Chair of Structural and Functional Ceramics

Master's Thesis

Development and Automation of a Bulge Test Procedure for Polymer Supported Thin Films

Alexander Kuhn

September 2024

Master's Thesis

Development and Automation of a Bulge Test Procedure for Polymer Supported Thin Films

Alexander Kuhn

2024

Montanuniversität Leoben
Eidgenössische Materialprüfungs- und Forschungsanstalt Thun
Mechanics of Materials and Nanostructures

Ass.-Prof. Dr. Barbara Putz

Master's Thesis

Development and Automation of a Bulge Test Procedure
for Polymer Supported Thin Films
Study program: Advanced Materials Science and Engineering
2024

Submitted by Alexander Kuhn
Day of Submission: 10th September 2024

Supervisor: Ass.-Prof. Dr. Barbara Putz

This work was carried out at Empa, the Swiss Federal Laboratories for Materials Science and Technology, with support for research provided by the Swiss National Science Foundation (SNSF) through the project PZ00P2 202089

Montanuniversität Leoben
in Cooperation with
Eidgenössische Materialprüfungs- und Forschungsanstalt Thun
Mechanics of Materials and Nanostructures
Thun, Switzerland

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my family for their unwavering support throughout this journey. During difficult times, they encouraged me to keep going, and they never discouraged my decision to pursue a Master's degree, even as a "professional long-term student" after completing my Bachelor's. Their patience and love have been invaluable.

I would also like to thank Erwin Pieper from Empa Dübendorf for manufacturing a lid for the bulge setup. Thanks are also due to Jérémie Bérard for developing the pressure control system and to Peter Denninger for his advice on using Autodesk Inventor, as well as for the valuable discussions regarding the determination of the zero point.

A special thanks goes to the Erasmus Mundus Association, the EU, and all European taxpayers for granting and enabling me the Erasmus Mundus scholarship. I know this opportunity cannot be taken for granted, and I hope to repay it in some way through my future professional or academic contributions. I would also like to extend my gratitude to the great EUSMAT Team in Saarbrücken that coordinated the program and provided patient support throughout the AMASE program.

I am especially grateful to Ass.-Prof. Dr. Barbara Putz, the supervisor of this master's thesis, for her patient guidance. Despite her incredibly busy schedule, filled with professional travel and commitments, she found time to answer my questions, even when I showed up at her office unannounced. She is remarkably gifted and possesses an exceptional depth of expertise, which greatly benefited my work.

As a Christian, I also want to express my gratitude to my Lord and Savior for providing me with comfort and strength throughout this journey, especially during challenging times.

> *But they that wait upon the Lord shall renew their strength;*
> *they shall mount up with wings as eagles;*
> *they shall run, and not be weary; and they shall walk, and not faint.*
> *— Isaiah 40:31 (KJV)*

Finally, I would like to share a profound thought by Viktor Frankl, whose wisdom about finding meaning in adversity has been a source of inspiration for me during this journey:

> "*When we are no longer able to change a situation,*
> *we are challenged to change ourselves.*"
> *— Viktor Frankl*

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

AFFIDAVIT

I declare on oath that I wrote this thesis independently, did not use any sources and aids other than those specified, have fully and truthfully reported the use of generative methods and models of artificial intelligence, and did not otherwise use any other unauthorized aids.

I declare that I have read, understood and complied with the "Good Scientific Practice" of the Montanuniversität Leoben.

Furthermore, I declare that the electronic and printed versions of the submitted thesis are identical in form and content.

Date 10.09.2024

Signature Author
Alexander Kuhn

Use of AI Tools in Thesis Writing

In this master thesis, AI tools are used at different stages in the writing process to assist with drafting and refinement of content. AI was used in generating the abstract and the Summary Chapter 13, with only minimal revisions made by the author.

Since the author of this master thesis is not a native English speaker, in every chapter, some parts were written in English or "Denglish" (a mix of English and German) and then later translated *as precisely as possible* to keep the writing style using DeepL and AI. The sections written in English were further refined with prompts such as "*Suggest synonyms for [...]*" or "*Correct spelling and grammar errors without generating or modifying the text, while keeping my writing style.*" This approach ensured that the text was refined without altering the original intent, avoiding automatic generation of content. AI was also used to correct larger stylistic issues (*grobe Stilfehler*) by employing prompts like "*Which sentences are too colloquial or informal for a scientific work such as a master thesis? Provide suggestions how to rephrase.*"

Additionally, the flowcharts in Figure 5.7 and Figure 11.1 were created using AI and later modified by the author to better fit the context of the thesis.

The equations used in this thesis in the LaTeX environment were created as follows: the equations were first handwritten on paper, then photographed and processed by AI to convert them into LaTeX code. This method made it easier to integrate the equations into the LaTeX environment and ensured accuracy.

Ironically, the generative language model ChatGPT was also used to help reduce redundant text rather than generating new content. For example, prompts such as "*Show parts that are redundant and repetitive*" were employed to streamline the content.

In summary, AI tools were utilized for tasks such as translation, grammar refinement, synonyms suggestions and flowchart generation, LaTeX equation conversion, and text reduction, while trying to preserve the author's original writing style and intent throughout the thesis.

Use of LaTeX Template

The LaTeX template used for this master thesis was created by *Prof. Dr.-Ing. Ralf Steinmetz* from the *Technische Universität Darmstadt*, and made available via overleaf.com; I thank him for that. This template therefore provided a structured format that guided the process of writing, ensuring consistency with regard to formatting, citation, and layout within the thesis.

Remark: The equations 3.28 and 5.2 are wrongly displayed in Figure 6.1. Instead of a^4 in the cubic term, where the Young's modulus is, a^2 was displayed. However, the calculations in the program (see code in the appendix) were all done correctly. Due to time constraints, the displayed formulas in the mentioned figure could not be corrected. For determining the Young's modulus, the correct formulas with a^4 were used to calculate the Young's modulus based on the Nix and Timoshenko model.

Abstract

This thesis focuses on the development of an automated bulge test procedure that can be used to measure and analyze the mechanical properties of thin films on polymer substrates. The system was specifically designed to evaluate material characteristics like residual stress, elastic modulus, and crack formation under equi-biaxial loading condition. One of the main materials tested was aluminum-coated Kapton, which is commonly used in flexible electronics, aerospace applications, and as electrical insulation. By automating the testing process, the system ensures consistent and accurate results for the evaluation of thin metal films on polymer substrates. The setup also allows for detailed investigation into how cracks form and propagate in thin films, offering valuable insights into material deformation and potential failure points. Ultimately, this system could be an tool for improving the mechanical properties of polymer-based materials, particularly in industries like electronics and aerospace.

List of Abbreviations

MFC	Mass Flow Controller
ISO	International Organization for Standardization
SEM	Scanning Electron Microscopy
TEM	Transmission Electron Microscopy
LED	Light Emitting Diode
FEP	Fluorinated Ethylene Propylene
E-Modulus	Young’s Modulus
S neox	Model of Sensofar’s confocal microscope
SMR	Single Measurement Recipe
MMR	Multiple Measurement Recipe
CAD	Computer-Aided Design
CLI	Command Line Interface

Contents

Part I.

Introduction and State of the Art

1 Introduction to the Thesis

The goal of this master's thesis is to develop a comprehensive and automated procedure for a bulge testing setup, aimed at investigating the mechanical properties of thin films on polymer substrates. Specifically, the focus was to study the effects and crack formation of aluminium coatings on Kapton substrates, , which are commonly used in flexible electronics, aerospace components, and high-performance electrical insulation.

The main objective is to create a reliable procedure for the bulge testing setup that can accurately measure and analyze the mechanical properties of various materials under biaxial stress conditions. This involves automating the testing process, ensuring reproducibility, and applying the setup to different classes of materials, such as soft polymer membranes and polymers coated with metallic thin films. By doing so, the thesis aims to contribute to the broader understanding of how thin films behave under stress and how these properties can be optimized for practical applications in industries such as aerospace and electronics.

1.1 Motivation

The motivation for this research stems from the need to accurately determine crack onset and propagation in thin films on polymer substrates, particularly under biaxial loading conditions, which are common in real-world applications. This study starts with the investigation of bare Kapton substrates, followed by examining the effects of aluminum coatings on these substrates. The ultimate goal is to refine the bulge testing setup, making it adaptable to a wide range of material systems and providing critical insights into the mechanical behavior of polymer-based materials under stress.

1.2 Contribution

The primary contribution of this thesis is the development of a fully automated bulge testing procedure, which allows for the precise analysis of mechanical properties such as elastic modulus, residual stress, and crack formation in thin films. This process has been applied to both bare and aluminum-coated Kapton substrates, providing valuable insights into their behavior under biaxial stress conditions. These findings contribute to the broader optimization of polymer-based materials for practical applications. Furthermore, the framework established by this research offers a solid foundation for future studies, particularly in understanding crack propagation and the mechanical performance of various coated polymer substrates.

2 Background of Bulge Testing

The bulge test, a method used to measure thin film mechanics, works by applying pressure to a freestanding membrane and tracking its deflection. It is known for its precise sample fabrication and minimal handling, with a simple and portable design that offers cost-effectiveness compared to nanoindentation and uniaxial tensile stress tests. By altering the shape of the bulge window, the test can achieve diverse stress/strain states. Originally designed for freestanding films, the bulge test has now been applied to thin films on polymers as well. Theoretical development began with Hencky in 1915 [1], expanded by Beams [2] for material property measurement, and further evolved with Tsakalakos [3] for circular membranes, Small [4] considering intrinsic stress, and Vlassak and Pratt [5],[6] for square membranes. Tabata and Vlassak [5],[6] extended this to rectangular membranes, enhancing its application scope

In the context of this thesis, the term 'sample' may refer to the films on substrates or to the uncoated substrate itself. The bulge equation, based on linear elasticity, relates central deflection to strain and applied pressure to stress. From the measurement of this pressure/deflection relationship together with the size and shape of the window, mechanical properties of the sample can be deduced. Further, monitoring the surface with a microscope provides detailed insights into the mechanical behavior of the coating.

2.1 Bulge Test Setup

The Bulge Test Setup used at Empa Thun is shown in Figure 2.1. The setup consists of a chamber designed to hold a sample with a diameter of 20 mm, with a window size of 14 mm in diameter, which is the area being subjected to bulging. The chamber has a volume of 2.5 cm^3. The system includes an inlet valve for introducing pressure into the chamber, an outlet valve for pressure release, and a third valve connected to a pressure sensor for monitoring the pressure inside the chamber.

Additionally, the sample is clamped by the lid which is secured by screws, ensuring a tight and secure fit during testing. The pressure sensor can withstand a maximum pressure of 10 bar and the system allows a flow rate of up to 1 liter per minute. The compressors can compress air to a maximum of 6 bar, but the limiting factor is the Mass Flow Controllers (MFCs), which can only handle pressures up to 3.8 bar without risk of damage.

2.2 Types of Bulge Tests and Their Setups

Addititional Bulge tests up exist, which will be briefly discussed in this section. In addition to the pneumatic bulge test setup hydrostatic bulge testing exists too. This test is regulated by international standards such as ISO 16808 [7], ensuring consistency and reliability in test results.

Pneumatic systems are well-suited for testing thin, soft materials like foils due to their speed and efficiency, but they can introduce shock waves, particularly in denser or stronger materials. In contrast, hydraulic systems offer greater versatility, allowing for the testing of a wider range of materials, including strong metals. While hydraulics provide better sealing and more consistent results, they tend to be messier and require a more complex, time-consuming setup between tests [8].

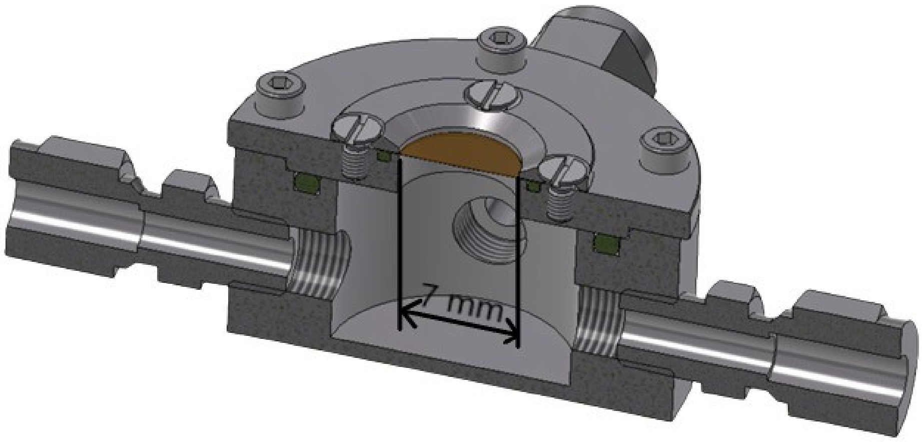


Figure 2.1.: Bulge Test Apparatus. This apparatus configuration was used to apply controlled pressure to thin films in the experiments conducted for this master thesis.

3 Derivation of the Circular Window Model in the Bulge Test

In this chapter, the derivation of one model for the bulge test is presented: the Circular Window Model. The term "window" in the context of bulge testing refers to the portion of the thin film that is exposed to pressure and deformation. Different window shapes—such as circular, rectangular, or square—are used depending on the specific test setup and the material properties being evaluated. Each window shape affects the distribution of stress and strain in the film. The focus of this chapter will be on the Circular Window Model, as it is the one used in this thesis. The derivation of the rectangular and square models can be found in [9] and [10].

3.1 Derivation of the Equations for the Circular Window Model

The bulge test is a method used to measure the mechanical properties of thin films. In this model, a thin film clamped by a circular window is subjected to a pressure differential, resulting in a bulge forming in the film. The analysis of this bulge can provide information about the stress and strain in the material.

3.1.1 Understanding the Basic Geometry

We begin by considering the geometry of the problem. A thin film is clamped along a circular edge of radius a and subjected to a uniform pressure p that causes a central deflection or bulge height h. The profile of the bulge can be approximated as a segment of a sphere with a radius of curvature R.

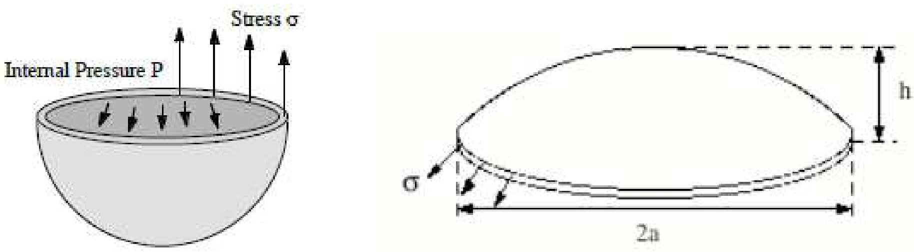


Figure 3.1.: Schematic sketch of a bulged film in spherical shape [9]

3.1.2 Establishing the Relationship Between Pressure and Stress

The force exerted by the pressure p on the circular window is given by:

$$p \cdot \pi R^2 = \sigma \cdot 2\pi R t \tag{3.1}$$

Here:

- $p \cdot \pi a^2$ is the force exerted by the pressure on the film (since pressure is force per unit area and the area of the circular window is πa^2),
- σ is the stress in the film, defined as force per unit area,
- $2\pi R t$ is the area over which the force acts, where t is the film thickness.
- R is the radius of curvature

Thus, solving for σ gives:

$$\sigma = \frac{p \cdot R}{2t} \tag{3.2}$$

This equation expresses the stress σ in terms of the applied pressure p, the radius curvuture R, and the film thickness t.

3.1.3 Relating Geometric Parameters to Stress

The radius of curvature R of the bulge is related to the bulge height h and the window radius a by the following relation:

$$R = \frac{a^2 + h^2}{2h} \tag{3.3}$$

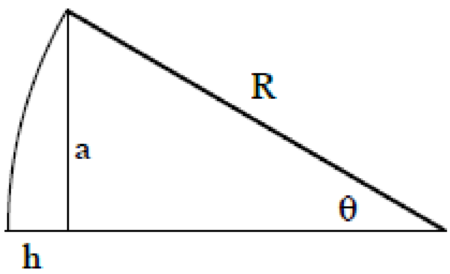


Figure 3.2.: Schematic sketch of cross section of substrate/film and its geometry [9]

3.2 Stress in Terms of Geometric Parameters

We begin by inserting the expression for R from equation (3.3) into the stress equation (3.2). The radius of curvature R is given by:

$$R = \frac{a^2 + h^2}{2h} \tag{3.4}$$

Substituting this into the stress equation (3.2), we get:

$$\sigma = \frac{p \cdot R}{2t} = \frac{p \cdot \frac{a^2+h^2}{2h}}{2t} \tag{3.5}$$

Simplifying, we arrive at:

$$\sigma = \frac{p(a^2+h^2)}{4th} \tag{3.6}$$

3.2.1 Small Deflection Approximation

For small deflections, where h^2 is negligible in comparison to a^2, we can approximate the stress equation (3.6) as follows:

$$\sigma_{\text{elastic}} \approx \frac{p \cdot a^2}{4th} \tag{3.7}$$

This simplified equation is valid for cases where the bulge height h is much smaller than the window radius a.

3.2.2 Deriving the Strain from Deflection

The strain ϵ in the film is given by:

$$\epsilon = \frac{R\theta - a}{a} = \frac{\theta - \left(\frac{a}{R}\right)}{\left(\frac{a}{R}\right)} \tag{3.8}$$

Using a Taylor expansion, we can approximate the angle θ as:

$$\theta = \arcsin\left(\frac{a}{R}\right) \approx \left(\frac{a}{R}\right) + \frac{1}{6}\left(\frac{a}{R}\right)^3 + \frac{3}{40}\left(\frac{a}{R}\right)^5 + 7\text{th order term} + \dots \tag{3.9}$$

By terminating the expansion at the third-order term, before including the 5th and 7th order terms, etc., the strain can be expressed as:

$$\epsilon = \frac{a^2}{6R^2} = \frac{2h^2}{3a^2} \tag{3.10}$$

3.2.3 Combining Stress and Strain for the Biaxial Modulus

The stress in the film is related to the strain by the biaxial modulus B, which is a function of the material properties (Young's modulus E and Poisson's ratio ν):

$$\sigma_{\text{elastic}} = B \cdot \epsilon \tag{3.11}$$

where $B = \frac{E}{1-\nu}$.

3.2.4 Incorporating Material Properties

By combining equations 3.2 and 3.11, we can derive the relationship between the applied pressure and the deflection of the film:

$$p = \frac{8Bth^3}{3a^4} \tag{3.12}$$

3.2.5 Accounting for Residual Stress

In thin films, residual stress, denoted as σ_0, can arise during fabrication processes such as deposition. This stress exists in the film prior to any applied pressure and can significantly affect the film's mechanical response. To account for this, the pressure-deflection relationship is modified to include the contribution from the residual stress.

The general condition for mechanical equilibrium is that the sum of all forces in the system must equal zero:

$$\sum_i F_i = 0 \tag{3.13}$$

The total force exerted on the film by the applied pressure p is given by the product of the pressure and the area of the circular window:

$$F_{\text{total}} = p \cdot \pi R^2 \tag{3.14}$$

This total force must be balanced by the force generated by the stress in the film. In the presence of residual stress, the total stress in the film consists of two components:

1. The elastic stress generated by the applied pressure, σ_{elastic},
2. The residual stress σ_0, which exists in the film prior to loading.

It is important to note that the following derivation assumes the film is in the *elastic regime*. Therefore, we specifically refer to *elastic stress* σ_{elastic}, which is proportional to the strain induced by the applied pressure.

The force due to the elastic stress in the film is proportional to the cross-sectional area of the film, $2\pi at$, where t is the thickness of the film. Thus, the force due to elastic stress can be written as:

$$F_{\sigma_{\text{elastic}}} = \sigma_{\text{elastic}} \cdot 2\pi Rt \tag{3.15}$$

Similarly, the force due to the residual stress is:

$$F_{\sigma_0} = \sigma_0 \cdot 2\pi Rt \tag{3.16}$$

The total force acting on the film is the sum of the elastic force and the residual stress force:

$$F_{\text{total}} = F_{\sigma_{\text{elastic}}} + F_{\sigma_0} \tag{3.17}$$

Substituting the expressions for the forces, the total force becomes:

$$p \cdot \pi a^2 = \sigma_{\text{elastic}} \cdot 2\pi Rt + \sigma_0 \cdot 2\pi at \tag{3.18}$$

This total force must be balanced by the force generated by the stress in the film. In the presence of residual stress, the total stress in the film consists of two components:

1. The elastic stress generated by the applied pressure, σ_{elastic},
2. The residual stress σ_0, which exists in the film prior to loading.

The force due to the elastic stress in the film is proportional to the cross-sectional area of the film, $2\pi at$, where t is the thickness of the film. Thus, the force due to elastic stress can be written as:

$$F_{\sigma_{\text{elastic}}} = \sigma_{\text{elastic}} \cdot 2\pi Rt \tag{3.19}$$

Similarly, the force due to the residual stress is:

$$F_{\sigma_0} = \sigma_0 \cdot 2\pi R t \tag{3.20}$$

The total force acting on the film is the sum of the elastic force and the residual stress force:

$$F_{\text{total}} = F_{\sigma_{\text{elastic}}} + F_{\sigma_0} \tag{3.21}$$

Substituting the expressions for the forces, the total force becomes:

$$p \cdot \pi a^2 = \sigma_{\text{elastic}} \cdot 2\pi R t + \sigma_0 \cdot 2\pi R t \tag{3.22}$$

3.3 Deriving Pressure from Stress and Deflection

We begin with the equilibrium condition for pressure p, the elastic stress σ_{elastic}, and the surface stress σ_0:

$$p \cdot \pi R^2 = \sigma_{\text{elastic}} \cdot 2\pi R t + \sigma_0 \cdot 2\pi R t \tag{3.23}$$

Now, using the relation $R = \frac{a^2+h^2}{2h}$ (eq. 3.4) and neglecting h^2 results in $R = \frac{a^2}{2h}$. Cancelling π from both sides:

$$p \cdot R^2 = \sigma_{\text{elastic}} \cdot 2Rt + \sigma_0 \cdot 2Rt \tag{3.24}$$

Substituting $R = \frac{a^2}{2h}$ and $\sigma_{\text{elastic}} = B \cdot \epsilon$:

$$p\left(\frac{a^4}{4h^2}\right) = (B \cdot \epsilon) \cdot 2\left(\frac{a^2}{2h}\right)t + \sigma_0 \cdot 2\left(\frac{a^2}{2h}\right)t \tag{3.25}$$

Now, using the strain relation $\epsilon = \frac{2h^2}{3a^2}$ (eq. 3.10):

$$p \cdot \frac{a^4}{4h^2} = B \cdot \frac{2h^2}{3a^2} \cdot 2 \cdot \frac{a^2}{2h} \cdot t + \sigma_0 \cdot 2 \cdot \frac{a^2}{2h} \cdot t \tag{3.26}$$

Simplifying the terms by dividing both sides by a^2 and multiplying by h^2:

$$p \cdot \frac{a^2}{4} = B \cdot \frac{2h^2}{3a^2} \cdot h \cdot t + \sigma_0 \cdot t \cdot h \tag{3.27}$$

Thus, solving for p:

$$p = \frac{8Bt}{3a^4} \cdot h^3 + \frac{4\sigma_0 t}{a^2} \cdot h \tag{3.28}$$

Explanation of the Terms:

- The first term, $\frac{8Bth^3}{3a^4}$, corresponds to the contribution from the elastic deformation of the film, where B is the biaxial modulus, t is the film thickness, h is the deflection height, and a is the window radius
- The second term, $\frac{4\sigma_0 th}{a^2}$, accounts for the effect of the residual stress σ_0 in the film. The residual stress acts along the film's cross-sectional area, contributing to the overall mechanical response under the applied pressure

This final equation shows how both the elastic deformation and the residual stress influence the pressure-deflection relationship. The residual stress term increases the overall force needed to create a given deflection, thereby modifying the film's response to applied pressure.

3.4 Conclusion of the Derivations

The presented equations link the observed deformation of a bulged thin film under known pressure to its material properties such as Young's modulus and Poisson's ratio, including any residual stresses. This set of relationships forms the theoretical basis for analyzing the results of the bulge test and extracting valuable information about the mechanical behavior of thin films.

4 Recording the Pressure-Deflection Curves

In the previous chapter, the derivation of the bulge test models provided a means to extract mechanical properties, such as stress and strain, from pressure-deflection relationships. This chapter focuses on how to record these pressure-deflection curves, which are critical for evaluating the behavior of thin films under mechanical load. There are various methods described in the literature for measuring deflection in bulge tests, including those standardized by ISO, which are widely used in industry. These methods differ in precision, applicability, and the kind of information they provide about the material under test.

4.1 Methods to Measure Deflection

One common approach involves the use of a tri-profilometer or height measurement devices. These devices measure the height of the bulged membrane by profiling the surface, providing accurate deflection data across the membrane. However, these methods come with some limitations, particularly the inability to capture visual data, such as the evolution of crack patterns or localized defects, which can be crucial for a comprehensive material analysis.

4.2 Limitations of Tri-Profilometer and Height Measurement Devices

Although tri-profilometers and other height measurement devices offer high precision in measuring deflection, their major disadvantage lies in the lack of visual data. Without image capture capabilities, these devices cannot document crack initiation and propagation during testing. This lack of imaging makes it difficult to link crack-onsets to specific points on the pressure-deflection curve with certainty. As a result, crack patterns cannot be analyzed in real time or correlated with the data.

4.3 Advantages of Imaging for Crack Analysis

In contrast, methods that incorporate imaging — such as using optical or scanning electron microscopy (SEM) — allow for real-time observation of crack patterns, stress concentrations, and other failure mechanisms as they develop during testing. These techniques can provide detailed insights into the material's mechanical properties beyond just deflection measurements. The ability to capture and analyze crack patterns will be explored in greater detail in chapter 11

4.4 Confocal Microscopy and Characterization

In this thesis, Sensofar's S neox confocal microscopy technique is employed to measure surface deformations with high precision. The microscope uses multispectral LEDs to illuminate a small spot on the film's surface and measures the intensity of the reflected light. By scanning across the surface, it creates a detailed topographical map, which is crucial for analyzing how the thin film deforms under pressure. This data is essential for calculating material properties such as stress and strain [11].

Confocal microscopy works by using an aperture to block out-of-focus light, capturing only the focused plane of the sample. This selective focusing allows for the generation of high-resolution 2D profiles and 3D surface images, making it ideal for analyzing complex surfaces, such as those found in thin films under mechanical stress. It is widely used in fields like materials science and nanotechnology due to its ability to provide accurate surface measurements with fine detail.

One of the advantages of the S neox system is its higher resolution compared to conventional tri-profilometers, despite using LEDs instead of lasers, which are typically favored for their higher coherence and spatial resolution. This makes the S neox suitable for most surface topography applications, though lasers might be preferred for extremely fine detail.

The key specifications of the S neox optical 3D profiler are summarized in Table 4.1. These specifications are critical for understanding the system's measurement range and precision. For instance, the high numerical aperture (NA) of 0.95 allows for fine detail to be captured, while the Z-stage resolution as low as 0.75 nm with a piezo stage enables precise vertical measurements. This table is included to show that the system can measure both large-scale deformations and very small changes in surface topology, making it versatile for different types of thin films.

Table 4.1.: Key Specifications of the S neox Optical 3D Profiler

Specification	**Details**
Magnification (MAG)	2.5X to 150X
Numerical Aperture (NA)	0.075 to 0.95
Working Distance (WD)	0.2 mm to 23.5 mm
Field of View (FOV)	Up to 6800x5675 μm
Spatial Sampling	As low as 0.09 μm
Optical Resolution	As fine as 0.14 μm
Measurement Array	1232 x 1028 pixels
Vertical Range	Up to 200 μm with a piezo stage
Z Stage Resolution	2 nm with a linear stage, 0.75 nm with a piezo stage
Step Height Accuracy	0.5%

One limitation of the S neox system is the absence of a command line interface (CLI), which would enable external systems to be directly coupled to the Sensofar device. This would allow the recording of measurements to be synchronized with specific pressure levels, helping to mitigate overshooting issues discussed in Chapter **??**. A CLI could significantly improve the integration of this system into more automated experimental setups, where precise timing of measurements is critical.

