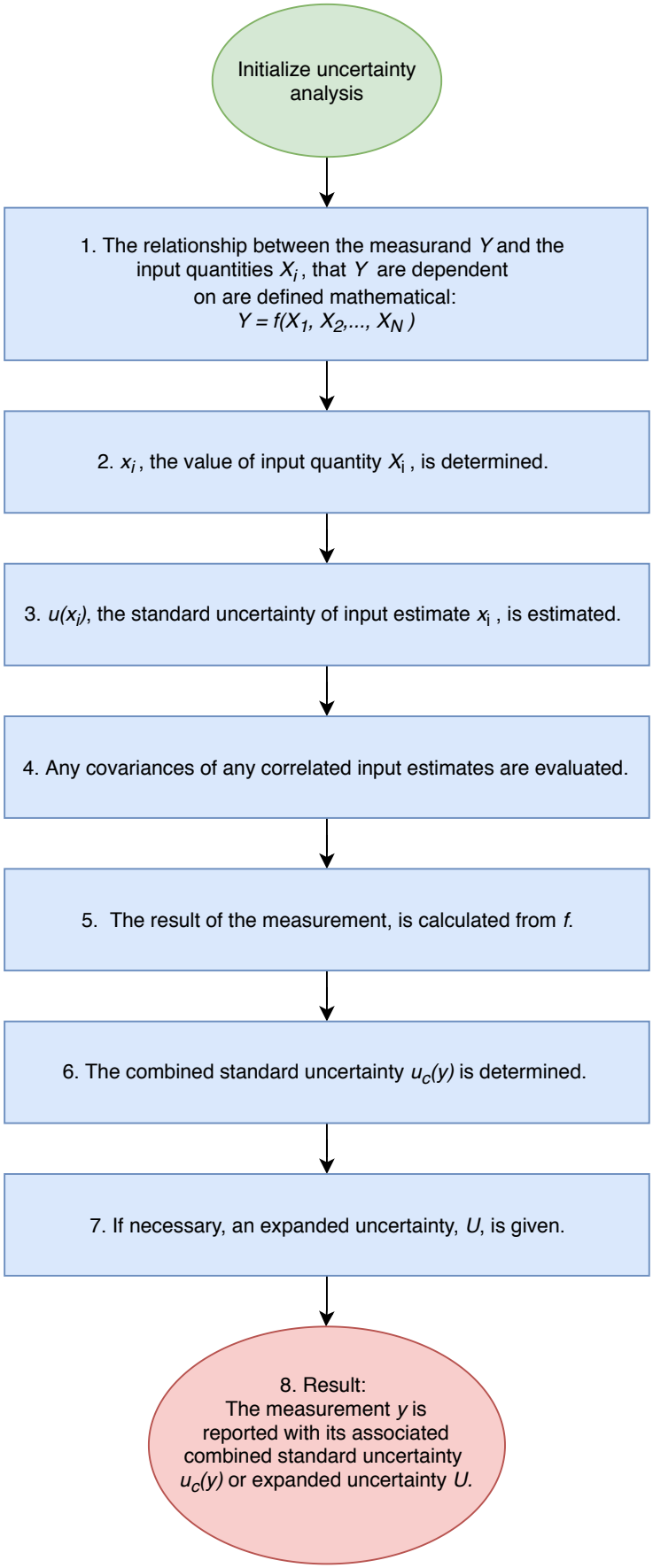


Figure 4: Flow chart of an uncertainty analysis (ISO/IEC 98-3:2008, 2008)

The guide presents the following steps of an uncertainty analysis. The steps complement the corresponding numbers in the flowchart. Chapters in the guide are referred in the list.

1. Express mathematically the relationship between the measurand Y and the input quantities X_i on which Y depends: $Y = f(X_1, X_2, ..., X_N)$. The function f should contain every quantity, including all corrections and correction factors, that can contribute a significant component of uncertainty to the result of the measurement (see 4.1.1 and 4.1.2).

2. Determine x_i, the estimated value of input quantity X_i, either on the basis of the statistical analysis of series of observations or by other means (see 4.1.3).

3. Evaluate the *standard uncertainty* $u(x_i)$ of each input estimate x_i. For an input estimate obtained from the statistical analysis of series of observations, the standard uncertainty is evaluated as described in 4.2 (*Type A evaluation of standard uncertainty*). For an input estimate obtained by other means, the standard uncertainty $u(x_i)$ is evaluated as described in 4.3 (*Type B evaluation of standard uncertainty*).

4. Evaluate the covariances associated with any input estimates that are correlated (see 5.2).

5. Calculate the result of the measurement, that is, the estimate y of the measurand Y, from the functional relationship f using for he input quantities X_i the estimates x_i obtained in step 2 (see 4.1.4).

6. Determine the *combined standard uncertainty* $u_c(y)$ of the measurement result y from the standard uncertainties and covariances associated with the input estimates, as described in Clause 5. If the measurement determines simultaneously more than one output quantity, calculate their covariances (see 7.2.5, H.2, H.3, and H.4).

7. If it is necessary to give an *expanded uncertainty* U, whose purpose is to provide an interval $y - U$ to $y + U$ that may be expected to encompass a large fraction of the distribution of values that could reasonably be attributed to the measurand Y, multiply the combined standard uncertainty $u_c(y)$ by a *coverage factor* k, typically in the range 2 to 3, to obtain $U = ku_c(y)$. Select k on the basis of the level of confidence required of the interval (see 6.2, 6.3, and especially Annex G, which discusses the selection of a value of k, that produces an interval having a level of confidence close to a specified value).

8. Report the result of the measurement y together with its combined standard uncertainty $u_c(y)$ or expanded uncertainty U as discussed in 7.2.1 and 7.2.3; use one of the formats recommended in 7.2.2 and 7.2.4. Describe, as outlined also in Clause 7, how y and $u_c(y)$ or U were obtained.

2.3 Calibration process

A calibration process in metrology is a procedure where a device is tested by comparing the measurement values of the device to a calibration standard. The calibration standard has a calibration certificate where the known uncertainty is stated. By regularly calibrating a device, the created measurement data is ensured to have low uncertainty and be within the MPE.

The calibration process can lead to a significant error being discovered. If no significant error is discovered, there is no need for an adjustment. If a significant error is discovered, either no adjustment is made, or an adjustment to correct the error is made. A calibration process is to be conducted once more after a correction to ensure that the error is at an acceptable level. The potential adjustment is not a part of the calibration process itself, as the calibration process is the process of comparison.

A National Metrological Institute, in Norway - Justervesenet, has national standards that the calibration standard often is traceable to.

A scientific guidance standard is made by the European Accreditation (EA), "Evaluation of the Uncertainty of Measurement in Calibration". It encourages to make a report after a calibration process. It should include a title describing the calibration, a description of how the measurement is conducted and the applied evaluation model with a description of applied symbols. All inputs data with descriptions of how they are obtained, the evaluation of statistical parameters and the observations listed, a table of an uncertainty budget, the expanded uncertainty of measurement and the complete result of measurement to be reported should also be included. (EA-4/02 M, 2013)

2.4 Dimensional measuring devices

As metrology is dependent on high numerical accuracy in a measurement process, it is necessary with highly specialized metrology equipment. The metrology laboratory at NTNU possesses two measuring devices, both from Hexagon Manufacturing Intelligence. A coordinate measurement machine (CMM), Leitz PMM-C 600 with the software PC-DMIS, and a laser tracker, Leica Absolute Tracker AT960 with the software Inspire.

The CMM has a higher accuracy than the laser tracker. The functionality is better, however, the associated software, PC-DMIS, is advanced and requires more from the operator than the associated software of the laser tracker. When 'up-close' measurement is not an option, due to

the size or complexity of the geometric object, the laser tracker is used.

2.4.1 Coordinate measurement machine: Leitz PMM-C 600

Leitz PMM-C 600 is an ultra-high precision CMM and gear measuring machine. By sensing discrete points on the surface of a physical object with a probe, the machine measures the geometry.

Figure 5: Leitz PMM-C 600

Figure 5 shows the CMM. The closed frame design and the materials used to build it will ensure long-term stability. The base is made of granite and has a fixed portal made of cast iron and a crossbeam made of granite. To ensure consistent accuracy over the complete measurement volume, the stiffness of the measurement axes are high. The moving measuring table ensures efficient courses of motion, with no twisting or tilting, as well as ensuring a constant dimensional relationship. The measurement results are highly repeatable because of the high-resolution scales. An active pneumatic damping system eliminates any influence of vibrations.

The measurement machine uses tactile probes to ensure the highest accuracy. It can also be equipped with optical probes for non-contact measurements.

The maximum permissible error (MPE) in the measurement machine, as defined by the

ISO 10360 Standards for CMMs (ISO 10360-2:2009, 2009) (ISO 10360-4:2000, 2000) (ISO 10360-5:2010, 2010), is defined in Equation 10, where L is measured in mm. The probing frequency is 40 points/min.

$$MPE_{CMM} = \left(0.6 + \frac{L}{600}\right)\mu m \tag{10}$$

2.4.2 Laser tracker: Leica Absolute Tracker AT960

The laser tracker is a portable laser measurement system available with probe, reflector and non-contact scanner measurement. It can be looked at as a walk-around CMM.

Figure 6a shows the laser tracker. It is easier to operate than the stationary CMM. It can be transported, as well as unpacked and powered up in any location in minutes.

NTNU possesses a probe, seen in Figure 6b. It is a hand-held, wireless device that lets the operator walk around with the device. It is installed with an automated probe identification, ensuring the connection with the laser tracker and reduces operator errors. The probe can be exchanged without any need for calibration. The measurement device is applicable for volumes up to ⌀40m.

(a) Leica Laser Tracker AT960

(b) Leica T-Probe

Figure 6: The laser tracker and the probe associated to the laser tracker

The MPE of the laser tracker is defined in Equation 11.

$$MPE_{laser\_tracker} = \left(\pm 15 + 0.006 \frac{1}{\text{mm}}\right)\mu\text{m} \tag{11}$$

The probe has an MPE as defined in Equation 12. To find the complete accuracy of the measurement, the MPE of the probe needs to be added to the MPE of the laser tracker.

$$MPE_{probe} = \pm 35\,\mu\text{m} \tag{12}$$

2.5 Computational geometry

The branch within computer science that applies to the study of geometric algorithms is called computational geometry. It can be divided into two, combinatorial computational geometry and numerical computational geometry. Where numerical computational geometry is devoted to the problems of representing real-world objects, for example in CAD or CAM systems, combinatorial computational geometry is devoted to algorithms where discrete entities describe the geometry. An example could be a convex hull problem of a data set of points in three dimensions.

2.5.1 Convex hull

Given a two- or three-dimensional set of n points, $P = p_1, p_2, ..., p_n$, a fundamental problem in computational geometry is to construct an efficient, unambiguous representation of the needed convex configuration. The convex hull is the subset of points that creates the smallest convex contour, containing P.

The hull is convex if any two points, p_i, p_j, where $i, j \leq n$, in the set can connect and the segment is entirely inside the hull.

In two dimensions, the convex hull is a polygon. It can be seen as the remaining points that hold an elastic rubber band from collapse, if the rubber band can not go through a point, as seen in Figure 7.

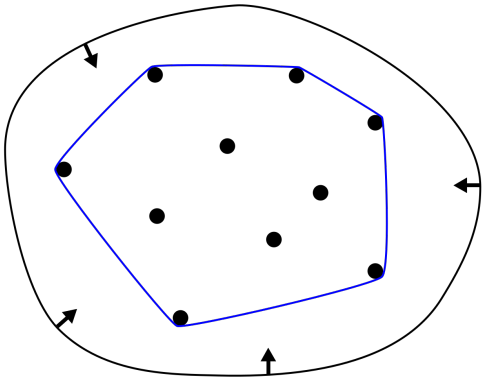


Figure 7: Convex hull of points in two dimensions

In three dimensions, the convex hull is a polyhedron. A polyhedron is a region of space consisting of faces, edges and vertices, where all the faces intersect and the surface is connected, as seen in Figure 8.

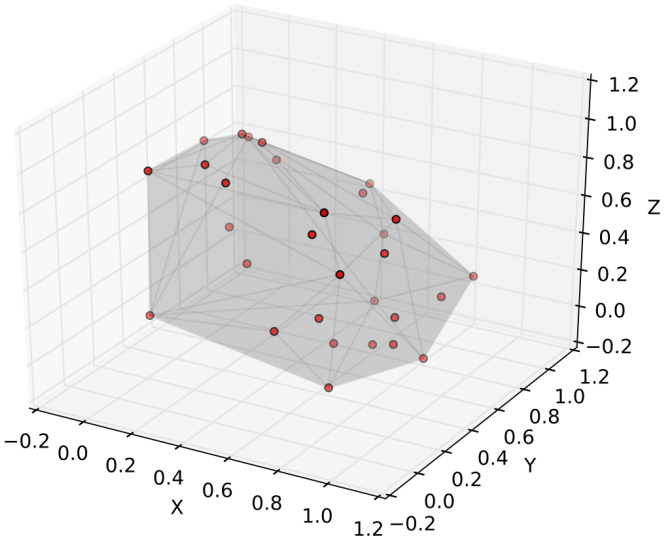


Figure 8: Convex hull of points in three dimensions (Moritz, 2013)

Common algorithms to find the convex hull of a set of points in three dimensions are gift wrapping (Jarvis, 1973), divide and conquer(Preparata and Hong, 1977) and QuickHull(Bykat, 1978; Eddy, 1977; Green and Silverman, 1979; Preparata and Shamos, 1985), among others.

2.6 Rotation matrix

A 3 x 3 rotation matrix defines a rotation of points in the three-dimensional Euclidean space. An axis of rotation and an angle, defining the amount of rotation about an axis, can define any arbitrary rotation. To find an arbitrary rotation, an element wise rotation can be performed around the $x-$, $y-$ and $z-$axis. The arbitrary rotation matrix, R, can be found by matrix multiplication of R_x, R_y, R_z.

A rotation about the $x-$axis with an angle α is defined in Equation 13, a rotation about the $y-$axis with an angle of β is defined in Equation 14 and a rotation about the $z-$axis with an angle of γ is defined in Equation 15.

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \tag{13}$$

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \tag{14}$$

$$R_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{15}$$

The order of the element wise rotation decide the final rotation, R. There are six different orders to rotate around the $x-$, $y-$, and $z-$axis. It can be $R_xR_yR_z$, $R_xR_zR_y$, $R_yR_xR_z$, $R_yR_zR_x$, $R_zR_xR_y$, $R_zR_yR_x$, all six orders resulting in different rotations. The standard is to rotate around the $z-$axis, then the $y-$axis and then the $x-$axis, as this corresponds to roll, pitch and yaw. This gives the arbitrary rotation as in Equation 16.

$$R = R_{zyx} = R_z(\gamma)R_y(\beta)R_x(\alpha) \tag{16}$$

To rotate a point, the original point (x, y, z) is represented as a row vector, $[x, y, z]$ or a column

vector, $[x, y, z]^T$, and multiplied with the rotation matrix, R, giving the rotated point (x', y', z'), shown in Equation 17.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} R \tag{17}$$

2.7 Minimum bounding objects

There exist several methods and algorithms to find the minimal enclosing object to a data set of points. Both brute force methods and optimization algorithms can be used. The development of a solution, when a data set consists of points in two dimensions, can be quite straight forward. However, even with two dimensions, the methods differ from simple solutions to more advanced, depending on the accuracy expected from the result. When a data set consists of points in three dimensions, such as a data set from a measured object in the metrology laboratory at NTNU, the development of a solution is more advanced. As the minimal enclosing object to a data set may not be axis-aligned, the algorithm needs to try different orientations to find the minimal enclosing object. Especially when the object is a parallelepiped, the optimal orientation of the parallelepiped can be advanced.

2.7.1 Minimum bounding cylinder

A cylinder is a three-dimensional geometric solid. Figure 9 shows a standard cylinder with circular ends of radius r. The ends of the cylinder are perpendicular to the axis of the cylinder, and the perpendicular distance between them is the height h. The radius and the height define the size of the cylinder.

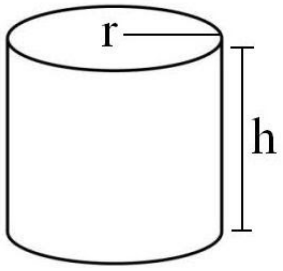


Figure 9: A cylinder with radius r and height h

The volume, defined in Equation 18, is minimized to find the minimum cylinder.

$$V_c = \pi r^2 h \tag{18}$$

The minimum enclosing cylinder of a data set of 3D points minimizes the volume while all the points are within the boundaries of the shape of the cylinder. The two-dimensional problem is explored, where the object is to find the minimum enclosing circle to a data set of 2D points in one plane, before the three-dimensional problem is explored.

Minimum enclosing circle in 2D

The problem of finding the minimum enclosing circle in 2D was first presented by Sylvester (1857). The area of a circle with radius r, defined in Equation 19 is minimized to find the minimum circle.

$$A_c = \pi r^2 \tag{19}$$

A common technique to solve the geometric problem is to find the minimum circle defined by two or three points in the data set, that lay on the boundary of the minimum circle. If three points define the minimum circle, the triangle defined by the respective points cannot be obtuse. Thus it has to be acute (all three angles are less than 90°). If two points define the minimum circle, the line segment of the respective points must be defined as the diameter of the minimum circle. The minimum circle is defined to be unique.

Several solutions to the two-dimensional problem are available as open-source code in different programming languages. A program, "Smallest enclosing circle" produced by Project Nayuki (2018), is used in this project for exploration. It is free software, open to redistribution and modification under the terms of the GNU Lesser General Public License as published by the Free Software Foundation. The algorithm is included in section A.4.

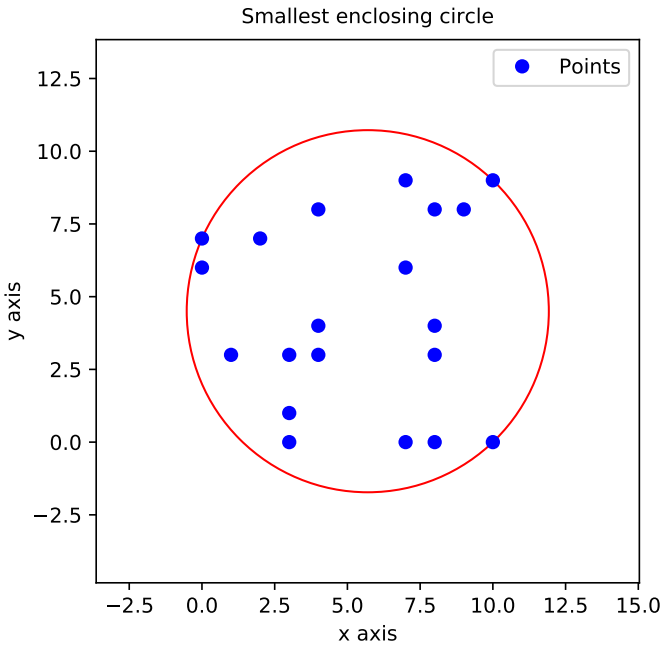


Figure 10: Minimum enclosing circle

Figure 10 shows a plot of the smallest circle enclosing data points on a plane. The input in the algorithm is 20 random generated points and the output is the center of the circle, defined by an x- and y-coordinate, and the radius of the circle. As visualized in the figure, three of the data points lay on the boundary of the circle and the remaining points are enclosed by the circle.

Minimum bounding cylinder in 3D

In some applications, such as a calibration process of a cylinder in a CMM in the industry, high numerical accuracy is crucial to the results. The object is to find the minimum circumscribed cylinder from a data set of 3D points. For other applications, such as object detection, autonomous navigation, manufacturing and quality control, a robust cylinder fitting can be sufficient (Nurunnabi et al., 2019).

In Figure 11, the least square circle (LSC), the maximum inscribed circle (MIC) and the minimum circumscribed circle (MCC) to a cross-section of a cylinder is drawn to visualize the differences between the definitions.

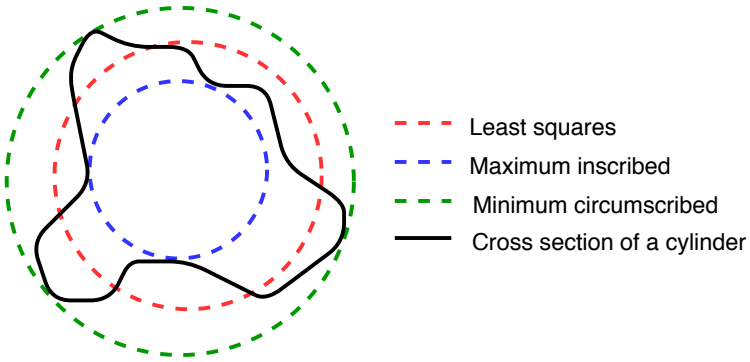


Figure 11: Cross section of a cylinder

Figure 12 shows three cross-sections of a cylinder drawn with exaggeration. The cylinder fitted with the least squares method, the maximum inscribed cylinder and the minimum circumscribed cylinder is drawn. It visualizes how the dimensions of the cylinder can vary, depending on the relation of the cylinder to the underlying geometry. Note that the height of the cylinders in the figure is the same and is affected by the longest distance measured and a potential rotation. A potential rotation is not included in this figure.

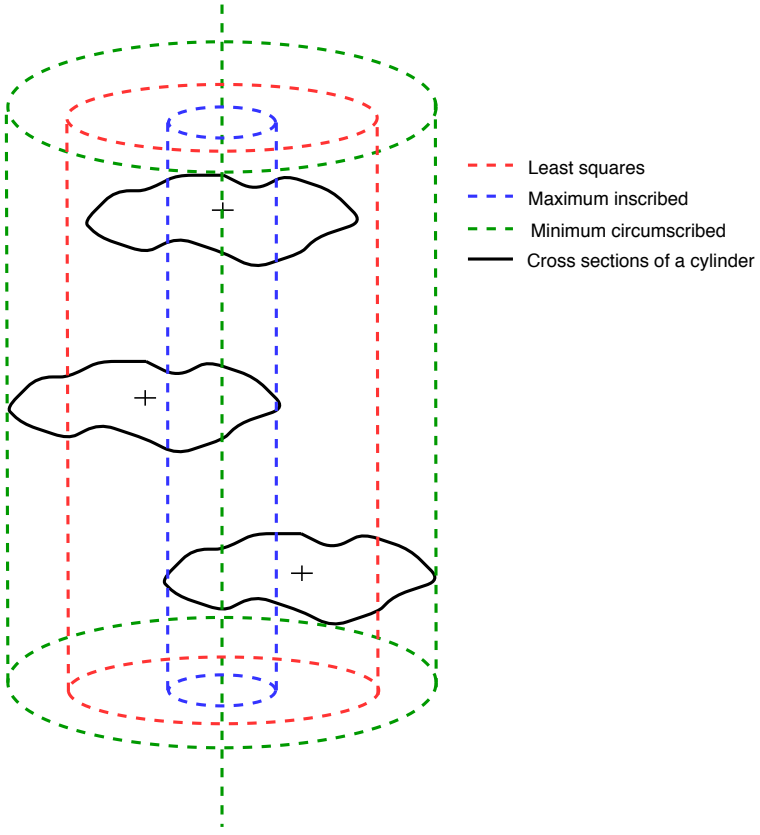


Figure 12: A measured object in the shape of a cylinder

Pan (2017) has developed an algorithm in Python, based on Eberly (2019) pseudo-code of "Fitting a Cylinder to 3D Points". The algorithm is suitable where the underlying geometry of the data is in the shape of a cylinder with possibly small errors. However, this is not a suitable algorithm for cylinders measured in the metrology laboratory as the resulting cylinder is fitted with the least squares method. Nurunnabi et al. (2019) proposes two new variants of robust cylinder fitting, comparing them to existing known approaches such as the least squares method, singular value decomposition (SVD) and principal component analysis (PCA). An algorithm presented by Schömer et al. (2000), combines a general linearization technique with a parametric search, that is efficient for finding the smallest cylinder. However, as stated by Schömer et al. (2000), the results seem mainly of theoretical interest and not for practical applicability, such as metrology. Chan and Tan (2004) has developed an algorithm to check if a given object can fit inside a cylindrical bounded volume. They propose that the algorithm can be a solution to the optimization problem of the smallest enclosing cylinder of a data set of 3D points. However, their solution is restricted to find the minimum height or the minimum diameter of the enclosing cylinder, but not both. An algebraic method to compute the smallest enclosing and circumscribing cylinder of a data set of 3D points is presented by Brandenberg and Theobald (2004). This solution is only applicable when the finite point set P consists of $n = 4$ points. Petitjean (2012) proposes an algebraic algorithm to compute the smallest enclosing cylinder. As well as in the algorithm of Brandenberg and Theobald (2004), the algorithm is limited by the number of points in the data set. Petitjean (2012) pays special attention to $n = 4$ and $n = 5$, n being the number of points in the data set.

2.7.2 Minimum bounding box

In this project, a box is defined as a parallelepiped; a three-dimensional geometric figure, formed by six parallel planes. All parallelepipeds in this project are rectangular. Hence the planes are either perpendicular or parallel to each other. Figure 13 shows how the dimensions of a rectangular parallelepiped is defined by the width, w, the length, l and the height, h.

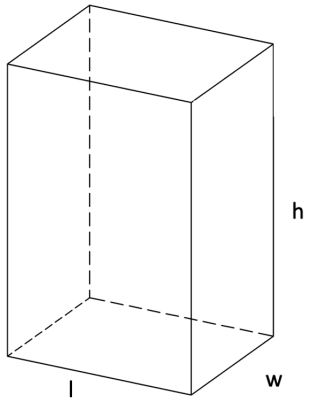


Figure 13: A rectangular parallelepiped with width w, length l and height h

The volume, defined in Equation 20, is minimized to find the minimum parallelepiped.

$$V_p = wlh \tag{20}$$

The minimum enclosing parallelepiped of a data set of 3D points is found when the volume is minimized while all the points are within the boundaries of the shape of the parallelepiped. The problem is explored in two dimensions before the three-dimensional solution is explored and presented.

Minimum enclosing rectangle in 2D

The minimum enclosing rectangle also called the minimum bounding rectangle (MBR) and minimum area rectangle (MAR), is a two-dimensional case of the minimum bounding box in three dimensions. It is also a subproblem of the minimum volume box enclosing a convex polyhedron presented by O'Rourke (1985) and discussed in subsection 2.8.2.

The area of a rectangle with width w and length l, defined in Equation 21, is minimized to find the minimum rectangle. With a data set of 2D points in one plane, the minimum bounding rectangle contains all the points.

$$A_r = wl \tag{21}$$

Given a data set of 2D points, a convex polygon, also defined as the convex hull, can be created. Freeman and Shapira (1975) presented three theorems to prove that one side of the minimum enclosing rectangle must be coincident with one of the edges of the convex hull.

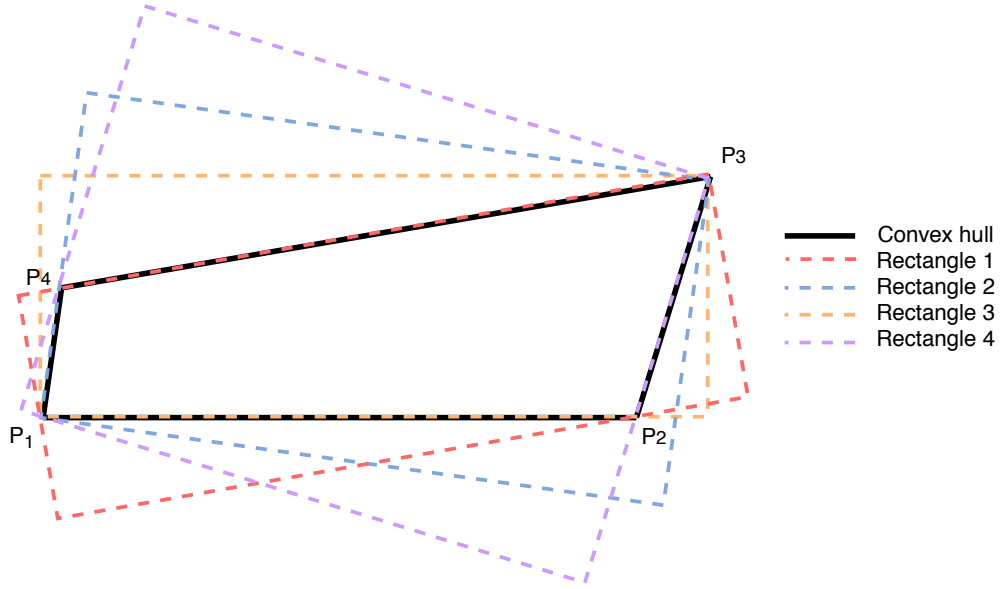


Figure 14: A convex hull with four enclosing rectangles in two dimensions

Figure 14 shows a convex hull of a data set of m points in 2D. The convex hull is defined as $P = (P_1, P_2, ..., P_n)$, enclosing all the points in the data set. Shown in the figure, $n = 4$. n is possibly much larger than m. Rectangle 1, 2, 3 and 4 are drawn to visualize different bounding rectangles. From the proof of Freeman and Shapira (1975), the four rectangles are the candidates of the minimum area rectangle. Rectangle 1 coincide with the edge P_3P_4 of the convex hull, Rectangle 2 is coincide with the edge P_1P_4, Rectangle 3 coincide with the edge P_1P_2 of the convex hull, while Rectangle 4 is coincide with the edge P_2P_3. Calculation of the areas of the rectangles give $A_{R_1} < A_{R_3} < A_{R_2} < A_{R_4}$. Rectangle 1, which is the minimum bounding rectangle of the convex hull, coincide with the longest edge of the convex hull.

The brute force approach to find the minimal bounding rectangle of the convex hull, presented in Freeman and Shapira (1975) will use $O(n^2)$ time. First, n bounding rectangles are constructed, one for each edge of the convex hull, and then the rectangle with the minimum area is chosen as the minimum bounding rectangle. Another method, presented by Toussaint (1983) finds the minimal bounding rectangle of the convex hull in $O(n)$ time. The algorithm is more efficient, as it uses the rotating calipers method (Shamos, 1978).

Regardless of which of the two algorithms that are used, the brute force approach or the more

efficient method with rotating calipers, the convex hull of the data set needs to be computed. This applies to both two-dimensional and three-dimensional cases.

Types of bounding boxes

There are different definitions of bounding boxes. The different types are used for different purposes and give very different results.

The axis aligned bounding box (AABB) is defined as the minimized bounding box with axes parallel to the coordinate frame. Figure 15 shows the AABB of a convex hull, in two dimensions.

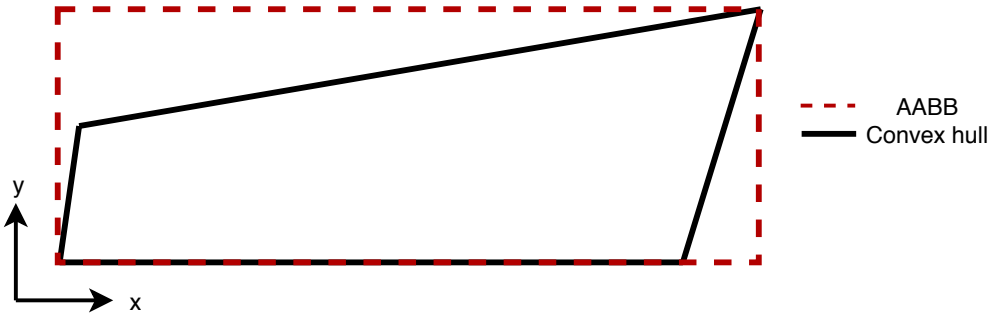


Figure 15: AABB of a convex hull in two dimensions

The dimensions of the AABB are decided by the minimum and maximum values along the axes, x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}. The region S of the AABB is defined in Equation 22.

$$S = \{ (x, y, z) \mid x_{min} \leq x \leq x_{max},\ y_{min} \leq y \leq y_{max},\ z_{min} \leq z \leq z_{max} \} \tag{22}$$

An AABB can be simple to compute within a short amount of time. However, it often does not fit the underlying geometry.

The oriented bounding box (OBB) is defined as a rectangular parallelepiped, arbitrarily oriented around the axes of the coordinate frame. No axes limit the OBB that is minimized around the convex hull, enclosing all the points in the data set. Figure 16 shows the OBB of a convex hull, in two dimensions.

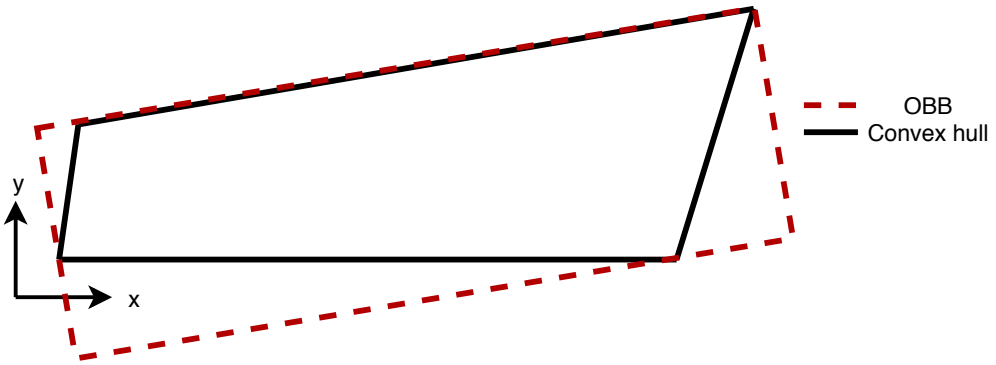


Figure 16: OBB of a convex hull in two dimensions

An OBB is defined by a center, defined in Equation 23, by dimensions, defined in Equation 24, and by an orientation, defined in Equation 25.

$$X \in \mathbb{R}^3 \tag{23}$$

$$\triangle \in \mathbb{R}^3 \tag{24}$$

$$R \in SO(3, \mathbb{R}) \tag{25}$$

2.8 Bounding box algorithms

When a bounding box is computed, the solution can be found by a brute force search or by mathematical optimization. A brute force search is a very general problem-solving technique. It will search through all possible solutions to decide the best one. As it always finds an optimal solution and can be easy to implement, it is a popular method when the number of possible solutions is low. The cost of the function is proportional to the number of possible solutions. Thus the method is not preferred to use when the size of a problem grows. When a problem is complex and is subject to specific limitations, conditions and assumptions, mathematical

optimization can be the right solution. An optimization algorithm will seek an optimal solution within the specifications.

2.8.1 Available bounding box algorithms

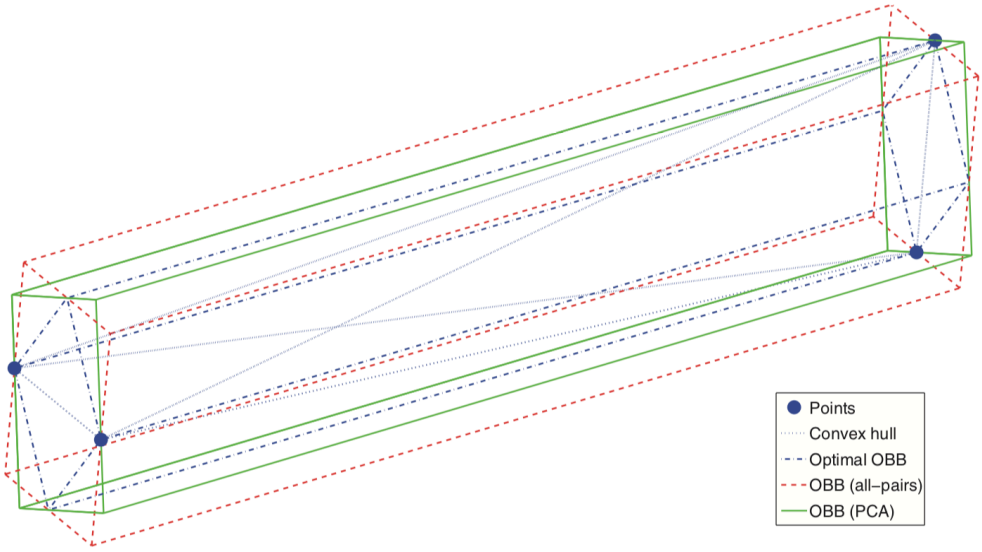
Several algorithms to compute a bounding box in three dimensions are available. Some of them use a brute force method, while others are optimization algorithms. The algorithm of choice is dependent on the application. The most important factors are the computation time of a solution and the accuracy of this solution.

Table 2 shows selected algorithms that are looked into. The complexities of the algorithms are dependant on either n, the number of points describing an object, or n_v, the number number of points in the convex hull (number of vertices).

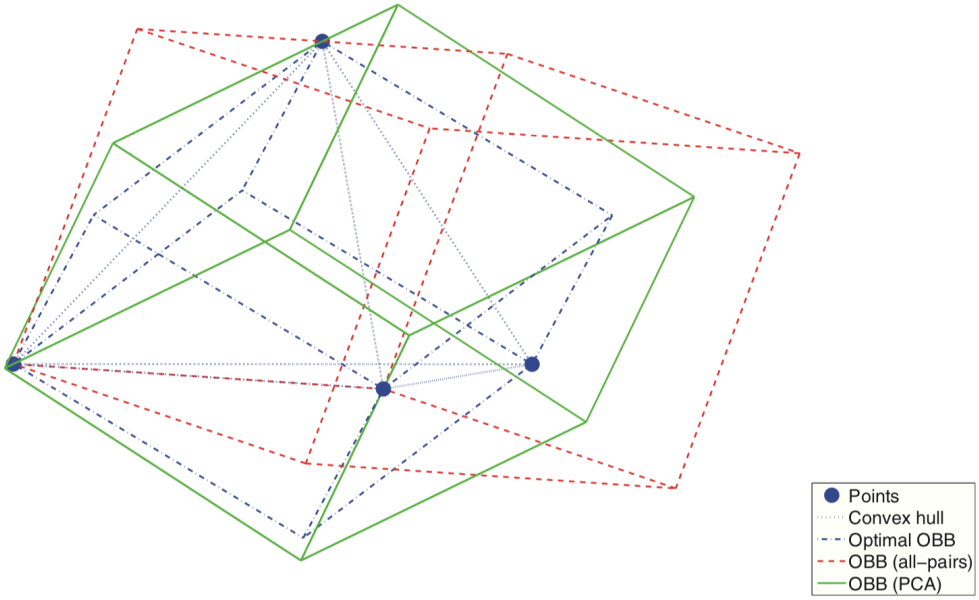
Table 2: Available bounding box algorithms

Algorithm	**Features**			
	Accuracy			**Complexity**
	Exact	**Exact in practice**	**suboptimal**	
O'Rourke	x	x		$O\left(n_v^3\right)$
TriangleMesh_IntersectRay_SSE		x		$O\left(n^{\frac{3}{2}}\left(\log n\right)^2\right)$
HYBBRID		x		$O(n_v)$
PCA			x	$O\left(n\right)$
All-pairs			x	$O\left(n_v^3\right)$
Korsawe			x	$O\left(n_v^2\right)$ or $O\left(n_v^3\right)$

There exist several PCA-based methods. Common for the algorithms is that they are easy to implement. However, the algorithms are sensitive to the distribution of points in the geometry. Found by Dimitrov et al. (2009), the most complex PCA-based method still provide a volume of the OBB that is four times larger than the exact solution in some instances. A brute force method, solving the three-dimensional problem by computing the associated two-dimensional problem is called "all-pairs" by Barequet and Har-Peled (2001). An algorithm developed by Korsawe, J. (2008) uses a two-dimensional minimum bounding rectangle approach to find a bounding box with one side coincide with one face of the convex hull.



(a) One optimal OBB and two fitted OBBs (Chang et al., 2011)



(b) One optimal OBB and two fitted OBBs (Chang et al., 2011)

Figure 17: Two examples of optimal OBB versus fitted OBBs

The PCA-based methods, "all-pairs" algorithm and the algorithm of Korsawe are suboptimal solutions more suitable for OBB fitting than finding the minimal OBB. Figure 17 shows the bounding box solution from the HYBBRID algorithm (optimal OBB), the "all-pairs" algorithm and the most complex PCA method. To be able to use a minimum bounding box algorithm in the measurement technology, the suboptimal solutions are not sufficient.

The algorithm of O'Rourke is an algorithm giving the exact solution of the minimum bounding box (O'Rourke, 1985). However, the algorithm has a high complexity giving a high computation time. The algorithm is described in subsection 2.8.2. The algorithm "TriangleMesh_IntersectRay_SSE" by Jylänki (2015) is not exact by proof. However, thorough testing by Jylänki (2015) has not discovered any scenarios where the found bounding box is not optimal. The complexity of the algorithm provides a lower computation time than the algorithm of O'Rourke. The HYBBRID algorithm by Chang et al. (2011) has a complexity providing a lower computation time than the algorithms of both O'Rourke and Jylänki. Even though the algorithm is an approximation algorithm, the bounding box found by the algorithm is exact in practice (Chang et al., 2011). The algorithm is described in subsection 2.8.3.

An implementation of the HYBBRID algorithm by Chang et al. (2011) in this master thesis turned out to be easier than an implementation of the "TriangleMesh_IntersectRay_SSE" algorithm by Jylänki (2015).

2.8.2 O'Rourke's algorithm

An exact algorithm to find the minimum enclosing box is described in an article of O'Rourke (1985). He investigated how to find the minimal volume box, circumscribing a given set of points in three dimensions.

As written in section 2.7.2, Freeman and Shapira (1975) proved that the solution to the corresponding two-dimensional problem, must have one edge coincident with one of the edges of the convex hull. A simple search for the three-dimensional solution could be to follow the corresponding strategy. However, O'Rourke shows that such a solution is problematic. The minimum volume box does not need to have any sides in flush with a face of the convex hull of the set of points (O'Rourke, 1985).

It is addressed that the solution to the minimum enclosing box can be arbitrarily oriented and the faces must meet orthogonally. However, the box is not necessarily unique.

Two conditions are stated to simplify the problem(O'Rourke, 1985): "First, it is obvious that every box circumscribing a set of n points also circumscribes the convex hull of those points." and "Second, it is obvious that a minimal box must touch the inscribed polyhedron on each of its six faces; otherwise a face would be moved inwards reducing the volume." A third condition that is less obvious is stated: "every minimal box must have at least two adjacent faces flush with the edges of the enclosed polyhedron." and proved by two theorems in the article (O'Rourke, 1985).

The algorithm described by O'Rourke requires $O(n_v^3)$ time to find the minimum bounding box, where n_v is the number of points in the convex hull. The algorithm provides an exact solution, but as the complexity is high, O'Rourke admits that the algorithm may not be optimal.

2.8.3 HYBBRID algorithm

A new approach to find the optimized minimum bounding box was introduced by Chang et al. (2011). The name HYBBRID was decided as it combines both genetic- and Nelder-Mead algorithms, resulting in a HYbrid Bounding Box Rotation IDentification algorithm (Chang et al., 2011). It is inspired by the algorithm of O'Rourke, described in subsection 2.8.2, and based on the method (a combination of Nelder-Mead and genetic algorithm) presented by Durand and Alliot (1999). HYBBRID was developed to result in an easily implemented optimization algorithm with a fast computation time, in contrast to the algorithm of O'Rourke, which is extremely hard to implement and has slow computation time (Chang et al., 2011).

HYBBRID solves the problem of finding the minimum volume of an arbitrarily oriented bounding box, enclosing a given set of N points, denoted as $X \subset \mathbb{R}^3$. That results in a rectangular parallelepiped with a minimum volume enclosing X. The solution is based on an unconstrained optimization problem of the rotation group $SO(3,\mathbb{R})$. The solutions should have good accuracy and find the optimal OBB in most cases, with low computational time. If a sub-optimal solution is returned, it should be close to the optimal solution.

$X \in \mathbb{R}^3$, $\triangle \in \mathbb{R}^3$ and $R \in SO(3,\mathbb{R})$ denotes the center, dimensions and orientation, respectively. The rotation group is defined as follows:

$$SO(3,\mathbb{R}) = \{\, R \in GL(3,\mathbb{R}) \mid R^T R = I = RR^T,\ \det(R) = 1 \,\} \tag{26}$$

$GL(3,\mathbb{R})$ is a set of three by three invertible real matrices, defined as the general linear group of degree three.

The problem to be solved can be written as follows:

$$\min_{R \in SO(3,\mathbb{R})} f(R) \tag{27}$$

where the volume of the AABB of X rotated by R, defines the objective function $f(R)$. $f(R)$ is

defined in Equation 28, where $X_i \in X$. $\triangle = \triangle_\xi \triangle_\eta \triangle_\zeta$ specifies the dimensions of the OBB. The center after rotation by R is defined as Ξ.

$$f(R) = \left(\begin{array}{cc} \min\limits_{\triangle, \Xi \in \mathbb{R}} & \triangle_\xi \triangle_\eta \triangle_\zeta \\ s.t. & -\frac{\triangle}{2} \leq RX_i - \Xi \leq \frac{\triangle}{2} \ \forall i \in \{1,\ldots,N\} \end{array} \right) \tag{28}$$

The iterative optimization solution ensures that the position, dimensions and orientation minimize the volume of the OBB while enclosing X. It is a derivative-free solution, as the objective function in Equation 28 is non-differentiable, with a global search technique and a fast convergence rate. The genetic algorithm is used as the global exploration component and the Nelder-Mead simplex algorithm ensures a high convergence rate. More precisely, the genetic algorithm finds a global optimum and the Nelder-Mead algorithm converges to a local minimum. The combination of these two finds a global minimum of the volume function in Equation 28.

As described in Chang et al. (2011), the HYBBRID algorithm can be decomposed into six steps. Since the rotation group has three dimensions, four rotation matrices, $R = \{R_1, R_2, R_3, R_4\}$, define a simplex. With the R_j at its vertices, a tetrahedron is formed at the manifold. A_k, an element of the population A has a simplex R and a "fitness" defined as $\min_{j \in \{1,\ldots,4\}} f(R_j)$. As stated by Chang et al. (2011), the HYBBRID algorithm is decomposed in the following steps:

1. *Initialization.* Let M be the size of the total population. It is initialized with random simplices, that is, the four vertices R_j of each simplex are obtained by QR factorization of random 2-by-2 matrices.

2. *Selection.* The fitness of all the simplices is evaluated. The best $\frac{M}{2}$ simplices are selected, the other are discarded. From this reduced population, four groups $A_1^I, A_2^I, A_1^{II}, A_2^{II}$ are created at random using a uniform distribution. Each group has $\frac{M}{2}$ elements, and one population member can be in one group, several groups, or none, and can be selected any number of times in each group.

3. *Crossover I.* A standard mixing crossover is applied between A_1^I and A_2^I. A pair of parents is constituted by choosing the k^{th} element of both subpopulations: $A_1 \in A_1^I$ and $A_2 \in A_2^I$. They produce an offspring $A_{0,i}$. Each vertex of the simplex A_0 is either the corresponding vertex of A_1 or of A_2, the selection being random, but the parent with the best fitness having higher probability of being chosen. This give us $\frac{M}{2}$ new simplices.

4. *Crossover II.* The other $\frac{M}{2}$ new simplices are given by an affine crossover between A_1^{II} and

A_2^{II}. Let $A_1 \in A_1^{II}$, $A_2 \in A_2^{II}$ be the k^{th} pair of parents as before. The four vertices $A_{0,j}$ of the corresponding offspring A_0 are defined by $A_{0,j} = \lambda A_{1,j} + (1-\lambda)A_{2,j}$, where the value of λ depends on whether A_1 is better or worse than A_2. For example, λ can have the value 0.4 (respectively 0.6) if the fitness of A_1 is smaller than that of A_2.

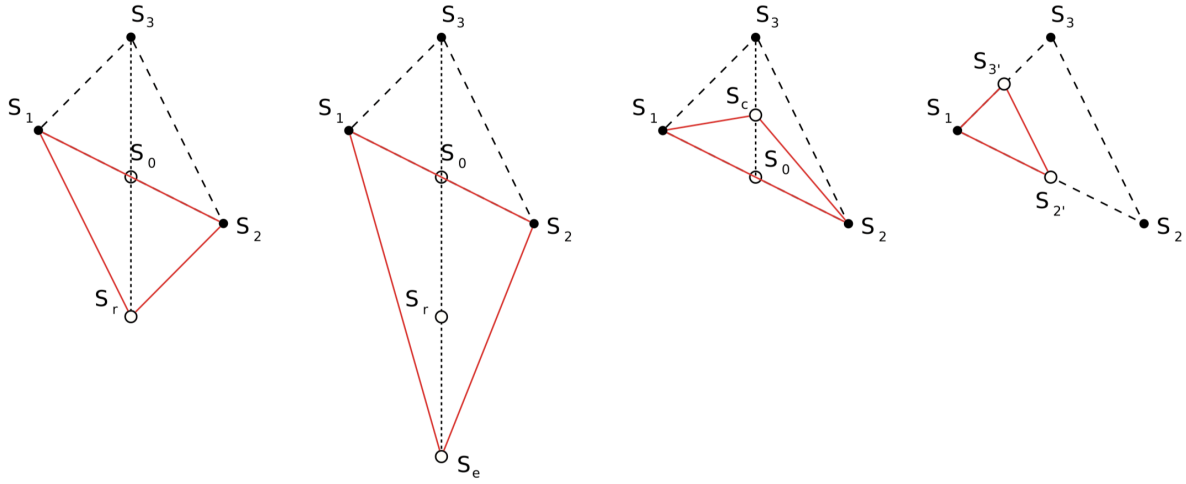
5. *Mutation.* K Nelder-Mead iterations are applied on all these M new simplices to obtain the new generation of the population.