Part II.

Methodology and Development of the Bulge Test Procedure

5 Development of a Bulge Test Setup

5.1 Development of a Bulge Testing Setup

The body of the bulge test setup was already fabricated and did not need to be developed from scratch. However, modifications were necessary to integrate confocal imaging, which the previous setup could not support. To enable this, the bulge testing setup was equipped with an S neox confocal microscope from Sensofar [11], which provides high-precision surface measurements with a vertical resolution of 1 nm and a lateral resolution of 140 nm.

Figure 5.1 provides an overview of the setup, showing the key components integrated for the experiment. The S neox microscope is positioned to capture high-resolution images of thin film deformations. Pressure control is handled by an in-house developed LabView program, which allows precise regulation of the pressure, displayed in mbar. The multi-flow controllers manage the inlet and outlet pressures, and the setup is connected to a compressor capable of generating pressures up to 6 bar.

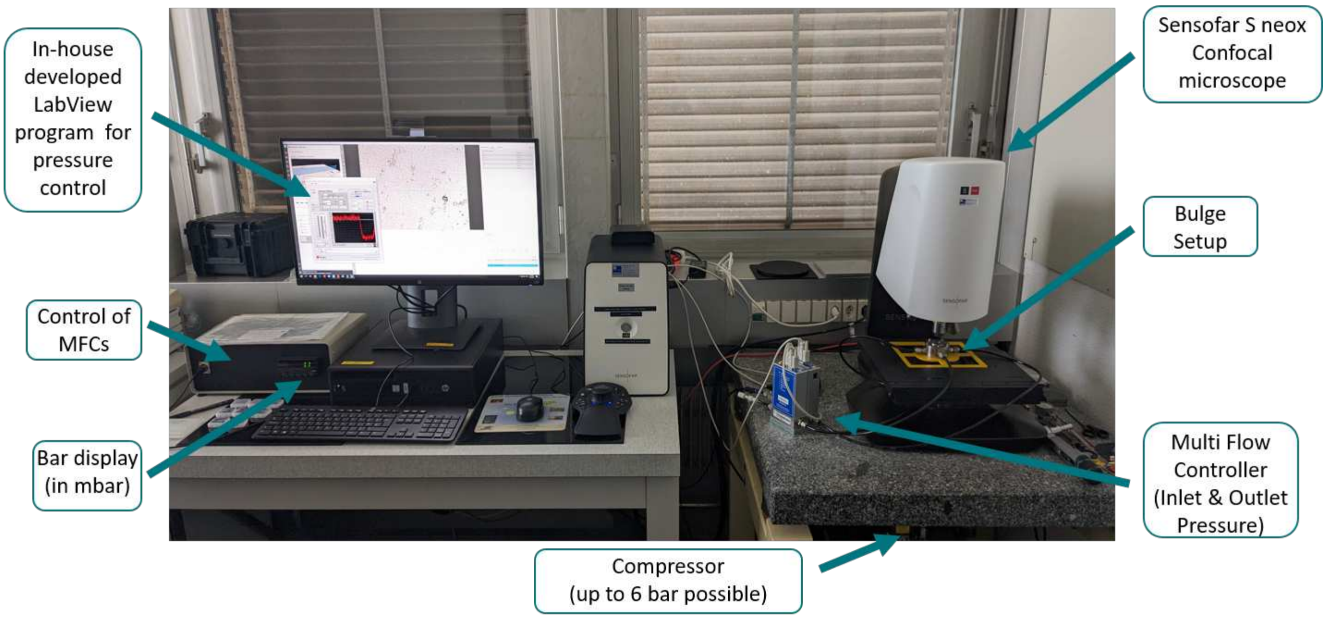


Figure 5.1.: Overview of the Bulge Testing Setup integrated with the S neox confocal microscope. Key components include the LabView program for pressure control, Multi Flow Controller, and Compressor.

5.2 Leak Detection and Pressure Control

There were initial leaks that had to be detected and addressed. Air was leaking from the hose connections, which was quickly resolved by wrapping Teflon tape around the threads. Fixing these leaks was critical to maintaining a stable pressure during the experiment, as any leakage would affect the accuracy of the pressure readings and could lead to inconsistent results.

The pressure control system was developed using LabVIEW. The pressure is generated by a compressor, and the mass flow controllers (MFCs) — one for inflow and one for outflow — regulate the set pressure entering the chamber. Although the compressor can generate up to 6 bar, the limiting factor for the pressure range is the MFCs, which, according to specifications, only allow up to 3.9 bar. While experiments were conducted up to 6 bar, it is not advisable, as exceeding 3.9 bar could potentially damage the valves that regulate the mass flow based on the opening angle.

At Empa, two pressure sensors are used: one for pressures up to 1 bar with a resolution of 0.001 bar, and another for pressures up to 10 bar with a resolution of 0.01 bar. These sensors provide the necessary precision for accurately controlling the pressure during the tests.

5.3 Fixation of the Bulge Test Setup via Framework

To capture the maximum bulging, it is essential that the maximum deflection stays within the field of view of the microscope. This ensures that all deformations can be properly observed and recorded. The field of view was discussed earlier in relation to the Sensofar microscope, which highlights the importance of positioning during measurements. To avoid the need to manually locate the center each time, a 3D-printed fixture was designed and printed to fit the dimensions of the turntable on the stage, ensuring the setup remains centered throughout the experiment (see Figure 5.2).

As shown in Figure 5.2, the setup includes two components: a 3D-printed fixture and an aluminum plate. The 3D-printed fixture was designed to fit the bulge test setup perfectly, allowing for easy and repeatable placement of the samples.

To further improve accuracy, an aluminum plate was used to prevent any bending during measurements. This ensures that the bulge setup remains stable and that the center of the bulging membrane does not need to be located and aligned each time, allowing for more consistent and repeatable results.

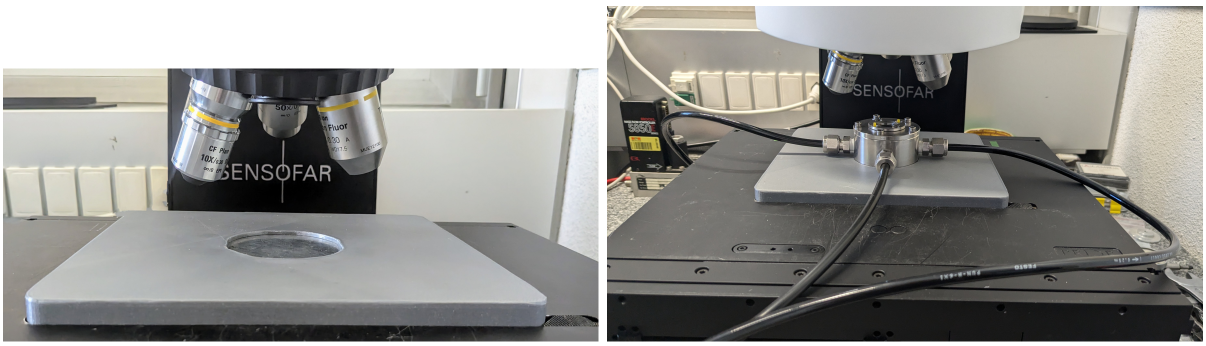


Figure 5.2.: (a) 3D-printed fixture used for the bulge testing setup to ensure proper positioning and (b) aluminum plate used to improve accuracy by preventing bending during measurements. These components help maintain stability and ensure repeatability in positioning the samples.

5.4 Zero Point Determination

Determining the zero point, the starting Z-value in the global coordinate system of the Sensofar microscope from which deflection measurements are taken, proved to be anything but trivial. The current bulge setup presents a major challenge: the samples are clamped, which induces stresses and deformations. During a discussion with Prof. Vlassak, he pointed out that this is not ideal, and his team uses epoxy resin to avoid these clamping issues.

Experimentally, it was concluded that the zero point should be set at 0.01 bar to best fit the formula (**??**). This pressure allows the sample to be nearly straight with minimal deformation. At 0.00 bar, the sample tends to bulge slightly inward due to the stress caused by clamping. To prevent this, applying a slight pre-tension at 0.01 bar ensures that the sample remains in a more stable and straight condition.

Alternatively, the zero point can be set at 0.00 bar, with the Z-value at this pressure manually focused using the confocal microscope and recorded in the global coordinate system.However, experience has shown that the global z-value at 0.00 bar can vary between samples, even of the same type, while setting the zero point with a small amount of pre-tension at 0.01 bar provides more consistent Z-values across samples.

Once the Z-values are recorded with the lid on, but before the sample is clamped, the screws are tightened to 8-10 Nm to secure the sample. This process often induces slight curvature. When a pressure of 0.01 bar is applied, the sample returns to an almost straight condition. For stiff materials, the deformation at 0.01 bar is negligible, but for materials with an E-Modulus below 1 GPa, this small deformation should be considered and a lower pre-tension is advised. To minimize additional stress, it is important to tighten the screws homogeneously in a cross pattern to prevent uneven clamping, which can affect measurement accuracy.

When conducting the test, the user is free to choose whether to use the measured value (where the lid simply rests on the sample but is not yet clamped) as the zero point or the zero point with a slight pre-tension of 0.01 bar. (Applying a pre-tesnion is also common in tensile testing machines). Evidently, the chosen zero point will affect the measured deflections. From experience, the results vary by less than 1 percent (typically around 0.7%).

5.5 Objective Selection and Measurement Duration

The selection of the appropriate objective is critical for accurate bulge testing measurements, as it affects both the field of view and the ability to capture key deformation points. The following table lists the four available objectives and their respective fields of view:

Table 5.1.: Objectives and Corresponding Fields of View (in micrometers, μm)

Objective	**10x**	**20x**	**50x**	**150x**
Field of View (in μm)	1700 x 1418.6	850 x 709	340 x 283.7	113.3 x 94.6

Lower magnifications (10x, 20x, and 50x) provide a larger field of view, which increases the likelihood of capturing the point of maximum deflection, even during large pressure jumps. These objectives are ideal for measuring residual stresses and calculating the modulus of elasticity, as they offer sufficient coverage to ensure the highest point of deflection is within the frame.

In contrast, the 150x objective is better suited for detailed studies of crack initiation and propagation. While it provides the level of detail necessary for studying crack onset (see Section 12.2), its smaller field of view makes it more challenging to center on the point of maximum deflection, which can lead to underestimating deflection and consequently overestimating stiffness.

5.6 Measurement Process and Data Extraction

Once the zero point is determined as discussed in 5.4, the Inficon (pressure program) is started. After 60 seconds, the Sensofar SMR program is initiated. The 60-second delay is optional but strongly recommended, as the pressure control tends to overshoot, especially at pressures below 0.8 bar, and it takes some time during the holding phases to stabilize (see Figure 5.4 for details on pressure stabilization and overshooting behavior).

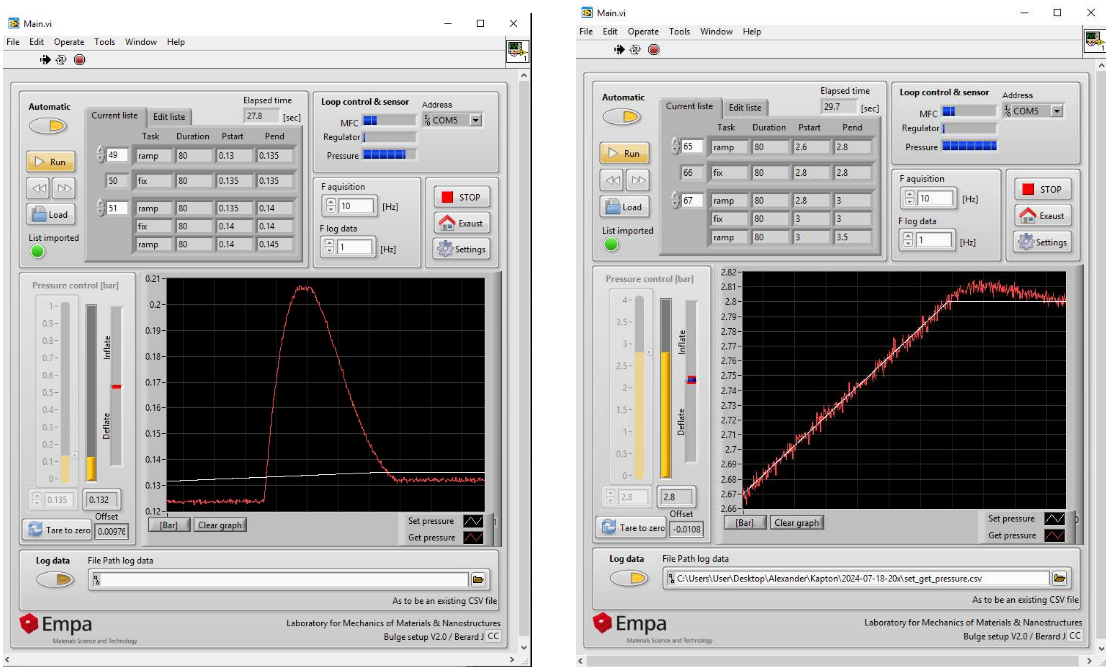


Figure 5.3.: (a) Overshooting caused by the valve opening too quickly and widely, which is difficult to mitigate. (b) Overshooting that can be managed by reducing the steepness of the ramp phase and increasing pressure more gradually. These images illustrate the different types of overshooting behavior observed in the pressure control system during bulge testing.

After each pressure increase (increment), a measurement is taken by the Sensofar microscope. Sensofar uses its own file format, .plux, which is essentially a zip file. It contains all topography data along with absolute coordinate system values. As the bulging (the curvature) of the sample increases with rising pressure, Sensofar adjusts the Z-value (the height) in each autofocused capture.

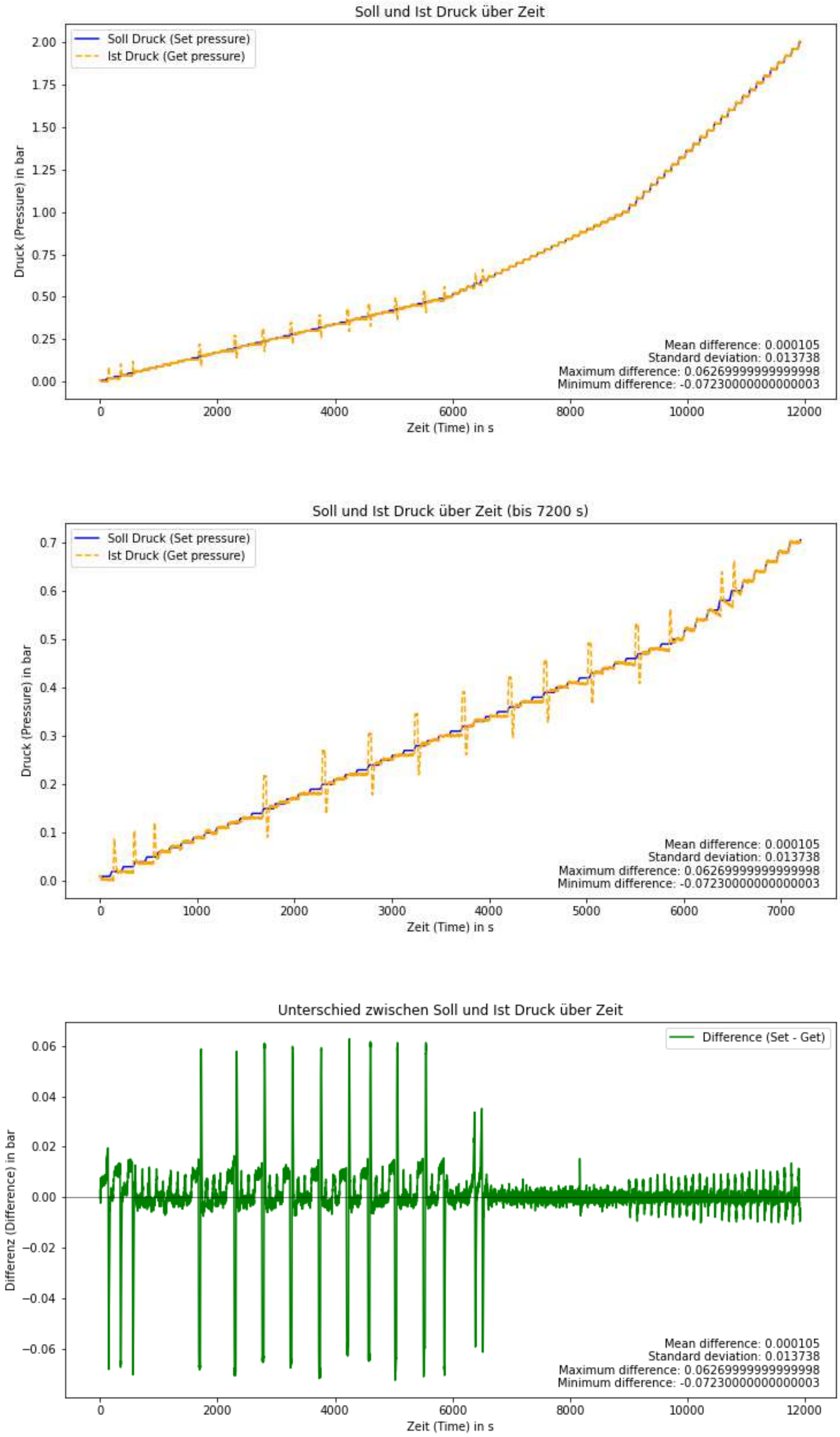


Figure 5.4.: Pressure measurement data collected during the experiment. (a) Shows the overall set and actual pressure over time. (b) Most overshootings occur up to 0.7 bar. (c) Highlights the difference between the set and actual pressure over time.

After the experiment, the Z-values are extracted from the .xml file (contained within the .plux file) and stored in a Z-array file. The first value represents the zero point. By subtracting the other values from the zero point, the measured deflection is calculated. It is important to note that only the Z-value of a single pixel is stored in the .xml file, which corresponds to the point displayed as the zero line (see the horizontal dashed black line in the cross profile in Figure 10.1). Since this point is not the highest point, the highest point within the capture must be identified and added.

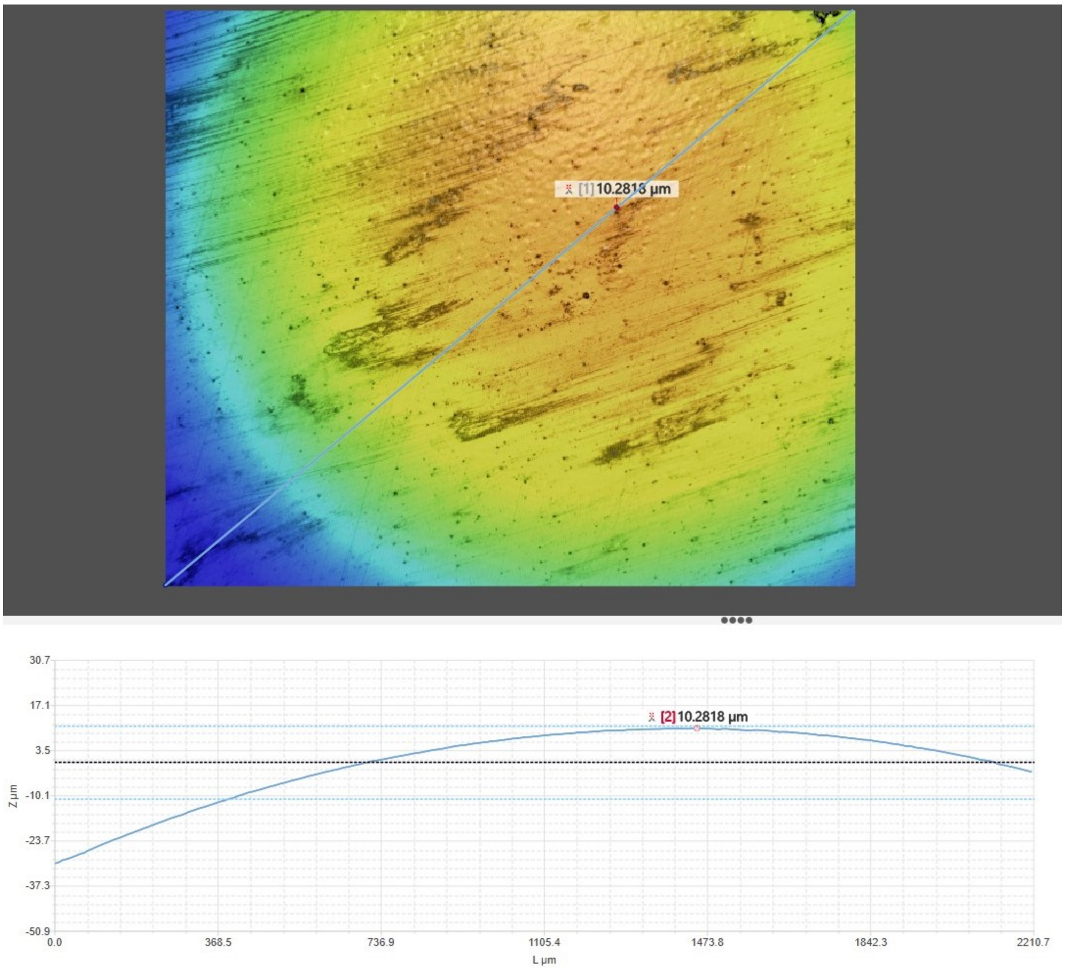


Figure 5.5.: Screenshot from Sensofar showing the measurement of the maximum deflection point. The highest point within the capture, which is not the zero line, is automatically identified and recorded.

The highest point, or the point of maximum deflection, is automatically detected when you select the appropriate template (.plut file) under "choose template" in the SMR recipe, where the maximum value is stored. (Further details can be found in the manual in the appendix.)

Depending on the selected pressure range and the pressure increment, the measurement can take up to 6 hours. The system automatically adjusts the focal plane to accommodate the increasing deflection. However, the 150x objective struggles to detect the focal plane if the pressure increment is too large, as excessive deflection exceeds the range for automated focal plane detection. To prevent this, the deflection between measurements with the 150x objective should remain small. This can be achieved by choosing a lower pressure increment.

5.7 Alternative Approaches for Measuring the Deflection

Instead of using the absolute z-values, which involves extracting the deflection of the highest point from .xml files, it is also possible to measure the deflection directly from a stitched image. A stitched image, as shown in Figure 5.6(b), is a composite image made from multiple individual images. The profile of this stitched image allows for the direct reading of the highest point. It is also not necessary to stitch the entire surface; it is sufficient to measure across the curved sample. Strictly speaking, one would only need to measure up to the midpoint, as this is the location with the highest curvature, as shown in Figure 5.6(b).

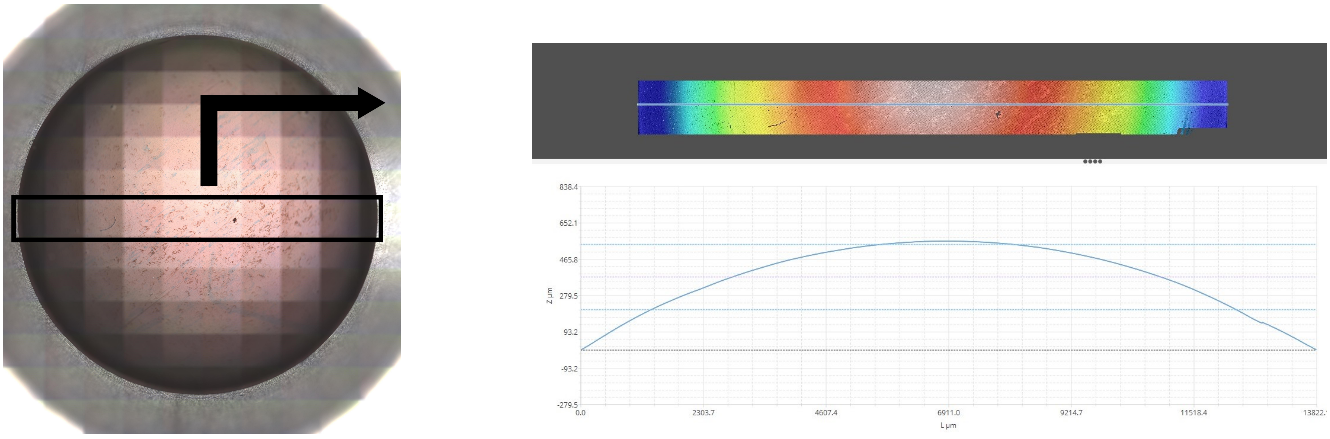


Figure 5.6.: (a) A top view of the curved sample for visual reference, and (b) a stitched image showing the profile of the curved sample. This alternative approach allows the highest point in the profile to be directly identified.

However, with the setup used in this thesis, this approach was not feasible, because the design of the lid has a sharp edge at the clamping point, which prevents the confocal microscope from taking measurements in those areas. Additionally, performing a Bulge Test using this method would take significantly longer, as the entire cross-sectional profile would need to be captured and stitched together after each pressure increase.

5.8 Data Analysis and Report Generation

Once the sample is manually positioned and the zero point is set, the entire bulge test process becomes fully automated. After the bulge test is completed and the automated detection of the Z-values is performed, all captures are stored in a folder at predefined pressure values along a set file path. Subsequently, a Python script is executed to automate the analysis and documentation of the bulge test experiments. The script extracts relevant data, performs fitting analyses to determine material properties, and generates comprehensive reports, including both visual and tabular representations of the results. The full code is provided in the appendix.

The main functionalities of the script are as follows:

1. **Data Extraction:** The script extracts Z-position values from .plux files stored within a specified directory. It retrieves both the set pressure and the actual pressure values from

CSV files, text files, or filenames. Additionally, it accounts for missing data and ensures a complete dataset for analysis. Missing data or recordings can occur if the pressure drops after the objective (Objektiv) is set at a given global z-value, causing all planes to be out of focus. If the image is more than 50% black, the recording is not saved. The script is designed to handle these cases accordingly.

2. **Curve Fitting and Analysis:** The script fits pressure-deflection data according to two primary models: the Nix Model and the Timoshenko method. Both models are discussed in Chapter 3. These models are used to determine fitting parameters for material properties such as residual stress (σ_0) and Young's modulus (E). The key difference between these models lies in how they account for boundary conditions and deformation behavior under pressure.
The Nix Model [12], given by:

$$P = \frac{4\sigma_0 th}{a^2} + \frac{8Eth^3}{3a^4(1-\nu)} \tag{5.1}$$

assumes that the film behaves according to the spherical membrane approximation, where the bulge height (h) is much smaller than the film radius (a). It simplifies the relation between pressure and deflection, making it suitable for certain types of materials and conditions. The complete derivation for the Nix model is shown in chapter 3.
The Timoshenko Model [12], given by:

$$P = \frac{4\sigma_0 th}{a^2} + \frac{(7-\nu)Eth^3}{3a^4(1-\nu)} \tag{5.2}$$

accounts for additional material behavior, such as the influence of Poisson's ratio (ν) on deformation. This method is typically used for more complex cases where the film exhibits larger deflections and is more compliant than predicted by the Nix Model.
In [12], according to Martha K. Small and W. D. Nix, the Timoshenko model, based on the energy minimization method, "*predicts more compliant film behavior than the spherical membrane model and a different dependence on Poisson's ratio. It should be pointed out that in Timoshenko's energy-minimization calculations, he generally assumes a value of 0.25 or 0.30 for the Poisson's ratio at some point in the derivation. This should be noted in reporting values of Young's modulus using these equations.*"
The choice between the two models depends on the specific material being tested and the deformation conditions. In general, the Timoshenko model is more appropriate when larger deflections are observed, as it includes higher-order terms and more accurately captures the behavior of the film. In this thesis, both models were used to calculate E and σ_0, allowing for a comparison of the material properties derived from each.

The script evaluates the fitting accuracy through R^2 scores and visualizes the fitted curve alongside the measured data points. Only the elastic regime is considered for fitting because the equations 5.1 and 5.2 are only valid in that regime.

3. **Strain and Stress Analysis:** The script uses the measured deflections of the material under varying pressures to calculate strain, membrane stress, and von Mises stress. The strain is calculated using the equation:

$$\epsilon = \frac{R\theta - a}{a} \tag{5.3}$$

Here, R is the radius of curvature, θ is the angular displacement, and a is the original radius. The membrane stress is calculated as:

$$\sigma_{\text{membrane}} = \frac{PR}{2t} \tag{5.4}$$

where P is the applied pressure, R is the radius of curvature, and t is the thickness of the film.

The von Mises stress, which is used to evaluate yield criteria in ductile materials, is given by:

$$\sigma_{\text{von Mises}} = \sqrt{\frac{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2}{2}} \tag{5.5}$$

where σ_1, σ_2, and σ_3 are the principal stresses in the x, y, and z directions, respectively. For circular bulge testing, assuming equibiaxial stress ($\sigma_1 = \sigma_2$) and $\sigma_3 = 0$, this simplifies to:

$$\sigma_{\text{von Mises}} = \sqrt{\frac{(\sigma_1 - \sigma_1)^2 + (\sigma_1 - 0)^2 + (0 - \sigma_1)^2}{2}} = \sqrt{\frac{0 + \sigma_1^2 + \sigma_1^2}{2}} = \sqrt{\frac{2\sigma_1^2}{2}} = \sqrt{\sigma_1^2} = \sigma_1$$

Therefore, the von Mises stress is equal to the membrane stress σ_1 under these conditions [13].

$$\sigma_{\text{von Mises}} = \sigma_1 \tag{5.6}$$

4. **Image Processing:** Optional, the recorded images can be processed from the bulge test to identify and analyze surface modification or crack formations in thin films on polymers.

The programm also calculates the number, type, and distances between cracks as shown in Chapter 12 in Figure 12.2

5. **Report Generation:** Finally, the program creates detailed PDF reports that include tables of pressure and deflection data, annotated images, and plots of various analyses. This provides a visual summary of the pressure vs. deflection, strain vs. stress, and other key metrics and, furthermore, it offers the option to generate videos with overlaid pressure, stress, and strain values.
6. **User Interaction:** The program collects input parameters such as measurement mode, window radius (Empa Thun has two setups with different diameters), thickness, Poisson's ratio, and magnification. It allows for customization of the analysis process, including setting zero points, defining maximum elastic regime values, and choosing to perform crack analysis or video creation. Currently, the program is tailored for circular windows only; for different geometries, the equations for stress and strain, as discussed in 3, should be adapted accordingly.
7. **File Management:** Organizes and saves extracted and processed data in structured directories. Generates filenames based on input parameters and current timestamps for easy tracking and retrieval of results.

This script ensures a consistent, automated process for analyzing and documenting bulge test results, as illustrated in the data extraction and analysis flowchart in Figure 5.7.

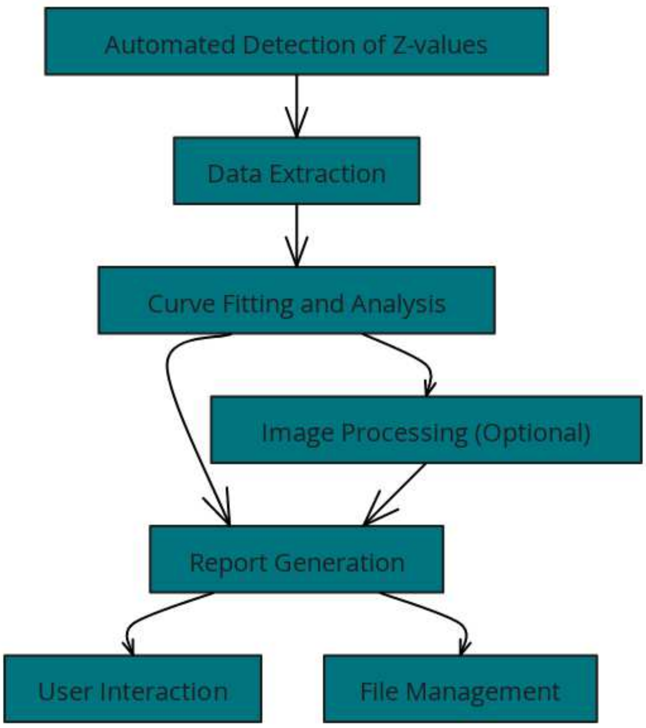


Figure 5.7.: Flowchart of data extraction and analysis process after the bulge test procedure is finished

6 Exemplary Analysis of a Bulge Test Experiment

6.1 Pressure-Deflection Curve Fitting

The pressure-deflection data for the Al/Kapton sample is fitted using a polynomial model [12]. The fitting parameters are used to derive the E-modulus (E) and the residual stress (σ_0) based on the Nix and Timoshenko models, as shown in Figure 6.1. The difference between the Nix and Timoshnko model are discussed in chapter 5.8

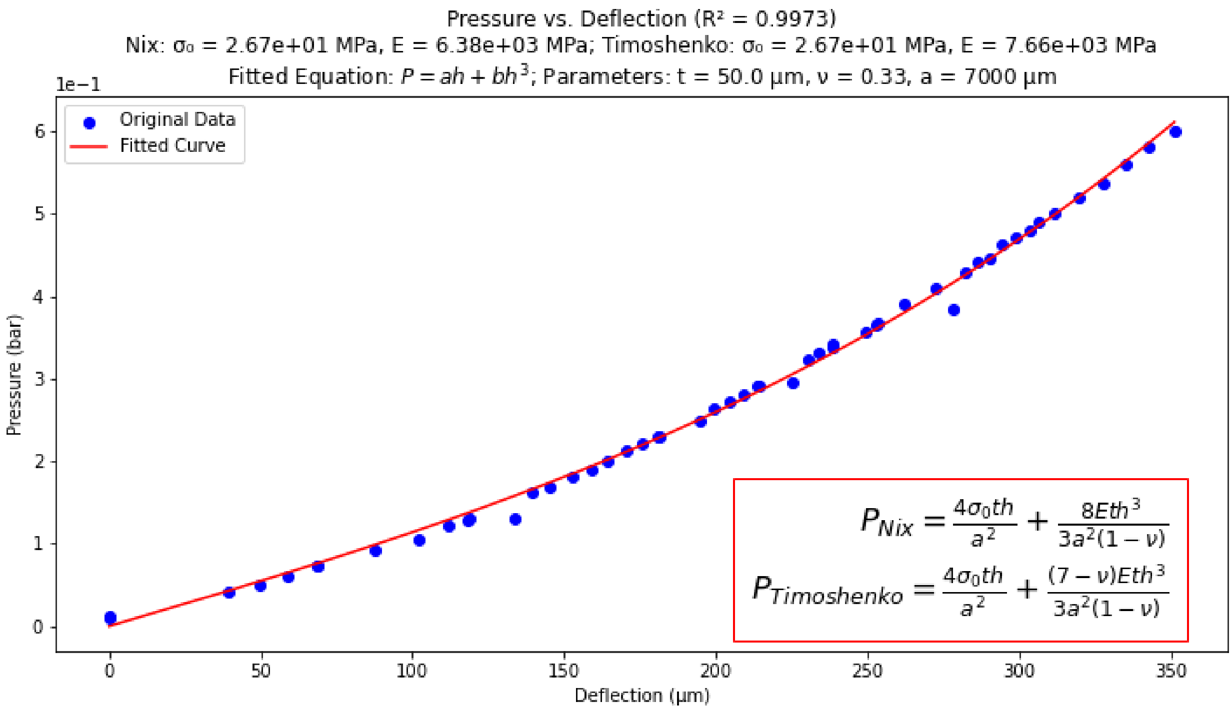


Figure 6.1.: Pressure vs. Deflection curve for Al (10 nm) on Kapton (50 μm).

6.2 Extrapolation and Shifted Stress-Strain Curve

Using the E-modulus obtained from the pressure-deflection fitting (Figure 6.1), the corresponding slope is identified in the stress-strain curve (Figure 6.2a). The strain is extrapolated back to the axis to determine the residual strain ϵ_0. The final step involves shifting the original stress-strain data to account for the residual strain ϵ_0, aligning the curve to provide an accurate representation of the film's mechanical behavior (Figure 6.2b).

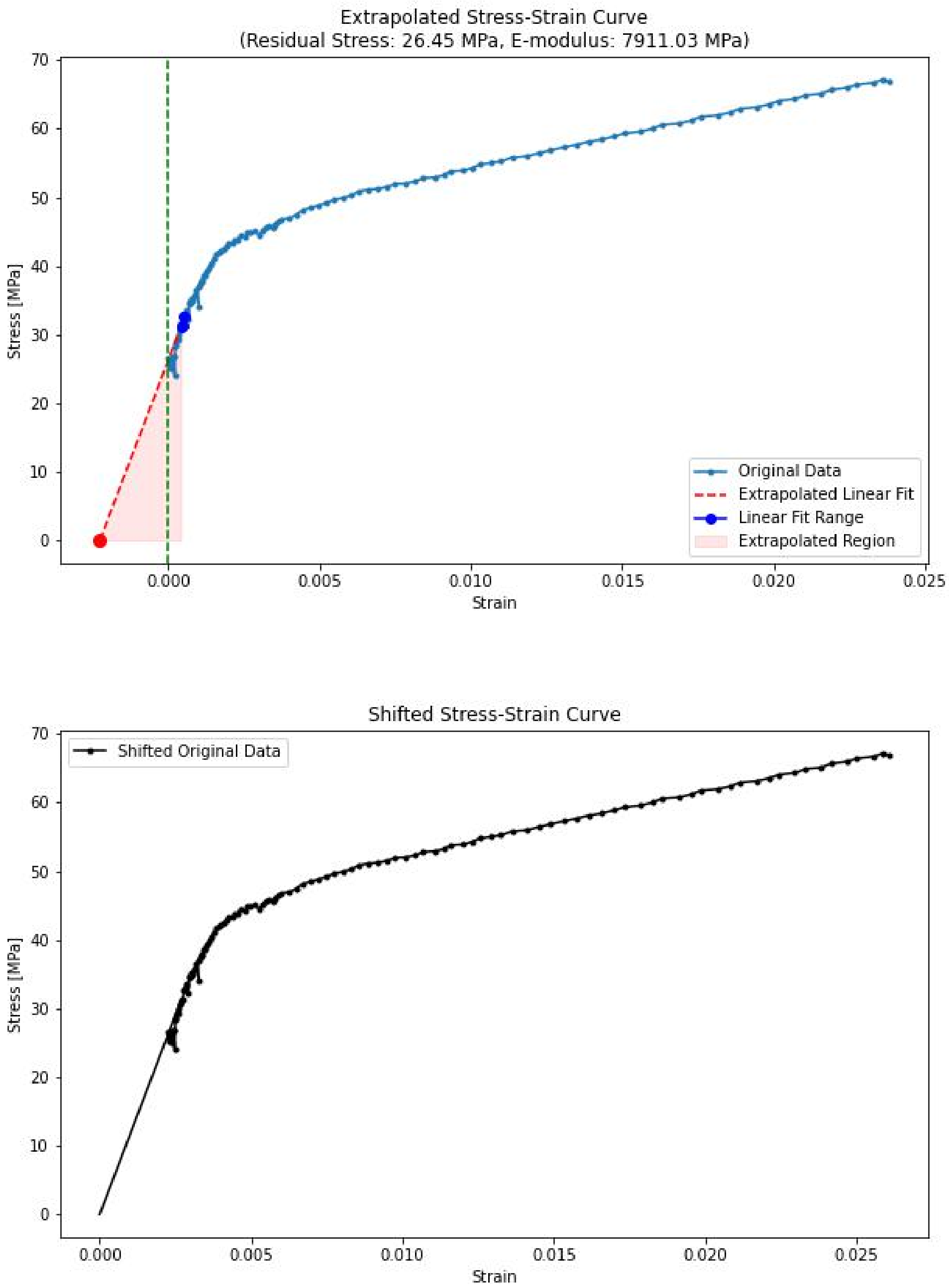


Figure 6.2.: (a) Extrapolated stress-strain curve where the intersection with the strain axis reveals ϵ_0. (b) Shifted stress-strain curve after accounting for ϵ_0.

The analysis demonstrates how residual strain ϵ_0 is calculated from a combination of pressure-deflection fitting and stress-strain extrapolation using Nix's fitting equations.

Residual strains ϵ_0 present in the material need to be taken into account for analysis of bulge test data. Reasons for residual strains are discussed in . In this exemplary analysis, the residual strain ϵ_0 in a 10 nm thick aluminum film deposited on a 50 micrometer Kapton substrate is evaluated. This process is exemplified for one sample but is also applicable to other samples. The steps involve fitting the pressure-deflection data (Figure 6.1), identifying the appropriate slope in the stress-strain curve (Figure 6.2), and finally calculating ϵ_0 from the extrapolated curve. For a circular window bulge setup, the stress σ determined from the applied pressure p and membrane deflection h using the following expressions [14]:

$$\sigma = \frac{p \cdot R}{2h}, \tag{6.1}$$

The strain ϵ in a circular window bulge test setup is derived based on the following steps: First, as shown in chapter 3.1.3 the radius of curvature R at the pole of the bulge is given by:

$$R = \frac{a^2 + h^2}{2h} \tag{6.2}$$

where:

- a is the window radius,
- h is the deflection at the center of the bulge.

Next, the angle θ subtended by the arc length over the deformed membrane is calculated as:

$$\theta = \arcsin\left(\frac{2ah}{a^2 + h^2}\right) \tag{6.3}$$

The arc length L of the membrane is then given by:

$$L = R \cdot \theta = \frac{a^2 + h^2}{2h} \cdot \arcsin\left(\frac{2ah}{a^2 + h^2}\right) \tag{6.4}$$

Strain ϵ is defined as the relative change in length between the deformed length L and the original radius a, normalized by the original length a:

$$\epsilon = \frac{L - a}{a} \tag{6.5}$$

Substituting the expression for L from (6.4):

$$\epsilon = \frac{\frac{a^2+h^2}{2h} \cdot \arcsin\left(\frac{2ah}{a^2+h^2}\right) - a}{a} \tag{6.6}$$

Finally, accounting for the presence of residual strain ϵ_0, the total strain is expressed as:

$$\epsilon = \frac{\frac{a^2+h^2}{2h} \cdot \arcsin\left(\frac{2ah}{a^2+h^2}\right) - a}{a} + \epsilon_0 \tag{6.7}$$

Thus, the final expression for the strain ϵ is:

$$\epsilon = \epsilon_0 + \frac{a^2+h^2}{2ah} \arcsin\left(\frac{2ah}{a^2+h^2}\right) - 1, \tag{6.8}$$

where a is the window radius, h is the deflection, t is the film thickness, and ϵ_0 is the residual strain. These equations are valid for spherical deformation and account for both elastic and plastic deformation regimes [15].

$$\epsilon = \epsilon_0 + \frac{a^2+h^2}{2ah} \arcsin\left(\frac{2ah}{a^2+h^2}\right) - 1, \tag{6.9}$$

where a is the window radius, h is the deflection, t is the film thickness, and ϵ_0 is the residual strain. These equations are valid for spherical deformation and account for both elastic and plastic deformation regimes [15].

Occasionally, the values obtained from the fitting curve's corresponding slope cannot be directly found in the strain-stress curve. In such cases, the Young's modulus can also be derived by applying a linear regression within the linear range of the strain-stress curve. However, this approach typically yields a higher Young's modulus compared to the value retrieved from the pressure-deflection curve using the Nix model. The Nix model connects equations 6.1 and 6.9 with the biaxial modulus formula, neglecting higher-order deflections:

$$\frac{\sigma}{\epsilon} = \frac{E}{(1-\nu)} = E', \tag{6.10}$$

The underlying reason for this discrepancy may be that the Nix model provides a better description of the material behavior, as it incorporates Poisson's ratio. This inclusion might make it more suited for materials like Kapton. There are also modifications or evolutions of the equation 6.1 [14] used in this thesis that also apply to circular windows.

Some of these evolutions for calculating stress, which are applicable in both the elastic and plastic regimes, are shown in the table below. They won't be discussed in detail here, but they do illustrate how various methods can differ as shown in a table in [16] .

Methods	Expressions
Young et al. (1981)	$\sigma_1 = \sigma_2 = \frac{p \cdot R^O}{2t}$ $\quad R^O = \frac{R_a^2 + h^2}{2h}$
Yoshida (2013)	$\sigma_1 = \sigma_2 = \frac{p(R^O - t)^2}{2t\left(R^O - \frac{t}{2}\right)}$
ISO 16808 (2014)	$\sigma_1 = \sigma_2 = \frac{p \cdot R^O}{2t}, R^O = 2/\left(\frac{1}{R_1^O} + \frac{1}{R_2^O}\right)$
Current work	$\sigma_1 = \frac{p \cdot R_2^O (R_1^O - t)(R_2^O - t)}{t\left(R_2^O - \frac{t}{2}\right)\left(R_1^O + R_2^O\right)}, \sigma_2 = \frac{p \cdot R_1^O (R_1^O - t)(R_2^O - t)}{t\left(R_1^O - \frac{t}{2}\right)\left(R_1^O + R_2^O\right)}$

Table 6.1.: "Summary of equations used to calculate stresses at the pole of bulge specimens. 'Current work' refers to the paper by Min et al. (2017) [16]. The Yoshida method is discussed in [17]."

7 Reproducibility of Measurement Results in Bulge Testing

7.1 Importance of Reproducibility in Bulge Testing

Reproducibility in bulge testing means consistently getting the same measurement results for the same type of material. Ensuring reproducibility is crucial for validating experimental data and making research conclusions reliable [18]. Some variation in results is normal as seen in Figure 7.1 , but significant deviations can indicate problems with how the experiment was conducted.

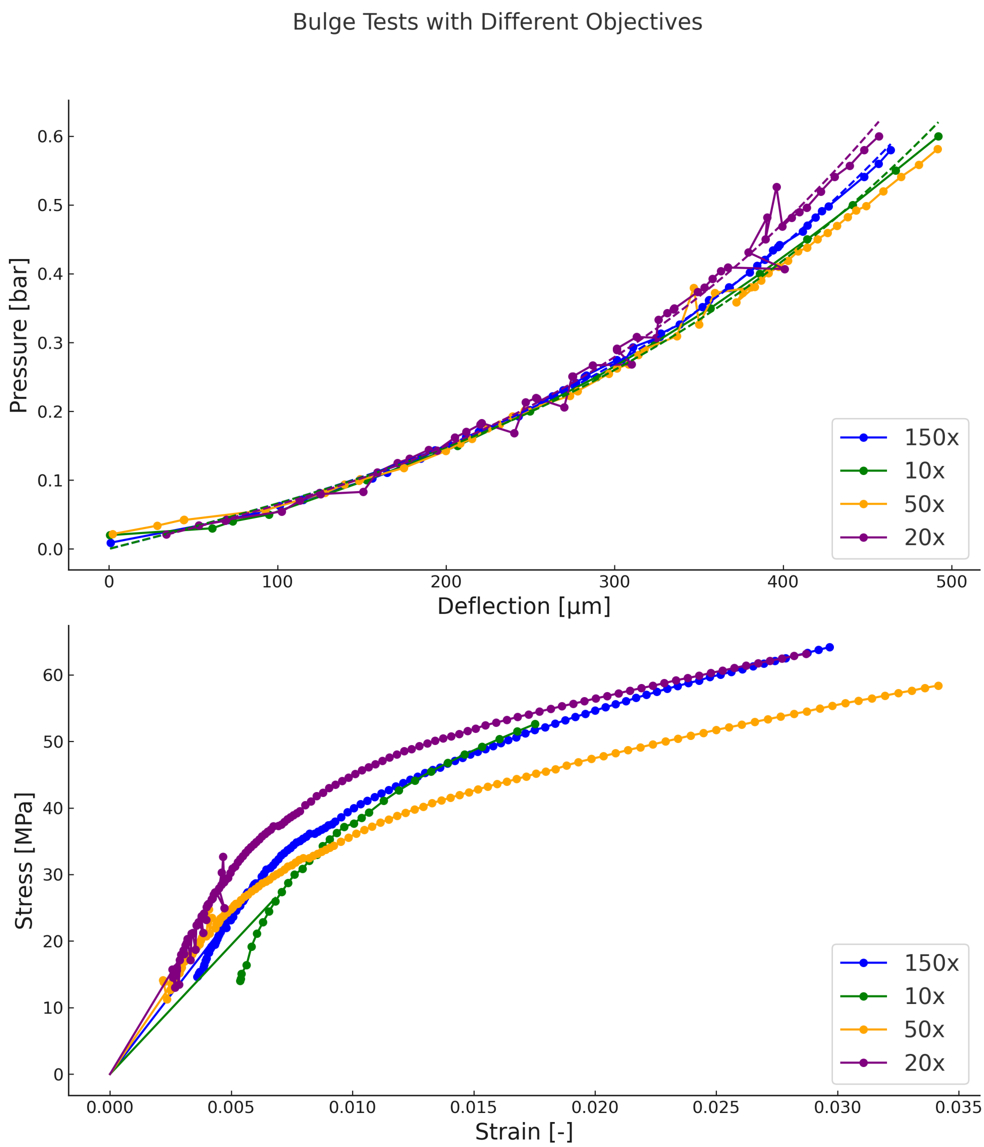


Figure 7.1.: Comparison of (a) pressure-deflection and (b) stress-strain curves for Kapton measured using different objectives. The experiment was conducted up to 3.5 bar, except for 10x, which was conducted up to 2 bar.

To ensure that measurement accuracy is independent of the chosen objective, bulge tests on Kapton were conducted using different objectives. Logically, the selection of the objective should not alter the material properties. The Young's modulus for all samples with the different objectives is, on average, 5001.75 MPa, ranging from 3885 MPa to 6048 MPa, with a corresponding Timoshenko E-modulus averaging 4113.01 MPa, ranging from 3740.85 MPa to 5250 MPa. However, some obvious deviations are noticeable, as shown in Table **??**. These deviations could arise due to slight differences in focus or resolution at different magnifications, but they do not indicate a change in the material's intrinsic properties.

Additionally, as discussed in the previous chapter 6, in some strain-stress curves, the corresponding slope of the pressure-deflection curves was not found. A direct fit of the strain-stress curve shows a higher Young's modulus compared to the modulus determined from the fitting curve. Since the strain-stress curve uses the full equations 6.1 and 6.9, without simplifying the higher-order deflections, it is believed that the Young's modulus directly extracted from the strain-stress curve is the most accurate.

Objective	**Strain-Stress Young's Modulus (MPa)**	**Nix Model Young's Modulus (MPa)**	**Timoshenko Model Young's Modulus (MPa)**
150x	4744	3581.31	4295.43
50x	5330	3349.38	4017.25
20x	6048	4380.00	5250.00
10x	3885	3118.93	3740.85
Average	$\mathbf{5001.75}\,^{+1046.25}_{-1116.75}$	$\mathbf{3429.93}\,^{+772.59}_{-488.48}$	$\mathbf{4113.01}\,^{+924.12}_{-585.03}$

Table 7.1.: Young's Modulus values calculated by suing hooke's law directly from Strain-Stress curve and retrieved from the pressure-deflection curves using Nix, and Timoshenko models, along with the average and range.

7.2 Factors Influencing Reproducibility in Bulge Testing

Several factors can affect the reproducibility of bulge test results [18] [19], including:

- **Variations in Sample Preparation:** Differences in how films are deposited or substrates are handled can cause variations in material properties.
- **Measurement System Variability:** Differences in the measurement setup, such as how precisely pressure is applied and the sensitivity of the detection system, can contribute to variability.
- **Environmental Conditions:** Changes in temperature, humidity, or other environmental factors during testing can impact the material's response.
- **Operator Influence:** Human factors, including how the sample is handled and the testing apparatus is operated, can introduce variability in the measurements.

The author has created a procedure in the appendix that must be strictly followed, as deviations can lead to varying results. Increased clamping pressure influences residual strain and bulge behavior.

8 Critique of the Setup

8.1 Design Limitations

The current lid design has a notable shortcoming: it doesn't allow for deflection measurements via cross-sectional imaging. This limitation, as discussed in Section 5.7, restricts the possibilty to measure higher deflection. Another negative point about the Design is the way samples are being placed. During a personal discussion with Professor Vlassak, he pointed out that clamping samples is not ideal. This method can introduce residual strain or cause the samples to wrinkle, which can compromise the results. To avoid these issues, Professor Vlassak's team used disposable fixtures with epoxy resin that are discarded after each experiment.

8.2 Measurement Constraints

Another significant issue with the current setup is the reliance on Mass-Flow Controllers (MFCs), which are not reliable below 0.7 bar, since overshootings can occur, as noted in Section 5.6 and shown in Figure 5.3. This overshooting leads to measurement errors and outliers, which undermine the accuracy of the results [20].

8.3 General Error Propagation

As every system, the bulge setup is not perfect. The uncertainties of the components were considered and are integrated in the code attached in the appendix of this master thesis. The detailed derivation of the uncertainties can be found in the Appendix B.

9 Validation of the Bulge Testing Setup Using Polymer Substrates and Soft Membranes

To validate the bulge setup and analysis procedure several known polymer materials were conducted to extract values for the Young's modulus and compared with the literature values or technical data sheets provided by the manufacturer. These validations are crucial to ensure the accuracy and reliability of the bulge testing setup before applying it to more complex material systems. The materials tested represent two extreme cases: one very stiff polymer, Kapton, and a much softer artificial skin material. Additionally, Fluorinated Ethylene Propylene (FEP), another well-known polymer, was included, although for the interpretation we need to consider the presecne of an Ag-Inconel layer, which influences the effective modulus. Kapton, a polyimide from DuPont, had a thickness of 50 micrometers. The artificial skin material was manufactured in-house by Empa Thun for another project by a collegue and had a thickness of 2200 micrometers. The thickness of the FEP sample was not directly measured but estimated to be around 50 micrometers.

9.1 Summary of Validation Results

The table below summarizes the measured Young's modulus and residual stress values for Kapton, artificial skin, and FEP. These results are compared to the corresponding literature values, with sources provided for reference.The pressure-deflection curves for Kapton, Inconel, and artificial skin material are shown in Figure 9.1.

Table 9.1.: Comparison of Measured and Literature Values for Various Materials

Material	Measured Young's Modulus	Literature Value	Source
Kapton	3.42 GPa_{Nix}, 4.11 $GPa_{Timoshenko}$	2.76 GPa	[21]
Artificial Skin	120 kPa_{Nix}, 144 $kPa_{Timoshenko}$	100-150 kPa	[22]
FEP	748 MPa_{Nix}, 897 $MPa_{Timoshenko}$	300-700 MPa	[23]

9.2 Discussion of Results

Kapton is a polyimide film with well-documented mechanical properties, making it an ideal candidate for validating the bulge testing setup. According to the manufacturer DuPont, Kapton has a Young's modulus of 2.76 GPa. The measured values from our tests were slightly higher, which could be due to differences in sample preparation, batch variability, or measurement techniques. Nonetheless, the results are within a reasonable range, indicating that the setup is functioning as expected.

The artificial skin material, fabricated at Empa, showed a Young's modulus of 120 kPa, which is in good accordance with the expected range of 100-150 kPa [22]. This consistency supports the accuracy of the bulge testing setup for softer materials.

The FEP sample, which was deposited with an Ag-Inconel layer, showed a measured Young's modulus of 748 - 897 MPa. According to the literature, FEP has a Young's modulus between 300-700 MPa [23]. The difference might be due to the Ag/Inconel coating.