6. *Stopping criterion.* This process (Selection - Crossover - Mutation) is repeated until a stopping criterion is met, usually if the fitness of the best simplex stalls for several iterations with respect to the desired tolerance, or if a maximal number of iterations is reached. In our case, the algorithm stops after k consecutive generations where the objective value does not improve by at least x% compared to the current best value, with $k = 5$ and $x = 1$ as default values for these parameters.

The steps of the HYBBRID algorithm benefit from the correlations between the initial conditions computed by the genetic component. If the genetic component of the algorithm computes the initial conditions as intended, the Nelder-Mead algorithm will converge to a global minimum. The minimum volume of the arbitrarily oriented bounding box is found.

Nelder-Mead algorithm

The Nelder-Mead algorithm was first presented by Nelder and Mead (1965). It is a numerical method, using a direct search to find the minimum/maximum of an objective function in a multidimensional space. It is a derivative-free method that can be applied to nonlinear optimization problems. Due to the simplicity and empirical efficiency of the algorithm, the heuristic method is well known and used for problems with dimensions less than five (Chang et al., 2011).



(a) Reflection (b) Expansion (c) Contraction (d) Reduction

Figure 18: Steps of the Nelder-Mead algorithm (Chang et al., 2011)

Figure 18 shows the steps of the Nelder-Mead algorithm in two dimensions. The objective function $f(X)$, where $X \in \mathbb{R}^2$ is minimized. A simplex $S \subset \mathbb{R}^2$ is formed by $S_1S_2S_3$, where $f(S_1) \leq f(S_2) \leq f(S_3)$. The worst point is removed at each iteration, in this illustration it is S_3. S_0 is defined as the centroid of the remaining points and S_r is defined as the reflection of S_3 through S_0. If S_r is better than the worst remaining point, in this case S_2, a new simplex is defined, $S_1S_rS_2$, as seen in Figure 18a. If S_r is better than the current best point, in this case S_1, the simplex is expanded, as Figure 18b shows. This happens in the direction of S_0S_r, so that $S_e = S_r + (S_r - S_0)$. $S_eS_1S_2$ is the new simplex. If S_r is worse than all of the current points, the algorithm aims to contract the simplex. Figure 18c shows the contracted point S_c, defined as $S_c = \frac{1}{2}(S_0 + S_3)$. If S_c is better than S_3, $S_cS_1S_2$ defines the new simplex. Finally, after all steps are repeated possibly several times, a reduction, as in Figure 18d, is performed. All points, except the current best, which here is S_1, is reduced to $S_{i'} = \frac{1}{2}(S_1 + S_i)$ for all i. The new simplex is $S_1S_{2'}S_{3'}$.

The algorithm needs a termination criterion for the iterative process to break. Another critical aspect of the algorithm is the initial simplex, as the search can get stuck and lead to a local search. In the HYBBRID algorithm, the initial simplex is decided by the genetic algorithm.

Genetic algorithm

A genetic algorithm is inspired by the evolution and the principles of natural selection; heredity, variation and selection. As written in Mitchell (1996), genetic algorithms have a population and a selection based on a fitness function. New offspring are produced by a crossover function and from random mutation.

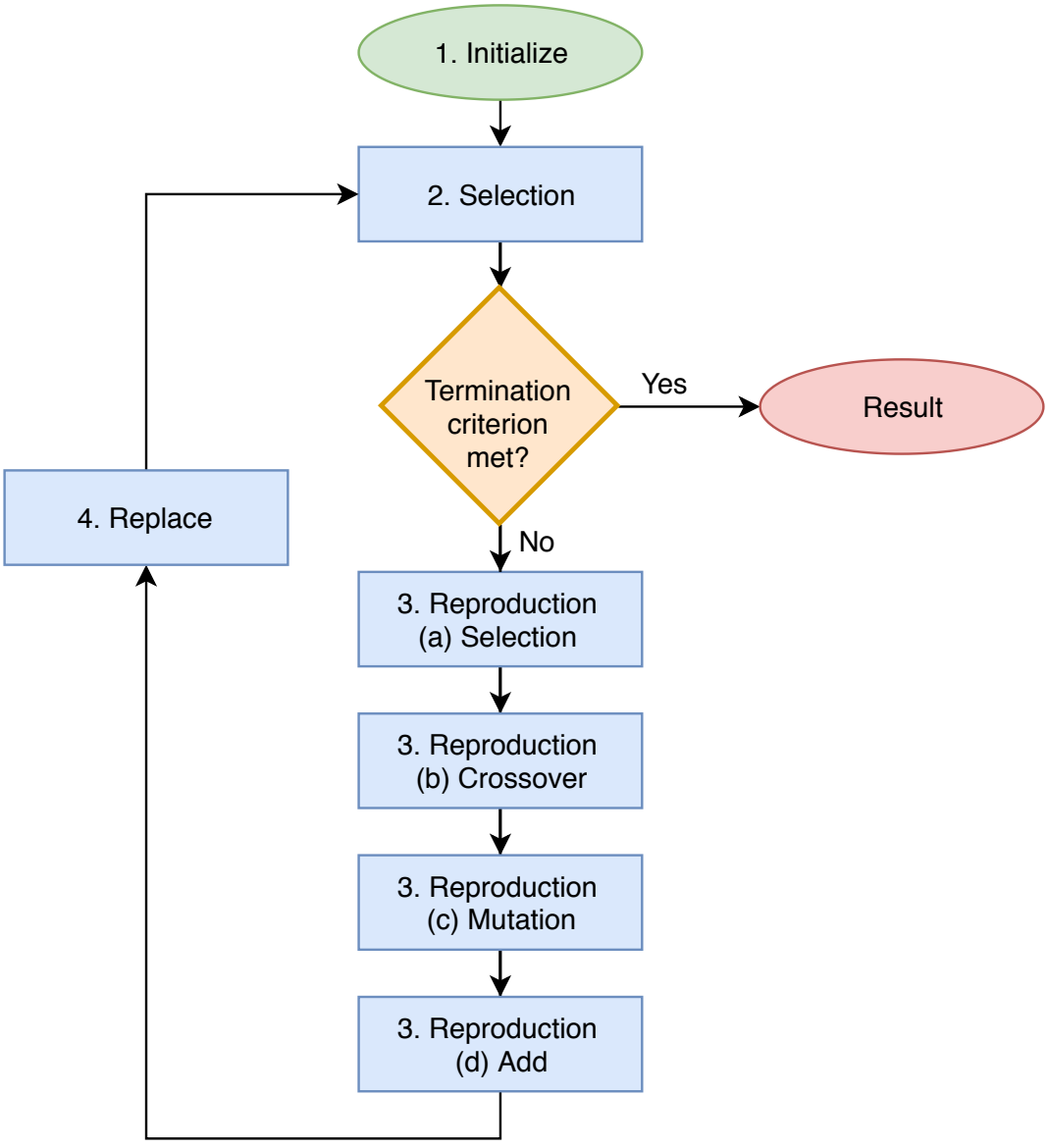


Figure 19: Flow chart of the main steps of the optimization process in a genetic algorithm

The flow chart in Figure 19 visualizes the main steps of the optimization process in the genetic algorithm. They could be as follows:

1. *Initialize.* Create a random population of N elements.
2. *Selection.* Calculate and evaluate fitness $f(x)$ for all the N elements in the population.
3. *Reproduction.* Until N offspring are created:
   (a) Pick a pair of "parents" from the current population, with the probability of selection according to the fitness.
   (b) *Crossover.* Cross over the parents at a random point, chosen with uniform probability, to create an offspring, based on the crossover probability. With no crossover taking place, the offspring is an exact copy of the parents.
   (c) *Mutation.* Mutate the offspring data, based on the mutation probability.
   (d) Add the offspring to the new population.
4. Replace the current population with the new population.
5. Return to step 2.

Each iteration in the algorithm is defined as a *generation*, where the entire set of generations is defined as a *run*. A run often ends with one or more elements with high fitness. The result often relies on the chosen values of the crossover and mutation probabilities.

The algorithm needs a termination criterion to break the iterative process. This can be when some generations or a minimum criterion are reached.

Chapter 3

Development of HYBBRID in Python

An algorithm to find the minimal enclosing object to a measured object from the metrology laboratory is developed in Python and presented in this chapter. The measured object has an underlying geometry as a rectangular parallelepiped. The developed algorithm is based on the minimum bounding box algorithm HYBBRID, presented in subsection 2.8.3. The algorithm, developed by Chang et al. (2011), is originally developed in Matlab®.

3.1 HYBBRID in Python

For this master thesis, Python is the chosen programming language. Python is an open-source language. This means that Python is free software, there is no financial expense for NTNU, redistribution is free and the source code is available for everyone. As Python is a technical programming language, the open-source library SciPy can be used for scientific and technical computing with packages such as NumPy, numerical Python. NumPy enables multi-dimensional array objects to be defined and associated math functions to be used.

HYBBRID is free software, open to redistribution and modification under the terms of the GNU General Public License as published by the Free Software Foundation. The algorithm, developed by Chang et al. (2011), is available in Matlab®. During this master thesis, a new algorithm converted from the HYBBRID algorithm in Matlab®, is developed in Python.

Included in Appendix A are the 14 scripts of the developed HYBBRID algorithm in Python. The main functions are "HYBBRID.py", "genetic.py", "localOptiRC.py", "rotatingCalipers.py", "nelderMeadBreed.py", "nelderMead.py", "volumeOBB.py" and "volumeAABB.py". "Params.py" is a class defining parameters, "affine.py" and "karcher.py" are helping functions.

To extract the 3D data points from the raw data measured from the laser tracker and the CMM, the algorithms "PCDMIS_to_numpy_array.py" and "Laser_tracker_to_numpy_array.py" are developed. They extract the 3D data points, represented as x-, y- and z-coordinates from the raw data, a .txt- or .csv file, and translates them into NumPy arrays, arranged as a matrix with one 3D data point in each row. The script "run_HYBBRID_algorithm_on_data.py" runs the program with the 3D data points.

Figure 20 shows the layout of the algorithm. The program "starts" at the top of the figure and is initiated by running "run_HYBBRID_algorithm_on_data.py". The arrows in the figure explain in what order the scripts are called and how they are connected. The initiating script, "run_HYBBRID_algorithm_on_data.py", receive data from both "PCDMIS_to_numpy_array.py" and "Laser_tracker_to_numpy_array.py", where only one of the two data sets is used in the algorithm.

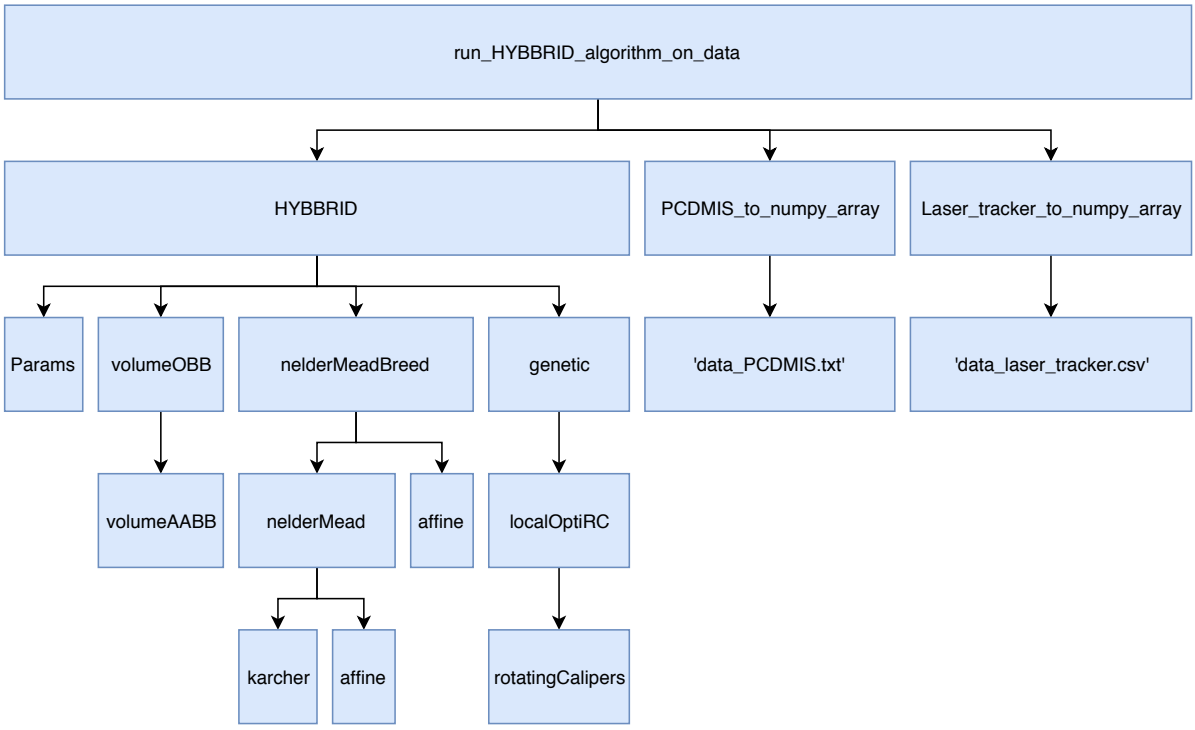


Figure 20: Layout of the HYBBRID algorithm in Python

Converting a data set of 3D points to NumPy array in Python

After a complete calibration process in the metrology laboratory, the measured 3D data points are saved. A data set from the CMM or the laser tracker is saved in a .txt and .csv file, respectively. The 3D data points need to be arranged as a matrix in a NumPy array, with one point per row,

to be used in the HYBBRID algorithm. The open-source library Pandas in Python enables data structures and data analysis tools to be used to extract the desired information from the files and convert the 3D data points into a NumPy array.

In section B.2 a .txt file is included. Planes of the object are named PLN_X+, PLN_X-, PLN_Y+, PLN_Y-, PLN_Z+ and PLN_Z- to identify lines with a data point. Every line with one of these names followed by HITS is identified as a line with a data point. Lines and columns without a coordinate from a data point are removed. The remaining data frame consists of three columns with the x-, y- and z-coordinates of the 3D data points. This data frame is converted into a NumPy array. The full algorithm is attached in subsection A.1.3.

A .csv file is included in section B.1. Every line in the file contains coordinates of a 3D data point. The algorithm in "Laser_tracker_to_numpy_array", included in subsection A.1.2, only needs to remove unnecessary columns. The remaining data frame consists of six columns, x-, y- and z-coordinates and the corresponding decimal numbers of the 3D data points. The integer is put together with the corresponding decimal number. This data frame is converted into a NumPy array.

Params

A class Params, attached in subsection A.1.14, is created to keep track of different parameters in the algorithm. The parameters are defined as attributes of Params, and described in Table 3.

Table 3: Parameters in the HYBBRID algorithm

Params		
Attribute	**Description**	**Default value**
test_repeat	Number of times the program is executed	1
opt_convhull	if == 1: Data is preprocessed by extracting the convex hull	1
g_popsize	Size of the population in the genetic algorithm	30
g_maxiter	Maximum number of generations in the genetic algorithm	100
g_tolval, g_toliter	The stopping criterion in the genetic algorithm: if the relative improvement is less than g_tolval during g_toliter iterations	10^{-2}, 5
g_verbose	if == 1: information is displayed at each generation in the genetic algorithm	1
g_randmut	The probability of random mutations at each generation of the genetic algorithm	0.0
nm_maxiter	The number of Nelder-Mead iterations at each generation	20

The default values are carefully defined after exploration by Chang et al. (2011). The performance of the algorithm is highly dependent on the parameters, especially g_popsize and

nm_maxiter, defining the size of the population in the genetic algorithm and the number of Nelder-Mead iterations at each generation, respectively. An increase in these two parameters increases both the reliability and the computation time. The default values are a trade-off between the two qualities. The default value of nm_maxiter is set to 20, as Chang et al. (2011) discovered that the performance of the algorithm increased significantly until this number. The population size should be defined according to the needs of the algorithm, where 30 provides sufficient reliability, with acceptable computation time.

nelderMead and nelderMeadBreed

The nedlerMeadBreed algorithm breeds a new population from a successful generation in the genetic algorithm, using the nelderMead simplex algorithm. The input in the nelderMeadBreed algorithm is the current population, the fitness value of each current population member, the volumeOBB function, a data set of 3D data points and the parameters defined in Params.

Each population member is defined as a simplex. As the rotation group is in three dimensions, a set of four rotation matrices define a simplex , $R = \{R_1, R_2, R_3, R_4\} \subset SO(3, \mathbb{R})$. The volumeOBB function is called to obtain the fitness value of each population member. The parameter used in the nelderMeadBreed algorithm is g_randmut; the probability of random mutations at each generation of the genetic algorithm.

The output of the nelderMeadBreed algorithm is a new population of simplices.

The nelderMead simplex algorithm has four inputs; a simplex, the volumeOBB function, a data set of 3D data points and the parameters defined in Params. The simplex is the starting point. The function, volumeOBB, is called to obtain the value of a simplex and minimizes the simplex. The parameter nm_maxiter is the only parameter used in the algorithm, deciding the number of Nelder Mead iterations for each generation.

The output of the nelderMead algorithm is the simplex after the decided number of Nelder Mead iterations.

Genetic

The genetic algorithm computes the initial conditions of the HYBBRID algorithm and approximates the optimal OBB enclosing a data set of 3D data points. The input of the algorithm is the volumeOBB function, the nelderMeadBreed function, a data set of 3D data points and the parameters defined in Params. The parameters used in the genetic algorithm is the g_popsize;

the size of the population, g_maxiter; the maximum number of generations, g_tolval, g_toliter; defining the stopping criterion, g_verbose; defining if the information is displayed at each generation.

The fitness of the OBB is defined as the volume, where the optimal OBB has the minimum fitness, hence the minimum volume. The initial population is generated at random. When the current minimum fitness value is improved, a local optimization, with the function localOptiRC, is performed. The function volumeOBB is called to decide the fitness of an OBB. The function nelderMeadBreed is called to obtain the population's next generation, modifying the current population with mutation and crossover steps. The population and the fitness are sorted by their value of fitness, by increasing order.

The output of the function is the minimum fitness of a population member, the respective population member, the log of the elapsed time and the best fitness and the log of the best argument.

localOptiRC and rotatingCalipers

The localOptiRC algorithm performs a local optimization of an OBB in three directions. The input of the algorithm is a data set of 3D points in a NumPy array, arranged as a matrix with one 3D point per row, and a rotation matrix, defining three axes.

The convex hull of the 3D data points, a convex boundary fitting around the points, is computed as a preprocessing step in the algorithm. Figure 21 shows the convex hull of a data set measured with the laser tracker.

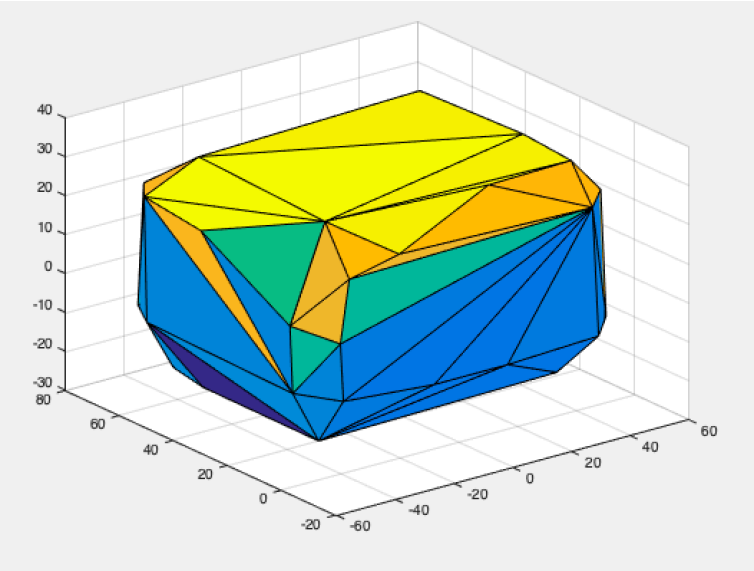


Figure 21: The convex hull of the data from laser tracker

By using the rotating calipers method in 2D on one of the three axes, different OBBs are obtained. These OBBs define a neighborhood. The algorithm computes the volumes of the OBBs in this neighborhood. The OBB in the neighborhood with the minimum volume is defined as the optimal OBB. The volume of the optimal OBB and the associated rotation matrix is returned.

The rotatingCalipers algorithm finds the volume of the minimal oriented rectangle, enclosing a data set of 2D points. The volume and the orientation, in radians, of the minimal oriented bounding rectangle, is returned to the local optimization algorithm.

volumeOBB and volumeAABB

The algorithm volumeOBB computes the volume of an OBB. The input in the algorithm is a set of rotation matrices, where one rotation matrix is associated with one OBB, and a data set of 3D points in a NumPy array arranged as a matrix with one 3D point per row. After the calculation, the volume of the minimal OBB in the set and an array of all the volumes of the OBBs in the set is returned to the genetic algorithm. To compute the volume of the OBBs, the algorithm volumeAABB is used.

The volumeAABB algorithm is used in the computation of the volume of the minimal AABB of the data set of 3D points. The input in the algorithm is a data set of 3D points and a rotation matrix. The calculated volume is returned to the volumeOBB algorithm.

HYBBRID

The 3D points in the data set, measured in the metrology laboratory, are described by the Cartesian coordinate system, as in Figure 22.

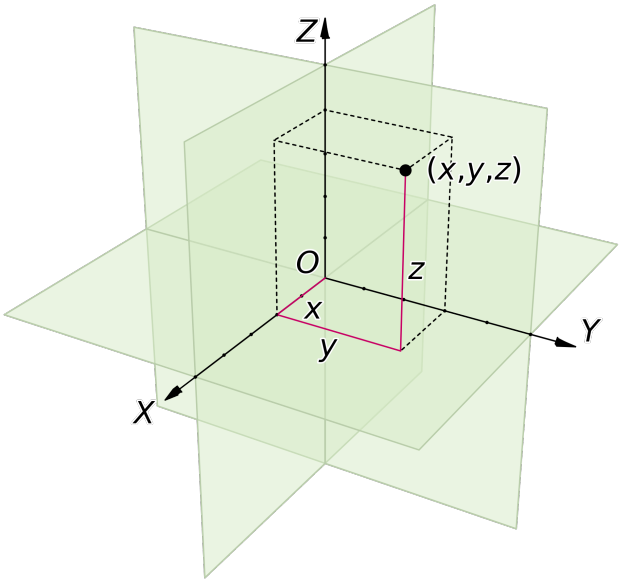


Figure 22: Cartesian coordinate system in three dimensions

When the rotation of the optimal OBB is found, there is a need for translating the points in the data set to a new coordinate frame. The rotation matrix rotates the points around the axis of the coordinate system. Figure 23 shows an arbitrary rotation from x_a-, y_a- and z_a-frame to x_b-, y_b- and z_b-frame. The HYBBRID script rotates the points in the convex hull of the measured points from the coordinate frame of the measurement table in the metrology laboratory, to the coordinate frame of the optimal OBB.

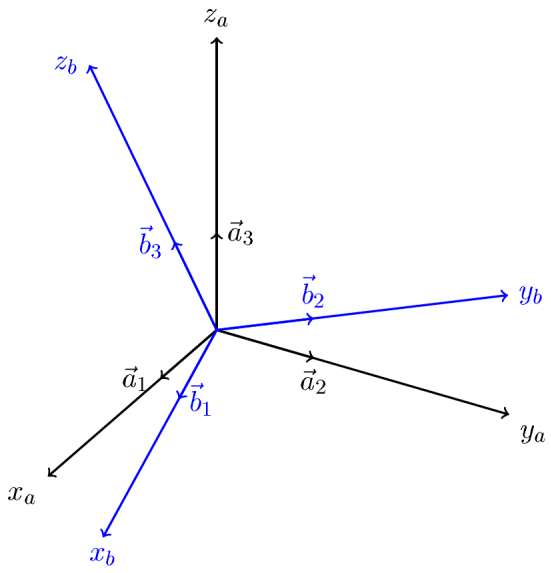


Figure 23: Rotation of coordinate frame a to b

The input of the HYBBRID algorithm is a data set of 3D points. The optimal OBB enclosing all the points in the data set is approximated. The output of the algorithm is the rotation matrix of the optimal OBB, the minimum volume of the optimal OBB, a log of the elapsed time and the best fit (minimum volume), a log of the best argument (rotation matrix), and the data set after the preprocessing (the points of the convex hull).

In the algorithm "run_HYBBRID_algorithm_on_data", the dimensions of the optimal OBB are calculated. When the data set originates from the laser tracker, a radius compensation of the radius of the stylus tip of the laser tracker is necessary.