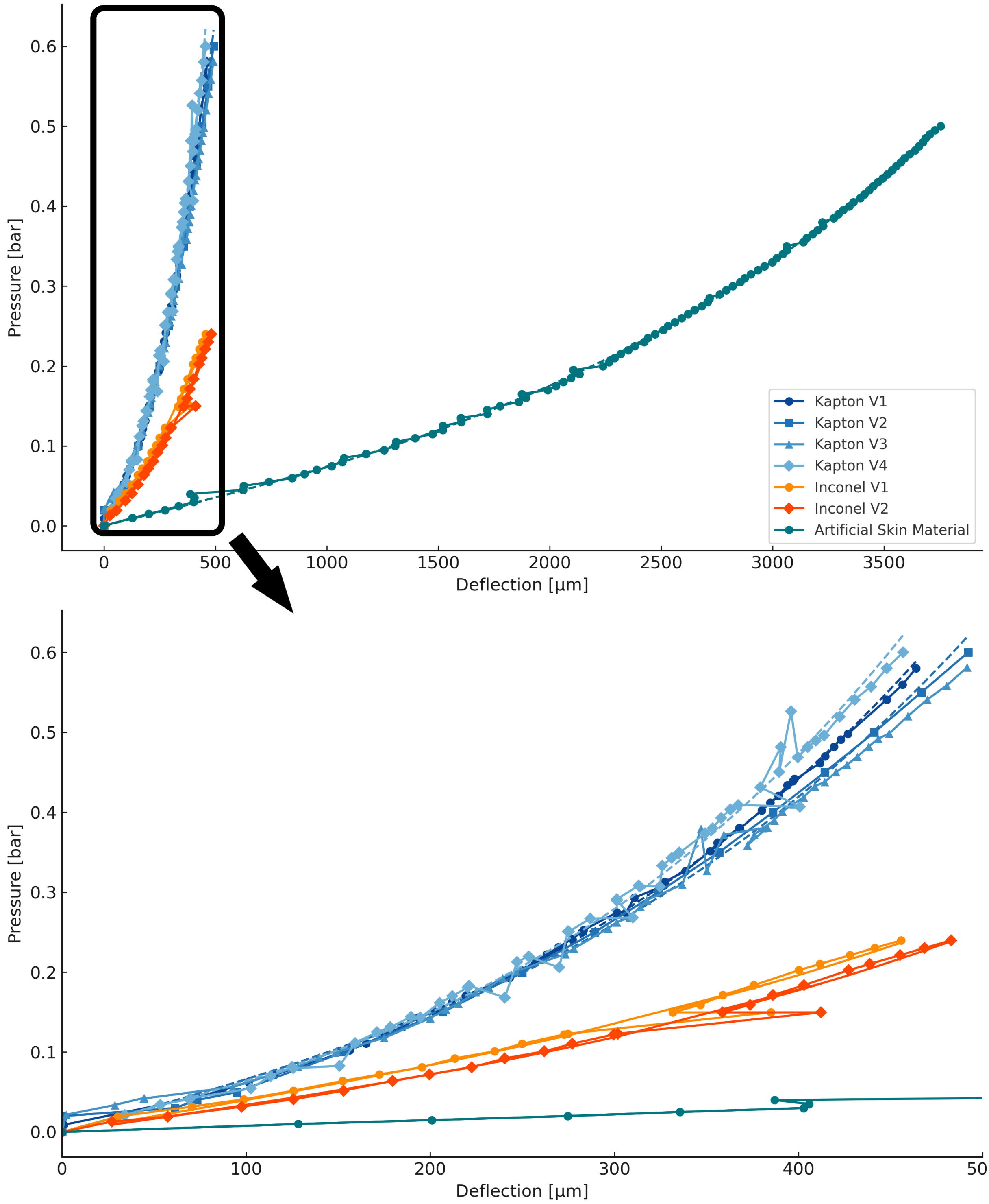


Figure 9.1.: Pressure vs. deflection curves for Kapton, Inconel, and artificial skin materials. The upper graph displays the overall pressure-deflection behavior for the materials, while the lower graph provides a magnified view of the deflection range up to 500 µm to highlight the differences in the behavior of the materials

As discussed in chapter 6, the fitting formula with a linear term and a cubic term used is: $p = a \cdot h + b \cdot h^3$. The coefficients of determination (R^2) for each curve, as shown in Table 9.2, demonstrate that the fits are highly accurate, with values consistently above 0.93. This indicates a very good agreement between the model and the experimental data.

Table 9.2.: Coefficients of Determination for Each Curve

Curve	R^2
Kapton V1	0.98
Kapton V2	0.95
Kapton V3	0.93
Kapton V4	0.94
Inconel V1	0.96
Inconel V2	0.97
Artificial Skin	0.99

9.3 Conclusion of Validation Tests

The validation tests for Kapton, artificial skin, and FEP demonstrate that the bulge testing setup is capable of producing reliable and consistent measurements of mechanical properties. The results confirm that the setup is suitable for further experimental work, especially for materials with known properties. This validation process provides the necessary confidence to apply the setup to more complex material systems in the subsequent chapters.

Part III.

Results and Discussion of Thin Films on Polymer Substrates

10 Effect of Aluminum Deposition on Kapton

The mechanical properties for three different samples were analyzed: a 50 micrometer Kapton substrate, Kapton with 10 nm aluminum deposition, and Kapton with 240 nm aluminum deposition. The stress-strain behavior and deflection data were examined to understand how aluminum deposition affects the mechanical properties of Kapton.

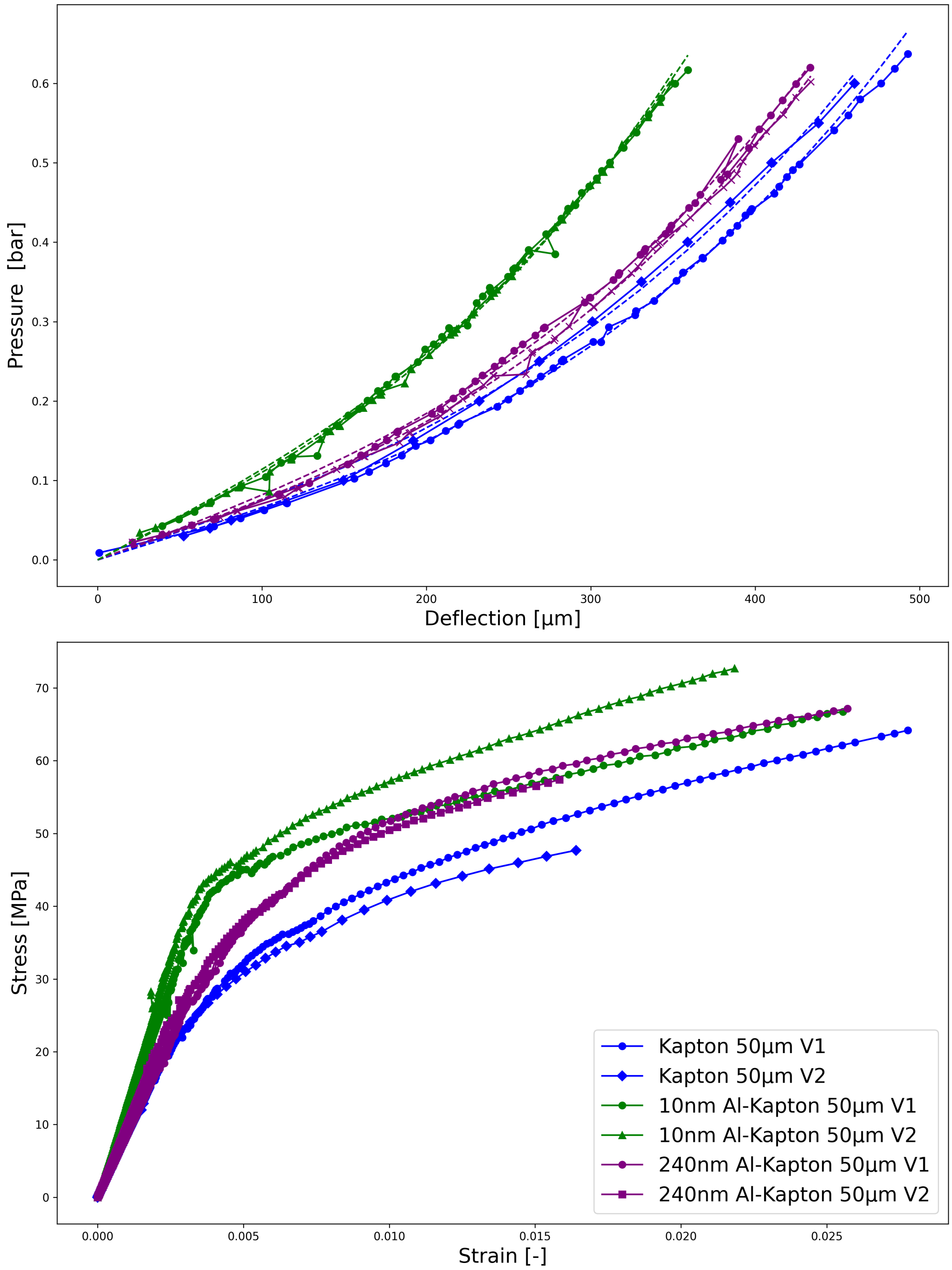


Figure 10.1.: Comparison of mechanical properties of Kapton samples with different aluminum depositions. (a) shows the pressure-deflection response, and (b) shows the stress-strain behavior.

10.1 Observations

Figure 10.1a provides a comparison of the pressure-deflection behavior and Figure 10.1b shows the stress-strain curves and of the three samples. Surprisingly, the sample with the 10 nm aluminum layer exhibits a higher effective modulus of elasticity and higher residual stresses compared to the sample with the 240 nm aluminum layer. This result is counterintuitive, as one might expect according to the rule of mixtures that a thicker aluminum layer would result in a higher stiffness. Possible explanations are discussed at the end of this chapter.

The data suggests that aluminum deposition has a significant effect on the mechanical properties of Kapton. However, the unexpected result where the 10 nm Al-Kapton sample shows less deflection at a given pressure than the 240 nm Al-Kapton samples and has a higher effective Young's modulus.

In table 10.1 and in table , the image on the left in each row capture the point where cracks first started to appear, which are highlighted in the strain-stress data shown in Figure 10.1. The images on the right display the final state of the samples after they were fully loaded and cracks had formed.

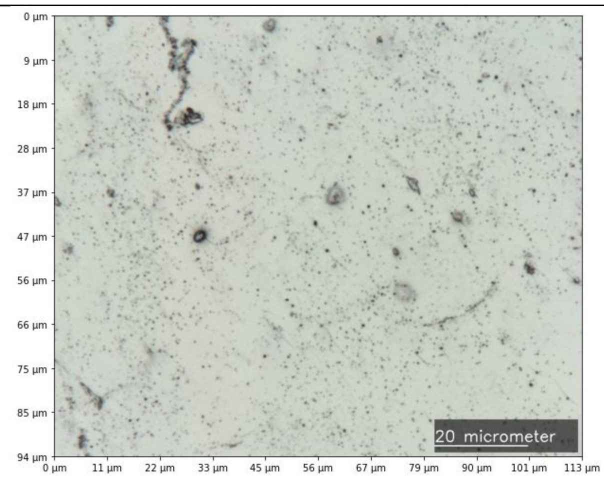
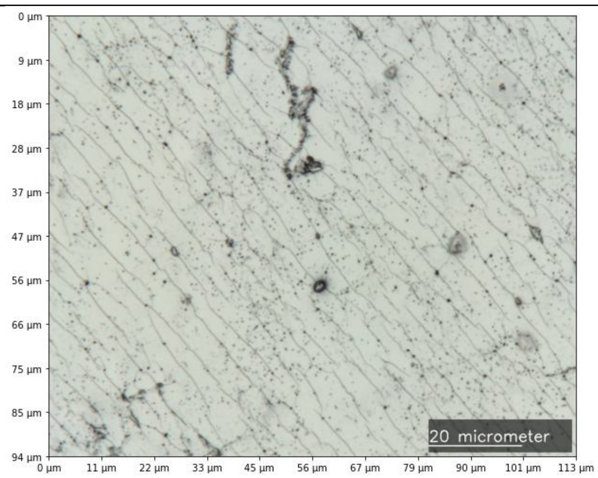
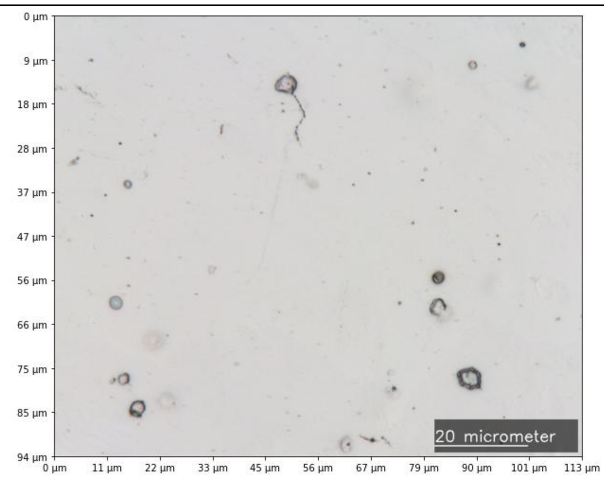
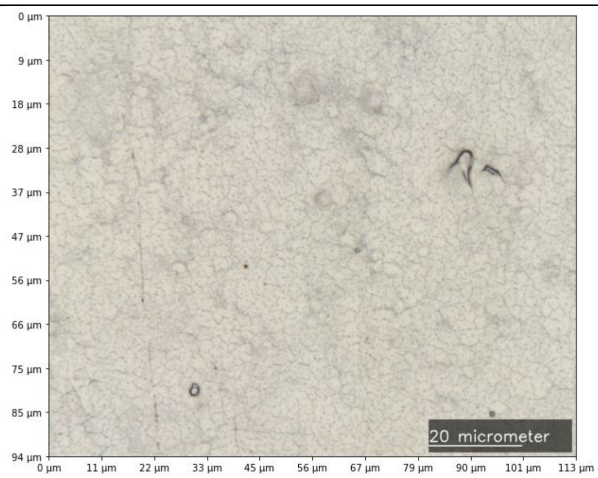
10nm Al-Kapton, 50μm V1		
10nm Al-Kapton, 50μm V2		

Table 10.1.: 10nm Al-Kapton, 50μm V1 and V2 sample images.

The visualization of the respective samples reveals an interesting observation. The differences between the 10nm and 240nm aluminum-coated Kapton samples are especially clear. They exhibit distinct crack patterns: the 10nm sample shows a very fine crack pattern, whereas the 240nm sample has a coarser one. There is a well-known relation between film thickness and crack spacing. Higher film thickness results in larger crack spacing, as shown in [24]. Perhaps the grain size and distribution are finer, and the cracking may be intergranular, occurring along the grain boundaries. An SEM analysis could provide further clarity.

Both show a similar type of primary cracks that differ from the primary cracks of the 240 nm-Al samples shown in 10.2. In the 10 nm samples, the Initiation of small crack points are observed, resembling a "pore opening," which are very finely distributed.

Interestingly, these two 10 nm-Al samples differ in their crack patterns. Typically, a biaxial crack pattern is expected for biaxial loading conditions, as shown in the image on page 15 in [25]. Even though a uniaxial crack pattern was observed in the 10 nm Al V1 sample, this is not typical for bulge testing, which generally induces biaxial loading. The presence of a uniaxial pattern may be due to local imperfections or anisotropy in the film, but this behavior requires further investigation. However, as shown in [25], certain effects and confinements due to reinforcement can lead to a uniaxial crack pattern, even under biaxial loading conditions.

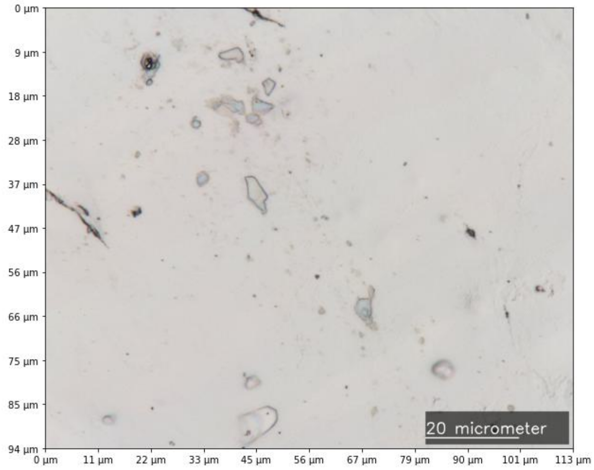
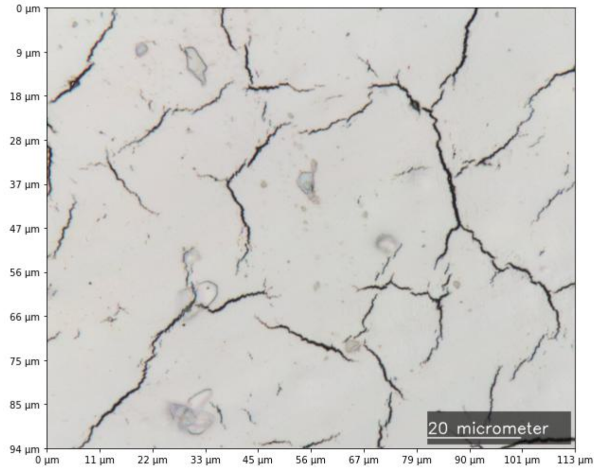
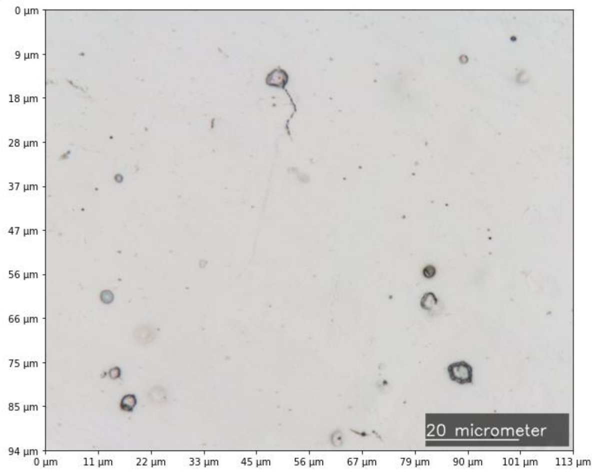
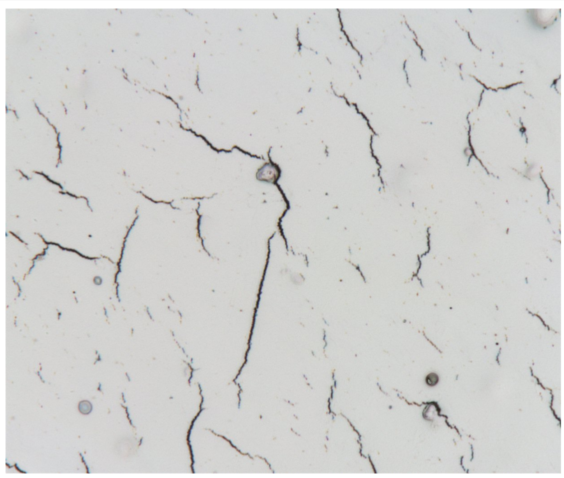
240nm Al-Kapton, 50μm V1		
240nm Al-Kapton, 50μm V2		

Table 10.2.: 240nm Al-Kapton, 50μm V1 and V2 sample images.

The corresponding values for the discussed samples are summarized in the following tables.

Table 10.3.: Young's Modulus and Residual Stress for Various Samples

Sample	E_{Nix} (MPa)	$E_{Timoshenko}$ (MPa)	(MPa)
Kapton 50µm V1	3581.31	4295.43	14.61
Kapton 50µm V2	3511.94	4212.22	11.9
240nm Al-Kapton 50µm V1	4650.73	5578.09	19.1
240nm Al-Kapton 50µm V2	4347.95	5214.93	17.8
10nm Al-Kapton 50µm V1	6384.06	7657.04	26.7
10nm Al-Kapton 50µm V2	7001.87	8398.05	25.6

Determining the crack-onset strain and stress by visual inspection can be challenging. Even at a magnification of 150x, which is relatively high, it's most likely that the initial microcracks form earlier than what can be observed.

Sample	Crack-Onset Strain ($\bar{x} \pm \Delta$)	Crack-Onset Stress ($\bar{x} \pm \Delta$)
10nm Al	1.29% ± 0.04%	53.25 ± 0.66
240nm Al	0.93% ± 0.11%*	49.44 ± 1.88

Table 10.4.: Average crack-onset and crack-onset stress values with range for 10nm and 240nm samples.

Interestingly enough there is a correlation (and perhaps a causality) between residual stresses and the determined effective Young's modulus. The 10nm samples have the highest, followed by 240nm samples and pure Kapton has 14.6

Sample	ϵ_{cr}	σ_{cr} (MPa)	Remarks
10nm Al V1	1.25%	53.9	Uniaxial crack pattern during loading
10nm Al V2	1.32%	52.59	Biaxial crack pattern as expected for bulge testing, however much finer distributed than in 240nm Al
240nm Al V1	1.03%	50.82	Coarse biaxael crack pattern [24]
240nm Al V2	0.82%	48.064	

Table 10.5.: Individual sample results for crack-onset strain (ϵ_{cr}) and crack-onset stress (σ_{cr}).

10.2 Possible Explanations for Higher Residual Stresses in Thinnes Films

The effective Young's modulus of a composite material can be estimated using the rule of mixtures [26], usually only valid if proportions are similar.

$$E_{\text{effective}} = \frac{E_{\text{Kapton}} \cdot t_{\text{Kapton}} + E_{\text{Al}} \cdot t_{\text{Al}}}{t_{\text{Kapton}} + t_{\text{Al}}} \tag{10.1}$$

where: - $E_{\text{Kapton}} = 4.2$ GPa (Young's modulus of Kapton), - $E_{\text{Al}} = 70$ GPa (Young's modulus of aluminum), - $t_{\text{Kapton}} = 50,000$ nm (thickness of the Kapton substrate), - $t_{\text{Al}} = 10$ nm (thickness of the aluminum film).

$$E_{\text{effective}} = \frac{4.2 \times 50000 + 70 \times 10}{50000 + 10} \approx 4.213 \text{ GPa}$$

According to this rule of mixture rule this calculation shows that the addition of a 10 nm aluminum layer should only slightly increase the effective Young's modulus of the Kapton substrate from 4.2 GPa to approximately 4.213 GPa, suggesting that the observed changes in mechanical behavior must have other contributing factors.

It is interesting to note that the residual stress for the 10nm Al-Kapton sample is higher than that of the 240nm Al-Kapton sample, as listed in Table 10.4. While thermal mismatch initially contributes to residual stress, it remains largely independent of film thickness beyond a certain threshold [27]. Instead, the key factor influencing thickness-dependent residual stresses is the density, not the distribution, of interface misfit dislocations. In thinner films, fewer dislocations tend to form because there isn't enough space to accommodate them, which means that the mismatch strain between the film and substrate isn't fully relieved. As a result, residual stresses are higher in thinner films. On the other hand, thicker films can support a higher density of dislocations, which allows more of the strain to be released, reducing the overall residual stress. So, it's this reduced dislocation density in thinner films that drives the increase in residual stresses, according to [27]. This aligns with the well-known principle that 'smaller is stronger,' where thinner films experience increased stress due to their reduced capacity for strain relief.

The fact that the 10nm Al-Kapton sample exhibits higher residual stress can be attributed to the significant role of dislocation mechanisms at such thin layers. Moridi et al. demonstrated that thinner films tend to have a lower density of interface dislocations, which are less effective in relieving stress buildup [27]. This suggests that the lower density of misfit dislocations in the 10nm sample likely plays a crucial role in the increased stress observed. So, it's this reduced dislocation density in thinner films that drives the increase in residual stresses, according to [27]. Further, regarding the deposition process, the 240nm layer takes much longer to deposit, causing the sample to heat up more. However, the samples were deposited at room temperature (RT) without intentional substrate heating, which may limit the full relaxation of stress in thicker films.

Additionally, for a film as thin as 10nm, surface and interface effects may dominate the stress response, further enhancing residual stress. In contrast, the thicker 240nm layer likely allows for stress relaxation mechanisms, such as grain boundary movement or dislocation formation, to occur more readily, thus reducing the overall residual stress. As Moridi et al. pointed out, the formation of dislocations becomes more pronounced with increasing thickness, which allows for greater stress relaxation and explains why the 240nm layer exhibits lower residual stress. Further [27] notes without explaining it in great detail, "that the residual stresses in a thinner film are much larger than those in a thicker film due to the effects of lattice defect."

The data suggests that aluminum deposition significantly affects the mechanical properties of Kapton. However, the unexpected finding that the 10 nm Al-Kapton sample shows a higher E-modulus than the 240 nm Al-Kapton. Variability in the base Kapton material seems unlikely, given that the substrates were sourced from the same roll and manufactured by the same producer. Additionally, the clear differences observed in crack formation and the range of residual stresses for each sample type argue against variability in the base material as the cause of this discrepancy. Specifically, the 10 nm Al-Kapton sample exhibits higher residual stresses than the 240 nm Al-Kapton sample, with the lowest residual stresses found in the pure Kapton. This pattern suggests that the differences in mechanical properties are more likely caused by the different film thickness rather than by any inconsistencies in the base material. The observed trend reinforces the notion that 'smaller is stronger,' where thinner films not only exhibit higher residual stress but also a higher modulus of elasticity.

11 Analysis of Discontinuities and Crack Formation in Ag/Inconel on FEP

One of the advantages of not only measuring deflection (e.g., with a point-based laser system) but also capturing images is the ability to study whether discontinuities can be observed in the pressure-deflection or strain-stress curves correspond and can be linked to to crack initiation and propagation in the thin film. The visual confirmation of cracks, coupled with the associated stress relief, would indicate that a measured discontinuity is indeed real and not noise or uncertainty in the measurement. Discontinuities in these curves might indicate the onset of crack formation. However, the system is not sensitive enough or, more precisely the pressure control is not stable enough, to observe discontinuities or label such with confidence. It appears to work only for the first crack initiation and substrate failure as shown in 11.3.

11.1 Observations in Ag/Inconel on FEP Samples

The samples investigated in this chapter are Ag/Inconel films on FEP. The Inconel layer has a thickness of 30 nm, while the Ag layer is 150 nm thick [24]. Initially, the Ag/Inconel film on FEP was crack-free but had some surface defects, as seen in Figure 11.3a. The plot on the top left of each subfigure in Fig. 11.3 shows the pressure-deflection curve. The bottom left plot has two axes: the left axis shows the increase in deflection Δh represented by bars, while the right axis shows the pressure increase ΔP in points. The sum of all the bars at any given point represents the total deflection up to that point. A bar is marked as a discontinuity when the **increase in deflection is larger than the previous one**, while the **increase in pressure is smaller or equal to the previous one**. These bars are marked as discontinuities, reflecting a deviation from the expected trend of the fitting curve in Equation 11.1. These conditions help to differentiate real discontinuities from regular deflection changes. Yellow bars indicate points where these conditions are met, indicating mechanisms like crack initiation or propagation.

However, even blue bars could represent discontinuities. This can occur after overshootings, where the deflection increase is significantly large, or when the pressure control system increases the pressure higher than it is set to be. In these cases, the subsequent bars may still show an atypically large increase in deflection, even though the pressure increment is smaller. This behavior can still indicate a discontinuity, as the deflection increment remains **disproportionately high relative to the pressure increase**. Therefore, it is essential to consider both yellow and certain blue bars when identifying potential crack initiation or propagation mechanisms. A flowchart illustrating the criteria for identifying discontinuities is shown below.

Currently, within the frame of this thesis no precise quantitative method has been developed to determine how atypical an increase is in comparison to the others. The assessment of discontinuities is based on a qualitative judgment of the patterns in the data. However, theoretically, a more refined approach could involve comparing each deflection increase to the predicted increase from the fitting function in Equation 11.1. This method could help evaluate whether an increase is typical or atypical in relation to the expected behavior, allowing for a more objective identification of discontinuities. Mathematically, for a function

$$P = ah + bh^3 \quad (11.1)$$

which is the fitting function discussed in Section 6.1, under a constant pressure increment ΔP, the deflection increment Δh should decrease steadily. Therefore, if the current deflection increment under a constant pressure increase is higher than the previous one, it indicates a discontinuity, suggesting an underlying mechanism such as crack initiation or propagation. Since the pressure increment is sometimes unstable, the corresponding pressure data is also plotted to account for any anomalies. The central image, captured using a microscope, shows the surface condition of the film, allowing direct visual comparison with the data. The plot on the right represents the strain-stress curve, with the red point marking the current data point, the blue points representing earlier measurements, and the grey points showing the following trajectory.

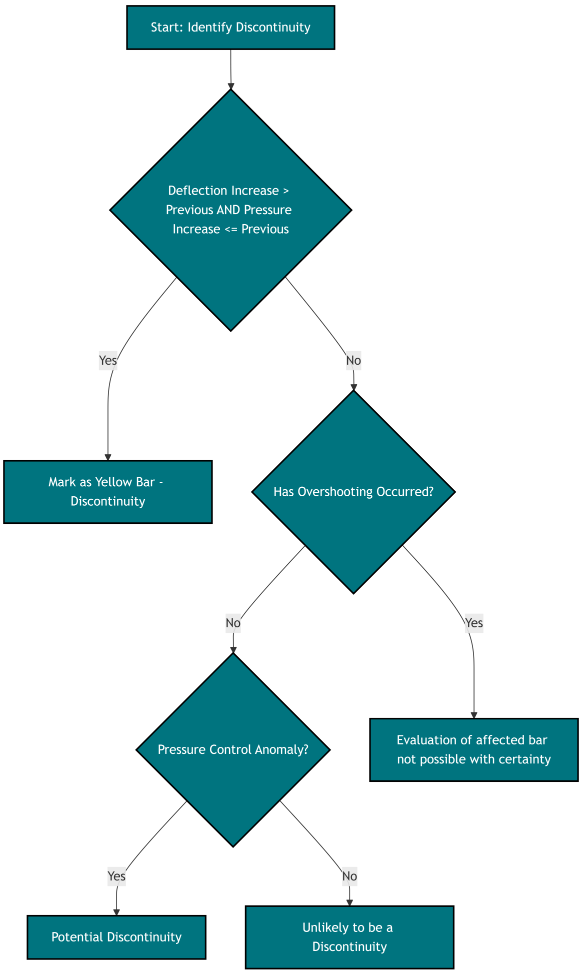


Figure 11.1.: Flowchart illustrating the criteria for identifying discontinuities in deflection and pressure data. The Yellow bars in in Fig. 11.3 indicate discontinuities when deflection increases while pressure remains constant or decreases. Blue bars are considered in cases of pressure control anomalies.

At a strain of 0.15%, the first crack initiation was observed, visible in Figure 11.3b. Since the residual stress was determined to be 10.08 MPa, we have to account for the residual strain ϵ_0, which was determined using the method described in section 6.2 and shown below in Figure 11.2 to be 1.051%. Accounting for residual strain ϵ_0, the first visible cracks observed with a magnification of 150x are at $\epsilon_{total} = \epsilon + \epsilon_0 = 0.15\% + 1.051\% = 1.201\%$. According to Putz et al. [24] the first primary cracks were observed at approximately 0.25% strain, and secondary cracks began to appear around 1% strain. So, there is a discrepancy between the values observed in this thesis and the values determined in [24].

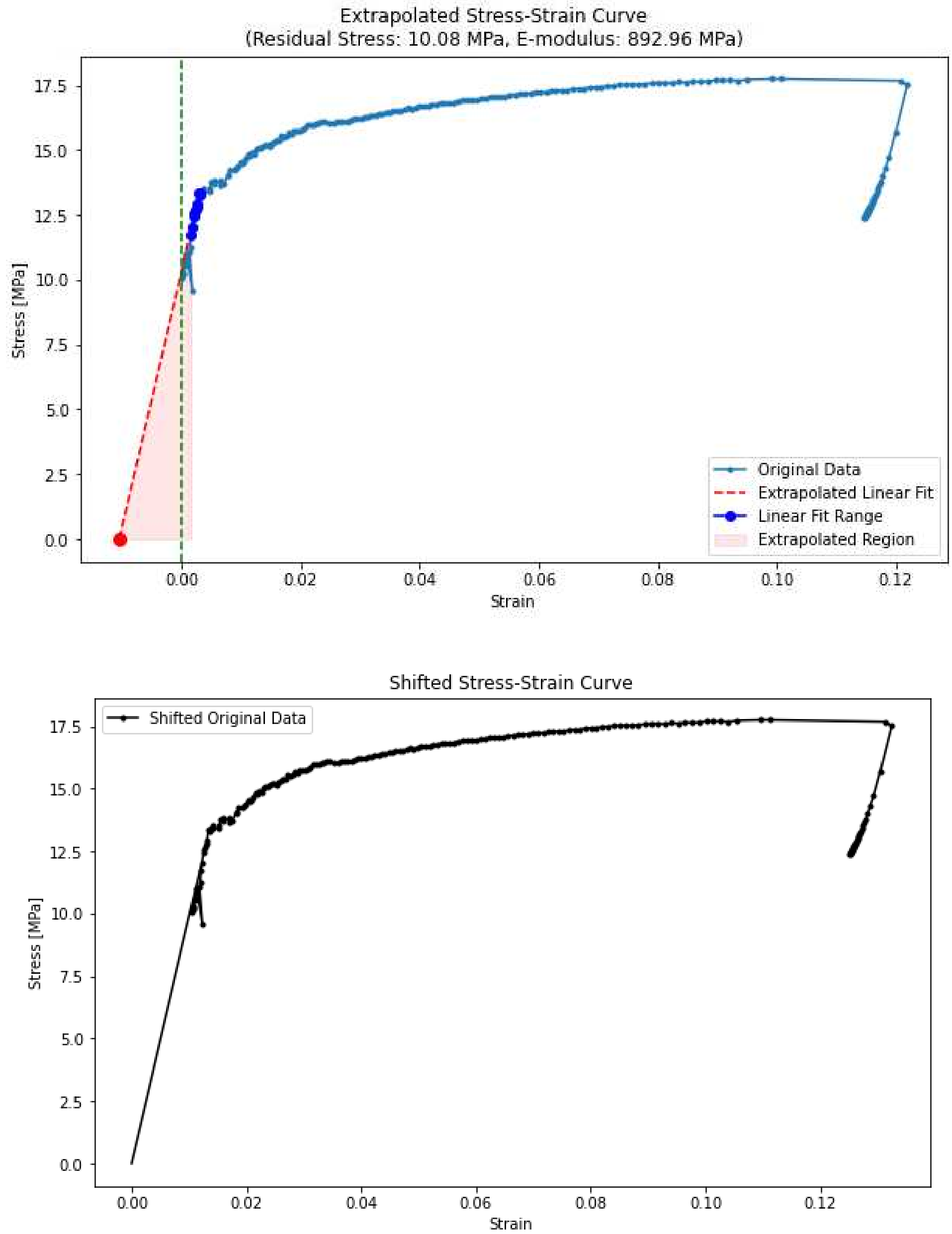
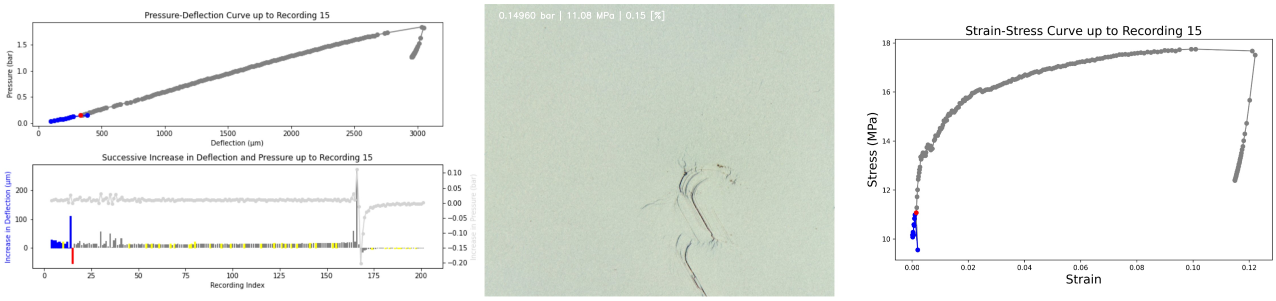


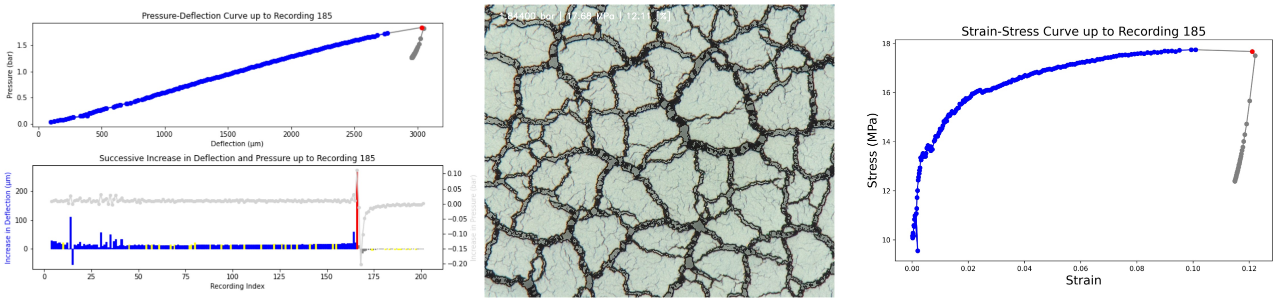
Figure 11.2.: (a) Pressure vs. strain plot showing the relationship between pressure and deformation. (b) Shifted stress-strain curve after accounting for initial strain ϵ_0. The samples consist of 30 nm Inconel and 150 nm Ag films on a 50 µm thick FEP substrate.

(a) Initial observation showing surface marks without cracks



(b) First crack initiation at at $\epsilon_{\text{total}} = \epsilon + \epsilon_0 = 0.15\% + 1.051\% = 1.201\%$

(c) Progression of crack formation showing a biaxial pattern $\epsilon_{\text{total}} = \epsilon + \epsilon_0 = 0.72\% + 1.051\% = 1.771\%$



(d) Point of material failure (rip) with visible stress relief $\epsilon_{\text{total}} = \epsilon + \epsilon_0 = 12.215\% + 1.051\% = 13.266\%$

(e) Crack pattern stays constant $\epsilon_{\text{total}} = \epsilon + \epsilon_0 = 11.484\% + 1.051\% = 12.535\%$

Figure 11.3.: Progression of crack formation in Ag/Inconel on FEP during bulge testing. Each subfigure shows the development from initial surface marks (a) to full crack propagation (e), with corresponding discontinuities in the pressure-deflection and strain-stress curves.

The defects on the surface act as stress concentrates, so it is not surprising that cracks begin there. Correspondingly, the first crack formation is observed at a defect, as seen in image in the middle of Figure 11.3b.

As pressure continued to increase, the crack grew, and additional cracks formed at other locations, showing the typical biaxial crack pattern as shown in the image on page 15 in [25], and as observed in Figures 11.3c and 11.3d. Finally, in the last stage (Figure 11.3e), a rupture of the polymer occurred. Since the air can now escape, no further pressure can build up. The actual pressure could not reach the target pressure anymore, leading to a sort of unloading curve. The crack pattern did not change further after this point. In Figure 11.3d, the rupture is not visible directly in the image. Even though the theoretical highest strain and stresses in a perfect sample should be exactly in the middle, failure of the substrate might occur slightly offset from the midpoint. The image in Figure 11.3d was made with a 150x magnification, and the rip was outside the field of view. An image showing the formation of a rip on another sample (20x magnification, wider field of view). is shown in the appendix.

11.2 Conclusion

By looking at jumps/discontinuities in pressure-deflection and strain-stress curves, along with visual images, one can get a clearer picture of how Ag/Inconel on FEP behaves during bulge testing. These discontinuities seem to be directly connected to cracks forming. This method might be useful for identifying when cracks start and understanding why thin films and coatings might fail.

11.3 Comparison of Discontinuities in Pure Kapton vs. Kapton Coated with 240nm Al

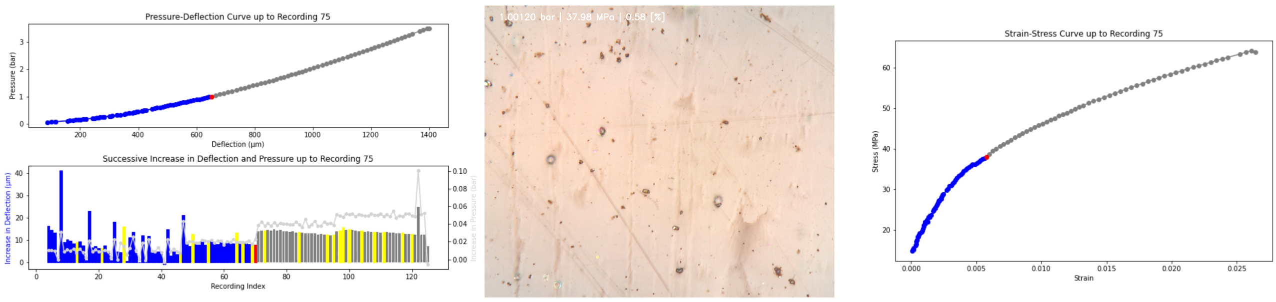
The analysis of discontinuities continues with a comparison between pure Kapton and Kapton coated with 240 nm of aluminum. As observed, pure Kapton exhibits fewer discontinuities in the same strain range, and some of these may indeed be attributed to measurement uncertainties. However, even in pure Kapton, there are some "jumps" in the pressure-deflection curve, which, frankly, should not be overinterpreted, as the cause is most likely due to measurement uncertainties rather than an underlying mechanism.

11.3.1 Pure Kapton

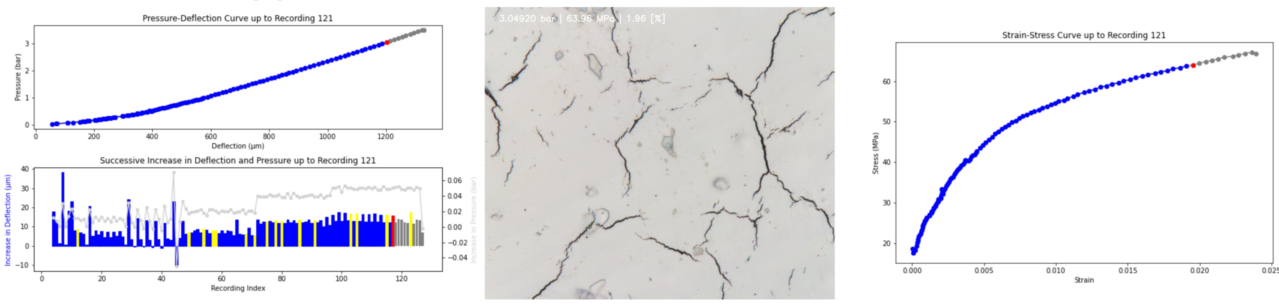
In the case of pure Kapton, as shown in Figure 11.4a, the pressure-deflection curve reveals a relatively smooth progression with a few minor discontinuities, which meet the conditions outlined at the beginning of this chapter to be classified as discontinuities in the context of this study. This approach, however, is somewhat simplistic. These minor jumps might be due to the material's intrinsic properties or minor experimental variations. The stress-strain curve similarly shows a gradual increase in deflection, with the increment of deflection gradually decreasing and minimal irregularities, even though the pressure is set to increase with the same increment.

11.3.2 Kapton Coated with 240nm Al

In contrast, the Kapton sample coated with 240nm of aluminum, as seen in Figure 11.4b, exhibits significantly more discontinuities. As discussed earlier, these discontinuities are indicative of crack formation within the coated layer. However, unlike the Ag/Inconel on FEP sample, the link between the observed discontinuities and the crack formations is less direct visible. It is more challenging to correlate specific discontinuities with crack initiation in the coated Kapton based on recorded images. Despite this, the overall trend is clear: the addition of a thin film like aluminum increases the likelihood and severity of discontinuities in the pressure-deflection curve, which correspond to the onset and progression of cracks.



(a) Pressure-deflection and strain-stress behavior of pure Kapton



(b) Pressure-deflection and strain-stress behavior of Kapton coated with 240nm Al

Figure 11.4.: Comparison of discontinuities and crack formation in pure Kapton (a) and Kapton coated with 240nm Al (b). The coated film exhibits significantly more discontinuities, indicative of crack formation in the Al film, although the correlation is more complex than in the Ag/Inconel on FEP case.

11.4 Conclusion

The comparison between pure Kapton and Kapton coated with 240nm Al highlights the potential of this method for crack detection. While pure Kapton shows relatively few discontinuities, the coated sample presents a complex pattern of jumps in the pressure-deflection curve, potentially corresponding to the onset and progression of cracks in the coating. Although the link between these discontinuities and crack formation is less direct than in the Ag/Inconel on FEP case, the overall trend confirms the coating deformation (cracking) in coating can influence the measured mechanical behavior of the sample during bulge testing.

Part IV.
Outlook and Conclusion

12 Outlook: Potential Applications of the Bulge Test Setup

The versatility of the Bulge Testing Setup offers possibilities for further research and applications. One promising area is fatigue testing. Since the setup includes both an inlet and an outlet valve, it is possible to control the pressure precisely, allowing for both loading and unloading experiments. These experiments could help verify whether the Young's modulus, determined through linear regression of the initial points in the linear range, is consistent with the unloading slope, as shown exemplarily in Figure 12.1c. One loading and unloading experiment was conducted with Ag/Inconel on FEP, but no comprehensive analyses or additional experiments related to fatigue testing were conducted. This represents an opportunity for future research.

12.1 Loading-Unloading

The consistency between the Young's modulus obtained from initial loading and the slope of the unloading curve can be validated through these experiments, offering a more comprehensive view of the material's elastic properties. Cracks present in the coating are known to change the slope of the unloading curve [28].

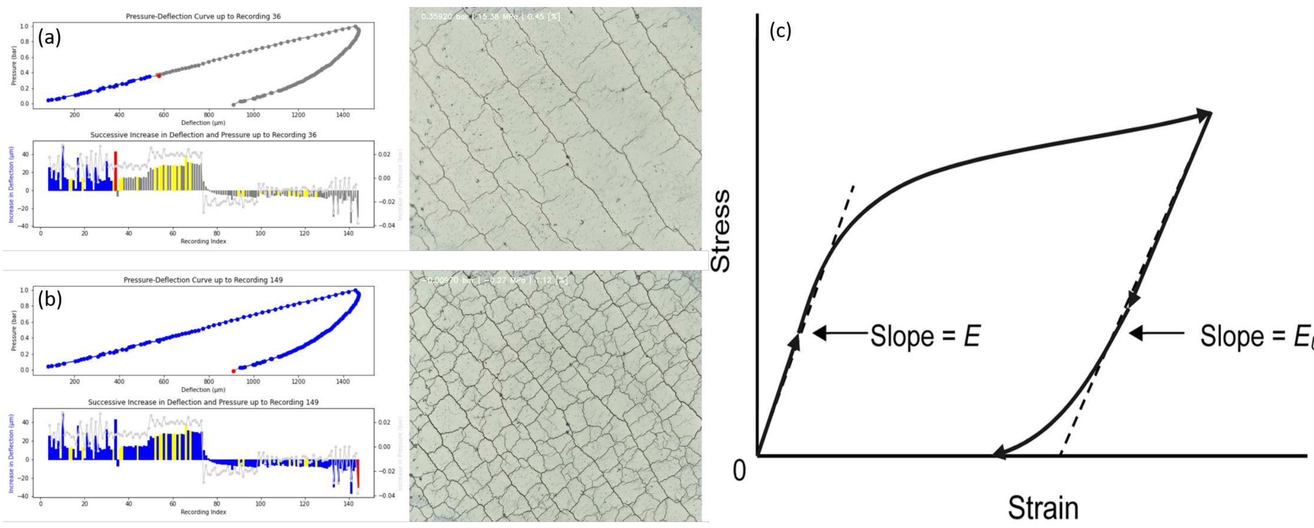
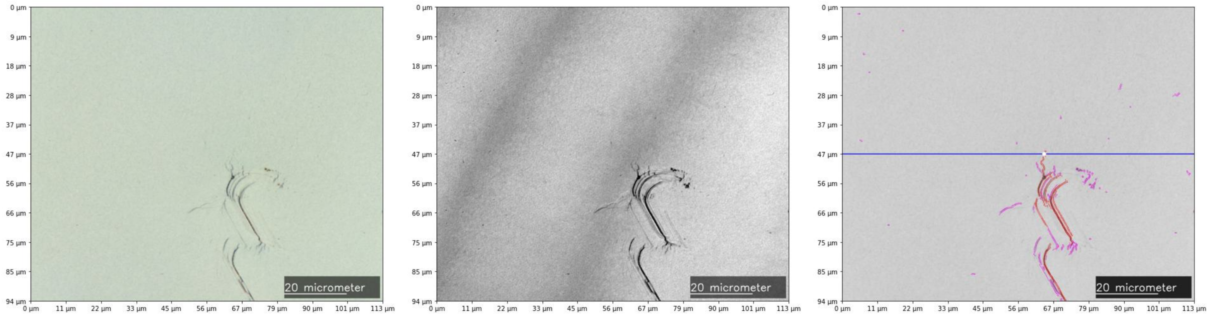


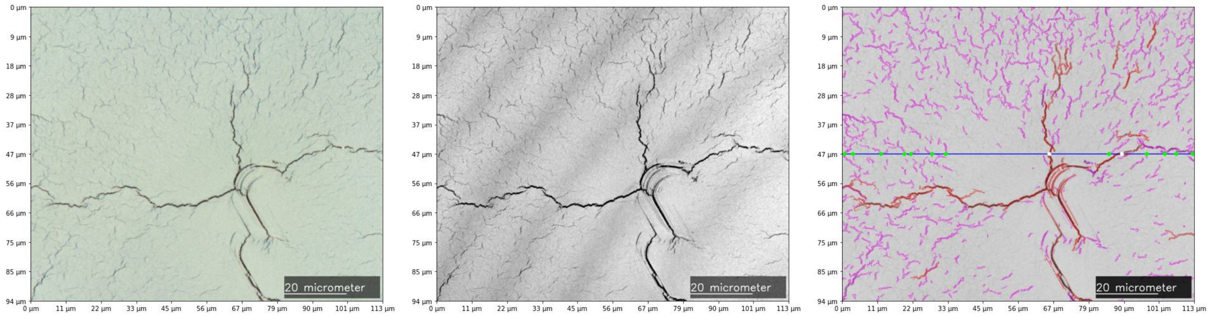
Figure 12.1.: Potential applications of the Bulge Testing Setup: (a) and (b) demonstrate loading-unloading experiments, while (c) presents a schematic illustrating the modulus of elasticity and unloading slope [29]

12.2 Outlook: Crack Formation and Pattern Analysis

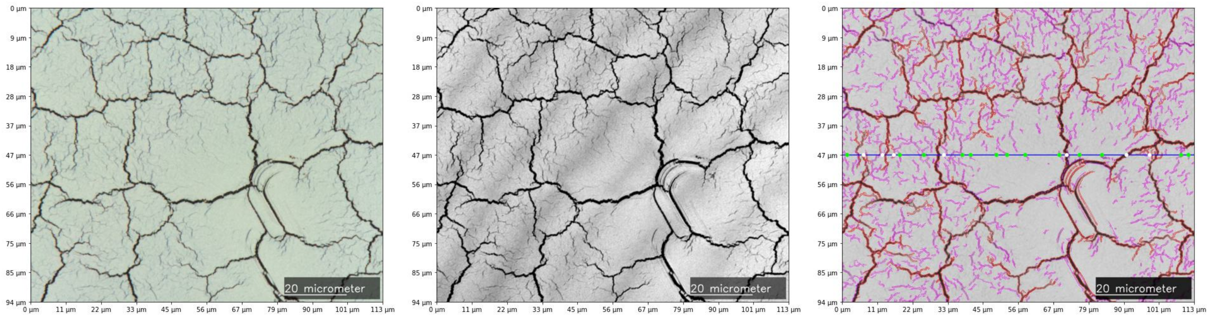
Another promising area of research involves studying crack formation and patterns under biaxial tension, to analogies fragmentation analysis with uniaxial tensile tests. A code has been developed to detect and classify crack formation into primary and secondary cracks, as illustrated in Figure 12.2. This two stage cracking mechanism has been reported in literature for uniaxial tension. Primary cracks form in the Inconel layer at 0.25% applied strain [24]. Upon further straining, secondary cracks form in the underlying Ag layer.



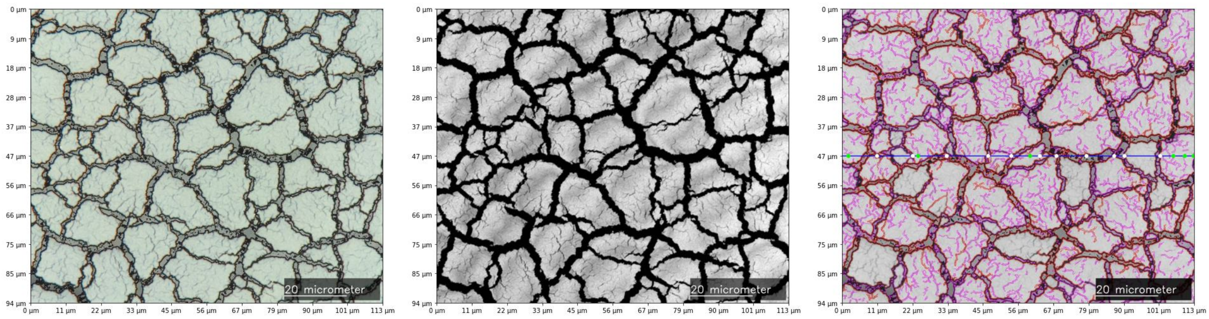
(a) 0.15 bar, 11.08 MPa, 0.15%



(b) 0.34 bar, 13.74 MPa, 0.52%



(c) 0.78 bar, 15.86 MPa, 2.09%



(d) 1.83 bar, 17.51 MPa, 12.21%

Figure 12.2.: Images taken at various pressure, stress, and strain conditions. Each row represents a set of images: (a) 0.15% ($\epsilon_{\text{total}} = \epsilon + \epsilon_0 = 0.15\% + 1.051\% = 1.201\%$); (b) 0.52% ($\epsilon_{\text{total}} = 0.52\% + 1.051\% = 1.571\%$); (c) 2.09% ($\epsilon_{\text{total}} = 2.09\% + 1.051\% = 3.141\%$); (d) 12.21% ($\epsilon_{\text{total}} = 12.21\% + 1.051\% = 13.261\%$). The second column shows the stacked images, and the third column shows the processed images. Green points indicate primary cracks, while white points indicate secondary cracks.

By using edge detection libraries in Python, it is possible to distinguish and count primary and secondary cracks. Since circular bulge testing involves biaxial loading stress, cracks propagate perpendicular to both directions, resulting in patterns that resemble islands. This contrasts with classical tension tests, where cracks propagate in a uniaxial manner.

12.3 Outlook: Detailed Crack Analysis with SensofarView Software

The SensofarView Software also offers advanced tools to study cracks in greater detail, if desired and useful. For instance, it allows for the examination of crack width in relation to the applied strain or stress, similar to the methodology used in [9]. For the presented Ag-Inconel system only secondary cracks can be detected by the software with certainty.