Chapter 4

Uncertainty in a calibration process

This chapter presents the results and discussions of the work done during the project. Three rectangular parallelepipeds are manufactured to function as reference standards in calibration processes. JV will bring the reference standards to different laboratories in Norway, evaluating their dimensional measuring devices. In the metrology laboratory at NTNU, the dimensional measurements of the reference standards are measured. The developed HYBBRID algorithm estimates the dimensional measurements and is used in the uncertainty analysis, on which the calibration certificates are based.

4.1 Uncertainty analysis

As the true value of the measurand cannot be known, the uncertainty analysis identifies, quantifies and combines the different contributions to error. This section described the procedure and the contributors to uncertainty in the calibration process.

4.1.1 Procedure of a calibration process of a rectangular parallelepiped

The procedure of a calibration process of a reference object in the CMM is as follows. The reference object, a rectangular parallelepiped, is placed on the measurement table of the CMM. The side with the serial number is facing up, and the measurement table of the CMM defines the surface of the bottom side of the object. The CMM measures the five visual sides with a measurement density of 30 mm, thus a distance of 30 mm between each measuring point.

The current procedure used by NTNU defines the distance from the measurement table to the highest measured point on the top surface as the height of the reference object. The surface normals of the four vertical surfaces are calculated and projected down to the measurement table — the projection results in four possible references for the orientation of the reference object. The orientation of the rectangular parallelepiped enclosing the reference object with the smallest volume defines the ultimate reference.

The use of the HYBBRID algorithm renews the procedure of a calibration process of a rectangular parallelepiped. The orientation and the dimensional measurement of the rectangular parallelepiped, enclosing the measured points of the reference object, is calculated by the HYBBRID algorithm, as described in chapter 3.

4.1.2 Contribution A: Repetition

Repeatability is investigated by repeating measurements several times, with the same measuring points each time. Repeatability tests have been conducted at the metrology laboratory at NTNU with similar manufactured objects as the reference objects. The repeated measurements were done with a measurement density of 30 mm and resulted in a calculation of the standard deviation.

Based on previous results from the metrology laboratory at NTNU, the standard uncertainty associated with repeatability is estimated to be 0.02 mm.

4.1.3 Contribution B: Reproducibility

Reproducibility is investigated by repeating measurements where the measuring points are moved between each repeated measurement. Reproducibility tests are conducted at the metrology laboratory at NTNU with similar manufactured objects as the reference objects. The movement of the measuring points is achieved by rotating the coordinate frame of the object with 180°.

Based on previous results from the metrology laboratory at NTNU, the standard uncertainty associated with reproducibility is estimated to be 0.02 mm.

4.1.4 Contribution C: Number of measuring points

It is desirable with a sensible balance between low uncertainty and time consumption in a calibration process. While a high number of measuring points is desired to keep the uncertainty low, a lower number of measuring points is desired for lower time consumption. Thus it is valuable to explore the significance of the number of measuring points and decide an uncertainty associated with the number of measuring points.

The analysis uses data sets of three reference objects. The intended dimensions of the parallelepipeds are 200 mm x 300 mm x 900 mm. The three reference objects are measured in the metrology laboratory with a measurement density of 10 mm; thus, the original data sets consist of measuring points with this measurement density. Extracting data points from the original data sets creates new data sets representing a lower number of measuring points.

Figure 24, Figure 25 and Figure 26 shows the relation between the measurement density and the measured length, width and height, respectively, of the reference object with the original data set "DkS10".

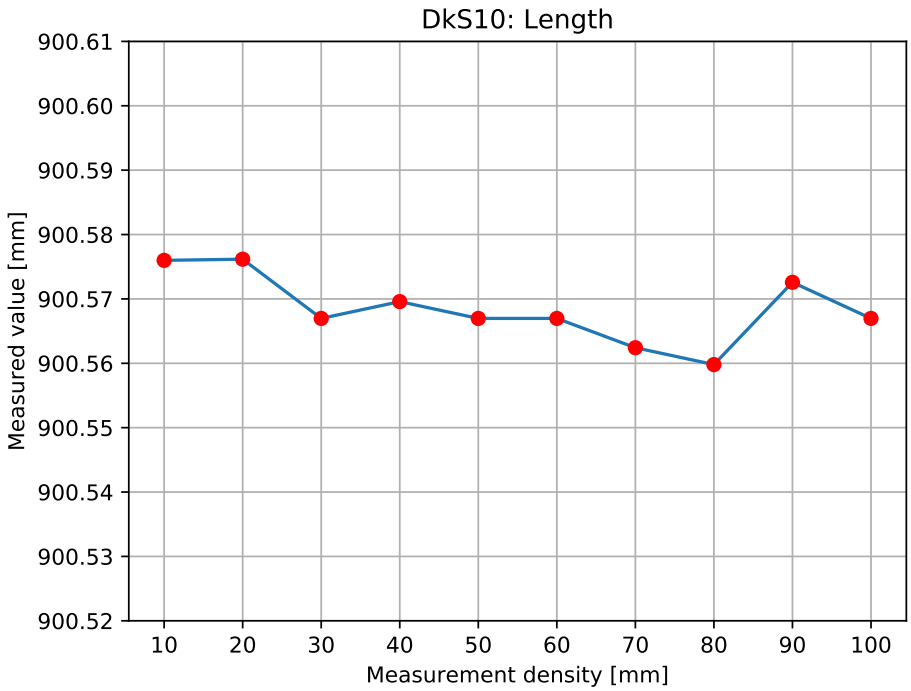


Figure 24: DkS10 length

The measured value of the length of DkS, visualized in Figure 24, ranges from 900.5598 mm to

900.57617 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 20 mm
- Minimum value: Measurement density of 80 mm

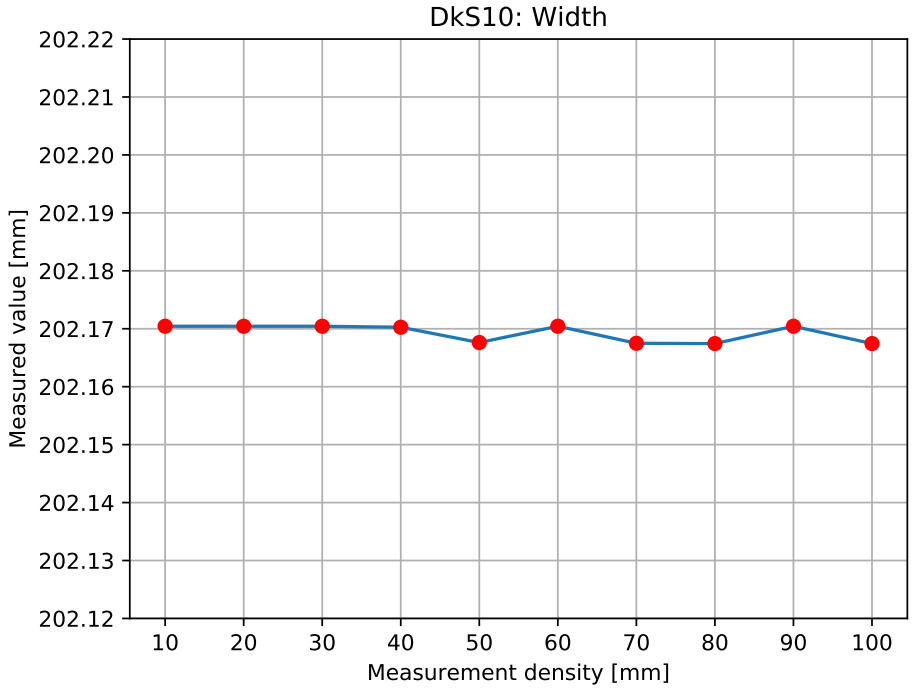


Figure 25: DkS10 width

The measured value of the width of DkS, visualized in Figure 25, ranges from 202.16744 mm to 202.17043 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 10 mm, 20 mm, 30 mm, 60 mm, 90 mm
- Minimum value: Measurement density of 80 mm, 100 mm

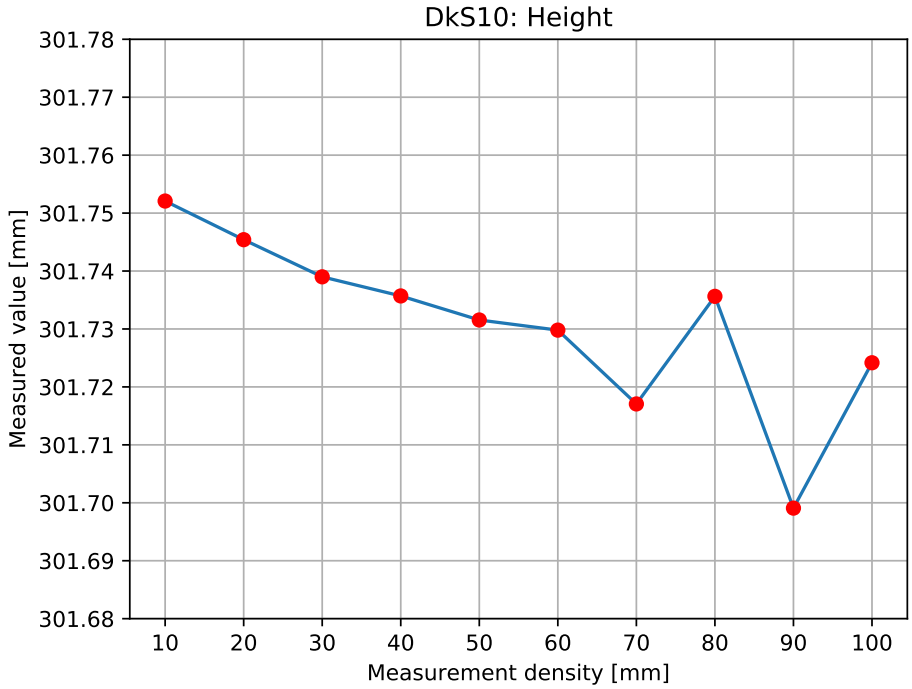


Figure 26: DkS10 height

The measured value of the height of DkS, visualized in Figure 26, ranges from 301.69908 mm to 301.75208 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 10 mm
- Minimum value: Measurement density of 90 mm

Figure 27, Figure 28 and Figure 29 shows the relation between the measurement density and the measured length, width and height, respectively, of the reference object with the original data set "DkN10".

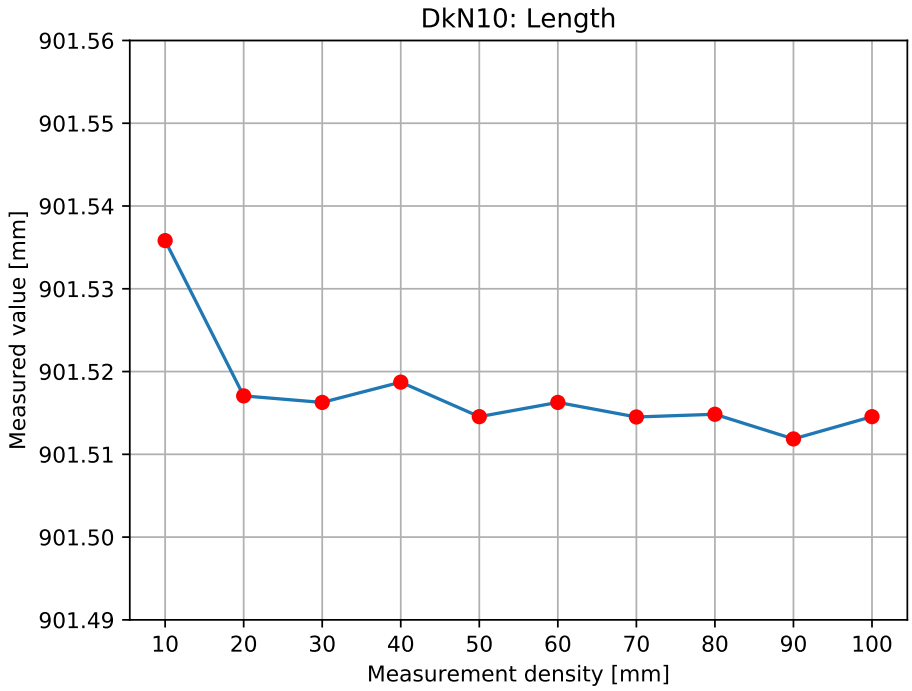


Figure 27: DkN10 length

The measured value of the width of DkN, visualized in Figure 27, ranges from 901.51186 mm to 901.53582 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 10 mm
- Minimum value: Measurement density of 90 mm

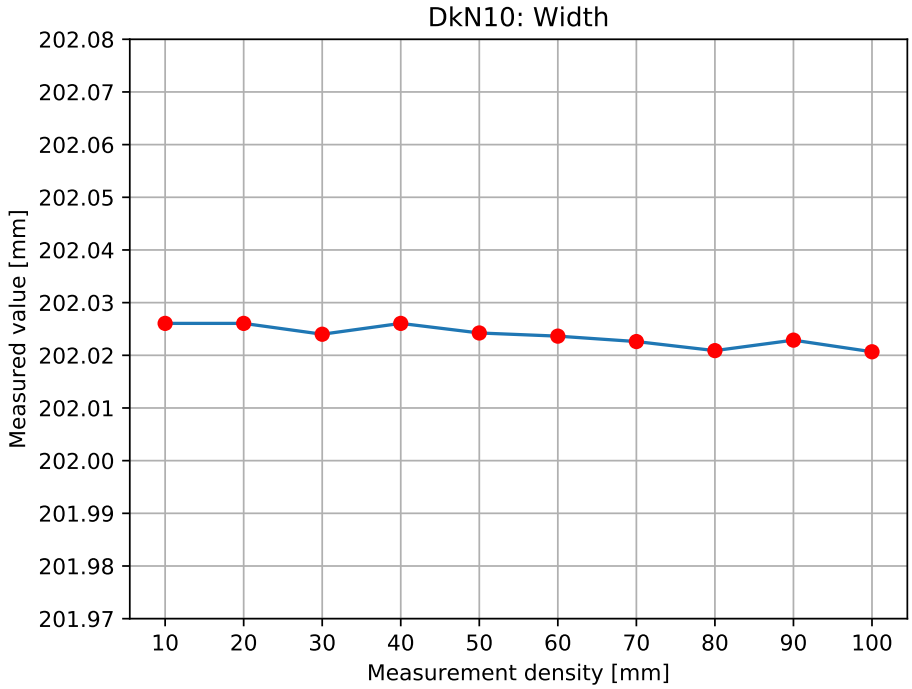


Figure 28: DkN10 width

The measured value of the width of DkN, visualized in Figure 28, ranges from 202.02067 mm to 202.020606 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 10 mm, 20 mm, 40 mm
- Minimum value: Measurement density of 100 mm

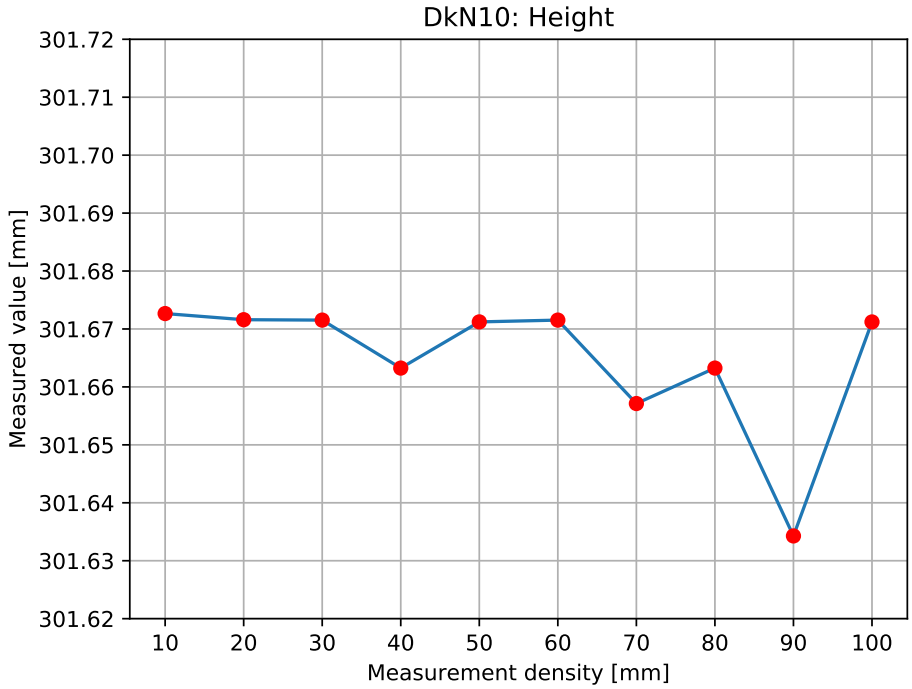


Figure 29: DkN10 height

The measured value of the width of DkN, visualized in Figure 29, ranges from 301.63428 mm to 301.67266 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 10 mm
- Minimum value: Measurement density of 90 mm

Figure 30, Figure 31 and Figure 32 shows the relation between the measurement density and the measured length, width and height, respectively, of the reference object with the original data set "DkMn10".

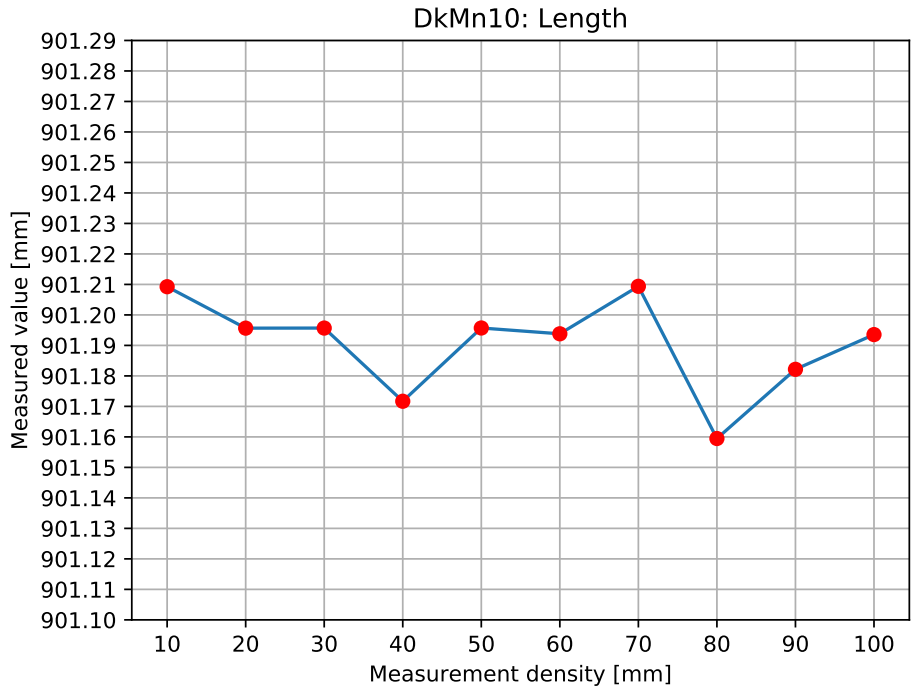


Figure 30: DkMn10 length

The measured value of the length of DkMn, visualized in Figure 30, ranges from 901.15947 mm to 901.20938 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 70 mm
- Minimum value: Measurement density of 80 mm

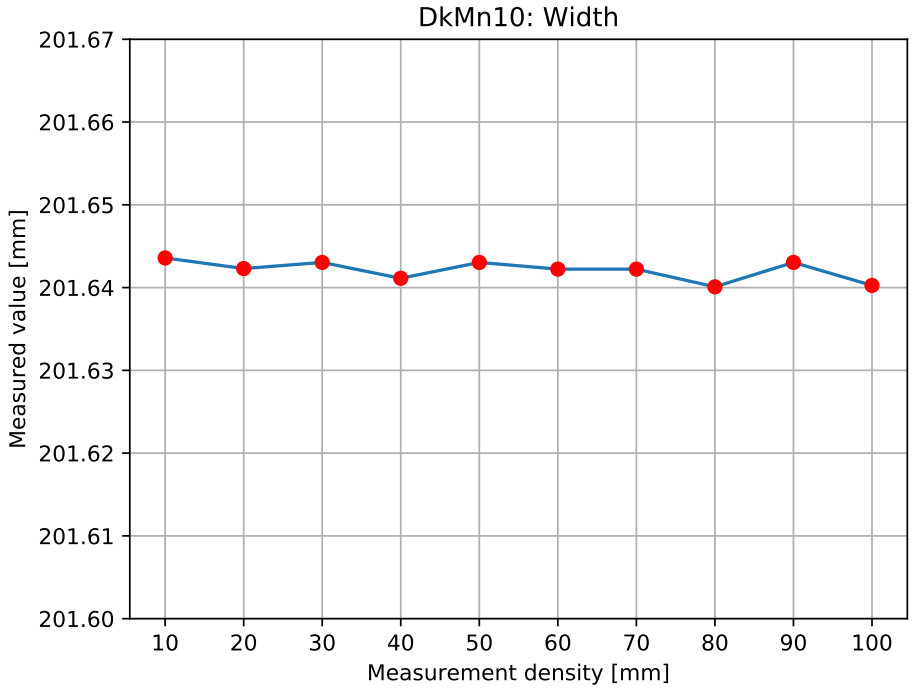


Figure 31: DkMn10 width

The measured value of the width of DkMn, visualized in Figure 31, ranges from 201.64009 mm to 201.64358 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 10 mm
- Minimum value: Measurement density of 80 mm

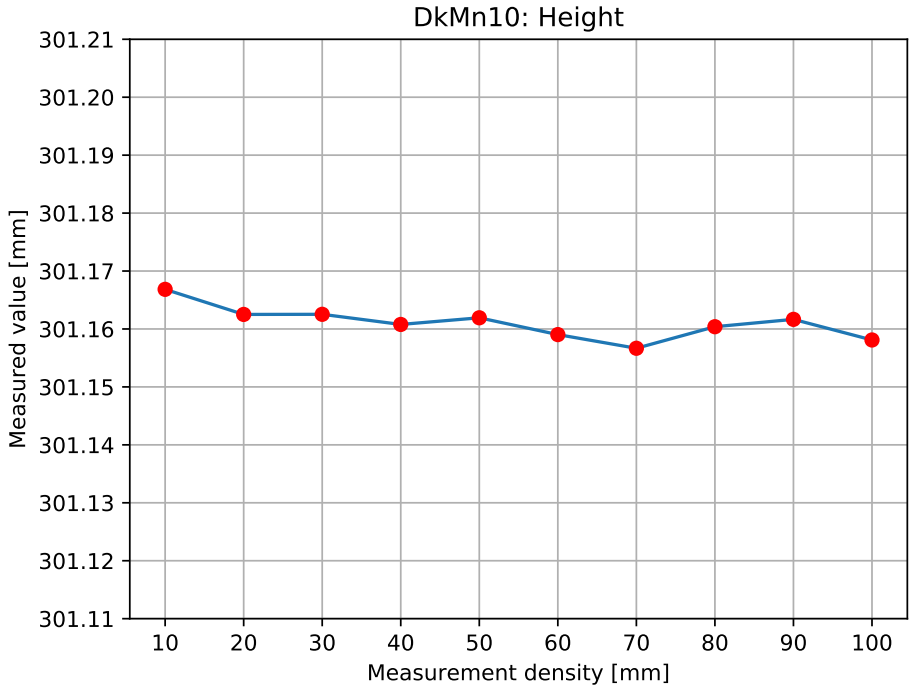


Figure 32: DkMn10 height

The measured value of the height of DkMn, visualized in Figure 32, ranges from 301.15667 mm to 301.16684 mm. Maximum and minimum value are given by the following:

- Maximum value: Measurement density of 10 mm
- Minimum value: Measurement density of 70 mm

Table 4 presents the results of the measured dimensions when the measurement densities are 10 mm, 30 mm, 50 mm and 70 mm.

Table 4: Measured dimensions at different measurement densities

Reference object	Measurement density [mm]	Length [mm]	Width [mm]	Height [mm]
DkS	10	900.57599	202.17043	301.75208
DkS	30	900.56697	202.17043	301.739
DkS	50	900.56697	202.16762	301.73155
DkS	70	900.56242	202.16749	301.71706
DkN	10	901.53582	202.02606	301.67266
DkN	30	901.51627	202.024	301.67153
DkN	50	901.51455	202.02424	301.67122
DkN	70	901.51451	202.02261	301.65713
DkMn	10	901.20925	201.64358	301.16684
DkMn	30	901.19569	201.64304	301.16253
DkMn	50	901.19571	201.64304	301.16193
DkMn	70	901.20938	201.64223	301.15667

A measurement density of 10 mm is assumed to measure all extrema on the surface. There is an uncertainty associated with a measured value with a measurement density other than 10 mm. Calculation of the deviation between a measured value with a measurement density of 10 mm and a measurement density other than 10 mm estimates this. The uncertainty associated with a measurement value due to a measurement density other than 10 mm is twice the value of the maximum estimated deviation.

The graphs in Figure 24 to Figure 32 and the values in Table 4 estimates that a measurement density of 30 mm provides an uncertainty of 0.039 mm, a measurement density of 50 mm provides an uncertainty of 0.043 mm and a measurement density of 70 mm provides an uncertainty of 0.07 mm.