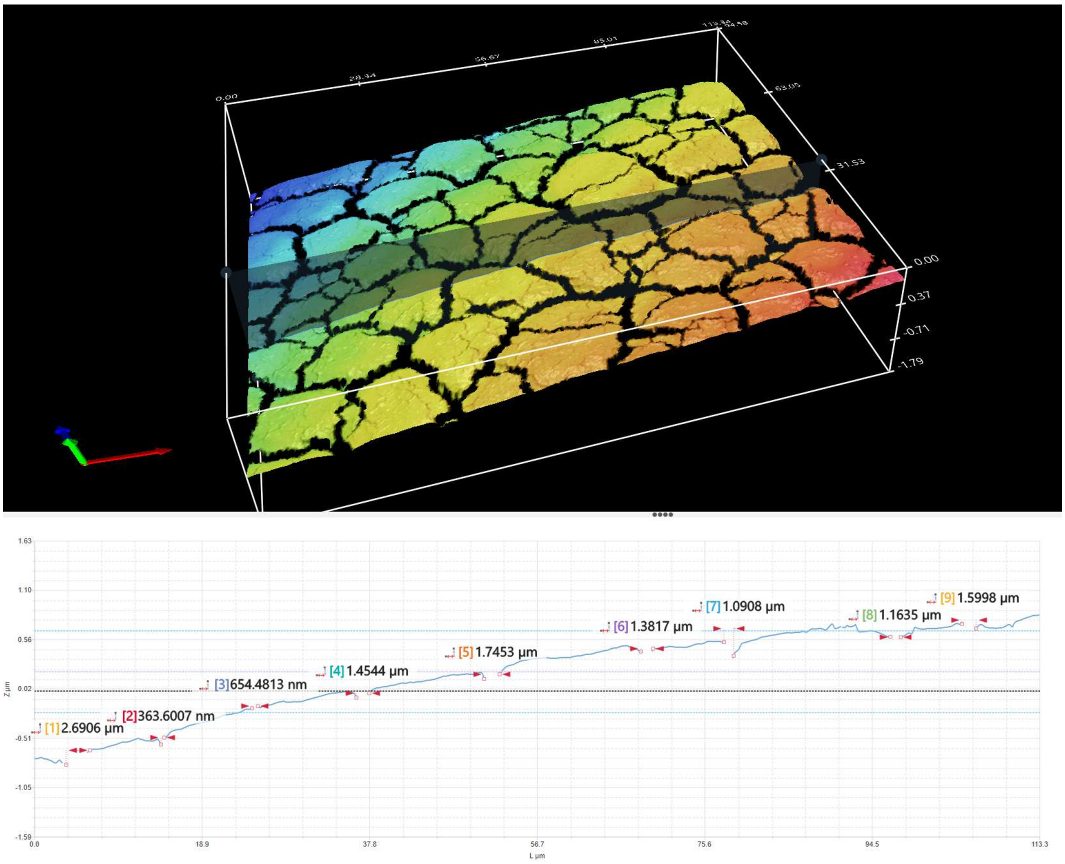


Figure 12.3.: (a) 3D view with cross-sectional plane and (b) corresponding profile analysis of crack width using SensofarView Software

12.4 Conclusion

In summary, the Bulge Testing Setup offers significant potential for advancing the understanding of material behavior under various conditions. The ability to precisely control loading and unloading opens up opportunities for in-depth analysis of material durability and elastic properties. Additionally, the setup allows for detailed study of crack patterns and provides tools for comprehensive crack analysis.

13 Summary

This thesis developed and implemented a fully automated bulge testing setup and methodology that allows for the detailed analysis of the mechanical properties of thin films, with a particular emphasis on polymers and thin metal films on polymer substrates. The setup provides consistent and reproducible results, making it a reliable tool for the study of material behavior under equi-biaxial loading conditions.

The research primarily examined two classes of materials: single-layer soft membranes and polymers with metallic coatings. For the determination of elastic properties, two models (Nix & Timoshenko) are contrasted and residual stresses are taken into account. The measurements of elastic modulus for the the artificial skin material aligned reasonably well with reported literature values. For polymers coated with thin films, even a very thin metallic coating (10nm Al) had a significant impact on the pressure-deflection curve, highlighting the effect of coating on substrate deformation.

Beyond mechanical characterization, the setup also enabled the observation and analysis of crack onset and propagation through recorded images. By integrating imaging with pressure-deflection and strain-stress curves crack initiation and growth under stress can be observed.

In conclusion, this work establishes a strong foundation for future research in material science, especially in the study of thin films and composite materials. The bulge testing setup developed and validated in this thesis proves to be a versatile and powerful tool for both academic research and industrial applications.

Bibliography

[1] F. Maseeh and S.D. Senturia. Viscoelasticity and creep recovery of polyimide thin films. *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, 13(1):94–100, 1990.

[2] Y. Xiang, T.Y. Tsui, J.J. Vlassak, and A.J. McKerrow. Measuring the elastic modulus and ultimate strength of low-k dielectric materials by means of the bulge test. In *Proceedings of IEEE 2004 International Interconnect Technology Conference*, pages 133–136, Piscataway, NJ, 2004.

[3] M.T. Perez-Prado and J.J. Vlassak. Microstructural evolution in electroplated cu thin films. *Scripta Materialia*, 47(8):817–822, 2002.

[4] Y. Xiang and J.J. Vlassak. Bauschinger effect in thin metal films. *Scripta Materialia*, 53(2):177–182, 2005.

[5] G.E. Dieter. *Mechanical Metallurgy*. McGraw-Hill, New York, 3 edition, 1986.

[6] Y. Xiang, X. Chen, and J.J. Vlassak. Plane-strain bulge test for thin films. *Journal of Materials Research*, 20(8):2360–2370, 2005.

[7] International Organization for Standardization. Iso 21844:2018 - polymeric materials — bulge test method, 2018. Accessed: 2024-09-06.

[8] Fadi Amt. Resources: Bulge testing. Available online: `https://fadi-amt.com/resources-bulge-testing.html`. Accessed: 2023-12-19.

[9] Xuying Xian. Development of a bulge test experimental setup for the in-situ mechanical characterization of metal thin films on polymer substrate. Master's thesis, Politecnico di Milano, Milan, Italy, 2016.

[10] Benoit Merle. *Mechanical Properties of Thin Films Studied by Bulge Testing*. Phd thesis, Technische Fakultät, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany, June 2013. Charakterisierung der mechanischen Eigenschaften dünner Membranen mittels Bulge-Experimenten.

[11] Sensofar. S neox optical 3d profilometer. Available online: `https://www.sensofar.com/de/metrology/industry-research/s-neox/`. Accessed: 2023-12-21.

[12] Martha K. Small and W. D. Nix. Analysis of the accuracy of the bulge test in determining the mechanical properties of thin films. *Department of Materials Science and Engineering, Stanford University*, Stanford, California 94305:1–15, 1992. Received 7 October 1991; accepted 18 February 1992.

[13] Ramin Ebrahimi and Faraz Rahimzadeh Lotfabad. A comprehensive mathematical analysis on achieving stress–strain behavior at large strains in bulge test. *Iranian Journal of Science and Technology, Transactions of Mechanical Engineering*, 46(3):901–909, 2022. Published online: 25 September 2021.

[14] R. F. Young, J. E. Bird, and J. L. Duncan. An automated hydraulic bulge tester. *Journal of Applied Metalworking*, 2(1):11–18, 1981.

[15] J. J. Vlassak and W. D. Nix. A new bulge test technique for the determination of young's modulus and poisson's ratio of thin films. *Journal of Materials Research*, 7(12):3242–3249, 1992.

[16] Jaewon Min, Thomas B. Stoughton, John E. Carsley, Bruce E. Carlson, Jian Lin, and Xiang Gao. Accurate characterization of biaxial stress-strain response of sheet metal from bulge testing. *International Journal of Plasticity*, 94:192–213, 2017.

[17] F. Yoshida, H. Hamasaki, and T. Uemori. Modeling of anisotropic hardening of sheet metals. In *AIP Conference Proceedings*, volume 1567, pages 1476–1481. AIP Publishing, 2013.

[18] Anthony J Hall. Gauge repeatability and reproducibility study for a bulge test fixture for corrugated boxes. *Senior Projects*, 2011. A senior project presented to the faculty of the Materials Engineering Department, California Polytechnic State University, San Luis Obispo.

[19] Anne L. Plant and Robert J. Hanisch. Reproducibility in science: A metrology perspective. *Harvard Data Science Review*, 2(4), Fall 2020. Published on: Dec 16, 2020.

[20] ASME. An experimental, analytical, and numerical study of the hydraulic bulge test for sheet metal forming. *Journal of Manufacturing Science and Engineering*, 139(3):031005, 2023.

[21] DuPont. Kapton hn polyimide film: Technical data sheet. `https://www.dupont.com/content/dam/dupont/amer/us/en/ei-transformation/public/documents/en/EI-10206-Kapton-HN-Data-Sheet.pdf`, n.d. Accessed: 2024-08-22.

[22] Anubha Kalra, A. Lowe, and A. Al-Jumaily. Mechanical behaviour of skin: A review. *Journal of Material Sciences & Engineering*, 5(1):1–7, 2016.

[23] Kelux Kunststoffe GmbH. Fep datenblatt. `https://www.kelux-kunststoffe.de/wp-content/uploads/FEP-Datenblatt.pdf`, n.d. Accessed: 2024-08-22.

[24] B. Putz, C. May-Miller, V. Matl, B. Völker, D.M. Többens, C. Semprimoschnig, and M.J. Cordill. Two-stage cracking of metallic bi-layers on polymer substrates under tension. *Scripta Materialia*, 145:5–8, 2018.

[25] 2011 Kreitman. Evaluation of concrete structures affected by alkali-silica reaction and delayed ettringite formation. `https://www.researchgate.net/figure/Crack-patterns-of-a-plain-concrete-and-b-reinforced-concrete-from-Kreitman-2011_fig3_271270924`. Accessed: 9 Sept 2024.

[26] Dryver R Huston, Wolfgang Sauter, Patricia S Bunta, and Brian Esser. Bulge testing of single and dual layer thin films. In *Proceedings of SPIE-The International Society for Optical Engineering*, volume 4407, pages 20–29. International Society for Optics and Photonics, 2001.

[27] Alireza Moridi, Haihui Ruan, Liangchi Zhang, and Mei Liu. Residual stresses in thin film systems: Effects of lattice mismatch, thermal mismatch and interface dislocations. *International Journal of Solids and Structures*, 50(24):3562–3569, 2013.

[28] Barbara Putz, Thomas E.J. Edwards, Emese Huszar, Laszlo Pethö, Patrice Kreiml, Megan J. Cordill, Dominique Thiaudiere, Stephane Chiroli, Fatih Zighem, Damien Faurie, Pierre-Olivier Renault, and Johann Michler. In situ fragmentation of al/al2o3 multilayers on flexible substrates in biaxial tension. *Materials & Design*, page 112081, 2023. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

[29] Joamin Gonzalez-Gutierrez and Martin Scanlon. *Rheology and Mechanical Properties of Fats*, pages 127–172. Academic Press, 2012.

[30] Nicolas Cloutier. Propagation of uncertainty calculator. https://nicoco007.github.io/Propagation-of-Uncertainty-Calculator/, 2023. Accessed: 2024-08-27.

[31] Gm50a metal sealed, digital mass flow controller. https://www.mksinst.com/mks/inst/downloads/gm50a.pdf, 2021.

[32] 3500 series oem pressure transmitter. https://www.gemssensors.com/gemssensors/media/files/documents/Gems_3500_eng_tds.pdf, 2015.

[33] S neox 3d optical profiler. https://www.sensofar.com/en/products/sneox/, 2020.

Part V.

Appendix

A Rupture

As mentioned, in Figure 11.3d, the rupture is not visible directly in the image. Even though the theoretical highest strain and stresses in a perfect sample should be exactly in the middle, failure of the substrate might occur slightly offset from the midpoint. The image in Figure 11.3d was made with a 150x magnification, and the rip was outside the field of view. The images below show the formation of a rip on another sample (20x magnification, wider field of view).

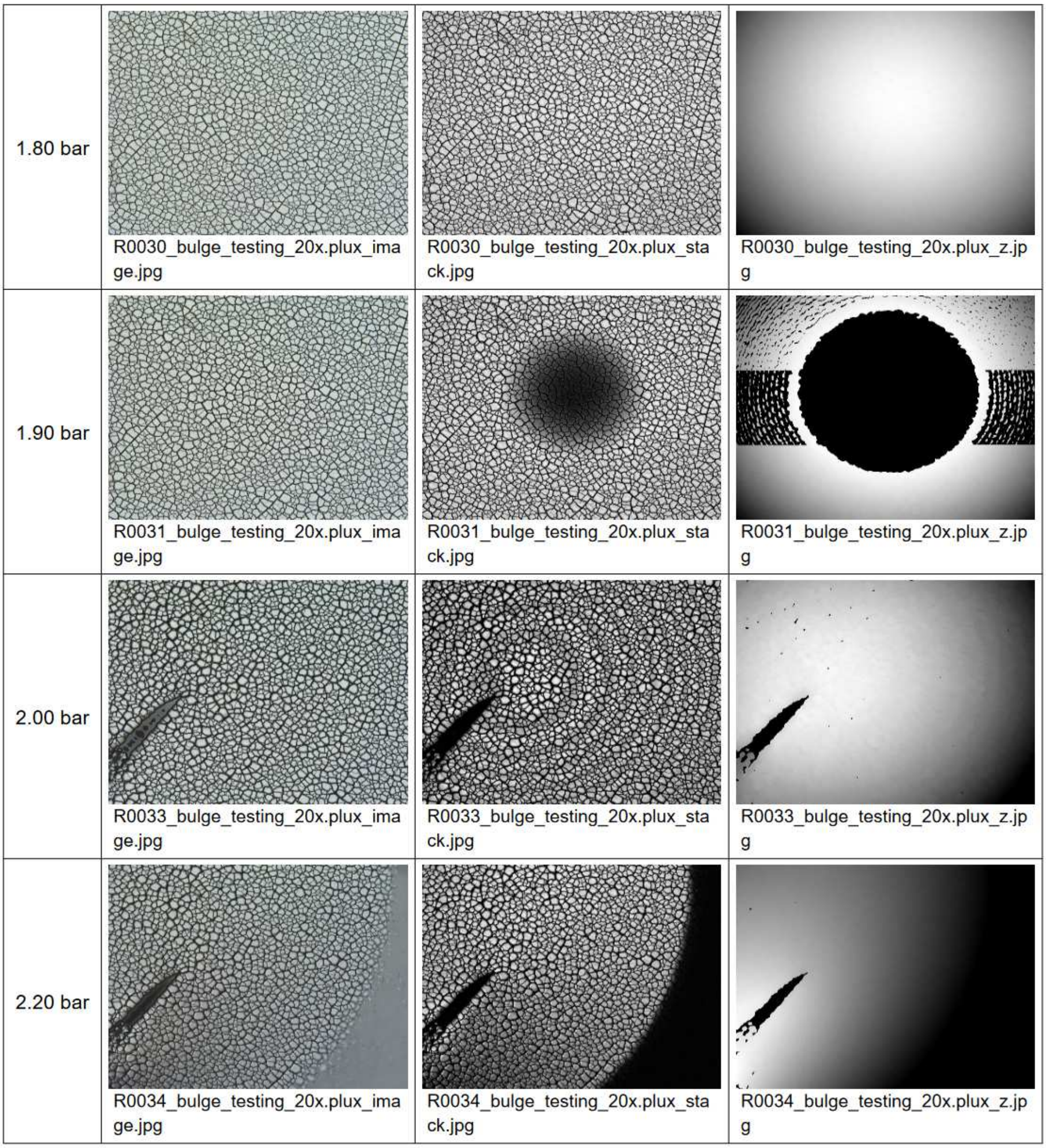


Figure A.1.: Formation of a rip captured with 20x magnification: Lucky hit

B General Error Propagation

Remark: This section was partly done by AI utilizing [30], instructed by the author!

As refered to in 8.3 the uncertainties associated with the measurements in this analysis are derived from three primary sources:

Table B.1.: Measurement Uncertainties and Sources

Measurement Source	Uncertainty Type	Value	Source
Mass Flow Controller (MFC) GM50A	Accuracy (20 to 100% FS)	±1% of reading	[31]
	Repeatability	±0.3% of reading	[31]
Pressure Sensor (3500 Series)	Accuracy	±0.25% full scale	[32]
	Zero Tolerance	±0.5% of span	[32]
	Span Tolerance	±0.5% of span	[32]
Height Measurement (Sensofar S Neox)	Step Height Accuracy	±0.5%	[33]
	Step Height Repeatability	±0.1%	[33]

The general formula for error propagation when dealing with multiple independent variables is:

$$\Delta y = \sqrt{\left(\frac{\partial y}{\partial x_1} \cdot u_{x_1}\right)^2 + \left(\frac{\partial y}{\partial x_2} \cdot u_{x_2}\right)^2 + \cdots} \tag{B.1}$$

This formula is widely used in uncertainty analysis and can be verified or calculated using tools such as the Propagation of Uncertainty Calculator [30].

Here:

- y is the calculated quantity (e.g. strain or stress)
- $x_1, x_2, \ldots$ are the independent variables that contribute to the uncertainty in y
- $u_{x_1}, u_{x_2}, \ldots$ are the uncertainties in those independent variables
- $\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots$ are the partial derivatives of y with respect to each independent variable

1. Strain (ϵ) Calculation:

The strain ϵ is calculated using the equation:

$$\epsilon = \frac{R \cdot \theta - a}{a} \tag{B.2}$$

2. Stress (σ) Calculation:

The stress σ is calculated using the equation:

$$\sigma = \frac{p \cdot R}{2 \cdot t} \tag{B.3}$$

B.0.1 Error Propagation for Strain and Stress

The uncertainty in ϵ (strain) and σ (stress) can be propagated as follows:

$$u_\epsilon = \sqrt{\left(\frac{\partial \epsilon}{\partial h} \cdot u_h\right)^2 + \left(\frac{\partial \epsilon}{\partial p} \cdot u_p\right)^2} \tag{B.4}$$

$$u_\sigma = \sqrt{\left(\frac{\partial \sigma}{\partial h} \cdot u_h\right)^2 + \left(\frac{\partial \sigma}{\partial p} \cdot u_p\right)^2} \tag{B.5}$$

Here:

- u_h is the uncertainty in deflection h.
- u_p is the uncertainty in pressure p.

Partial Derivatives:

- For Strain (ϵ):

$$\frac{\partial \epsilon}{\partial h} \quad \text{and} \quad \frac{\partial \epsilon}{\partial p}$$

- For Stress (σ):

$$\frac{\partial \sigma}{\partial h} \quad \text{and} \quad \frac{\partial \sigma}{\partial p}$$

$$\text{Strain:} \quad \epsilon = \frac{-a + \frac{\left(a^2+h^2\right)\operatorname{asin}\left(\frac{2ah}{a^2+h^2}\right)}{2h}}{a} \tag{B.6}$$

$$\text{Membrane Stress:} \quad \sigma = \frac{p\left(a^2+h^2\right)}{4ht} \tag{B.7}$$

$$\text{Derivative of Strain w.r.t h:} \quad \frac{\partial\epsilon}{\partial h} = \frac{\operatorname{asin}\left(\frac{2ah}{a^2+h^2}\right) + \frac{\left(a^2+h^2\right)\left(-\frac{4ah^2}{\left(a^2+h^2\right)^2}+\frac{2a}{a^2+h^2}\right)}{2h\sqrt{-\frac{4a^2h^2}{\left(a^2+h^2\right)^2}+1}} - \frac{\left(a^2+h^2\right)\operatorname{asin}\left(\frac{2ah}{a^2+h^2}\right)}{2h^2}}{a} \tag{B.8}$$

$$\text{Derivative of Strain w.r.t p:} \quad \frac{\partial\epsilon}{\partial p} = 0 \tag{B.9}$$

$$\text{Derivative of Membrane Stress w.r.t h:} \quad \frac{\partial\sigma}{\partial h} = \frac{p}{2t} - \frac{p\left(a^2+h^2\right)}{4h^2t} \tag{B.10}$$

$$\text{Derivative of Membrane Stress w.r.t p:} \quad \frac{\partial\sigma}{\partial p} = \frac{a^2+h^2}{4ht} \tag{B.11}$$

$$\text{Error Propagation for Strain:} \quad u_\epsilon = \sqrt{\frac{u_h^2\left(\operatorname{asin}\left(\frac{2ah}{a^2+h^2}\right) + \frac{\left(a^2+h^2\right)\left(-\frac{4ah^2}{\left(a^2+h^2\right)^2}+\frac{2a}{a^2+h^2}\right)}{2h\sqrt{-\frac{4a^2h^2}{\left(a^2+h^2\right)^2}+1}} - \frac{\left(a^2+h^2\right)\operatorname{asin}\left(\frac{2ah}{a^2+h^2}\right)}{2h^2}\right)^2}{a^2}} \tag{B.12}$$

$$\text{Error Propagation for Membrane Stress:} \quad u_\sigma = \sqrt{u_h^2\left(\frac{p}{2t} - \frac{p\,(a^2+h^2)}{4h^2t}\right)^2 + \frac{u_p^2\,(a^2+h^2)^2}{16h^2t^2}} \tag{B.13}$$

C Bulge Test Program: Python-Script

The following section presents the Python script developed to analyze the bulge test experiments. The script automates the process of extracting data, performing calculations, and generating reports, ensuring consistency and accuracy in the analysis of the bulge test results.

It is also important to note that AI, particularly ChatGPT, was used in the process of debugging, adding comments, and creating some lines of code in the script. AI contributed to improving the script's efficiency, though it's still far from lean and could be optimized further to achieve the same functionality with fewer lines of code. Additionally, it ensured the script is well-documented for easier use and future maintenance.

Listing C.1: Python script for Analysis of the Automated Bulge Test

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 27 14:41:24 2024

@author: alku
"""

import os
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np
import pandas as pd
from scipy.optimize import curve_fit
import datetime
#from zipfile import ZipFile

import re
import xlsxwriter
import shutil
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Image, Paragraph, Spacer
from reportlab.lib import colors

from reportlab.lib.units import inch
from reportlab.lib.pagesizes import letter
from reportlab.lib.styles import getSampleStyleSheet

import cv2
from moviepy.editor import ImageSequenceClip


import sympy as sp
from uncertainties import ufloat
from uncertainties.umath import asin

from sklearn.metrics import r2_score
import zipfile
from scipy.cluster.hierarchy import fclusterdata

from scipy.stats import linregress

from lxml import etree

```

```
def extract_and_save_z_values(source_directory, automatic_mode=True):
    z_values_dict = {}
    rxx_regex = re.compile(r'R(\d+)' if automatic_mode else r'^(\d+)_')

    for file in os.listdir(source_directory):
        if file.endswith('.plux'):
            match = rxx_regex.search(file)
            if match:
                r_number = int(match.group(1))
                plux_path = os.path.join(source_directory, file)
                tag_found = False

                with zipfile.ZipFile(plux_path, 'r') as zip_ref:
                    for extracted_file in zip_ref.namelist():
                        if extracted_file.endswith('.xml'):
                            with zip_ref.open(extracted_file) as xml_file:
                                xml_content = xml_file.read()

                                try:
                                    tree = etree.fromstring(xml_content)
                                    position_z_element = tree.find('.//POSITION_Z')
                                    if position_z_element is not None:
                                        z_value = float(position_z_element.text)
                                        z_values_dict[r_number] = z_value
                                        tag_found = True
                                        break # Exit loop once the tag is found
                                except etree.XMLSyntaxError as e:
                                    print(f"Error parsing XML in {file}: {e}")

                if not tag_found:
                    print(f"POSITION_Z tag not found in {file}")
                    z_values_dict[r_number] = 0

    max_r_number = max(z_values_dict.keys(), default=0)
    z_positions = [z_values_dict.get(i, 0) for i in range(1, max_r_number + 1)]
    missing_indices = [i for i, z in enumerate(z_positions, start=0) if z == 0]
    filtered_z_positions = [z for z in z_positions if z != 0]

    return filtered_z_positions, missing_indices



def parse_fix_entries(file_path):
    unique_bars = set()
    with open(file_path, 'r') as file:
        for line in file:
            if line.startswith('fix'):
                parts = line.split()
                bar_value = float(parts[2])
                unique_bars.add(bar_value)
    return sorted(unique_bars)

def fit_function(h, b, c):
    return b * h + c * h**3

def analyze_fitting(z_values, pressure_values, window_radius, output_directory, t, nu,
     max_bar_value_elastic):
    deflections = np.array([z - z_values[0] for z in z_values])
    pressures = np.array(pressure_values)
```

```python

    valid_indices = np.where(pressures <= max_bar_value_elastic)[0]
    deflections = deflections[valid_indices]
    pressures = pressures[valid_indices]

    params, _ = curve_fit(fit_function, deflections, pressures)
    a_fit, b_fit = params
    fitted_pressures = fit_function(deflections, *params)
    r_squared = r2_score(pressures, fitted_pressures)

    # Calculate sigma_0 and E_modul according to Nix
    sigma_0_manual_Nix = (a_fit * window_radius**2 / (4 * t)) * 0.1 # Convert to MPa
    E_modul_Nix = (3 * b_fit * window_radius**4 * (1 - nu) / (8 * t)) * 0.1 # Convert to MPa

    # Calculate sigma_0 and E_modul according to Timoshenko
    sigma_0_manual_Timoshenko = (a_fit * window_radius**2 / (4 * t)) * 0.1 # Convert to MPa
    E_modul_Timoshenko = (3 * b_fit * window_radius**4 * (1 - nu) / ((7 - nu) * t)) * 0.1 # Convert to MPa

    # Calculate mean values
    sigma_0_mean = (sigma_0_manual_Nix + sigma_0_manual_Timoshenko) / 2
    E_modul_mean = (E_modul_Nix + E_modul_Timoshenko) / 2

    base_path = output_directory
    os.makedirs(base_path, exist_ok=True)

    # Combined original and fitted pressures with deflections
    df_combined = pd.DataFrame({
        'Deflection (m)': deflections,
        'Original Pressure (bar)': pressures,
        'Fitted Pressure (bar)': fitted_pressures
    })
    combined_excel_path = os.path.join(base_path, 'combined_data.xlsx')
    df_combined.to_excel(combined_excel_path, index=False)

    # Save sigma_0 and E_modul along with a_fit and b_fit, and formula with parameters
    formula_with_params = f"P(h) = {a_fit:.4e} * h + {b_fit:.4e} * h^3"

    # Save sigma_0 and E_modul along with a_fit and b_fit
    df_params = pd.DataFrame({
        'a_fit': [a_fit],
        'b_fit': [b_fit],
        'sigma_0_Nix (MPa)': [sigma_0_manual_Nix],
        'E_modul_Nix (MPa)': [E_modul_Nix],
        'sigma_0_Timoshenko (MPa)': [sigma_0_manual_Timoshenko],
        'E_modul_Timoshenko (MPa)': [E_modul_Timoshenko],
        'sigma_0_mean (MPa)': [sigma_0_mean],
        'E_modul_mean (MPa)': [E_modul_mean],
        'Fitted Formula': [formula_with_params]
    })

    params_csv_path = os.path.join(base_path, 'fit_parameters.csv')
    params_excel_path = os.path.join(base_path, 'fit_parameters.xlsx')
    df_params.to_csv(params_csv_path, index=False)
    df_params.to_excel(params_excel_path, index=False)

    # Plotting with a finer grid for the fitted curve
    fine_deflections = np.linspace(deflections.min(), deflections.max(), 1000)
    fine_fitted_pressures = fit_function(fine_deflections, *params)

    plt.figure(figsize=(10, 6))
```

```python
    plt.scatter(deflections, pressures, label='Original Data', color='blue')
    plt.plot(fine_deflections, fine_fitted_pressures, label='Fitted Curve', color='red')
    plt.xlabel('Deflection (m)')
    plt.ylabel('Pressure (bar)')
    plt.title(f'Pressure vs. Deflection (R = {r_squared:.4f})\n'
              f'Nix:      = {sigma_0_manual_Nix:.2e} MPa, E = {E_modul_Nix:.2e} MPa; '
              f'Timoshenko:     = {sigma_0_manual_Timoshenko:.2e} MPa, E = {E_modul_Timoshenko:.2e} MPa\n'
              f'Fitted Equation: $P = a h + b h^3$; Parameters: t = {t} m,  = {nu}, a = {window_radius} m')