4.1.5 Contribution D: Calculation of the minimum bounding box from measured points

The current procedure of selecting reference direction for a parallelepiped in a calibration process, described in subsection 4.1.1, has an associated uncertainty. The standard uncertainty is estimated to be 0.1 mm.

The use of the developed HYBBRID algorithm is assumed to eliminate the standard uncertainty. Thus it is zero.

4.1.6 Contribution E: Maximum permissible error in the CMM

As described in subsection 2.4.1, the supplier of the CMM used in the metrology laboratory specifies that the MPE of the CMM as in Equation 10. In addition to the MPE, there are uncertainties associated with probes that are extra long or other conditions in the CMM.

Based on previous results from the metrology laboratory at NTNU, the maximal permitted error of the measured reference objects is assumed to contribute to a standard uncertainty of 3 µm = 0.003 mm.

4.1.7 Contribution F: Temperature effects

The material of the reference objects is RenShape BM 5460 (polyurethane). During the calibration process, the temperature in the metrology laboratory is assumed to range between 19.5 °C and 20.5 °C with a rectangular distribution. It is also assumed that the reference objects have been in the metrology laboratory for sufficient time for the temperature of the reference objects to the same as in the room. The coefficient of thermal expansion (CTE) of the CMM is assumed to be small in comparison with the CTE of the reference objects. Thus the temperature effect associated with the CTE of the CMM is neglected.

Equation 29 specifies the CTE of the material of the reference objects at 20 °C.

$$CTE = 55 \cdot 10^{-6}\,\mathrm{K}^{-1} = 55 \cdot 10^{-6}\,^{\circ}\mathrm{C}^{-1} \tag{29}$$

Equation 30 specifies the estimated uncertainty due to the temperature effects. The estimation is associated to the reference objects with length $l = 900$ mm and a difference between the actual temperature and the reference temperature $\Delta T = \pm\, 0.5$ °C.

$$u_{temp} = \frac{\Delta T}{\sqrt{3}} \cdot CTE \cdot l = \frac{0.5\,^{\circ}\mathrm{C}}{\sqrt{3}} \cdot 55 \cdot 10^{-6}\,^{\circ}\mathrm{C}^{-1} \cdot 900\,\mathrm{mm} = 0.014\,\mathrm{mm} \tag{30}$$

4.1.8 Calculation of the combined uncertainty

Table 5 presents an uncertainty budget with a combined uncertainty and the expanded uncertainty. The expanded uncertainty has a coverage factor of two, $k = 2$, providing a confidence level of approximately 95 %.

The standard uncertainty associated to contribution C, the number of measuring points, is estimated for a measurement density of 30 mm, 50 mm and 70 mm, which gives three different estimations of the combined uncertainty.

Table 5: Estimated uncertainty budget

Contribution	**Standard uncertainty [mm]**
A: Repetition	0.02
B: Reproducibility	0.02
C: Measurement density (= 30 mm)	0.039
C: Measurement density (= 50 mm)	0.043
C: Measurement density (= 70 mm)	0.07
D: Reference of dimensional measurement	0
E: MPE in the CMM	0.003
F: Temperature effects	0.014
Combined uncertainty (= $\sqrt{u_A^2 + u_B^2 + u_C^2 + u_D^2 + u_E^2 + u_F^2}$**)**	
Combined uncertainty (k=1) Measurement density (= 30 mm)	0.0503
Expanded uncertainty (k=2, 95 %)	0.1006
Combined uncertainty (k=1) Measurement density (= 50 mm)	0.0534
Expanded uncertainty (k=2, 95 %)	0.1069
Combined uncertainty (k=1) Measurement density (= 70 mm)	0.0769
Expanded uncertainty (k=2, 95 %)	0.1537

The expanded uncertainty when the measurement density is 30 mm is estimated to be 0.1006 mm, the expanded uncertainty when the measurement density is 50 mm is estimated to be 0.1069 mm and the expanded uncertainty when the measurement density is 70 mm is estimated to be 0.1537 mm.

4.2 Discussion of the uncertainty analysis

All the steps of the uncertainty analysis require the experience and judgment of an engineer. This section discusses the evaluation of some of the contributions in the uncertainty analysis, the obtaining of the estimations, and other possible contributors to uncertainty.

4.2.1 Procedure of a calibration process of a rectangular parallelepiped

As described in subsection 4.1.1, the reference object is fastened in the measurement table during a measurement procedure with the CMM. One side is facing down on the table. As a result of the fastening method, the CMM measures five out of six sides of the parallelepiped. The five visual sides are measured. However, the side facing down is not measured as it is not accessible for the probe.

There is an uncertainty associated with the fastening of the object to the measuring machine. However, this uncertainty is not included in the uncertainty analysis. It is assumed to be at an acceptable level for a calibration process. The minimum rock requirement is used to minimize the uncertainty, ensuring that the positioning of the object eliminates the amount of rock in all directions. In practice, adding a piece of paper beneath the corner of the object that causes the object to rock accomplishes this. Removing all noticeable rock in any direction is assumed to minimize the uncertainty to an acceptable level.

If other fastening methods enabled the object to have all six sides measured, the possible impact from the current fastening method could be discovered. The uncertainty associated with the lack of measuring points on the sixth side and the uncertainty associated with an unnoticeable rock that is not eliminated by the minimum rock requirement could be estimated.

4.2.2 Procedure of generating data sets

Independent of the number of measuring points in a measurement process, it is desired to spread the measuring points evenly on the surface, ensuring measurement close to all edges of the surface. Figure 33 shows a desired and an undesired situation when reducing the measuring points in a measurement procedure. The number of measuring points in both the desired and the undesired situation is approximately half of the original number of measuring points. The original number of measuring points is 40. In the desired situation, the number of measuring points is 18, covering the surface evenly, out to all the corners and edges. In

the undesired situation, every second measuring point is removed from the original measuring points, resulting in 20 measuring points.

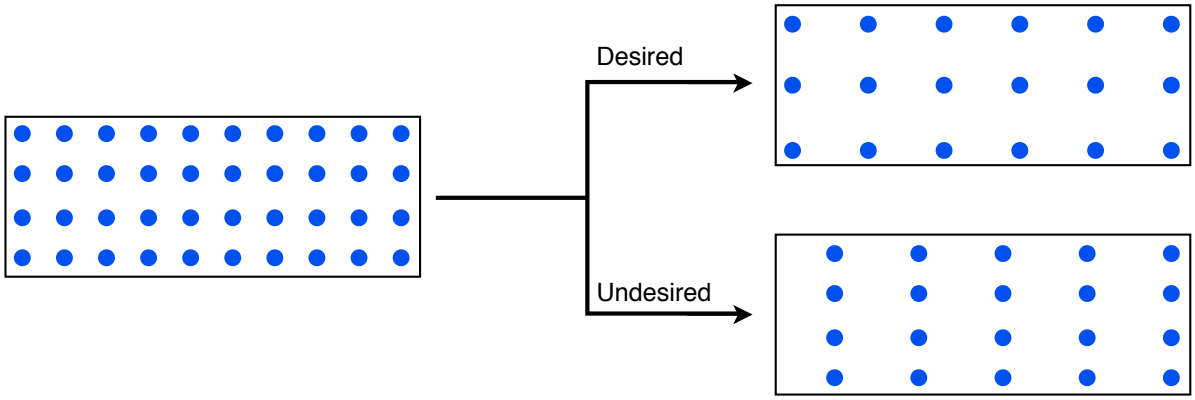


Figure 33: A desired and undesired situation when removing measuring points

The method of generating the data sets used in the uncertainty analysis can result in the undesired situation in Figure 33. The original data sets are created by measuring the visible surfaces of the parallelepipeds, evenly with a measurement distance of 10 mm. The other data sets, with measurement density of 20 mm, 30 mm, 40 mm, 50 mm, 60 mm, 70 mm, 80 mm, 90 mm and 100 mm are generated by extracting every second, third, fourth, ..., tenth point, respectively, from the original data set.

As described in subsection 4.1.1, the CMM measures five out of six sides of the rectangular parallelepiped due to the fastening method. To be able to use the developed HYBBRID algorithm to estimate the dimensional measurements of the objects, measuring points of the sixth surface are necessary. Changing the z-coordinates of the top surface to zero creates the points of the bottom surface.

Estimation of the possible uncertainties associated with data generation is not included in the uncertainty analysis. Possible uneven spreading of the measuring points on the surfaces and the plausible incorrect generation of the bottom surface with z-coordinates of zero could generate an uncertainty. However, the assumption of the uncertainties being at an acceptable level excludes them from the analysis.

4.2.3 Number of measuring points

The number of measuring points is established before a measurement procedure. The uncertainty of a dimensional measurement is assumed to decrease with an increase in the

number of measuring points. The probability of detecting extrema is increasing with a shorter distance between each measuring point. However, an increase in the number of measuring points increases the time consumption. A measurement density of 10 mm is assumed to detect all extrema, thus providing the highest measured value, and a measurement density of 100 mm is too high. A measurement density lower than 10 mm introduces an uncertainty associated with the number of measuring points.

All manufactured surfaces have some degree of irregularity. When the distance between each measuring point is higher than 10 mm, there is an uncertainty of whether the extrema of the surface is measured. As the highest points are more likely to be measured when the number of measuring points is high, it follows that the estimated length of an object increases with the number of measuring points. This assumption indicates that there should be a negative trend in the measured values in Figure 24 to Figure 32. However, the visualizations do not show a consistent negative trend between the measured values and the measurement density.

Figure 34 shows the surface of an object seen from the side. The surface in the figure has a visual roughness. A measurement procedure with a measurement density of 10 mm, 30 mm and 50 mm are shown with blue, orange and red dots, respectively. The figure shows a situation where a measurement conducted with a lower number of measuring points cover a higher point than a measurement conducted with a higher number of measuring points. A similar situation can explain the lack of negative trends in the graphs in Figure 24 to Figure 32.

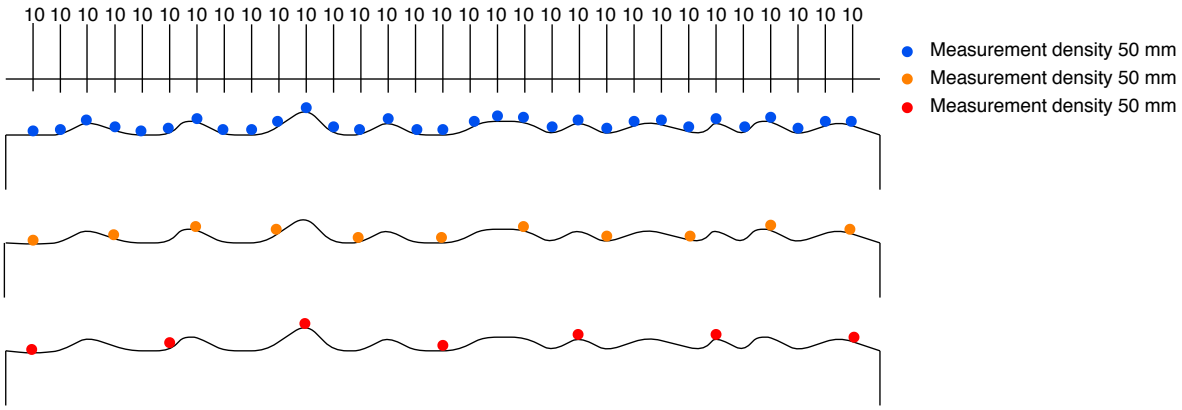


Figure 34: Side view of a measurement procedure with different measurement densities

Figure 24 to Figure 32 shows that the width of all three parallelepipeds (200 mm) has the most even measured values, independent of the measurement density. The length of all three parallelepipeds (900 mm) has the most uneven measured values with the different measurement densities. This could indicate that surfaces with a longer distance between

them require a higher number of measuring points than surfaces that are closer. However, as the manufacturing technique is similar for the three parallelepipeds, it is possible that the measured surfaces giving the most even measured values have very low roughness, due to the manufacturing process.

A measurement procedure with a measurement density of 10 mm measures the largest (or second-largest) measured values of the different widths, lengths and heights. The underlying assumption, of a measurement density of 10 mm detecting all extrema in a measured surface is likely to be correct. However, it could be interesting to have a data set with measuring points with a measurement density of 5 mm.

The convex hull of the data set is defined in the calculation of the minimum bounding box in the HYBBRID algorithm before the minimum bounding box is estimated. Figure 35, Figure 36 and Figure 37 show the relation between the number of measuring points in the data sets and the number of points in the convex hull by the different measurement densities.

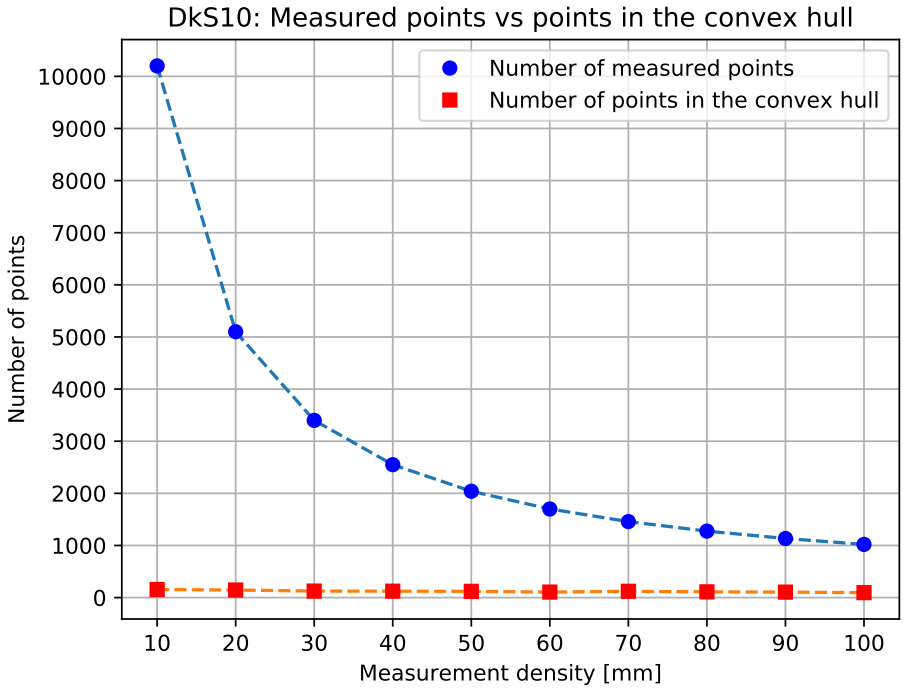


Figure 35: The number of points in the data set (DkS10) versus in the convex hull

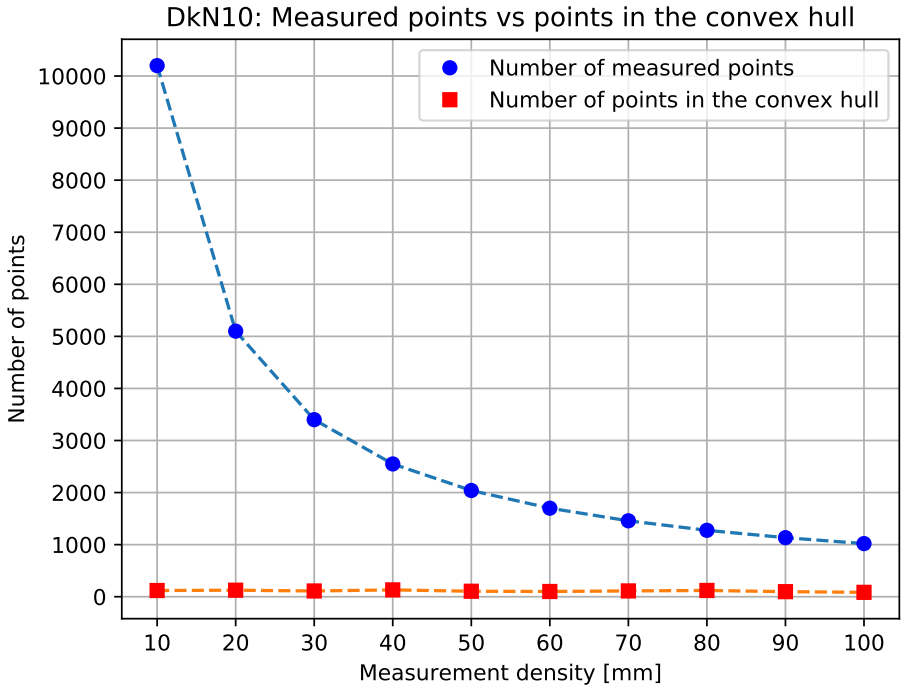


Figure 36: The number of points in the data set (DkN10) versus in the convex hull

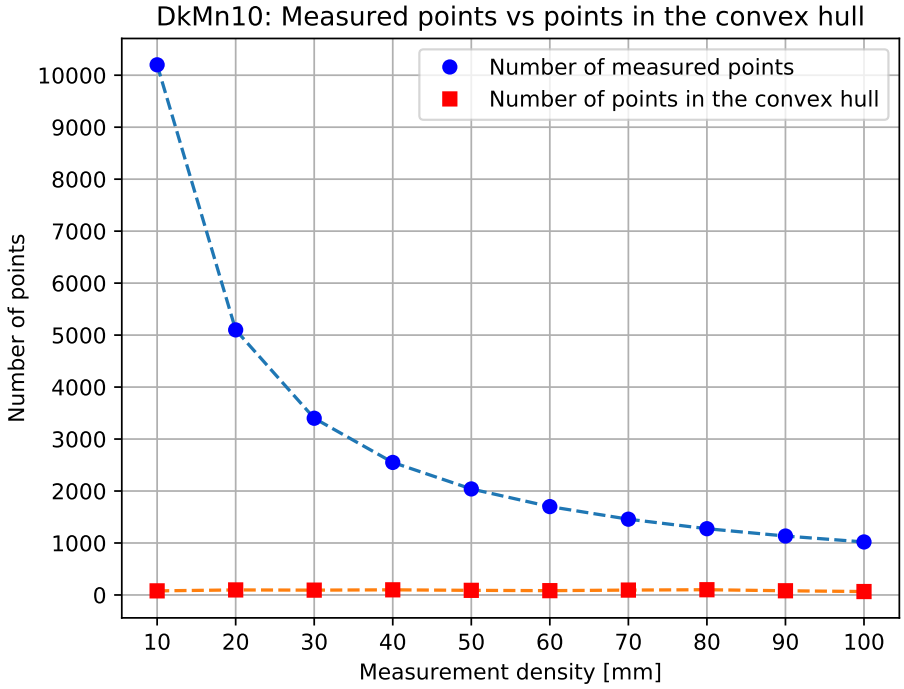


Figure 37: The number of points in the data set (DkMn10) versus in the convex hull

The graphs in Figure 35 to Figure 37 and the values in Table 6 show a significant decrease in the number of measured points as the distance between two measured points increases. The number of points in the convex hull stays quite stable. A measurement density of 10 mm causes 10200 measured points during a measurement procedure, while a measurement density of 30 mm, 50 mm and 70 mm causes 3400, 2040 and 1458 measured points, respectively. With a measurement density of 30 mm, 50 mm or 70 mm, the number of points in the convex hull ranges from 121 to 125, from 105 to 111 and from 89 to 94 for DkS, DkN and DkMn, respectively.

Table 6: Number of points at different measurement densities

Reference object	**Measurement density [mm]**	**Number of points**	
		Measured	**In the convex hull**
DkS	10	10200	155
DkS	30	3400	125
DkS	50	2040	120
DkS	70	1458	121
DkN	10	10200	119
DkN	30	3400	110
DkN	50	2040	105
DkN	70	1458	111
DkMn	10	10200	79
DkMn	30	3400	93
DkMn	50	2040	89
DkMn	70	1458	94

The difference between the number of points visualizes the importance of measuring the highest points of a surface, as the estimation of the dimensions only includes points in the convex hull. This is achieved by having a high number of measuring points. However, the difference also visualizes how much impact the measurement density has on the number of measured points, hence the time of the calibration process.

4.2.4 Calculation of the minimum bounding box from measured points

The developed HYBBRID algorithm is not an exact algorithm as the algorithm of O'Rourke is. However, it is exact in practice. Thus, the associated uncertainty is assumed to be zero. When computing the dimensional measurements of the data sets in Python, the program runs five times. Often the dimensional measurements of four of the results were equal to the fifth decimal, while one result where off. The repeated result was chosen, assumed to be the exact answer. However, it could be interesting to calculate the dimensional measurements of the data set with the algorithm of O'Rourke and compare the results.

4.3 Laser tracker vs CMM

A measurement procedure with the laser tracker was conducted to create a data set on which the developed HYBBRID algorithm practiced. The data set was not representative to conduct an uncertainty analysis, as the laser tracker measured a random number of measuring points randomly on the surfaces of a parallelepiped. The resulting data set contributed to create the program to translate data points from the laser tracker to the HYBBRID algorithm.

4.4 Discussion of laser tracker vs CMM

It could be interesting to compare data sets from the laser tracker and the CMM. A measurement procedure of the same parallelepiped with both the laser tracker and the CMM could accomplish this. In a measurement procedure with the laser tracker, all six sides are available for measurement. A measurement procedure with the laser tracker has two associated uncertainties, one with the laser tracker and one with the probe. A measurement procedure with the CMM has only one associated uncertainty. With two representative data sets, it could be interesting to compare the uncertainties.

Chapter 5

Conclusions and further work

The work of this master thesis has provided knowledge of uncertainty analysis and minimum bounding box algorithms. The objectives of this project have been achieved.

5.1 Conclusions

In section 2.1, a literature review on uncertainty analysis is performed and presented. The procedure of an uncertainty analysis is conducted to estimate the uncertainty associated with the measured value. The uncertainty of the measurement can be as significant as the measurement value. An uncertainty analysis is a methodical approach that is objective and standardized. However, detailed knowledge of the measurand's nature and the measurement procedure is crucial to achieve a satisfying measurement result — in addition to the critical thinking, integrity and professional skills of those performing the uncertainty analysis. The guide by ISO and the standard by ANSI/ASME are the two professional documents accepted on uncertainty analysis. They differ in terminology. However, the different categorizations are for convenience. The resulting uncertainty of a measurement is similar using the standard or the guide.

A literature review on finding minimal enclosing objects, both cylinder and rectangular parallelepiped, is performed and presented in section 2.7 and section 2.8. A data set from a measured object in the metrology laboratory at NTNU consists of points in three dimensions. The development of an algorithm to find the minimum enclosing object in three dimensions is advanced. Especially when the object is a parallelepiped, the optimal orientation can be complex to find. Several algorithms to compute a bounding box in three dimensions are

available. The HYBBRID algorithm turned out to be exact in practice, possible to implement in Python and have a low computation time. No new algorithm of finding the minimum bounding cylinder was developed. When the object has an underlying geometry of a cylinder, effective algorithms implemented in the CMM finds the minimum bounding cylinder.

An algorithm is developed in Python to find the minimal enclosing object to a measured reference object from the metrology laboratory. The underlying geometry of the reference object is a rectangular parallelepiped. The development of the algorithm is presented in chapter 3. It consists of 14 scripts and is based on the minimum bounding box algorithm HYBBRID. A data set of 3D points is the input in the algorithm approximating the optimal OBB. The optimal OBB encloses all the points in the data set. The algorithm rotates the computed convex hull with the rotation matrix associated with the optimal OBB. Then the dimensional measurements are calculated. The algorithm can use data sets from the CMM or the laser tracer.

The developed algorithm has been verified on simple physical parts in the metrology laboratory. Measurement procedures with both the CMM and the laser tracker created data sets. The data sets were used regularly in the development of the algorithm to verify the results, as the dimensional measurements of the parts were known. During the development of the programs converting raw data to NumPy arrays in Python, the data sets were also used.

An estimation of the uncertainty in calibration is conducted. The results of the uncertainty analysis are presented in chapter 4. Three rectangular parallelepipeds, manufactured to function as reference standards in calibration processes by the JV, are used in the uncertainty analysis. A measurement procedure with a measurement density of 10 mm created the original data sets. Data sets with measurement densities of 20 mm, 30 mm, ..., 100 mm were generated from the original data sets and used in the analysis.

The uncertainty of a dimensional measurement is assumed to decrease with an increase in the number of measuring points. However, the time consumption of a measurement procedure increases with the number of measuring points. The chosen measurement density decides the number of measuring points. As it is desirable with a sensible balance between low uncertainty and time consumption in a calibration process, the significance of the measurement density was explored. The uncertainty associated with the different measurement densities was estimated.

The results strongly indicate that the number of measuring points, given by a measurement density of 30 mm, currently used in calibration processes by NTNU can be reduced. The time consumption would significantly decrease, while keeping the associated uncertainty sufficiently low. The associated uncertainty with a measurement density of 30 mm is 0.039 mm, while

the uncertainty associated with a measurement density of 50 mm is 0.043 mm. By using a measurement density of 50 mm instead of 30 mm, the time of a measurement process will significantly decrease. A measurement density of 30 mm results in 3400 points measured. A similar box and a measurement density of 50 mm results in 2040 points measured.