    # LaTeX formatted text for the function
    formula_text_nix = r"$P_{Nix} = \frac{{4\sigma_0 th}}{{a^2}} + \frac{{8E th^3}}{{3a^2(1-\nu)}}$"
    formula_text_timoshenko = r"$P_{Timoshenko} = \frac{{4\sigma_0 th}}{{a^2}} + \frac{{(7-\nu)E th^3}}{{3a^2(1-\nu)}}$"
    plt.text(0.95, 0.05, formula_text_nix + '\n' + formula_text_timoshenko,
             horizontalalignment='right', verticalalignment='bottom',
             transform=plt.gca().transAxes, fontsize=18,
             bbox=dict(facecolor='white', alpha=1, edgecolor='red', pad=10.0))

    plt.legend()
    plt.grid(False)
    plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
    plt.tight_layout()
    plot_path = os.path.join(base_path, 'Pressure_vs_Deflection.png')
    plt.savefig(plot_path)
    plt.show() # Display the plot to the user

    return sigma_0_mean, E_modul_Timoshenko


def find_slope_near_target(strain, stress, target_slope, tolerance_percent=5, max_points=20):
    num_points = min(len(strain), 50) # Limit to the first 50 points
    lower_tolerance_percent = tolerance_percent / 2

    for segment_length in range(max_points, 4, -1): # Start with max_points and decrease to 5 points
        for start_index in range(num_points - segment_length + 1):
            segment_strain = strain[start_index:start_index + segment_length]
            segment_stress = stress[start_index:start_index + segment_length]

            # Using numpy.polyfit for linear regression
            slope, intercept = np.polyfit(segment_strain, segment_stress, 1)

            lower_tolerance = target_slope * lower_tolerance_percent / 100
            upper_tolerance = target_slope * tolerance_percent / 100



            if slope > 0 and (target_slope - lower_tolerance) <= slope <= (target_slope + upper_tolerance):
                print(f"Found positive slope: {slope} with {segment_length} points")
                return start_index, slope

    return None, None




def analyze_dynamic_check(z_values, get_pressures, window_radius, output_directory, t, E_modul, sigma_0, missing_indices):
    a = float(window_radius)
    t = t

```

```
    current_time = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M")

    excel_path = os.path.join(output_directory, f'analysis_dynamic_{current_time}.xlsx')
    plots_directory = os.path.join(output_directory, 'Plots')
    if not os.path.exists(plots_directory):
        os.makedirs(plots_directory)

    plot_folder = os.path.join(plots_directory, 'Succesive_Plots')
    os.makedirs(plot_folder, exist_ok=True)

    plot_folder_p_vs_d = os.path.join(plots_directory, 'Succesive_Plots_p_vs_d')
    os.makedirs(plot_folder_p_vs_d, exist_ok=True)

    workbook = xlsxwriter.Workbook(excel_path)
    worksheet = workbook.add_worksheet('Results')

    headers = [
        'Filename', 'Z Position', 'Deflection (m)',
        'Get Pressure (bar)',
        'Strain', 'Membrane Stress (MPa)',
        'Von Mises Stress (MPa)', 'Difference (m)'
    ]

    for col, header in enumerate(headers):
        worksheet.write(0, col, header)

    pressures_unfitted = []
    deflections = []
    strains_unfitted = []
    von_mises_stresses_unfitted = []

    min_length = min(len(z_values), len(get_pressures))
    z_values = z_values[:min_length]
    get_pressures = get_pressures[:min_length]

    for index, z in enumerate(z_values):
        pressure_unfitted = get_pressures[index] * 0.1
        pressures_unfitted.append(pressure_unfitted)

        difference = z - z_values[0]
        deflections.append(difference)
        h = difference

        if h == 0:
            strain_unfitted = 0
            von_mises_stress_unfitted = sigma_0
        else:
            R = (a ** 2 + h ** 2) / (2 * h)
            theta = np.arcsin(a / R)
            strain_unfitted = (R * theta - a) / a

            membrane_stress_unfitted = (pressure_unfitted * R) / (2 * t)
            von_mises_stress_unfitted = membrane_stress_unfitted

        strains_unfitted.append(strain_unfitted)
        von_mises_stresses_unfitted.append(von_mises_stress_unfitted)

    # Filter out the first data point and any outliers. Data points are not being discarded but simply not
        considered for linear regression
    filtered_strain = np.array(strains_unfitted[2:])
```

```python
    filtered_stress = np.array(von_mises_stresses_unfitted[2:])

    # Original Part: Find the slope near the target using provided E_modul
    start_index, found_slope = find_slope_near_target(filtered_strain, filtered_stress, E_modul)

    if start_index is not None:
        target_stress = filtered_stress[0]

        adjusted_strain = [s - filtered_strain[0] for s in filtered_strain]
        positive_strain_indices = [i for i, s in enumerate(adjusted_strain) if s >= 0]
        positive_strain = [adjusted_strain[i] for i in positive_strain_indices]
        positive_stress = [filtered_stress[i] for i in positive_strain_indices]

        linear_range_start = start_index
        linear_range_end = linear_range_start + 5
        linear_strain_positive = positive_strain[linear_range_start:linear_range_end]
        linear_stress_positive = positive_stress[linear_range_start:linear_range_end]

        slope, intercept, _, _, _ = linregress(linear_strain_positive, linear_stress_positive)

        negative_strain_at_intersection = -intercept / slope

        extrapolated_strain = np.linspace(negative_strain_at_intersection, positive_strain[linear_range_start], 100)
        extrapolated_stress = slope * extrapolated_strain + intercept

        combined_strain = list(extrapolated_strain) + positive_strain
        combined_stress = list(extrapolated_stress) + positive_stress

        plt.figure(figsize=(10, 6))
        plt.plot(positive_strain, positive_stress, marker='o', linestyle='-', markersize=3, label='Original Data')
        plt.plot(extrapolated_strain, extrapolated_stress, color='red', linestyle='--', label='Extrapolated Linear Fit')
        plt.plot(linear_strain_positive, linear_stress_positive, marker='o', linestyle='-', color='blue', label='Linear Fit Range')
        plt.axvline(0, color='green', linestyle='--')
        plt.plot(negative_strain_at_intersection, 0, 'o', color='red', markersize=8)
        plt.fill_between(extrapolated_strain, extrapolated_stress, color='red', alpha=0.1, label='Extrapolated Region')

        removed_strains = np.array(adjusted_strain)[np.array(adjusted_strain) < 0]
        removed_stresses = np.array(filtered_stress)[np.array(adjusted_strain) < 0]
        if len(removed_strains) > 1:
            plt.plot(removed_strains, removed_stresses, 'o', color='grey', alpha=0.5, label='Removed Data Points')

        plt.title(f'Extrapolated Stress-Strain Curve\n(Residual Stress: {target_stress:.2f} MPa, E-modulus: {found_slope:.2f} MPa)')
        plt.xlabel('Strain')
        plt.ylabel('Stress [MPa]')
        plt.grid(False)
        plt.legend()
        plt.show()

        if len(removed_strains) > 1:
            removed_strains = removed_strains[1:]
            removed_stresses = removed_stresses[1:]
        print(f"Number of points removed: {len(removed_strains)}")
        for strain_value, stress_value in zip(removed_strains, removed_stresses):
```

```
        print(f"Removed strain: {strain_value}, Removed stress: {stress_value}")

    adjusted_strain_for_plot = [s - negative_strain_at_intersection for s in combined_strain]
    adjusted_extrapolated_strain = adjusted_strain_for_plot[:len(extrapolated_strain)]
    adjusted_original_strain = adjusted_strain_for_plot[len(extrapolated_strain):]
    adjusted_original_stress = combined_stress[len(extrapolated_strain):]

    plt.figure(figsize=(10, 6))
    plt.plot(adjusted_extrapolated_strain, extrapolated_stress, color='black', linestyle='-')
    plt.plot(adjusted_original_strain, adjusted_original_stress, marker='o', color='black', linestyle='-', markersize=3, label='Shifted Original Data')

    plt.title('Shifted Stress-Strain Curve')
    plt.xlabel('Strain')
    plt.ylabel('Stress [MPa]')
    plt.grid(False)
    plt.legend()
    plt.show()

    print(f"epsilon_0 is: {negative_strain_at_intersection:.6f}")
else:
    print("Could not find a range with the desired slope.")

# Additional Part: Iteratively expand the window for a direct fit within constraints
max_R2 = 0
best_fit_slope = None
best_fit_intercept = None
best_linear_strain = []
best_linear_stress = []

# Iterate over starting points within the first 10 points
for i in range(10):
    for j in range(i + 10, min(i + 40, len(filtered_strain))): # Ensure at least 10 points in the range, up to 40th point
        linear_range_strain = filtered_strain[i:j+1]
        linear_range_stress = filtered_stress[i:j+1]

        slope, intercept, r_value, _, _ = linregress(linear_range_strain, linear_range_stress)
        if r_value**2 > 0.95: # Use R^2 > 0.98 as the threshold
            if r_value**2 > max_R2:
                max_R2 = r_value**2
                best_fit_slope = slope
                best_fit_intercept = intercept
                best_linear_strain = linear_range_strain
                best_linear_stress = linear_range_stress
        else:
            break # Stop expanding when R^2 drops below 0.98

if best_fit_slope is not None:
    negative_strain_at_intersection_direct = -best_fit_intercept / best_fit_slope
    direct_fit_E_modul = best_fit_slope # The slope of the linear fit represents the E-modul

    # Extend the extrapolation
    extrapolated_strain_direct = np.linspace(negative_strain_at_intersection_direct, best_linear_strain[0], 100)
    extrapolated_stress_direct = best_fit_slope * extrapolated_strain_direct + best_fit_intercept

    combined_strain_direct = list(extrapolated_strain_direct) + list(filtered_strain)
    combined_stress_direct = list(extrapolated_stress_direct) + list(filtered_stress)

```

```python
    plt.figure(figsize=(10, 6))
    plt.plot(filtered_strain, filtered_stress, marker='o', linestyle='-', markersize=3, label='Original Data')
    plt.plot(extrapolated_strain_direct, extrapolated_stress_direct, color='purple', linestyle='--', label='Direct Extrapolated Linear Fit')
    plt.plot(best_linear_strain, best_linear_stress, marker='o', linestyle='-', color='orange', label='Best Linear Fit Range')
    plt.axvline(0, color='green', linestyle='--')
    plt.plot(negative_strain_at_intersection_direct, 0, 'o', color='purple', markersize=8)
    plt.fill_between(extrapolated_strain_direct, extrapolated_stress_direct, color='purple', alpha=0.1, label='Direct Extrapolated Region')

    plt.title(f'Direct Extrapolated Stress-Strain Curve (E-modul: {direct_fit_E_modul:.2f} MPa)')
    plt.xlabel('Strain')
    plt.ylabel('Stress [MPa]')
    plt.grid(False)
    plt.legend()
    plt.show()

    adjusted_strain_for_plot_direct = [s - negative_strain_at_intersection_direct for s in combined_strain_direct]
    adjusted_extrapolated_strain_direct = adjusted_strain_for_plot_direct[:len(extrapolated_strain_direct)]
    adjusted_original_strain_direct = adjusted_strain_for_plot_direct[len(extrapolated_strain_direct):]
    adjusted_original_stress_direct = combined_stress_direct[len(extrapolated_stress_direct):]

    plt.figure(figsize=(10, 6))
    plt.plot(adjusted_extrapolated_strain_direct, extrapolated_stress_direct, color='black', linestyle='-')
    plt.plot(adjusted_original_strain_direct, adjusted_original_stress_direct, marker='o', color='black', linestyle='-', markersize=3, label='Shifted Original Data (Direct Fit)')

    plt.title(f'Shifted Stress-Strain Curve (Direct Fit, E-modul: {direct_fit_E_modul:.2f} MPa)')
    plt.xlabel('Strain')
    plt.ylabel('Stress [MPa]')
    plt.grid(False)
    plt.legend()
    plt.show()

    print(f"Direct fit epsilon_0 is: {negative_strain_at_intersection_direct:.6f}")


    # Create subplots with shared x-axis
    # Create subplots with shared x-axis
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12), sharex=True)

    # Plot on ax1: Shifted curve for the original method with Steigungsdreieck
    ax1.plot(adjusted_extrapolated_strain, extrapolated_stress, color='red', linestyle='-')
    ax1.plot(adjusted_original_strain, adjusted_original_stress, marker='o', color='red', linestyle='-', markersize=3)

    # Calculate and plot the Steigungsdreieck for the original method
    sigma_orig = extrapolated_stress[-1] - extrapolated_stress[0]
    epsilon_orig = adjusted_extrapolated_strain[-1] - adjusted_extrapolated_strain[0]

    # Draw the Steigungsdreieck (only area below the slope)
    ax1.plot([adjusted_extrapolated_strain[0], adjusted_extrapolated_strain[-1]],
             [extrapolated_stress[0], extrapolated_stress[-1]], 'k--', label="Steigungsdreieck")
    ax1.vlines(adjusted_extrapolated_strain[-1], 0, extrapolated_stress[-1], colors='k', linestyle='dotted')
```

```python
    ax1.fill_betweenx([0, extrapolated_stress[-1]], adjusted_extrapolated_strain[0],
        adjusted_extrapolated_strain[-1], color='red', alpha=0.1)

    # Positioning the annotation to the right of the vertical line and centered vertically
    midpoint_x_orig = adjusted_extrapolated_strain[-1]
    midpoint_y_orig = (extrapolated_stress[0] + extrapolated_stress[-1]) / 2
    ax1.text(midpoint_x_orig + 0.0002, midpoint_y_orig,
            f"$E_{{orig}} = \\frac{{\Delta\\sigma}}{{\Delta\\epsilon}} \\approx {found_slope:.2f} \\,
                MPa$",
            color='red', fontsize=12, verticalalignment='center')

    ax1.set_title(f'E-modul taken from pressure-deflection curve = {found_slope:.2f} MPa')
    ax1.set_ylabel('Stress [MPa]')
    ax1.legend()
    ax1.grid(False)

    # Plot on ax2: Shifted curve for the direct fit method with Steigungsdreieck
    ax2.plot(adjusted_extrapolated_strain_direct, extrapolated_stress_direct, color='blue',
        linestyle='-')
    ax2.plot(adjusted_original_strain_direct, adjusted_original_stress_direct, marker='o', color='blue',
        linestyle='-', markersize=3)

    # Calculate and plot the Steigungsdreieck for the direct fit method
    sigma_direct = extrapolated_stress_direct[-1] - extrapolated_stress_direct[0]
    epsilon_direct = adjusted_extrapolated_strain_direct[-1] - adjusted_extrapolated_strain_direct[0]

    # Draw the Steigungsdreieck (only area below the slope)
    ax2.plot([adjusted_extrapolated_strain_direct[0], adjusted_extrapolated_strain_direct[-1]],
            [extrapolated_stress_direct[0], extrapolated_stress_direct[-1]], 'k--',
                label="Steigungsdreieck (Direct Fit)")
    ax2.vlines(adjusted_extrapolated_strain_direct[-1], 0, extrapolated_stress_direct[-1], colors='k',
        linestyle='dotted')
    ax2.fill_betweenx([0, extrapolated_stress_direct[-1]], adjusted_extrapolated_strain_direct[0],
        adjusted_extrapolated_strain_direct[-1], color='blue', alpha=0.1)

    # Positioning the annotation to the right of the vertical line and centered vertically
    midpoint_x_direct = adjusted_extrapolated_strain_direct[-1]
    midpoint_y_direct = (extrapolated_stress_direct[0] + extrapolated_stress_direct[-1]) / 2
    ax2.text(midpoint_x_direct + 0.0002, midpoint_y_direct,
            f"$E_{{direct}} = \\frac{{\Delta\\sigma}}{{\Delta\\epsilon}} = \\frac{{{sigma_direct:.2f}
                \\, MPa}}{{{epsilon_direct:.4f}}} \\approx {direct_fit_E_modul:.2f} \\, MPa$",
            color='blue', fontsize=12, verticalalignment='center')

    ax2.set_title(f'Direct Fit Method: E-modul = {direct_fit_E_modul:.2f} MPa')
    ax2.set_xlabel('Strain')
    ax2.set_ylabel('Stress [MPa]')
    ax2.legend()
    ax2.grid(False)

    plt.tight_layout()
    plt.show()

    # Combined plot with both shifted stress-strain curves
    plt.figure(figsize=(10, 6))

    # Plot for the original method
    plt.plot(adjusted_extrapolated_strain, extrapolated_stress, color='red', linestyle='-')
    plt.plot(adjusted_original_strain, adjusted_original_stress, marker='o', color='red', linestyle='-',
        markersize=3, label='Pressure-Deflection Method')

```

```python
        # Plot for the direct fit method
        plt.plot(adjusted_extrapolated_strain_direct, extrapolated_stress_direct, color='blue', linestyle='-')
        plt.plot(adjusted_original_strain_direct, adjusted_original_stress_direct, marker='o', color='blue', linestyle='-', markersize=3, label='Direct Fit Method')

        # Title comparing the E-moduli
        plt.title(f'Comparison of Shifted Stress-Strain Curves\nNix E-modul: {found_slope:.2f} MPa vs Direct Fit E-modul: {direct_fit_E_modul:.2f} MPa')
        plt.xlabel('Strain')
        plt.ylabel('Stress [MPa]')
        plt.grid(False)
        plt.legend()
        plt.show()

    else:
        print("Could not find a good linear fit directly from strain-stress data.")


def analyze_dynamic(z_values, get_pressures, window_radius, output_directory, t, E_modul, sigma_0, missing_indices):
    a = float(window_radius)
    t = t

    current_time = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M")

    excel_path = os.path.join(output_directory, f'analysis_dynamic_{current_time}.xlsx')
    plots_directory = os.path.join(output_directory, 'Plots')
    if not os.path.exists(plots_directory):
        os.makedirs(plots_directory)

    plot_folder = os.path.join(plots_directory, 'Succesive_Plots')
    os.makedirs(plot_folder, exist_ok=True)

    plot_folder_p_vs_d = os.path.join(plots_directory, 'Succesive_Plots_p_vs_d')
    os.makedirs(plot_folder_p_vs_d, exist_ok=True)

    workbook = xlsxwriter.Workbook(excel_path)
    worksheet = workbook.add_worksheet('Results')

    headers = [
        'Filename', 'Z Position', 'Deflection (m)',
        'Get Pressure (bar)',
        'Strain', 'Membrane Stress (MPa)',
        'Von Mises Stress (MPa)', 'Difference (m)'
    ]

    for col, header in enumerate(headers):
        worksheet.write(0, col, header)

    pressures_unfitted = []
    deflections = []
    strains_unfitted = []
    von_mises_stresses_unfitted = []

    min_length = min(len(z_values), len(get_pressures))
    z_values = z_values[:min_length]
    get_pressures = get_pressures[:min_length]

    for index, z in enumerate(z_values):
```

```
            pressure_unfitted = get_pressures[index] * 0.1
            pressures_unfitted.append(pressure_unfitted)

            difference = z - z_values[0]
            deflections.append(difference)
            h = difference

            if h == 0:
                strain_unfitted = 0
                von_mises_stress_unfitted = sigma_0

                worksheet.write(index + 1, 0, f'Analysis_{index}')
                worksheet.write(index + 1, 1, z)
                worksheet.write(index + 1, 2, '(NA)')
                worksheet.write(index + 1, 3, '(NA)')
                worksheet.write(index + 1, 4, '(NA)')
                worksheet.write(index + 1, 5, '(NA)')
                worksheet.write(index + 1, 6, '(NA)')


            else:
                R = (a ** 2 + h ** 2) / (2 * h)
                theta = np.arcsin(a / R)
                strain_unfitted = (R * theta - a) / a

                membrane_stress_unfitted = (pressure_unfitted * R) / (2 * t)
                von_mises_stress_unfitted = membrane_stress_unfitted

                formulas = {
                    'Z Position': f'={z}',
                    'Deflection (m)': f'=(B{index + 2} - $B$2)',
                    'Get Pressure (bar)': f'={pressure_unfitted}*10',
                    'Strain': f'=(({a}^2 + C{index + 2}^2) / (2 * C{index + 2}) * ASIN({a} / (({a}^2 + C{index + 2}^2) / (2 * C{index + 2}))) - {a}) / {a}',
                    'Membrane Stress (MPa)': f'=(D{index + 2} * (({a}^2 + C{index + 2}^2) / (2 * C{index + 2})) / (2 * {t})/10',
                    'Von Mises Stress (MPa)': f'=F{index + 2}',
                }

                worksheet.write(index + 1, 0, f'Analysis_{index}')
                worksheet.write(index + 1, 1, z)
                worksheet.write_formula(index + 1, 2, formulas['Deflection (m)'])
                worksheet.write_formula(index + 1, 3, formulas['Get Pressure (bar)'])
                worksheet.write_formula(index + 1, 4, formulas['Strain'])
                worksheet.write_formula(index + 1, 5, formulas['Membrane Stress (MPa)'])
                worksheet.write_formula(index + 1, 6, formulas['Von Mises Stress (MPa)'])


            strains_unfitted.append(strain_unfitted)
            von_mises_stresses_unfitted.append(von_mises_stress_unfitted)

        workbook.close()
        print(f"Dynamic analysis results saved to: {excel_path}")



        pressures_unfitted_bar = np.array(pressures_unfitted) * 10 # from MPa to bar

        title_index = 1

```

```python
    # Compute the increase in deflection and pressure for all points first
    increase_in_deflection = [0] * 3 # Initialize the first three points as 0
    increase_in_pressure = [0] * 3 # Initialize the first three points as 0

    # Initialize a list to store discontinuity flags
    discontinuity_flags = [False] * 3

    for i in range(3, min_length):
        increase_in_deflection.append(deflections[i] - deflections[i - 1])
        increase_in_pressure.append(pressures_unfitted_bar[i] - pressures_unfitted_bar[i - 1])

        # Check for discontinuity: deflection increase larger, pressure increase smaller or equal
        if (increase_in_deflection[-1] > increase_in_deflection[-2]) and (increase_in_pressure[-1] <=
             increase_in_pressure[-2]):
            discontinuity_flags.append(True)
        else:
            discontinuity_flags.append(False)


    # Create the plots after computing all values
    for index in range(min_length):
        while title_index in [i + 1 for i in missing_indices]:
            title_index += 1

        # Strain-Stress Plot (for comparison)
        plt.figure(figsize=(10, 6))
        if index < 3:
            plt.plot(strains_unfitted[3:], von_mises_stresses_unfitted[3:], 'o-', color='grey', zorder=1)
        else:
            plt.plot(strains_unfitted[3:], von_mises_stresses_unfitted[3:], 'o-', color='grey', zorder=1)
            plt.plot(strains_unfitted[3:index], von_mises_stresses_unfitted[3:index], 'o-', color='blue',
                 zorder=2)
            plt.scatter(strains_unfitted[index], von_mises_stresses_unfitted[index], color='red', zorder=3)
        plt.xlabel('Strain')
        plt.ylabel('Stress (MPa)')
        plt.title(f'Strain-Stress Curve up to Recording {title_index}')
        plt.grid(False)
        plot_path = os.path.join(plot_folder, f'strain_stress_{index + 1}.png')
        plt.savefig(plot_path)
        plt.close()

        # Create a 2x1 plot for Pressure-Deflection and Successive Increase in Deflection/Pressure using
             GridSpec
        fig = plt.figure(figsize=(10, 6)) # Keep the figure size the same as the strain-stress plot
        gs = gridspec.GridSpec(2, 1, height_ratios=[1, 1]) # 2 rows, 1 column

        ax1 = fig.add_subplot(gs[0]) # First row for Pressure-Deflection Curve
        ax2 = fig.add_subplot(gs[1]) # Second row for Successive Increase in Deflection

        # Top plot: Pressure-Deflection Curve, skipping first 3 points
        if index < 3:
            ax1.plot(deflections[3:], pressures_unfitted_bar[3:], 'o-', color='grey', zorder=1)
        else:
            ax1.plot(deflections[3:], pressures_unfitted_bar[3:], 'o-', color='grey', zorder=1)
            ax1.plot(deflections[3:index], pressures_unfitted_bar[3:index], 'o-', color='blue', zorder=2)
            ax1.scatter(deflections[index], pressures_unfitted_bar[index], color='red', zorder=3)
        ax1.set_xlabel('Deflection (m)')
        ax1.set_ylabel('Pressure (bar)')
        ax1.set_title(f'Pressure-Deflection Curve up to Recording {title_index}')
        ax1.grid(False)
```

```
        # Bottom plot: Successive Increase in Deflection as bars
        x_data = list(range(4, min_length + 1))
        y_data_deflection = increase_in_deflection[3:]
        y_data_pressure = increase_in_pressure[3:]

        # Plot all bars in grey
        bars_deflection = ax2.bar(x_data, y_data_deflection, color='grey')

        # Color the discontinuity bars in yellow from the start
        for i in range(len(bars_deflection)):
            if discontinuity_flags[i + 3]:
                bars_deflection[i].set_color('yellow')

        # Color all previous bars in blue, keep yellow bars as they are, and color the current one in red
        if index >= 3:
            for i, bar in enumerate(bars_deflection[:index - 2]):
                if not discontinuity_flags[i + 3]: # Only recolor if it's not a yellow bar
                    bar.set_color('blue')
            if discontinuity_flags[index]:
                bars_deflection[index - 3].set_color('yellow')
            else:
                bars_deflection[index - 3].set_color('red')

        ax2.set_xlabel('Recording Index')
        ax2.set_ylabel('Increase in Deflection (m)', color='blue')
        ax2.set_title(f'Successive Increase in Deflection and Pressure up to Recording {title_index}')
        ax2.grid(False)

        # Add a second y-axis on the right for the increase in pressure
        ax2_pressure = ax2.twinx()
        ax2_pressure.plot(x_data, y_data_pressure, 'o-', color='lightgrey', markersize=4)
        ax2_pressure.set_ylabel('Increase in Pressure (bar)', color='lightgrey')

        plt.tight_layout()
        plot_path_p_vs_d = os.path.join(plot_folder_p_vs_d, f'pressure_deflection_{index + 1}.png')
        plt.savefig(plot_path_p_vs_d)
        plt.close()

        title_index += 1


    plot_defs_unfitted = [
        (deflections, pressures_unfitted_bar, 'Deflection [m]', 'Pressure [bar]', 'Deflection vs. Pressure',
             'Deflection_vs_Pressure.jpg'),
        (pressures_unfitted_bar, strains_unfitted, 'Pressure [bar]', 'Strain', 'Pressure vs. Strain',
             'Pressure_vs_Strain.jpg'),
    ]

    for x_data, y_data, xlabel, ylabel, title, filename in plot_defs_unfitted:
        plt.figure(figsize=(10, 6))
        plt.plot(x_data[2:], y_data[2:], 'o-')
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.title(title)
        plt.grid(False)
        plt.savefig(os.path.join(plots_directory, filename))
        plt.close()

    plt.figure(figsize=(10, 6))
```

```python
    plt.plot(np.array(strains_unfitted[2:]), np.array(von_mises_stresses_unfitted[2:]), 'o-')
    plt.xlabel('Strain')
    plt.ylabel('Von Mises Stress (MPa)')
    plt.title('Strain vs. Von Mises Stress')
    plt.grid(False)
    plt.savefig(os.path.join(plots_directory, 'Strain_vs_Von_Mises_Stress.jpg'))
    plt.close()

    print(f"Plots saved in directory: {plots_directory}")

    # Filter out the first data point and any outliers
    filtered_strain = strains_unfitted[2:]
    filtered_stress = von_mises_stresses_unfitted[2:]

    # Find the slope near the target
    start_index, found_slope = find_slope_near_target(filtered_strain, filtered_stress, E_modul)

    if start_index is not None:
        # Use the first plotted data point as the target stress
        target_stress = filtered_stress[0]

        adjusted_strain = [s - filtered_strain[0] for s in filtered_strain]
        positive_strain_indices = [i for i, s in enumerate(adjusted_strain) if s >= 0]
        positive_strain = [adjusted_strain[i] for i in positive_strain_indices]
        positive_stress = [filtered_stress[i] for i in positive_strain_indices]

        linear_range_start = start_index
        linear_range_end = linear_range_start + 5
        linear_strain_positive = positive_strain[linear_range_start:linear_range_end]
        linear_stress_positive = positive_stress[linear_range_start:linear_range_end]

        slope, intercept, _, _, _ = linregress(linear_strain_positive, linear_stress_positive)

        negative_strain_at_intersection = -intercept / slope

        # Extend the extrapolation to the first fitting data point
        extrapolated_strain = np.linspace(negative_strain_at_intersection,
            positive_strain[linear_range_start], 100)
        extrapolated_stress = slope * extrapolated_strain + intercept

        combined_strain = list(extrapolated_strain) + positive_strain
        combined_stress = list(extrapolated_stress) + positive_stress

        plt.figure(figsize=(10, 6))
        plt.plot(positive_strain, positive_stress, marker='o', linestyle='-', markersize=3, label='Original
            Data')
        plt.plot(extrapolated_strain, extrapolated_stress, color='red', linestyle='--', label='Extrapolated
            Linear Fit')
        plt.plot(linear_strain_positive, linear_stress_positive, marker='o', linestyle='-', color='blue',
            label='Linear Fit Range')
        plt.axvline(0, color='green', linestyle='--')
        plt.plot(negative_strain_at_intersection, 0, 'o', color='red', markersize=8)
        plt.fill_between(extrapolated_strain, extrapolated_stress, color='red', alpha=0.1,
            label='Extrapolated Region')

        # Plot removed data points
        removed_strains = np.array(adjusted_strain)[np.array(adjusted_strain) < 0]
        removed_stresses = np.array(filtered_stress)[np.array(adjusted_strain) < 0]
        if len(removed_strains) > 1:
```

```
        plt.plot(removed_strains, removed_stresses, 'o', color='grey', alpha=0.5, label='Removed Data Points')

    plt.title(f'Extrapolated Stress-Strain Curve\n(Residual Stress: {target_stress:.2f} MPa, E-modulus: {found_slope:.2f} MPa)')
    plt.xlabel('Strain')
    plt.ylabel('Stress [MPa]')
    plt.grid(False)
    plt.legend()
    plt.savefig(os.path.join(plots_directory, 'Extrapolated_Stress-Strain.jpg'))
    plt.show()

    if len(removed_strains) > 1:
        removed_strains = removed_strains[1:]
        removed_stresses = removed_stresses[1:]
    print(f"Number of points removed: {len(removed_strains)}")
    for strain_value, stress_value in zip(removed_strains, removed_stresses):
        print(f"Removed strain: {strain_value}, Removed stress: {stress_value}")

    adjusted_strain_for_plot = [s - negative_strain_at_intersection for s in combined_strain]
    adjusted_extrapolated_strain = adjusted_strain_for_plot[:len(extrapolated_strain)]
    adjusted_original_strain = adjusted_strain_for_plot[len(extrapolated_strain):]
    adjusted_original_stress = combined_stress[len(extrapolated_strain):]

    plt.figure(figsize=(10, 6))
    plt.plot(adjusted_extrapolated_strain, extrapolated_stress, color='black', linestyle='-')
    plt.plot(adjusted_original_strain, adjusted_original_stress, marker='o', color='black', linestyle='-', markersize=3, label='Shifted Original Data')

    plt.title('Shifted Stress-Strain Curve')
    plt.xlabel('Strain')
    plt.ylabel('Stress [MPa]')
    plt.grid(False)
    plt.legend()
    plt.savefig(os.path.join(plots_directory, 'Shifted_Stress-Strain.jpg'))
    plt.show()

    print(f"epsilon_0 is: {negative_strain_at_intersection:.6f}")
else:
    print("Could not find a range with the desired slope.")