Even though the results indicate that the measurement density can be decreased, the results only apply for rectangular parallelepipeds similar to the three used in the uncertainty analysis, with similar dimensions, material and manufacturing technique. A similar analysis of rectangular parallelepipeds manufactured with different processes, different materials and in different sizes will make it easier to conclude if the measurement density can be reduced in general.

NTNU wants to develop new working methods and enhancement when it comes to the validation of its calibration process. The development of the HYBBRID algorithm renews the working methods of deciding the dimensional measurements of a rectangular parallelepiped in a calibration process. The results of the uncertainty analysis indicate that the measurement density can be reduced to improve time usage while keeping the uncertainty low.

5.2 Recommendations for further work

It would be interesting to explore the possibilities of measuring all six sides of a parallelepiped. This requires the development of a new fastening method. It would also be desirable to conduct several measurement procedures with rectangular parallelepipeds with different dimensional measurements and measurement densities, as well as different materials and manufacturing techniques.

Appendix A

Source code

A.1 HYBBRID algorithm in Python

A.1.1 run_HYBBRID_algorithm_on_data.py

```
from HYBBRID import HYBBRID
import numpy as np
#################################################################
# GET DATA FROM ONE OF THE FOLLOWING SCRIPT:
#################################################################
# from PCDMIS_to_numpy_array import data_PCDMIS
# from dataset_HYBBRID1 import data_PCDMIS
# from dataset_HYBBRID2 import data_PCDMIS
# from dataset_HYBBRID3 import data_PCDMIS_10
# from dataset_HYBBRID4 import data_PCDMIS_10
from dataset_HYBBRID5 import data_PCDMIS_10
# from Laser_tracker_to_numpy_array import data_Laser_tracker


# Run HYBBRID algorithm: Find the minimum enclosing box to data
best_t, bestvalue_t, log_elapsedtime_best_fit_t, log_best_arg_t, data_t = \
    HYBBRID(data_PCDMIS_10)

# The data rotated with the R associated to the optimal OBB
data_optimal_OBB = np.matmul(data_t, best_t.conj().transpose())

# Find the dimensions of the optimal OBB
dimension_optimal_OBB = np.max(data_optimal_OBB, axis=0) - \
```

```
                        np.min(data_optimal_OBB, axis=0)

# Print the results of the HYBBRID(data) algorithm
print('Minimum volume (bestvalue): ', bestvalue_t)
print('Rotation matrix (best, all decimals): \n', best_t)
print('Rotation matrix (best, around to 5 decimals): \n', np.around(best_t, decimals=5))
print('Dimensions of the optimal OBB: ', dimension_optimal_OBB.round(5))
print('Convex hull: \n', data_t)
print('Log of elapsedtime and best_fit: \n', log_elapsedtime_best_fit_t)
print('Log of best_arg: \n', log_best_arg_t)


# Check the volume
check_volume = dimension_optimal_OBB[0]*dimension_optimal_OBB[1]*dimension_optimal_OBB[2]
if check_volume == bestvalue_t:
    print('Volume is the same!')
else:
    print('while check volume = ', check_volume, ' while bestvalue_t = ', bestvalue_t)

'''
# WHEN DATA FROM LASER TRACKER: RADIUS COMPENSATION
d_laser_tracker = 6  # diameter is 5 or 6 mm, have to check
dimension_optimal_OBB[0] -= d_laser_tracker
dimension_optimal_OBB[1] -= d_laser_tracker
dimension_optimal_OBB[2] -= d_laser_tracker

print('true dimensions: (-d)', dimension_optimal_OBB.round(5))
'''
```

A.1.2 Laser_tracker_to_numpy_array.py

```
import pandas as pd


# Create a data frame df from a txt file
df = pd.read_csv('Boks-Laser_Tracker.csv', header=None)

# Create names of each column. Need to know the number of columns (Seven columns)
df.columns = ["Point", "X", "x_decimal", "Y", "y_decimal", "Z", "z_decimal"]

# Print the data frame
# print(df)  # debug

```

```
# All lines in the df contain information of a coordinate

# Create data frame with only X Y Z coordinates of the measured points
df_data_temp = pd.DataFrame({'X': df.X, 'Y': df.Y, 'Z': df.Z})
df_data_decimals = pd.DataFrame({'x_decimal': df.x_decimal, 'y_decimal': df.y_decimal,
                                 'z_decimal': df.z_decimal})



# Print the data frame
# print('df_data_temp', df_data_temp)  # debug
# print('df_data_ decimals', df_data_decimals)  # debug

# Convert data frame to numpy array (matrix)
data_Laser_tracker = df_data_temp.to_numpy(dtype=float)
data_Laser_tracker_decimals = df_data_decimals.to_numpy()

# Updating the NumPy array with decimals
number_of_data = len(data_Laser_tracker)

for i in range(number_of_data):
    len_x_decimal = len(str(abs(data_Laser_tracker_decimals[i, 0])))
    len_y_decimal = len(str(abs(data_Laser_tracker_decimals[i, 1])))
    len_z_decimal = len(str(abs(data_Laser_tracker_decimals[i, 2])))

    x_decimal = data_Laser_tracker_decimals[i, 0]*(10**-len_x_decimal)
    y_decimal = data_Laser_tracker_decimals[i, 1]*(10**-len_y_decimal)
    z_decimal = data_Laser_tracker_decimals[i, 2]*(10**-len_z_decimal)
    if data_Laser_tracker[i, 0] >= 0:
        data_Laser_tracker[i, 0] += x_decimal
    else:
        data_Laser_tracker[i, 0] -= x_decimal
    if data_Laser_tracker[i, 1] >= 0:
        data_Laser_tracker[i, 1] += y_decimal
    else:
        data_Laser_tracker[i, 1] -= y_decimal
    if data_Laser_tracker[i, 2] >= 0:
        data_Laser_tracker[i, 2] += z_decimal
    else:
        data_Laser_tracker[i, 2] -= z_decimal

# Print the numpy array of data
# print('data_Laser_tracker, numpy array: \n', data_Laser_tracker)  # debug
```

A.1.3 PCDMIS_to_numpy_array.py

```
import pandas as pd

# Create a data frame df from a txt file
# Alt 1:
df = pd.read_csv('2019-03-05-boks.txt', header=None)
# Alt 2:
# df = pd.read_csv('boks_200x300x120.csv', header=None, skiprows=2,
# float_precision='.6f')

# Create names of each column. Need to know the number of columns (Seven columns)
df.columns = ["Feature", "X", "Y", "Z", "x", "y", "z", "extra"]

# Print the data frame
# print(df)  # debug

# Remove lines without necessary information
# Alt 1:
count = 0
for line in df.Feature:
    if line == 'PLN_X+ HITS' or line == 'PLN_X- HITS' or line == 'PLN_Y+ HITS' or \
            line == 'PLN_Y- HITS' or line == 'PLN_Z+ HITS' or line == 'PLN_Z- HITS':
        pass
    else:
        df = df.drop([count], axis=0)
    count = count + 1
'''
Alt 2:
count = 0
for line in df.Feature:
    if line == 'PLN_X_PLUS HITS' or line == 'PLN_X_MINUS HITS' or
    line == 'PLN_Y_PLUS HITS'
    or line == 'PLN_Y_MINUS HITS' or line == 'PLN_Z_PLUS HITS' or
    line == 'PLN_Z_MINUS HITS':
        pass
    else:
        df = df.drop([count], axis=0)
    count = count + 1
'''
# Print the data frame
# print(df)  # debug

```

```
# Create data frame with only X Y Z coordinates of the measured points
df_data = pd.DataFrame({'X': df.X, 'Y': df.Y, 'Z': df.Z})

# Print the data frame
# print('df_data \n', df_data)  # debug

# Convert data frame to numpy array (matrix)
data_PCDMIS = df_data.to_numpy()

# Print the numpy array of data
print('data_PCDMIS, numpy array: \n', data_PCDMIS)
```

A.1.4 HYBBRID.py

```
import numpy as np
from datetime import datetime
from scipy.spatial import ConvexHull
from Params import Params
from genetic import genetic
from volumeOBB import volumeOBB
from nelderMeadBreed import nelderMeadBreed


def HYBBRID(data):
    # HYBBRID(data) approximates the minimum-volume OBB enclosing
    # the set of points described in data
    # (one point per row in the matrix(numpy array))

    # Default parameters
    params = Params(1, 1, 30, 100, 1e-2, 5, 1, 0.0, 20)

    # Create a six element array containing the current date and
    # time in decimal form:
    now = datetime.now()
    starttime = np.array([now.year, now.month, now.day, now.hour, now.minute,
                          now.second + 0.000001 * now.microsecond])

    # Preprocessing with convex hull (default)
    if params.opt_convhull == 1:
        idx = ConvexHull(data).simplices
        data = data[np.unique(idx), :]
    if params.test_repeat == 1:
        # Genetic Nelder-Mead
```

```
        bestvalue, best, log_elapsedtime_best_fit, log_best_arg = \
            genetic(volumeOBB, nelderMeadBreed, data, params)
        later = datetime.now()
        newtime = np.array([later.year, later.month, later.day, later.hour, later.minute,
                            later.second + 0.000001 * later.microsecond])
        etime_array = newtime - starttime
        etime = round(etime_array[5], 5)
        print('Elapsed time : %6.3g s' % etime)

    best = np.around(best, decimals=3)  # changed from 4 to 3
    bestvalue = np.around(bestvalue, decimals=3)  # changed from 4 to 3
    return best, bestvalue, log_elapsedtime_best_fit, log_best_arg, data
```

A.1.5 genetic.py

```
import numpy as np
import numpy.matlib as M
from scipy import linalg
import math
from datetime import datetime
from localOptiRC import localOptiRC


def genetic(volumeOBB, nelderMeadBreed, data, params):
    # Randomly generate initial population. data is a n x p matrix
    n, p = np.shape(data)
    m = params.g_popsize

    # Pre allocating empty zeros matrix:
    # m x 1 matrix consisting (p+1) x (p x p) matrices.
    # pop[][] access a p x p matrix
    pop = np.zeros((m, p + 1, p, p))

    for i in range(m):
        for j in range(p + 1):
            q, r = linalg.qr(M.random.rand(p, p))
            pop[i][j] = q

    best_arg = pop[0]
    best_fit = volumeOBB(best_arg, data)[0]
    # print('best_fit fra volumeOBB')  # debug
    check_fit = math.inf
    stop = 0
```

```
29
30      # Evolution
31      # Create a six element array containing the current date and time in decimal form
32      now = datetime.now()
33      starttime = np.array([now.year, now.month, now.day, now.hour, now.minute,
34                            now.second + 0.000001*now.microsecond])
35
36      # Log divided into log of elapsed time [0] and best_fit [1] and log of
37      # best_arg (rotation matrix 3x3)
38      log_elapsedtime_best_fit = np.zeros((params.g_maxiter, 2))
39      log_best_arg = np.zeros((params.g_maxiter, 3, 3))
40
41      for iter in range(params.g_maxiter):
42          # Compute fitness
43          fitness = np.zeros((m, 2))
44          for i in range(m):
45              # the smallest volume of the OBBs
46              fitness[i][0], details = volumeOBB(pop[i], data)
47              # the index of the smallest volume in details
48              fitness[i][1] = np.argmin(details)
49          # sorts the rows of fitness in ascending order
50          fitsort = fitness[fitness[:, 0].argsort(), ]
51          # returns the indices of the sorted rows
52          fitidx = fitness.argsort(axis=0)[:, 0]
53          old_fit = best_fit
54
55          # Check best fitness
56          if fitsort[0][0] < check_fit:
57              check_fit = fitsort[0][0]
58              candidate, better_fit = localOptiRC(data,
59                                                  pop[fitidx[0]][fitsort[0][1].astype(int)])
60              if better_fit < best_fit:
61                  best_arg = candidate
62                  best_fit = better_fit
63                  # data_optimal_OBB = noe
64
65          # Create new population
66          pop = nelderMeadBreed(pop[fitidx], fitsort, volumeOBB, data, params)
67          later = datetime.now()
68          newtime = np.array([later.year, later.month, later.day, later.hour, later.minute,
69                              later.second + 0.000001*later.microsecond])
70          elapsedtime_array = newtime - starttime
71          elapsedtime = round(elapsedtime_array[5], 5)
```

```
        if params.g_verbose == 1:
            print('[%7.5g] Iter #%3d : objective value = %17.15g --- best value = '
                  '%17.15g' % (elapsedtime, iter+1, fitsort[0][0], best_fit))
        log_elapsedtime_best_fit[iter][0] = elapsedtime
        log_elapsedtime_best_fit[iter][1] = best_fit
        log_best_arg[iter] = best_arg

        # Stopping criterion
        if np.abs(best_fit - old_fit) < params.g_tolval*best_fit:
            stop += 1
            if stop >= params.g_toliter:
                break
        else:
            stop = 0
    # the first iter rows of log
    log_elapsedtime_best_fit = log_elapsedtime_best_fit[0:iter + 1, :]
    # the first iter rows of log
    log_best_arg = log_best_arg[0:iter + 1, :]
    return best_fit, best_arg, log_elapsedtime_best_fit, log_best_arg
```

A.1.6 localOptiRC.py

```
import numpy as np
from scipy.spatial import ConvexHull
from rotatingCalipers import rotatingCalipers


def localOptiRC(data, R):
    #  Preprocessing step
    idx = ConvexHull(data).simplices
    data = data[np.unique(idx), :]
    # Using the complex conjugate transpose of R:
    rotatedData = np.dot(data, R.conj().transpose())

    # Apply the rotating calipers on the three axes
    rangeData = np.max(rotatedData, axis=0) - np.min(rotatedData, axis=0)
    volx, anglex = rotatingCalipers(rotatedData[:, [1, 2]])
    voly, angley = rotatingCalipers(rotatedData[:, [0, 2]])
    volz, anglez = rotatingCalipers(rotatedData[:, [0, 1]])
    volx *= rangeData[0]
    voly *= rangeData[1]
    volz *= rangeData[2]

```

```
    vol_xyz = np.array([volx, voly, volz])
    Vopt = np.min(vol_xyz)  # Smallest value of volx, voly, volz
    idx = np.argmin(vol_xyz)  # index of the smallest value of volx, voly, volz

    if idx == 0:
        Ropt = np.matmul(np.array([[1, 0, 0], [0, np.cos(anglex), np.sin(anglex)],
                                   [0, -np.sin(anglex), np.cos(anglex)]]), R)
    elif idx == 1:
        Ropt = np.matmul(np.array([[np.cos(angley), 0, -np.sin(angley)], [0, 1, 0],
                                   [np.sin(angley), 0, np.cos(angley)]]), R)
    elif idx == 2:
        Ropt = np.matmul(np.array([[np.cos(anglez), np.sin(anglez), 0],
                                   [-np.sin(anglez), np.cos(anglez), 0], [0, 0, 1]]), R)
    else:
        Ropt = np.zeros((3, 3))
        print('Error in localOptiRC, idx is not 0, 1 or 2. Ropt set to zeros((3,3))')

    return Ropt, Vopt
```

A.1.7 volumeOBB.py

```
import numpy as np
from volumeAABB import volumeAABB


def volumeOBB(set, data):
    # For sets with one rotation matrix
    if len(set.shape) < 3:
        set = np.array([set])

    p = len(set)  # number of rotation matrices
    details = np.zeros((1, p))
    for idx in range(0, p):
        # set[idx] accesses a rotation matrix(3x3).
        details[0][idx] = volumeAABB(data, set[idx])
    volume = np.min(details)
    details = details.flatten()
    return volume, details
```

A.1.8 volumeAABB.py

```
import numpy as np


def volumeAABB(data, R=None):
    if R is not None:
        data = np.matmul(data, R.conj().transpose())  # complex conjugate transpose of R
    vol = np.prod(np.max(data, axis=0) - np.min(data, axis=0))
    return vol
```

A.1.9 nelderMeadBreed.py

```
import numpy as np
from nelderMead import nelderMead
from affine import affine


def nelderMeadBreed(pop, fitness, volumeOBB, data, params):
    m = len(pop)
    p = len(pop[0])
    mmf = int(np.floor(m/2))
    mmc = int(np.ceil(m/2))

    # Creation of the four groups
    idx = np.floor(mmc*np.random.random((mmf,))).astype(int)
    pop1 = np.zeros((len(idx), len(pop[0]), len(pop[0][0]), len(pop[0][0])))
    for i in range(len(idx)):
        pop1[i] = pop[idx[i]]
    fit1 = np.zeros((len(idx), 1))
    for i in range(len(idx)):
        fit1[i] = fitness[idx[i]][0]

    idx = np.floor(mmc * np.random.random((mmf,))).astype(int)
    pop2 = np.zeros((len(idx), len(pop[0]), len(pop[0][0]), len(pop[0][0])))
    for i in range(len(idx)):
        pop2[i] = pop[idx[i]]
    fit2 = np.zeros((len(idx), 1))
    for i in range(len(idx)):
        fit2[i] = fitness[idx[i]][0]

    idx = np.floor(mmf * np.random.random((mmc,))).astype(int)
    pop3 = np.zeros((len(idx), len(pop[0]), len(pop[0][0]), len(pop[0][0])))
```

```
    for i in range(len(idx)):
        pop3[i] = pop[idx[i]]
    fit3 = np.zeros((len(idx), 1))
    for i in range(len(idx)):
        fit3[i] = fitness[idx[i]][0]

    idx = np.floor(mmf * np.random.random((mmc,))).astype(int)
    pop4 = np.zeros((len(idx), len(pop[0]), len(pop[0][0]), len(pop[0][0])))
    for i in range(len(idx)):
        pop4[i] = pop[idx[i]]
    fit4 = np.zeros((len(idx), 1))
    for i in range(len(idx)):
        fit4[i] = fitness[idx[i]][0]

    # Crossover I: pop1 x pop2
    # Preallocate a mmf x 1 matrix consisting of p x (3 x 3) matrices.
    # pop[][] access a p x p matrix
    newpop1 = np.zeros((mmf, p, len(pop[0][0]), len(pop[0][0])))
    cutoff = 0.5 + 0.1 * (fit1 <= fit2) - 0.1 * (fit1 >= fit2)
    for i in range(mmf):
        for j in range(p):
            if np.random.random() < cutoff[i]:
                newpop1[i][j] = pop1[i][j]
            else:
                newpop1[i][j] = pop2[i][j]

    # Crossover II: pop2 x pop3
    # Preallocate a mmc x 1 matrix consisting of p x (3 x 3) matrices.
    # pop[][] access a p x p matrix
    newpop2 = np.zeros((mmc, p, len(pop[0][0]), len(pop[0][0])))
    cutoff = 0.5 + 0.1 * (fit3 <= fit4) - 0.1 * (fit3 >= fit4)
    for i in range(mmc):
        for j in range(p):
            newpop2[i][j] = affine(pop3[i][j], np.array([]), cutoff[i],
                                   pop3[i][j], 1-cutoff[i], pop4[i][j])

    # Nelder-Mead mutation
    newpop = np.concatenate((pop1, pop2), axis=0)
    for i in range(m):
        newpop[i] = nelderMead(newpop[i], volumeOBB, data, params)

    return newpop
```

A.1.10 nelderMead.py

```
import numpy as np
from karcher import karcher
from affine import affine


def nelderMead(simplex, volumeOBB, data, params):
    p = len(simplex)  # number of rotation matrices
    N = p-1

    # Standard values for Nelder-mead simplex algorithm
    rho = 1/2
    sigma = 1/2
    Rg = simplex[0]

    for iter in range(params.nm_maxiter):

        # Step 1: Reordering
        # simplex_val gives an array with all the volumes
        simplex_val = volumeOBB(simplex, data)[1].flatten()
        # fi is simplex_val sorted
        fi = np.sort(simplex_val)
        # the original indexes after sorted
        idx = np.argsort(simplex_val)

        # Sort the rotation matrices with the index
        # from smallest volume
        simplex_copy = simplex.copy()
        # print('simplex', simplex)  # debug
        # print('simplex[0] rot mat', simplex[0])  # debug
        for i in range(len(simplex)):
            simplex[i] = simplex_copy[idx[i]]

        # Step 2: Computation of the center of gravity
        Rg = karcher(Rg, simplex[0:N][:])
        Rr_temp = np.dot(Rg, simplex[p-1].conj().transpose())
        Rr = np.dot(Rr_temp, Rg)
        fr = volumeOBB(Rr, data)[0]
        if fr < fi[N-1]:
            if fr >= fi[0]:
                # Step 3: Reflection
                simplex[p-1] = Rr
```

```
            else:
                # Step 4: Expansion
                Re_temp = np.dot(Rg, simplex[p-1].conj().transpose())
                Re = np.dot(Re_temp, Rr)
                fe = volumeOBB(Re, data)[0]
                if fe < fr:
                    simplex[p-1] = Re
                else:
                    simplex[p-1] = Rr
        else:
            # Step 5: Contraction
            Rc = affine(Rg, simplex[p-1], rho, Rg, -rho, simplex[p-1])
            fc = volumeOBB(Rc, data)[0]
            if fc <= fi[p-1]:
                simplex[p-1] = Rc
            else:
                # Step 6: Reduction
                for i in range(1, p):
                    simplex[i] = affine(Rg, simplex[0], sigma,
                                        simplex[i], -sigma, simplex[0])
    return simplex
```

A.1.11 rotatingCalipers.py

```
import numpy as np
from scipy.spatial import ConvexHull
import math


def rotatingCalipers(data):
    # Computation of the convex hull
    convHull = ConvexHull(data).vertices
    hullData = data[convHull, :]
    nbVertices = np.shape(hullData)[0]
    minxy = np.min(hullData, axis=0)        # unused
    argminxy = np.argmin(hullData, axis=0)
    maxxy = np.max(hullData, axis=0)        # unused
    argmaxxy = np.argmax(hullData, axis=0)

    # Initialization
    Theta = 0
    Right = argmaxxy[0]
    Up = argmaxxy[1]
```

```
    Left = argminxy[0]
    Down = argminxy[1]
    NextRight = np.mod(Right+1, nbVertices)
    NextUp = np.mod(Up+1, nbVertices)
    NextLeft = np.mod(Left+1, nbVertices)
    NextDown = np.mod(Down+1, nbVertices)
    DeltaRight = hullData[NextRight][:] - hullData[Right][:]
    DeltaUp = hullData[NextUp][:] - hullData[Up][:]
    DeltaLeft = hullData[NextLeft][:] - hullData[Left][:]
    DeltaDown = hullData[NextDown][:] - hullData[Down][:]
    AngleRight = np.arccos(DeltaRight[1] / np.linalg.norm(DeltaRight, 2))
    AngleUp = np.arccos(-DeltaUp[0] / np.linalg.norm(DeltaUp, 2))
    AngleLeft = np.arccos(-DeltaLeft[1] / np.linalg.norm(DeltaLeft, 2))
    AngleDown = np.arccos(DeltaDown[0] / np.linalg.norm(DeltaDown, 2))

    volume = math.inf
    angle = math.nan

    # Rotate the calipers
    while (2*Theta) < math.pi:
        rotatedData = np.dot(hullData, np.array([[np.cos(Theta), -np.sin(Theta)],
                                                 [np.sin(Theta), np.cos(Theta)]]))
        minxy = np.min(rotatedData, axis=0)
        argminxy = np.argmin(rotatedData, axis=0)        # unused
        maxxy = np.max(rotatedData, axis=0)
        argmaxxy = np.argmax(rotatedData, axis=0)        # unused
        curVolume = (maxxy[0] - minxy[0]) * (maxxy[1] - minxy[1])
        if curVolume < volume:
            volume = curVolume
            angle = Theta

        Theta = np.min([AngleRight, AngleUp, AngleLeft, AngleDown])
        argminth = np.argmin([AngleRight, AngleUp, AngleLeft, AngleDown])

        if argminth == 0:
            Right = NextRight
            NextRight = np.mod(Right+1, nbVertices)
            DeltaRight = hullData[NextRight][:] - hullData[Right][:]
            AngleRight = np.arccos(DeltaRight[1] / np.linalg.norm(DeltaRight, 2))
        elif argminth == 1:
            Up = NextUp
            NextUp = np.mod(Up+1, nbVertices)
            DeltaUp = hullData[NextUp][:] - hullData[Up][:]
```

```
63              AngleUp = np.arccos(-DeltaUp[0] / np.linalg.norm(DeltaUp, 2))
64          elif argminth == 2:
65              Left = NextLeft
66              NextLeft = np.mod(Left+1, nbVertices)
67              DeltaLeft = hullData[NextLeft][:] - hullData[Left][:]
68              AngleLeft = np.arccos(-DeltaLeft[1] / np.linalg.norm(DeltaLeft, 2))
69          elif argminth == 3:
70              Down = NextDown
71              NextDown = np.mod(Down+1, nbVertices)
72              DeltaDown = hullData[NextDown][:] - hullData[Down][:]
73              AngleDown = np.arccos(DeltaDown[0] / np.linalg.norm(DeltaDown, 2))
74
75      return volume, angle
```

A.1.12 karcher.py

```
1  import numpy as np
2  from scipy import linalg
3
4
5  def karcher(P, X):
6      # Karcher mean of Xi's with respect to the point P
7      # P is a rotation matrix, X is an array of rotation matrices
8      kmean = 0
9      for i in range(len(X)):
10         kmean = kmean + X[i]
11     kmean = kmean/len(X)
12     q, r = linalg.qr(kmean)
13     kmean = q/np.linalg.det(q)
14
15     return kmean
```

A.1.13 affine.py

```
1  import numpy as np
2  from scipy import linalg
3
4
5  def affine(foot, base, factd, dest, facto, orig):
6      # Affine combinations
7      # foot: rotation matrix
8      # base = simplex{p} : rotation matrix
9      # factd = rho: number
```

```
    # dest = Rg : rotation matrix
    # facto = -rho : -number
    # orig = simplex{p}: rotation matrix

    res = np.zeros(np.shape(foot))

    if base.size != 0:
        res += base
    if dest.size != 0:
        res += factd*dest
    if orig.size != 0:
        res += facto*orig
    q, r = linalg.qr(res)
    res = q/np.linalg.det(q)

    return res
```

A.1.14 Params.py

```
class Params:

    def __init__(self, test_repeat, opt_convhull, g_popsize, g_maxiter,
                 g_tolval, g_toliter, g_verbose, g_randmut, nm_maxiter):
        self.test_repeat = test_repeat
        self.opt_convhull = opt_convhull
        self.g_popsize = g_popsize
        self.g_maxiter = g_maxiter
        self.g_tolval = g_tolval
        self.g_toliter = g_toliter
        self.g_verbose = g_verbose
        self.g_randmut = g_randmut
        self.nm_maxiter = nm_maxiter
```

A.2 Data set generation

A.2.1 DkS10

```
import pandas as pd
import numpy as np
import sys


'''
# DATASET 5: Parallelepiped 900 x 300 x 200
# High measurement density: 10 mm
# Change the PLN_Z_PLUS z-coord to zero to get the PLN_Z_MINUS.

# Create a data frame df from a txt file
df = pd.read_csv('DkS10.txt', header=None, skiprows=2, float_precision='.6f')

# Create names of each column. Need to know the number of columns (Seven columns)
df.columns = ["Feature", "X", "Y", "Z", "x", "y", "z", "extra"]

# Print the data frame
# print(df)  # debug


# Remove lines without necessary information

# Store values of x- y- coord in PLN_Z_PLUS to create PLN_Z_MINUS
nb_of_pln_z_plus = 0
count = 0
for line in df.Feature:
    if line == 'PLN_X_PLUS HITS' or line == 'PLN_X_MINUS HITS' or
    line == 'PLN_Y_PLUS HITS' or line == 'PLN_Y_MINUS HITS':
        pass
    # Count the number of points in PLN_Z_PLUS
    elif line == 'PLN_Z_PLUS HITS':
        nb_of_pln_z_plus += 1
    # Remove to original PLN_Z_MINUS
    elif line == 'PLN_Z_MINUS HITS':
        df = df.drop([count], axis=0)
    else:
        df = df.drop([count], axis=0)
    count = count + 1

```

```
# Print the number of points on PLN_Z_PLUS
# print('number of pln z plus', nb_of_pln_z_plus)  # debug

# Print the data frame
# print(df)  # debug

# Create data frame with only X Y Z coordinates of the measured points
df_data_temp = pd.DataFrame({'X': df.X, 'Y': df.Y, 'Z': df.Z})

# Print the data frame
# print('df_data \n', df_data)  # debug

# Convert data frame to numpy array (matrix)
data_PCDMIS_temp = df_data_temp.to_numpy()

# Create PLN_Z_MINUS
PLN_Z_MINUS = np.zeros([nb_of_pln_z_plus, 3])
for i in range(nb_of_pln_z_plus):
    PLN_Z_MINUS[i][0] = data_PCDMIS_temp[i][0]
    PLN_Z_MINUS[i][1] = data_PCDMIS_temp[i][1]

# print(PLN_Z_MINUS)  # debug
# print(nb_of_pln_z_plus)  # debug

# Add PLN_Z_MINUS to the rest of the data set
data_PCDMIS_temp2 = np.concatenate((data_PCDMIS_temp, PLN_Z_MINUS), axis=0)

# Print the numpy array of data
#np.set_printoptions(threshold=sys.maxsize)
#print('data_PCDMIS, numpy array: \n', data_PCDMIS)

# Create a data frame (only six decimals)
data_PCDMIS = pd.DataFrame(data_PCDMIS_temp2)
data_PCDMIS = data_PCDMIS.round(6)

# Print the data frame
# print(data_PCDMIS)  # debug

# Only do this once
data_PCDMIS.to_csv('dataset_5_200x300x900_10.txt', header=None, index=None,
sep=',', mode='a')
'''

```

```
# VOL 2:
# DATASET 5: Parallelepiped 900 x 300 x 200
# High measurement density: 10 mm, correct z coord

# Create a data frame df from a txt file
df = pd.read_csv('dataset_5_200x300x900_10.txt', header=None)

# Create names of each column. Need to know the number of columns (Three)
df.columns = ["X", "Y", "Z"]

# Print the data frame
# print(df)  # debug

# Convert data frame to numpy array (matrix)
data_PCDMIS_10 = df.to_numpy()
data_PCDMIS_20 = np.array(data_PCDMIS_10)[::2]
data_PCDMIS_30 = np.array(data_PCDMIS_10)[::3]
data_PCDMIS_40 = np.array(data_PCDMIS_10)[::4]
data_PCDMIS_50 = np.array(data_PCDMIS_10)[::5]
data_PCDMIS_60 = np.array(data_PCDMIS_10)[::6]
data_PCDMIS_70 = np.array(data_PCDMIS_10)[::7]
data_PCDMIS_80 = np.array(data_PCDMIS_10)[::8]
data_PCDMIS_90 = np.array(data_PCDMIS_10)[::9]
data_PCDMIS_100 = np.array(data_PCDMIS_10)[::10]

# Print length of the arrays
# print('length data_PCDMIS_10:, ', len(data_PCDMIS_10))  # debug  # 10200
# print('length data_PCDMIS_20:, ', len(data_PCDMIS_20))  # debug  # 5100
# print('length data_PCDMIS_30:, ', len(data_PCDMIS_30))  # debug  # 3400
# print('length data_PCDMIS_40:, ', len(data_PCDMIS_40))  # debug  # 2550
# print('length data_PCDMIS_50:, ', len(data_PCDMIS_50))  # debug  # 2040
# print('length data_PCDMIS_60:, ', len(data_PCDMIS_60))  # debug  # 1700
# print('length data_PCDMIS_70:, ', len(data_PCDMIS_70))  # debug  # 1458
# print('length data_PCDMIS_80:, ', len(data_PCDMIS_80))  # debug  # 1275
# print('length data_PCDMIS_90:, ', len(data_PCDMIS_90))  # debug  # 1134
# print('length data_PCDMIS_100:, ', len(data_PCDMIS_100))  # debug  # 1020

# Print the numpy array
# print(data_PCDMIS_10, '\n')  # debug
# print(data_PCDMIS_20, '\n')  # debug
# print(data_PCDMIS_30, '\n')  # debug
# print(data_PCDMIS_40, '\n')  # debug
# print(data_PCDMIS_50, '\n')  # debug
```

A.2.2 DkN10

```
import pandas as pd
import numpy as np
import sys

'''
# DATASET 4: Parallelepiped 900 x 300 x 200
# High measurement density: 10 mm
# Change the PLN_Z_PLUS z-coord to zero to get the PLN_Z_MINUS.

# Create a data frame df from a txt file
df = pd.read_csv('DkN10.txt', header=None, skiprows=2, float_precision='.6f')

# Create names of each column. Need to know the number of columns (Seven columns)
df.columns = ["Feature", "X", "Y", "Z", "x", "y", "z", "extra"]

# Print the data frame
# print(df)  # debug


# Remove lines without necessary information

# Store values of x- y- coord in PLN_Z_PLUS to create PLN_Z_MINUS
nb_of_pln_z_plus = 0
count = 0
for line in df.Feature:
    if line == 'PLN_X_PLUS HITS' or line == 'PLN_X_MINUS HITS' or
    line == 'PLN_Y_PLUS HITS' or line == 'PLN_Y_MINUS HITS':
        pass
    # Count the number of points in PLN_Z_PLUS
    elif line == 'PLN_Z_PLUS HITS':
        nb_of_pln_z_plus += 1
    # Remove to original PLN_Z_MINUS
    elif line == 'PLN_Z_MINUS HITS':
        df = df.drop([count], axis=0)
    else:
        df = df.drop([count], axis=0)
    count = count + 1

# Print the number of points on PLN_Z_PLUS
# print('number of pln z plus', nb_of_pln_z_plus)  # debug

```

```
# Print the data frame
# print(df)  # debug

# Create data frame with only X Y Z coordinates of the measured points
df_data_temp = pd.DataFrame({'X': df.X, 'Y': df.Y, 'Z': df.Z})

# Print the data frame
# print('df_data \n', df_data)  # debug

# Convert data frame to numpy array (matrix)
data_PCDMIS_temp = df_data_temp.to_numpy()

# Create PLN_Z_MINUS
PLN_Z_MINUS = np.zeros([nb_of_pln_z_plus, 3])
for i in range(nb_of_pln_z_plus):
    PLN_Z_MINUS[i][0] = data_PCDMIS_temp[i][0]
    PLN_Z_MINUS[i][1] = data_PCDMIS_temp[i][1]

# print(PLN_Z_MINUS)  # debug
# print(nb_of_pln_z_plus)  # debug

# Add PLN_Z_MINUS to the rest of the data set
data_PCDMIS_temp2 = np.concatenate((data_PCDMIS_temp, PLN_Z_MINUS), axis=0)

# Print the numpy array of data
#np.set_printoptions(threshold=sys.maxsize)
#print('data_PCDMIS, numpy array: \n', data_PCDMIS)

# Create a data frame (only six decimals)
data_PCDMIS = pd.DataFrame(data_PCDMIS_temp2)
data_PCDMIS = data_PCDMIS.round(6)

# Print the data frame
# print(data_PCDMIS)  # debug

# Only do this once
data_PCDMIS.to_csv('dataset_4_200x300x900_10.txt', header=None, index=None,
sep=',', mode='a')
'''

# VOL 2:
# DATASET 4: Parallelepiped 900 x 300 x 200
# High measurement density: 10 mm, correct z coord
```

```
85
86 # Create a data frame df from a txt file
87 df = pd.read_csv('dataset_4_200x300x900_10.txt', header=None)
88
89 # Create names of each column. Need to know the number of columns (Three)
90 df.columns = ["X", "Y", "Z"]
91
92 # Print the data frame
93 # print(df)  # debug
94
95 # Convert data frame to numpy array (matrix)
96 data_PCDMIS_10 = df.to_numpy()
97 data_PCDMIS_20 = np.array(data_PCDMIS_10)[::2]
98 data_PCDMIS_30 = np.array(data_PCDMIS_10)[::3]
99 data_PCDMIS_40 = np.array(data_PCDMIS_10)[::4]
100 data_PCDMIS_50 = np.array(data_PCDMIS_10)[::5]
101 data_PCDMIS_60 = np.array(data_PCDMIS_10)[::6]
102 data_PCDMIS_70 = np.array(data_PCDMIS_10)[::7]
103 data_PCDMIS_80 = np.array(data_PCDMIS_10)[::8]
104 data_PCDMIS_90 = np.array(data_PCDMIS_10)[::9]
105 data_PCDMIS_100 = np.array(data_PCDMIS_10)[::10]
106
107 # Print length of the arrays
108 # print('length data_PCDMIS_10:, ', len(data_PCDMIS_10))  # debug, = 10200
109 # print('length data_PCDMIS_20:, ', len(data_PCDMIS_20))  # debug, = 5100
110 # print('length data_PCDMIS_30:, ', len(data_PCDMIS_30))  # debug, = 3400
111 # print('length data_PCDMIS_40:, ', len(data_PCDMIS_40))  # debug, = 2550
112 # print('length data_PCDMIS_50:, ', len(data_PCDMIS_50))  # debug, = 2040
113 # print('length data_PCDMIS_60:, ', len(data_PCDMIS_60))  # debug, = 1700
114 # print('length data_PCDMIS_70:, ', len(data_PCDMIS_70))  # debug, = 1458
115 # print('length data_PCDMIS_80:, ', len(data_PCDMIS_80))  # debug, = 1275
116 # print('length data_PCDMIS_90:, ', len(data_PCDMIS_90))  # debug, = 1134
117 # print('length data_PCDMIS_100:, ', len(data_PCDMIS_100))  # debug, = 1020
118
119 # Print the numpy array
120 # print(data_PCDMIS_10, '\n')  # debug
121 # print(data_PCDMIS_20, '\n')  # debug
122 # print(data_PCDMIS_30, '\n')  # debug
123 # print(data_PCDMIS_40, '\n')  # debug
124 # print(data_PCDMIS_50, '\n')  # debug
```

A.2.3 DkMn10

```
import pandas as pd
import numpy as np
import sys


'''
# DATASET 3: Parallelepiped 900 x 300 x 200
# High measurement density: 10 mm
# Change the PLN_Z_PLUS z-coord to zero to get the PLN_Z_MINUS.

# Create a data frame df from a txt file
df = pd.read_csv('DkMn10.txt', header=None, skiprows=2, float_precision='.6f')

# Create names of each column. Need to know the number of columns (Seven columns)
df.columns = ["Feature", "X", "Y", "Z", "x", "y", "z", "extra"]

# Print the data frame
# print(df)  # debug


# Remove lines without necessary information

# Store values of x- y- coord in PLN_Z_PLUS to create PLN_Z_MINUS
nb_of_pln_z_plus = 0
count = 0
for line in df.Feature:
    if line == 'PLN_X_PLUS HITS' or line == 'PLN_X_MINUS HITS' or
    line == 'PLN_Y_PLUS HITS' or line == 'PLN_Y_MINUS HITS':
        pass
    # Count the number of points in PLN_Z_PLUS
    elif line == 'PLN_Z_PLUS HITS':
        nb_of_pln_z_plus += 1
    # Remove to original PLN_Z_MINUS
    elif line == 'PLN_Z_MINUS HITS':
        df = df.drop([count], axis=0)
    else:
        df = df.drop([count], axis=0)
    count = count + 1

# Print the number of points on PLN_Z_PLUS
# print('number of pln z plus', nb_of_pln_z_plus)  # debug
```

```
42
43 # Print the data frame
44 # print(df)  # debug
45
46 # Create data frame with only X Y Z coordinates of the measured points
47 df_data_temp = pd.DataFrame({'X': df.X, 'Y': df.Y, 'Z': df.Z})
48
49 # Print the data frame
50 # print('df_data \n', df_data)  # debug
51
52 # Convert data frame to numpy array (matrix)
53 data_PCDMIS_temp = df_data_temp.to_numpy()
54
55 # Create PLN_Z_MINUS
56 PLN_Z_MINUS = np.zeros([nb_of_pln_z_plus, 3])
57 for i in range(nb_of_pln_z_plus):
58     PLN_Z_MINUS[i][0] = data_PCDMIS_temp[i][0]
59     PLN_Z_MINUS[i][1] = data_PCDMIS_temp[i][1]
60
61 # print(PLN_Z_MINUS)  # debug
62 # print(nb_of_pln_z_plus)  # debug
63
64 # Add PLN_Z_MINUS to the rest of the data set
65 data_PCDMIS_temp2 = np.concatenate((data_PCDMIS_temp, PLN_Z_MINUS), axis=0)
66
67 # Print the numpy array of data
68 #np.set_printoptions(threshold=sys.maxsize)
69 #print('data_PCDMIS, numpy array: \n', data_PCDMIS)
70
71 # Create a data frame (only six decimals)
72 data_PCDMIS = pd.DataFrame(data_PCDMIS_temp2)
73 data_PCDMIS = data_PCDMIS.round(6)
74
75 # Print the data frame
76 # print(data_PCDMIS)  # debug
77
78 # Only do this once
79 data_PCDMIS.to_csv('dataset_3_200x300x900_10.txt', header=None, index=None,
80 sep=',', mode='a')
81
82 '''
83 # VOL 2:
84 # DATASET 3: Parallelepiped 900 x 300 x 200
```

```
85 # High measurement density: 10 mm, correct z coord
86
87 # Create a data frame df from a txt file
88 df = pd.read_csv('dataset_3_200x300x900_10.txt', header=None)
89
90 # Create names of each column. Need to know the number of columns (Three)
91 df.columns = ["X", "Y", "Z"]
92
93 # Print the data frame
94 # print(df)  # debug
95
96 # Convert data frame to numpy array (matrix)
97 data_PCDMIS_10 = df.to_numpy()
98 data_PCDMIS_20 = np.array(data_PCDMIS_10)[::2]
99 data_PCDMIS_30 = np.array(data_PCDMIS_10)[::3]
100 data_PCDMIS_40 = np.array(data_PCDMIS_10)[::4]
101 data_PCDMIS_50 = np.array(data_PCDMIS_10)[::5]
102 data_PCDMIS_60 = np.array(data_PCDMIS_10)[::6]
103 data_PCDMIS_70 = np.array(data_PCDMIS_10)[::7]
104 data_PCDMIS_80 = np.array(data_PCDMIS_10)[::8]
105 data_PCDMIS_90 = np.array(data_PCDMIS_10)[::9]
106 data_PCDMIS_100 = np.array(data_PCDMIS_10)[::10]
107
108 # Print length of the arrays
109 # print('length data_PCDMIS_10:, ', len(data_PCDMIS_10))  # debug
110 # print('length data_PCDMIS_20:, ', len(data_PCDMIS_20))  # debug
111 # print('length data_PCDMIS_30:, ', len(data_PCDMIS_30))  # debug
112 # print('length data_PCDMIS_40:, ', len(data_PCDMIS_40))  # debug
113 # print('length data_PCDMIS_50:, ', len(data_PCDMIS_50))  # debug
114 # print('length data_PCDMIS_60:, ', len(data_PCDMIS_60))  # debug
115 # print('length data_PCDMIS_70:, ', len(data_PCDMIS_70))  # debug
116 # print('length data_PCDMIS_80:, ', len(data_PCDMIS_80))  # debug
117 # print('length data_PCDMIS_90:, ', len(data_PCDMIS_90))  # debug
118 # print('length data_PCDMIS_100:, ', len(data_PCDMIS_100))  # debug
119
120 # Print the numpy array
121 # print(data_PCDMIS_10, '\n')  # debug
122 # print(data_PCDMIS_20, '\n')  # debug
123 # print(data_PCDMIS_30, '\n')  # debug
124 # print(data_PCDMIS_40, '\n')  # debug
125 # print(data_PCDMIS_50, '\n')  # debug
```

A.3 Plot generation

```
import numpy as np
import matplotlib.pyplot as plt
# This import registers the 3D projection, but is otherwise unused:
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection, Line3DCollection
from matplotlib import cm
import pandas as pd
from sys import argv


################################################################################
#  DkN10  !!!!
################################################################################
'''
length = np.array([901.53582, 901.51706, 901.51627, 901.51872, 901.51455,
901.51627, 901.51451, 901.51484, 901.51186, 901.51455])
width = np.array([202.02606, 202.02606, 202.024, 202.02606, 202.02424,
202.02364, 202.02261, 202.02089, 202.02288, 202.02067])
height = np.array([301.67266, 301.6716, 301.67153, 301.66325, 301.67122,
301.67153, 301.65713, 301.66325, 301.63428, 301.67121])

x = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
plt.xticks(np.arange(min(x)-10, max(x)+10, 10))
plt.xlabel('Measurement density [mm]')
plt.grid(True)

# LENGTH

#plt.plot(x, length)
#plt.plot(x, length, 'ro')
#plt.ylabel('Measured value [mm]')
#plt.yticks(np.arange(901.53-0.04, 901.53+0.04, 0.01))

# WIDTH

#plt.plot(x, width)
#plt.plot(x, width, 'ro')
#plt.ylabel('Measured value [mm]')
#plt.yticks(np.arange(202.025-0.055, 202.025+0.055, 0.01))

# HEIGHT
```

```
42
43 plt.plot(x, height)
44 plt.plot(x, height, 'ro')
45 plt.ylabel('Measured value [mm]')
46 plt.yticks(np.arange(301.670-0.050, 301.670+0.050, 0.01))
47
48 plt.title('DkN10: Height')
49 plt.savefig('DkN10_height.pdf')
50 '''
51 ###############################################################################
52 # DkMn10 !!!!
53 ###############################################################################
54 '''
55 length = np.array([901.20925, 901.19566, 901.19569, 901.17169, 901.19571,
56 901.19381, 901.20938, 901.15947, 901.18216, 901.19355])
57 width = np.array([201.64358, 201.6423, 201.64304, 201.64112, 201.64304,
58 201.64223, 201.64223, 201.64009, 201.64304, 201.64026])
59 height = np.array([301.16684, 301.16251, 301.16253, 301.16078, 301.16193,
60 301.15903, 301.15667, 301.16039, 301.16166, 301.1581])
61 x = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
62 plt.xticks(np.arange(min(x)-10, max(x)+10, 10))
63 plt.xlabel('Measurement density [mm]')
64 plt.grid(True)
65
66 # LENGTH
67
68 #plt.plot(x, length)
69 #plt.plot(x, length, 'ro')
70 #plt.ylabel('Measured value [mm]')
71 #plt.yticks(np.arange(901.2-0.1, 901.2+0.1, 0.01))
72
73 # WIDTH
74
75 #plt.plot(x, width)
76 #plt.plot(x, width, 'ro')
77 #plt.ylabel('Measured value [mm]')
78 #plt.yticks(np.arange(201.64-0.04, 201.64+0.04, 0.01))
79
80 # HEIGHT
81
82 #plt.plot(x, height)
83 #plt.plot(x, height, 'ro')
84 #plt.ylabel('Measured value [mm]')
```

```
#plt.yticks(np.arange(301.16-0.05, 301.16+0.05, 0.01))

#plt.title('DkMn10: Height')
#plt.savefig('DkMn10_height.pdf')
'''
################################################################################
#  DkS10  !!!!
################################################################################
length = np.array([900.57599, 900.57617, 900.56697, 900.56958, 900.56697,
                   900.56697, 900.56242, 900.5598, 900.57258, 900.56697])
width = np.array([202.17043, 202.17043, 202.17043, 202.17026, 202.16762,
                  202.17043, 202.16749, 202.16744, 202.17043, 202.16744])
height = np.array([301.75208, 301.74541, 301.739, 301.7357, 301.73155,
                   301.7298, 301.71706, 301.73563, 301.69908, 301.72417])
x = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
plt.xticks(np.arange(min(x)-10, max(x)+10, 10))
plt.xlabel('Measurement density [mm]')
plt.grid(True)

# LENGTH

#plt.plot(x, length)
#plt.plot(x, length, 'ro')
#plt.ylabel('Measured value [mm]')
#plt.yticks(np.arange(900.57-0.05, 900.57+0.05, 0.01))

# WIDTH

#plt.plot(x, width)
#plt.plot(x, width, 'ro')
#plt.ylabel('Measured value [mm]')
#plt.yticks(np.arange(202.17-0.05, 202.17+0.05, 0.01))

# HEIGHT

#plt.plot(x, height)
#plt.plot(x, height, 'ro')
#plt.ylabel('Measured value [mm]')
#plt.yticks(np.arange(301.73-0.05, 301.73+0.05, 0.01))

#plt.title('DkS10: Height')
#plt.savefig('DkS10_height.pdf')

```

```
128 ##############################################################################
129 #  Measured points vs points in the convex hull (pre processing)
130 ##############################################################################
131 x = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
132 plt.xticks(np.arange(min(x)-10, max(x)+10, 10))
133 plt.xlabel('Measurement density [mm]')
134 plt.grid(True)
135
136 nb_measured_points = np.array([10200, 5100, 3400, 2550, 2040, 1700, 1458,
137                                1275, 1134, 1020])
138 # DkS10:
139 nb_points_convex_hull = np.array([155, 145, 125, 123, 120, 106, 121, 110,
140                                   105, 95])
141
142 # DkN10:
143 #nb_points_convex_hull = np.array([119, 125, 110, 131, 105, 100, 111, 120,
144 # 97, 84])
145
146 # DkMn10:
147 #nb_points_convex_hull = np.array([79, 97, 93, 99, 89, 83, 94, 101, 81, 66])
148
149 p1 = plt.plot(x, nb_measured_points, '--')
150 p2 = plt.plot(x, nb_measured_points, 'bo')
151 p3 = plt.plot(x, nb_points_convex_hull, '--')
152 p4 = plt.plot(x, nb_points_convex_hull, 'rs')
153
154 plt.ylabel('Number of points')
155 plt.yticks(np.arange(0, 10200, 1000))
156 #plt.legend((p2[0], p4[0]), ('Number of measured points',
157 # 'Number of points in the convex hull'))
158 plt.title('DkS10: Measured points vs points in the convex hull')
159 #plt.savefig('DkS10_measured_points_vs_convex_hull_2.pdf')
160
161 ##############################################################################
162 #  Include for all plots
163 ##############################################################################
164
165 plt.show()
```

A.4 Smallest enclosing circle

```
# Smallest enclosing circle - Library (Python)
#
# Copyright (c) 2018 Project Nayuki
# https://www.nayuki.io/page/smallest-enclosing-circle
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public License
# along with this program (see COPYING.txt and COPYING.LESSER.txt).
# If not, see <http://www.gnu.org/licenses/>.
#

import math
import random
import matplotlib.pyplot as plt


# Data conventions: A point is a pair of floats (x, y).
# A circle is a triple of floats (center x, center y, radius).

# Returns the smallest circle that encloses all the given points.
# Runs in expected O(n) time, randomized.
# Input: A sequence of pairs of floats or ints, e.g. [(0,5), (3.1,-2.7)].
# Output: A triple of floats representing a circle.
# Note: If 0 points are given, None is returned. If 1 point is given,
# a circle of radius 0 is returned.
#
# Initially: No boundary points known


def make_circle(points):
    # Convert to float and randomize order
    shuffled = [(float(x), float(y)) for (x, y) in points]
```

```
    random.shuffle(shuffled)

    # Progressively add points to circle or recompute circle
    c = None
    for (i, p) in enumerate(shuffled):
        if c is None or not is_in_circle(c, p):
            c = _make_circle_one_point(shuffled[: i + 1], p)
    print("center x, center y: ", c[0], ", ", c[1], "\n radius:", c[2])
    return c


# One boundary point known
def _make_circle_one_point(points, p):
    c = (p[0], p[1], 0.0)
    for (i, q) in enumerate(points):
        if not is_in_circle(c, q):
            if c[2] == 0.0:
                c = make_diameter(p, q)
            else:
                c = _make_circle_two_points(points[: i + 1], p, q)
    return c


# Two boundary points known
def _make_circle_two_points(points, p, q):
    circ = make_diameter(p, q)
    left = None
    right = None
    px, py = p
    qx, qy = q

    # For each point not in the two-point circle
    for r in points:
        if is_in_circle(circ, r):
            continue

        # Form a circumcircle and classify it on left or right side
        cross = _cross_product(px, py, qx, qy, r[0], r[1])
        c = make_circumcircle(p, q, r)
        if c is None:
            continue
        elif cross > 0.0 and (
                left is None or _cross_product(px, py, qx, qy, c[0], c[1]) >
```

```
                _cross_product(px, py, qx, qy, left[0], left[1])):
            left = c
        elif cross < 0.0 and (
                right is None or _cross_product(px, py, qx, qy, c[0], c[1]) <
                _cross_product(px, py, qx, qy, right[0], right[1])):
            right = c

    # Select which circle to return
    if left is None and right is None:
        return circ
    elif left is None:
        return right
    elif right is None:
        return left
    else:
        return left if (left[2] <= right[2]) else right


def make_diameter(a, b):
    cx = (a[0] + b[0]) / 2.0
    cy = (a[1] + b[1]) / 2.0
    r0 = math.hypot(cx - a[0], cy - a[1])
    r1 = math.hypot(cx - b[0], cy - b[1])
    return (cx, cy, max(r0, r1))


def make_circumcircle(a, b, c):
    # Mathematical algorithm from Wikipedia: Circumscribed circle
    ox = (min(a[0], b[0], c[0]) + max(a[0], b[0], c[0])) / 2.0
    oy = (min(a[1], b[1], c[1]) + max(a[1], b[1], c[1])) / 2.0
    ax = a[0] - ox
    ay = a[1] - oy
    bx = b[0] - ox
    by = b[1] - oy
    cx = c[0] - ox
    cy = c[1] - oy
    d = (ax * (by - cy) + bx * (cy - ay) + cx * (ay - by)) * 2.0
    if d == 0.0:
        return None
    x = ox + ((ax * ax + ay * ay) * (by - cy) + (bx * bx + by * by) *
              (cy - ay) + (cx * cx + cy * cy) * (ay - by)) / d
    y = oy + ((ax * ax + ay * ay) * (cx - bx) + (bx * bx + by * by) *
              (ax - cx) + (cx * cx + cy * cy) * (bx - ax)) / d
```

```
    ra = math.hypot(x - a[0], y - a[1])
    rb = math.hypot(x - b[0], y - b[1])
    rc = math.hypot(x - c[0], y - c[1])
    return x, y, max(ra, rb, rc)


_MULTIPLICATIVE_EPSILON = 1 + 1e-14


def is_in_circle(c, p):
    return c is not None and math.hypot(p[0] - c[0], p[1] - c[1]) <= \
           c[2] * _MULTIPLICATIVE_EPSILON


# Returns twice the signed area of the triangle defined by
# (x0, y0), (x1, y1), (x2, y2).
def _cross_product(x0, y0, x1, y1, x2, y2):
    return (x1 - x0) * (y2 - y0) - (y1 - y0) * (x2 - x0)


# points = {(0, 0), (0, 2), (1, 1)}
points = set()
while len(points) < 20:
    x, y = random.randint(0, 10), random.randint(0, 10)
    points.add((x, y))
# print(points)

x_cord, y_cord, rad = make_circle(points)
xs = [x[0] for x in points]
ys = [x[1] for x in points]
plt.scatter(xs, ys)

circle_1 = plt.Circle((x_cord, y_cord), rad, color='r', fill=False)
ax = plt.gca()
ax.cla()  # clear things for fresh plot

# change default range so that new circles will work
ax.set_xlim((x_cord - 3/2*rad, x_cord + 3/2*rad))
ax.set_ylim((y_cord - 3/2*rad, y_cord + 3/2*rad))
ax.plot(xs, ys, 'o', color='blue', label='Points')
ax.add_artist(circle_1)
ax.set_aspect('equal')
ax.set_xlabel('x axis')
```

```
171 ax.set_ylabel('y axis')
172 ax.legend()
173 ax.set_title("Smallest enclosing circle", fontsize=10,
174               verticalalignment='bottom')
175 plt.savefig('190226_smallest_enclosing_circle.pdf')
176 plt.show()
```

Appendix B

Raw data

B.1 Raw data - Laser tracker

P1	-38	599210244100988	-1	884437094638298	20	580877545358401
P2	-21	113531458201546	-1	919277938254828	23	97231990785524
P3	3	6017806847964362	-1	9749401832466389	22	415964742101721
P4	44	340726775577018	-2	0223979672802441	22	751260454463281
P5	46	591122074296536	-1	9552566918082164	6	9753185235985207
P6	10	003111797636294	-1	9209767387222996	6	5821459309348755
P7	-20	264978584843149	-1	8850504841555447	6	7403405815295185
P8	-41	321455519143591	-1	8069674231296062	4	6503155922046373
P9	-40	402180531050121	-1	6408437795220834	-11	474366562794097
P10	-9	1728756344027076	-1	6669607341232278	-13	278555732087849
P11	16	97012079977733	-1	7044346037333753	-13	775122307059402
P12	47	249920795005188	-1	7389338876009788	-12	872845437659699
P13	-38	153009064570142	8	2048767367009017	32	063844527529994
P14	-13	312990346653216	8	1196999979462419	31	892521319964612
P15	16	370098669424262	6	8658025315526112	31	702621970810355
P16	45	503601627017424	7	349411905589454	31	54109518044773
P17	47	138375071064083	27	472896121799753	31	640701445416685
P18	24	148340920025866	24	595163174301632	31	770032525409427
P19	20	465291304211327	40	560742686103133	31	855707114631393
P20	42	05279474282662	40	835365858619205	31	754328963449552
P21	45	920728280798564	63	697488204107337	31	636464245264047
P22	21	443419983372465	61	811023931776035	31	783848928277202
P23	-3	7830027950369525	61	888113538769545	31	944086235210353
P24	-35	788196225552134	58	797651503393844	32	195253562603448
P25	56	136916907794266	8	0661812134519373	21	847411941366971
P26	56	088440030080783	7	0329845370561088	5	0885903457499104
P27	56	07973963700465	6	174631652275405	-11	690753597231451
P28	56	074442455836774	26	650482797236247	-10	678965163990394
P29	56	040101353314093	26	070713231914333	7	8641815711928373
P30	56	205404843670387	26	896093796089485	22	529216562434335
P31	56	213349205259206	45	75264023947588	23	208892182981877
P32	56	167149154096755	47	307925766769614	6	0117694660885066
P33	56	058497799816188	48	651213236272014	-9	8828504368829897
P34	56	174827560813441	63	311441978495047	-8	3668468682967561
P35	56	201159339475453	63	314225975766384	5	8248307015861585

P37	46	359589564714391	73	829088734355395	21	359582973666647
P38	21	022599280127672	73	859331723047958	22	063130669642831
P39	-2	6590240519984731	73	922711507994052	24	301248003639529
P40	-39	713657155600878	74	056483488835838	21	045350093609134
P41	-35	388918888089002	74	212516934925191	4	7799894414905699
P42	-11	494043748326886	74	147579559364459	3	1519709115421177
P43	10	305468278945604	74	095072073206509	4	8975309075558107
P44	43	331052390242398	73	991651768282324	5	2008422773367871
P45	47	87410335919823	74	082402482838845	-11	055496581843412
P46	20	616163301166189	74	156986513536538	-10	639188837671371
P47	-2	3397830209796608	74	180079077188481	-11	53119325928863
P48	-26	680694603261109	74	239317823381427	-12	050501608754276
P49	-41	148196432550549	74	217074798470946	-10	465238417920139
P50	-49	308954882931616	63	721524706008715	23	652883007854445
P51	-49	350648892241267	62	054405362782788	5	712199122414388
P52	-49	362158200215667	62	893286503217908	-9	8890181222492402
P53	-49	381693949258484	42	935731268904561	-8	5529962040853498
P54	-49	34095337998383	42	350556236185525	6	3505895380114401
P55	-49	289817795668561	42	82843982613992	21	801762751240087
P56	-49	54269094367443	8	3937629256504227	-10	423254780721354
P57	-49	479048970997169	9	3865715571103294	7	6435228498796874
P58	45	417082631544325	60	315355092852428	-23	95616260276033
P59	23	613805568214048	62	556346617598166	-23	799205188062366
P60	48	011233362910332	38	392958709773062	-23	990241671197069
P61	35	237268062380579	37	759329619041722	-23	940264341352059
P62	41	779970222188112	8	4307146008308909	-23	906584639128656
P63	19	832727963240742	10	989005064113098	-23	759516357566483
P64	-13	480455307778923	59	602713229270343	-24	110843760971015
P65	-39	620148021555195	63	155934427284002	-23	924462741136264
P66	-40	360535509201242	51	474525029357302	-24	012005257738597
P67	-41	482408999512941	19	977514596972963	-23	932429170546683
P68	-42	230715294308638	6	3540031519849203	-23	883113197181473
P69	-17	267279205675209	12	961427119312065	-23	983542966705773

B.2 Raw data - CMM

```
LIN1, 7.682373, −0.000000, 23.933180, 1.000000, 0.000000, 0.000000, 80.488592
LIN1 HITS, 7.682373, −0.000000, 23.933570, 0.000000, −1.000000, 0.000000
LIN1 HITS, 88.170965, −0.000000, 23.932790, 0.000000, −1.000000, 0.000000
PNT1, −0.000000, 2.622369, 23.937910, −0.999927, −0.012079, 0.000000
PNT1 HITS, −0.000000, 2.622369, 23.937910, −0.999927, −0.012079, 0.000000
PNT2, 8.240182, 10.486465, 0.000000, 0.000000, 0.000000, −1.000000
PNT2 HITS, 8.240182, 10.486465, 0.000000, 0.000000, 0.000000, −1.000000
PLN_Y+, 53.326138, 70.074703, 26.398789, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 95.280256, 70.208220, 9.666484, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 82.147171, 70.236140, 9.664844, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 72.733278, 70.231680, 9.664674, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 59.972681, 70.255319, 9.666954, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 39.372693, 70.259075, 9.665344, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 23.557890, 70.277251, 9.665424, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 10.352599, 70.280377, 9.666964, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 10.328983, 70.192105, 22.379464, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 27.051958, 70.190899, 22.378824, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 45.423137, 70.167830, 22.378524, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 63.247387, 70.140525, 22.378174, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 76.950652, 70.112733, 22.377654, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 92.744767, 70.094326, 22.378464, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 92.725413, 69.785680, 44.055484, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 82.674084, 69.804433, 44.054494, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 72.371923, 69.816745, 44.055464, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 59.474087, 69.866878, 44.055524, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 46.716248, 69.885759, 44.054954, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 33.814414, 69.904241, 44.054034, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 23.421840, 69.923702, 44.056954, 0.001350, 0.999933, 0.011478
PLN_Y+ HITS, 9.487449, 69.934841, 44.055864, 0.001350, 0.999933, 0.011478
PLN_X+, 99.697511, 35.248401, 26.721370, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.754052, 65.583298, 44.087312, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.726536, 52.797663, 44.086752, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.688203, 40.347832, 44.087242, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.667183, 28.488113, 44.086322, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.673663, 29.442731, 31.677742, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.714786, 38.375669, 31.678122, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.686660, 47.338803, 31.678692, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.719128, 58.740907, 31.679042, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.722828, 58.762414, 8.319982, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.650946, 51.727562, 8.167592, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.661881, 43.803446, 8.167992, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.706907, 30.032045, 8.167812, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.734175, 10.756945, 8.681544, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.727326, 10.757652, 13.445554, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.695247, 10.754844, 22.009674, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.663672, 10.756733, 30.323304, 1.000000, −0.000424, 0.000074
PLN_X+ HITS, 99.664499, 10.756153, 43.918614, 1.000000, −0.000424, 0.000074
PLN_Y−, 51.165648, 0.008902, 27.095528, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 93.130209, −0.170626, 43.942995, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 82.488682, −0.171403, 43.943245, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 70.290179, −0.150446, 43.943225, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 57.455828, −0.140069, 43.943695, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 40.589439, −0.148456, 43.943265, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 30.606991, −0.147713, 43.943655, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 22.976698, −0.147118, 43.943665, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 14.924338, −0.151102, 43.944095, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 14.696220, −0.018497, 25.858715, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 22.485416, −0.018293, 25.858875, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 35.768087, −0.024304, 25.858575, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 53.752504, −0.028813, 25.858225, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 67.040076, −0.034875, 25.858155, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 80.882720, −0.038512, 25.858095, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 95.019817, −0.031491, 25.858275, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 95.010428, 0.289549, 6.074225, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 80.193162, 0.269262, 6.074675, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 55.779722, 0.260945, 6.074805, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 36.773581, 0.262932, 6.074915, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 20.226725, 0.251284, 6.075185, −0.000146, −0.999940, −0.010986
PLN_Y− HITS, 4.387777, 0.274680, 6.075515, −0.000146, −0.999940, −0.010986
PLN_X−, −0.002314, 31.699787, 23.793011, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.047068, 2.008152, 6.050401, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.061644, 2.118684, 9.553771, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.059442, 2.001792, 14.488981, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.057085, 2.005341, 20.761091, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.022878, 2.027365, 29.997491, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.014265, 2.135577, 41.277421, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.025243, 28.149393, 41.277831, −0.999999, 0.001295, −0.000080
PLN_X− HITS, 0.016359, 28.137104, 35.133781, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.004054, 28.143818, 28.937261, −0.999999, 0.001295, −0.000080
PLN_X− HITS, 0.028100, 28.142886, 23.646931, −0.999999, 0.001295, −0.000080
PLN_X− HITS, −0.012503, 28.142906, 19.462131, −0.999999, 0.001295, −0.000080
```

```
PLN_X- HITS, -0.011547, 28.199522, 10.644261, -0.999999, 0.001295, -0.000080
PLN_X- HITS, 0.009320, 64.914066, 46.336080, -0.999999, 0.001295, -0.000080
PLN_X- HITS, 0.020305, 64.893867, 35.754140, -0.999999, 0.001295, -0.000080
PLN_X- HITS, 0.042967, 64.892961, 26.723700, -0.999999, 0.001295, -0.000080
PLN_X- HITS, 0.045988, 64.893757, 19.756300, -0.999999, 0.001295, -0.000080
PLN_X- HITS, 0.046969, 64.893659, 13.174240, -0.999999, 0.001295, -0.000080
PLN_X- HITS, 0.064070, 64.895316, 5.298390, -0.999999, 0.001295, -0.000080
PLN_Z-, 49.161839, 108.510974, 0.296012, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 6.174643, 205.411376, 0.620607, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 11.238602, 205.472638, 0.619377, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 19.989684, 205.580018, 0.616777, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 41.312077, 205.837727, 0.651867, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 56.390598, 206.019712, 0.650897, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 66.711976, 206.144331, 0.640227, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 75.429035, 206.250031, 0.649867, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 90.071646, 206.426190, 0.653657, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 90.196566, 189.031210, 0.506307, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 90.262358, 181.258408, 0.472447, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 90.407996, 167.442709, 0.468797, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 8.796654, 199.086296, 0.567557, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 8.889763, 188.463385, 0.500097, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 8.988552, 167.201308, 0.488927, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 92.587084, -135.532539, -0.529975, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 81.740746, -135.662760, -0.515965, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 58.241267, -135.947198, -0.487245, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 27.374256, -135.402516, -0.458475, -0.000140, 0.003254, -0.999995
PLN_Z- HITS, 9.271445, -135.371825, -0.491515, -0.000140, 0.003254, -0.999995
PLN_Z+, 60.176838, 3.782908, 49.771480, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 13.883842, -135.464166, 49.312907, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 28.085473, -135.295714, 49.287327, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 41.871336, -135.126645, 49.230747, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 58.303834, -134.927034, 49.224587, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 71.874413, -134.763025, 49.218737, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 85.506160, -134.600947, 49.195567, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 93.922404, -134.498341, 49.180257, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 93.619899, -109.486551, 49.380097, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 83.808198, -109.608654, 49.406597, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 72.001582, -109.749064, 49.414217, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 64.828041, -109.836969, 49.420157, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 11.274540, 206.837004, 50.638849, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 21.426169, 206.960453, 50.595009, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 38.414029, 207.169033, 50.526189, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 54.982167, 207.366181, 50.503599, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 71.922155, 207.570462, 50.461639, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 83.733119, 207.713515, 50.452819, 0.001437, -0.003628, 0.999992
PLN_Z+ HITS, 93.725727, 207.832813, 50.437329, 0.001437, -0.003628, 0.999992
```

Bibliography

Abernethy, R. B. and Ringhiser, B. (1985). The History and Statistical Development of the New ASME-SAE-AIAA-ISO Measurement Uncertainty Methodology. In *Proc. AIAA/SAE/ASME/ASME 21st Joint Propulsion Conference*, pages 8–10. Citeseer.

AIAA (1995). Assessment of wind tunnel data uncertainty. Technical report, AIAA S-071-1995, Washington DC.

ASME PTC 19.1-2013 (2013). Test Uncertainty. Standard, The American Society of Mechanical Engingeers.

Barequet, G. and Har-Peled, S. (2001). Efficiently Approximating the Minimum-Volume Bounding Box of a Point Set in Three Dimensions. *Journal of Algorithms*, 38(1):91–109.

Brandenberg, R. and Theobald, T. (2004). Algebraic Methods for Computing Smallest Enclosing and Circumscribing Cylinders of Simplices. *Applicable Algebra in Engineering, Communication and Computing*, 14(6):439–460.

Bykat, A. (1978). Convex hull of a finite set of points in two dimensions. *Information Processing Letters*, 7(6):296–298.

Chan, C. and Tan, S. (2004). Putting objects into a cylindrical/rectangular bounded volume. *Computer-Aided Design*, 36(12):1189–1204.

Chang, C.-T., Gorissen, B., and Melchior, S. (2011). Fast oriented bounding box optimization on the rotation group SO (3, ℝ). *ACM Transactions on Graphics (TOG)*, 30(5):122.

Dieck, R. H. (2007). *Measurement Uncertainty: Methods and Applications*. ISA.

Dimitrov, D., Knauer, C., Kriegel, K., and Rote, G. (2009). Bounds on the quality of the PCA bounding boxes. *Computational Geometry*, 42(8):772–789.

Durand, N. and Alliot, J.-M. (1999). A combined nelder-mead simplex and genetic algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO*, volume 99, pages 1–7.

EA-4/02 M (2013). Evaluation of the Uncertainty of Measurement in Calibration. Standard, European Accreditation.

Eberly, D. (2000;2019). Least Squares Fitting of Data by Linear or Quadratic Structures. *Chapel Hill, NC: Magic Software*.

Eddy, W. F. (1977). A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3(4):398–403.

Figliola, R. S. and Beasley, D. (2015). *Theory and Design for Mechanical Measurements*. John Wiley & Sons.

Freeman, H. and Shapira, R. (1975). Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7):409–413.

Green, P. and Silverman, B. (1979). Constructing the convex hull of a set of points in the plane. *The Computer Journal*, 22(3):262–266.

ISO 10360-2:2009 (2009). Geometrical product specifications (GPS) – Acceptance and reverification tests for coordinate measuring machines (CMM) – Part 2: CMMs used for measuring linear dimensions. Standard, International Organization for Standardization.

ISO 10360-4:2000 (2000). Geometrical product specifications (GPS) – Acceptance and reverification tests for coordinate measuring machines (CMM) – Part 4: CMMs used in scanning measuring mode. Standard, International Organization for Standardization.

ISO 10360-5:2010 (2010). Geometrical product specifications (GPS) – Acceptance and reverification tests for coordinate measuring machines (CMM) – Part 5: CMMs using single and multiple stylus contacting probing systems. Standard, International Organization for Standardization.

ISO/IEC 98-3:2008 (2008). Uncertainty in Measurement – Part 3: Guide to the expression of uncertainty in measurement (GUM:1995). Standard, International Organization for Standardization.

Jarvis, R. A. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information processing letters*, 2(1):18–21.

Justervesenet (2019). Measuring technique. https://www.justervesenet.no/maleteknikk/. Accessed: 2019-02-05.

Jylänki, J. (2015). An exact algorithm for finding minimum oriented bounding boxes. *Semantic Scholar*.

Kline, S. J. (1953). Describing uncertainty in single sample experiments. *Mechanical Engineering*, 75:3–8.

Korsawe, J. (2008). Minimal bounding box. http://www.mathworks.com/matlabcentral/fileexchange/18264. Accessed: 2019-05-20.

Mitchell, M. (1996). An Introduction to Genetic Algorithms MIT Press. *Cambridge, Massachusetts. London, England.*

Moritz, D. (2013). Convex hull in 3D. https://commons.wikimedia.org/wiki/File:Convex_hull_in_3D.svg. Accessed: 2019-05-01.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313.

Nurunnabi, A., Sadahiro, Y., Lindenbergh, R., and Belton, D. (2019). Robust cylinder fitting in laser scanning point cloud data. *Measurement*, 138:632–651.

O'Rourke, J. (1985). Finding minimal enclosing boxes. *International journal of computer & information sciences*, 14(3):183–199.

Pan, X. (2017). cylinder_fitting. https://github.com/xingjiepan/cylinder_fitting. Accessed: 2019-04-02.

Petitjean, M. (2012). About the algebraic solutions of smallest enclosing cylinders problems. *Applicable Algebra in Engineering, Communication and Computing*, 23(3-4):151–164.

Preparata, F. P. and Hong, S. J. (1977). Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20(2):87–93.

Preparata, F. P. and Shamos, M. I. (1985). *Computational Geometry: An Introduction*. Springer-Verlag, Berlin.

Project Nayuki (2018). Smallest enclosing circle. https://www.nayuki.io/page/smallest-enclosing-circle. Accessed: 2019-04-01.

Schömer, E., Sellen, J., Teichmann, M., and Yap, C. (2000). Smallest Enclosing Cylinders. *Algorithmica*, 27(2):170–186.

Shamos, M. I. (1978). Computational geometry. *Ph. D. thesis, Yale University*.

Steele, W., Ferguson, R., Taylor, R., and Coleman, H. (1994). Comparison of ANSI/ASME and ISO models for calculation of uncertainty. *Isa Transactions*, 33(4):339–352.

Sylvester, J. J. (1857). A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1.

Toussaint, G. T. (1983). Solving Geometric Problems with the Rotating Calipers. In *Proc. IEEE Melecon*, volume 83, page A10.

Camilla Kjølstad

Uncertainty in Calibration of Reference Standards for Dimensional Measurements

NTNU
Norwegian University of
Science and Technology