# Additional Part: Iteratively expand the window for a direct fit within constraints
max_R2 = 0
best_fit_slope = None
best_fit_intercept = None
best_linear_strain = []
best_linear_stress = []

# Iterate over starting points within the first 10 points
for i in range(10):
    for j in range(i + 10, min(i + 40, len(filtered_strain))): # Ensure at least 10 points in the range, up to 40th point
        linear_range_strain = filtered_strain[i:j+1]
        linear_range_stress = filtered_stress[i:j+1]

        slope, intercept, r_value, _, _ = linregress(linear_range_strain, linear_range_stress)
        if r_value**2 > 0.95: # Use R^2 > 0.98 as the threshold
            if r_value**2 > max_R2:
                max_R2 = r_value**2
                best_fit_slope = slope
```

```python
                    best_fit_intercept = intercept
                    best_linear_strain = linear_range_strain
                    best_linear_stress = linear_range_stress
            else:
                break # Stop expanding when R^2 drops below 0.98

    if best_fit_slope is not None:
        negative_strain_at_intersection_direct = -best_fit_intercept / best_fit_slope
        direct_fit_E_modul = best_fit_slope # The slope of the linear fit represents the E-modul

        # Extend the extrapolation
        extrapolated_strain_direct = np.linspace(negative_strain_at_intersection_direct, best_linear_strain[0], 100)
        extrapolated_stress_direct = best_fit_slope * extrapolated_strain_direct + best_fit_intercept

        combined_strain_direct = list(extrapolated_strain_direct) + list(filtered_strain)
        combined_stress_direct = list(extrapolated_stress_direct) + list(filtered_stress)

        # Plot and save the direct extrapolated stress-strain curve
        plt.figure(figsize=(10, 6))
        plt.plot(filtered_strain, filtered_stress, marker='o', linestyle='-', markersize=3, label='Original Data')
        plt.plot(extrapolated_strain_direct, extrapolated_stress_direct, color='purple', linestyle='--', label='Direct Extrapolated Linear Fit')
        plt.plot(best_linear_strain, best_linear_stress, marker='o', linestyle='-', color='orange', label='Best Linear Fit Range')
        plt.axvline(0, color='green', linestyle='--')
        plt.plot(negative_strain_at_intersection_direct, 0, 'o', color='purple', markersize=8)
        plt.fill_between(extrapolated_strain_direct, extrapolated_stress_direct, color='purple', alpha=0.1, label='Direct Extrapolated Region')

        plt.title(f'Direct Extrapolated Stress-Strain Curve (E-modul: {direct_fit_E_modul:.2f} MPa)')
        plt.xlabel('Strain')
        plt.ylabel('Stress [MPa]')
        plt.grid(False)
        plt.legend()
        plt.savefig(os.path.join(plots_directory, 'Direct_Extrapolated_Stress-Strain.jpg'))
        plt.close()

        adjusted_strain_for_plot_direct = [s - negative_strain_at_intersection_direct for s in combined_strain_direct]
        adjusted_extrapolated_strain_direct = adjusted_strain_for_plot_direct[:len(extrapolated_strain_direct)]
        adjusted_original_strain_direct = adjusted_strain_for_plot_direct[len(extrapolated_strain_direct):]
        adjusted_original_stress_direct = combined_stress_direct[len(extrapolated_stress_direct):]

        # Plot and save the shifted stress-strain curve (direct fit)
        plt.figure(figsize=(10, 6))
        plt.plot(adjusted_extrapolated_strain_direct, extrapolated_stress_direct, color='black', linestyle='-')
        plt.plot(adjusted_original_strain_direct, adjusted_original_stress_direct, marker='o', color='black', linestyle='-', markersize=3, label='Shifted Original Data (Direct Fit)')

        plt.title(f'Shifted Stress-Strain Curve (Direct Fit, E-modul: {direct_fit_E_modul:.2f} MPa)')
        plt.xlabel('Strain')
        plt.ylabel('Stress [MPa]')
        plt.grid(False)
        plt.legend()
        plt.savefig(os.path.join(plots_directory, 'Shifted_Stress-Strain_Direct_Fit.jpg'))
        plt.close()
```

```
        print(f"Direct fit epsilon_0 is: {negative_strain_at_intersection_direct:.6f}")

        # Create subplots with shared x-axis and save
        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12), sharex=True)

        # Plot on ax1: Shifted curve for the original method with Steigungsdreieck
        ax1.plot(adjusted_extrapolated_strain, extrapolated_stress, color='red', linestyle='-')
        ax1.plot(adjusted_original_strain, adjusted_original_stress, marker='o', color='red', linestyle='-', markersize=3)

        # Calculate and plot the Steigungsdreieck for the original method
        sigma_orig = extrapolated_stress[-1] - extrapolated_stress[0]
        epsilon_orig = adjusted_extrapolated_strain[-1] - adjusted_extrapolated_strain[0]

        # Draw the Steigungsdreieck (only area below the slope)
        ax1.plot([adjusted_extrapolated_strain[0], adjusted_extrapolated_strain[-1]],
                [extrapolated_stress[0], extrapolated_stress[-1]], 'k--', label="Steigungsdreieck")
        ax1.vlines(adjusted_extrapolated_strain[-1], 0, extrapolated_stress[-1], colors='k', linestyle='dotted')
        ax1.fill_betweenx([0, extrapolated_stress[-1]], adjusted_extrapolated_strain[0], adjusted_extrapolated_strain[-1], color='red', alpha=0.1)

        # Positioning the annotation to the right of the vertical line and centered vertically
        midpoint_x_orig = adjusted_extrapolated_strain[-1]
        midpoint_y_orig = (extrapolated_stress[0] + extrapolated_stress[-1]) / 2
        ax1.text(midpoint_x_orig + 0.0002, midpoint_y_orig,
                f"$E_{{orig}} = \\frac{{\Delta\\sigma}}{{\Delta\\epsilon}} \\approx {found_slope:.2f} \\, MPa$",
                color='red', fontsize=12, verticalalignment='center')

        ax1.set_title(f'E-modul taken from pressure-deflection curve = {found_slope:.2f} MPa')
        ax1.set_ylabel('Stress [MPa]')
        ax1.legend()
        ax1.grid(False)

        # Plot on ax2: Shifted curve for the direct fit method with Steigungsdreieck
        ax2.plot(adjusted_extrapolated_strain_direct, extrapolated_stress_direct, color='blue', linestyle='-')
        ax2.plot(adjusted_original_strain_direct, adjusted_original_stress_direct, marker='o', color='blue', linestyle='-', markersize=3)

        # Calculate and plot the Steigungsdreieck for the direct fit method
        sigma_direct = extrapolated_stress_direct[-1] - extrapolated_stress_direct[0]
        epsilon_direct = adjusted_extrapolated_strain_direct[-1] - adjusted_extrapolated_strain_direct[0]

        # Draw the Steigungsdreieck (only area below the slope)
        ax2.plot([adjusted_extrapolated_strain_direct[0], adjusted_extrapolated_strain_direct[-1]],
                [extrapolated_stress_direct[0], extrapolated_stress_direct[-1]], 'k--', label="Steigungsdreieck (Direct Fit)")
        ax2.vlines(adjusted_extrapolated_strain_direct[-1], 0, extrapolated_stress_direct[-1], colors='k', linestyle='dotted')
        ax2.fill_betweenx([0, extrapolated_stress_direct[-1]], adjusted_extrapolated_strain_direct[0], adjusted_extrapolated_strain_direct[-1], color='blue', alpha=0.1)

        # Positioning the annotation to the right of the vertical line and centered vertically
        midpoint_x_direct = adjusted_extrapolated_strain_direct[-1]
        midpoint_y_direct = (extrapolated_stress_direct[0] + extrapolated_stress_direct[-1]) / 2
        ax2.text(midpoint_x_direct + 0.0002, midpoint_y_direct,
```

```
            f"$E_{{direct}} = \\frac{{\Delta\\sigma}}{{\Delta\\epsilon}} = \\frac{{{sigma_direct:.2f} \\, MPa}}{{{epsilon_direct:.4f}}} \\approx {direct_fit_E_modul:.2f} \\, MPa$",
            color='blue', fontsize=12, verticalalignment='center')

    ax2.set_title(f'Direct Fit Method: E-modul = {direct_fit_E_modul:.2f} MPa')
    ax2.set_xlabel('Strain [-]')
    ax2.set_ylabel('Stress [MPa]')
    ax2.legend()
    ax2.grid(False)

    plt.tight_layout()
    plt.savefig(os.path.join(plots_directory, 'Shifted_Curves_Comparison.jpg'))
    plt.close()

    # Combined plot with both shifted stress-strain curves and save
    plt.figure(figsize=(10, 6))

    # Plot for the original method
    plt.plot(adjusted_extrapolated_strain, extrapolated_stress, color='red', linestyle='-')
    plt.plot(adjusted_original_strain, adjusted_original_stress, marker='o', color='red', linestyle='-',
        markersize=3, label='Pressure-Deflection Method')

    # Plot for the direct fit method
    plt.plot(adjusted_extrapolated_strain_direct, extrapolated_stress_direct, color='blue',
        linestyle='-')
    plt.plot(adjusted_original_strain_direct, adjusted_original_stress_direct, marker='o', color='blue',
        linestyle='-', markersize=3, label='Direct Fit Method')

    # Title comparing the E-moduli
    plt.title(f'Comparison of Shifted Stress-Strain Curves\nOriginal E-modul: {found_slope:.2f} MPa vs
        Direct Fit E-modul: {direct_fit_E_modul:.2f} MPa')
    plt.xlabel('Strain [-]')
    plt.ylabel('Stress [MPa]')
    plt.grid(False)
    plt.legend()
    plt.savefig(os.path.join(plots_directory, 'Comparison_Shifted_Curves.jpg'))
    plt.close()

    workbook = xlsxwriter.Workbook(os.path.join(output_directory, 'Shifted_Stress_Strain_Data.xlsx'))
    worksheet = workbook.add_worksheet('Shifted Stress-Strain Data')

    # Headers for Original Method
    worksheet.write(0, 0, 'Nix Method - Extrapolated Strain')
    worksheet.write(0, 1, 'Nix Method - Extrapolated Stress')
    worksheet.write(0, 2, 'Nix Method - Original Strain')
    worksheet.write(0, 3, 'Nix Method - Original Stress')

    # Headers for Direct Fit Method
    worksheet.write(0, 5, 'Direct Fit - Extrapolated Strain')
    worksheet.write(0, 6, 'Direct Fit - Extrapolated Stress')
    worksheet.write(0, 7, 'Direct Fit - Original Strain')
    worksheet.write(0, 8, 'Direct Fit - Original Stress')

    # Save Original Method Data
    for i in range(len(adjusted_extrapolated_strain)):
        worksheet.write(i + 1, 0, adjusted_extrapolated_strain[i])
        worksheet.write(i + 1, 1, extrapolated_stress[i])
    for i in range(len(adjusted_original_strain)):
        worksheet.write(i + 1, 2, adjusted_original_strain[i])
        worksheet.write(i + 1, 3, adjusted_original_stress[i])
```

```

        # Save Direct Fit Method Data
        for i in range(len(adjusted_extrapolated_strain_direct)):
            worksheet.write(i + 1, 5, adjusted_extrapolated_strain_direct[i])
            worksheet.write(i + 1, 6, extrapolated_stress_direct[i])
        for i in range(len(adjusted_original_strain_direct)):
            worksheet.write(i + 1, 7, adjusted_original_strain_direct[i])
            worksheet.write(i + 1, 8, adjusted_original_stress_direct[i])

        workbook.close()

        print(f"Shifted stress-strain data saved to: {os.path.join(output_directory,
             'Shifted_Stress_Strain_Data.xlsx')}")

    else:
        print("Could not find a good linear fit directly from strain-stress data.")

        # Initialize the Excel workbook and worksheet
        workbook = xlsxwriter.Workbook(os.path.join(output_directory, 'Shifted_Stress_Strain_Data.xlsx'))
        worksheet = workbook.add_worksheet('Shifted Stress-Strain Data')

        # Headers for Original Method
        worksheet.write(0, 0, 'Nix Method - Extrapolated Strain')
        worksheet.write(0, 1, 'Nix Method - Extrapolated Stress')
        worksheet.write(0, 2, 'Nix Method - Original Strain')
        worksheet.write(0, 3, 'Nix Method - Original Stress')

        # Save Original Method Data
        for i in range(len(adjusted_extrapolated_strain)):
            worksheet.write(i + 1, 0, adjusted_extrapolated_strain[i])
            worksheet.write(i + 1, 1, extrapolated_stress[i])
        for i in range(len(adjusted_original_strain)):
            worksheet.write(i + 1, 2, adjusted_original_strain[i])
            worksheet.write(i + 1, 3, adjusted_original_stress[i])

        workbook.close()

        print(f"Shifted stress-strain data saved to: {os.path.join(output_directory,
             'Shifted_Stress_Strain_Data.xlsx')}")

    return von_mises_stresses_unfitted, strains_unfitted


def analyze_dynamic_with_uncertainties_and_sympy(z_values, get_pressures, window_radius, output_directory,
     t, E_modul, sigma_0, deflection_error_percent=0.5, pressure_error_percent=1.03):
    """
    Analyze dynamic stress-strain relationship considering error propagation using the uncertainties package,
    and save symbolic derivations with sympy as images and LaTeX files.
    """

    # Prepare lists for results
    deflections = []
    pressures_unfitted = []
    strains_unfitted = []
    von_mises_stresses_unfitted = []
    strains_error = []
    von_mises_stress_error = []

    # Convert the percentages to fractional uncertainties
    deflection_error_fraction = deflection_error_percent / 100.0
```

```python
    pressure_error_fraction = pressure_error_percent / 100.0

    # Prepare output paths for images and LaTeX files
    current_time = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M")
    latex_output_directory = os.path.join(output_directory, f'Latex_Images_{current_time}')
    os.makedirs(latex_output_directory, exist_ok=True)

    # Define symbolic variables
    h, a, p, t_sym, sigma_0_sym, u_h, u_p = sp.symbols('h a p t sigma_0 u_h u_p')

    # Define symbolic equations for strain and membrane stress
    R_sym = (a**2 + h**2) / (2 * h)
    theta_sym = sp.asin(a / R_sym)
    strain_sym = (R_sym * theta_sym - a) / a
    membrane_stress_sym = (p * R_sym) / (2 * t_sym)

    # Derivatives with respect to h and p
    strain_derivative_h = sp.diff(strain_sym, h)
    strain_derivative_p = sp.diff(strain_sym, p)
    stress_derivative_h = sp.diff(membrane_stress_sym, h)
    stress_derivative_p = sp.diff(membrane_stress_sym, p)

    # Error propagation formulas
    strain_error = sp.sqrt((strain_derivative_h * u_h)**2 + (strain_derivative_p * u_p)**2)
    stress_error = sp.sqrt((stress_derivative_h * u_h)**2 + (stress_derivative_p * u_p)**2)

    # LaTeX expressions for the symbolic computation
    latex_strain = sp.latex(strain_sym)
    latex_stress = sp.latex(membrane_stress_sym)
    latex_strain_derivative_h = sp.latex(strain_derivative_h)
    latex_strain_derivative_p = sp.latex(strain_derivative_p)
    latex_stress_derivative_h = sp.latex(stress_derivative_h)
    latex_stress_derivative_p = sp.latex(stress_derivative_p)
    latex_strain_error = sp.latex(strain_error)
    latex_stress_error = sp.latex(stress_error)

    # Full LaTeX expressions for documentation
    full_expression_latex = r"""
    \text{Strain:} \quad \epsilon = """ + latex_strain + r"""
    \\[1em]
    \text{Membrane Stress:} \quad \sigma = """ + latex_stress + r"""
    \\[2em]
    \text{Derivative of Strain w.r.t h:} \quad \frac{\partial \epsilon}{\partial h} = """ + latex_strain_derivative_h + r"""
    \\[1em]
    \text{Derivative of Strain w.r.t p:} \quad \frac{\partial \epsilon}{\partial p} = """ + latex_strain_derivative_p + r"""
    \\[1em]
    \text{Derivative of Membrane Stress w.r.t h:} \quad \frac{\partial \sigma}{\partial h} = """ + latex_stress_derivative_h + r"""
    \\[1em]
    \text{Derivative of Membrane Stress w.r.t p:} \quad \frac{\partial \sigma}{\partial p} = """ + latex_stress_derivative_p + r"""
    \\[2em]
    \text{Error Propagation for Strain:} \quad u_\epsilon = """ + latex_strain_error + r"""
    \\[1em]
    \text{Error Propagation for Membrane Stress:} \quad u_\sigma = """ + latex_stress_error

    # Save LaTeX code to a .tex file
    latex_file_path = os.path.join(latex_output_directory, 'symbolic_expressions_with_error_propagation.tex')
```

```python
    with open(latex_file_path, 'w') as latex_file:
        latex_file.write(full_expression_latex)

    # Function to save LaTeX as an image
    def save_latex_as_image(latex_code, output_path):
        fig, ax = plt.subplots(figsize=(12, 6))
        ax.text(0.5, 0.5, r"$" + latex_code + r"$", fontsize=20, ha='center', va='center')
        ax.axis('off')
        fig.savefig(output_path, bbox_inches='tight')
        plt.close(fig)

    # Save image of the full symbolic expression
    full_expression_image_path = os.path.join(latex_output_directory,
         'full_expression_with_derivatives_and_errors.png')
    save_latex_as_image(full_expression_latex, full_expression_image_path)

    min_length = min(len(z_values), len(get_pressures))
    z_values = z_values[:min_length]
    get_pressures = get_pressures[:min_length]

    # Iterate over z_values to calculate strain and stress with uncertainties
    for index, z in enumerate(z_values):
        pressure_unfitted = get_pressures[index] * 0.1 # Convert to MPa
        pressures_unfitted.append(pressure_unfitted)

        difference = z - z_values[0]
        deflections.append(difference)

        h_val = difference

        # If deflection is zero or negative, assign default values
        if h_val <= 0:
            strains_unfitted.append(0)
            von_mises_stresses_unfitted.append(sigma_0)
            strains_error.append(0)
            von_mises_stress_error.append(0)
            continue

        # Calculate the deflection with uncertainty
        h_with_uncertainty = ufloat(h_val, abs(h_val * deflection_error_fraction))
        p_with_uncertainty = ufloat(pressure_unfitted, abs(pressure_unfitted * pressure_error_fraction))

        # Calculate radius R
        R = (window_radius ** 2 + h_with_uncertainty ** 2) / (2 * h_with_uncertainty)
        theta = asin(window_radius / R)

        # Calculate strain and von Mises stress
        strain = (R * theta - window_radius) / window_radius
        membrane_stress = (p_with_uncertainty * R) / (2 * t)

        # Store results
        strains_unfitted.append(strain.nominal_value)
        von_mises_stresses_unfitted.append(membrane_stress.nominal_value)
        strains_error.append(strain.std_dev)
        von_mises_stress_error.append(membrane_stress.std_dev)

    # Plot the stress-strain curve with error bars
    plt.figure(figsize=(10, 6))
    plt.errorbar(
        x=strains_unfitted[3:],
```

```
        y=von_mises_stresses_unfitted[3:],
        xerr=strains_error[3:], # Add horizontal error bars for strain
        yerr=von_mises_stress_error[3:], # Vertical error bars for stress
        fmt='o', color='blue', label='Original Data with Error Bars',
        markersize=2 # Smaller points for better visibility of error bars
    )
    plt.xlabel('Strain')
    plt.ylabel('Von Mises Stress (MPa)')
    plt.title('Strain vs. Von Mises Stress with Error Propagation')
    plt.legend()
    plt.grid(False)
    plot_path = os.path.join(output_directory, f'Strain_vs_Von_Mises_Stress_with_Error_{current_time}.png')
    plt.savefig(plot_path, bbox_inches='tight')
    plt.close()

    # Save calculated values to an Excel file
    data = {
        'Deflections (m)': deflections,
        'Strains (Original)': strains_unfitted,
        'Von Mises Stress (Original)': von_mises_stresses_unfitted,
        'Strain Error': strains_error,
        'Von Mises Stress Error': von_mises_stress_error
    }

    df = pd.DataFrame(data)
    excel_path = os.path.join(output_directory,
         f'Stress_Strain_with_Error_Uncertainties_{current_time}.xlsx')
    df.to_excel(excel_path, index=False)
    print(f"Data saved to Excel file: {excel_path}")
    print(f"Full symbolic expression with error propagation saved to: {full_expression_image_path}")
    print(f"Full symbolic expression LaTeX saved to: {latex_file_path}")


def process_images(folder_path, output_directory, von_mises_stresses, magnification, plot_fractions=False):
    dimensions = {
        10: (1700, 1418.6),
        20: (850, 709),
        50: (340, 283.7),
        150: (113.3, 94.6)
    }
    scale_bar_lengths = {
        10: 500,
        20: 200,
        50: 50,
        150: 20
    }
    if magnification not in dimensions or magnification not in scale_bar_lengths:
        raise ValueError("Unsupported magnification. Choose from 10, 20, 50, or 150.")
    image_width_um, image_height_um = dimensions[magnification]
    scale_bar_length_um = scale_bar_lengths[magnification]

    output_folder = os.path.join(output_directory, "processed_images")
    os.makedirs(output_folder, exist_ok=True)

    image_files = [f for f in os.listdir(folder_path) if f.endswith(".plux_image.jpg")]

    primary_counts = []
    secondary_counts = []
    primary_distances = []
    secondary_distances = []
```

```
    primary_fractions = []
    secondary_fractions = []
    no_crack_indices = []

    for i, (image_file, von_mises_stress) in enumerate(zip(image_files, von_mises_stresses)):
        image_path = os.path.join(folder_path, image_file)
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            print(f"Error: Image {image_file} not found or the file is corrupt.")
            continue

        height, width = image.shape

        blurred_image = cv2.GaussianBlur(image, (3, 3), 0)
        clahe = cv2.createCLAHE(clipLimit=1, tileGridSize=(8, 8))
        contrasted_image = clahe.apply(blurred_image)
        edges_canny = cv2.Canny(contrasted_image, 110, 150)
        edges_blurred = cv2.Canny(blurred_image, 20, 70)
        combined_edges = cv2.bitwise_or(edges_canny, edges_blurred)
        dilated_edges = cv2.dilate(combined_edges, np.ones((1, 1), np.uint8), iterations=1)

        contours, _ = cv2.findContours(dilated_edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        contour_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)

        primary_mask = np.zeros_like(image, dtype=np.uint8)
        secondary_mask = np.zeros_like(image, dtype=np.uint8)

        contour_types = []

        for contour in contours:
            contour_length = cv2.arcLength(contour, closed=False)
            x, y, w, h = cv2.boundingRect(contour)
            if contour_length > 600 or w > 90 or h > 90:
                color = (0, 0, 255)
                contour_types.append('secondary')
                cv2.drawContours(secondary_mask, [contour], -1, 255, -1)
            else:
                color = (255, 0, 255)
                contour_types.append('primary')
                cv2.drawContours(primary_mask, [contour], -1, 255, -1)
            cv2.drawContours(contour_image, [contour], -1, color, 1)

        primary_fraction = np.sum(primary_mask) / (height * width * 255)
        secondary_fraction = np.sum(secondary_mask) / (height * width * 255)

        primary_fractions.append(primary_fraction)
        secondary_fractions.append(secondary_fraction)

        line_y = height // 2
        cv2.line(contour_image, (0, line_y), (width, line_y), (255, 0, 0), 2)
        crack_centers = []
        crack_indices = []

        for idx, contour in enumerate(contours):
            x_coords = [point[0][0] for point in contour if abs(point[0][1] - line_y) < 5]
            crack_centers.extend(x_coords)
            crack_indices.extend([idx] * len(x_coords))

        if not crack_centers:
            print(f"No crack centers found in image: {image_file}")
```

```python
            no_crack_indices.append(i)
            primary_counts.append(0)
            secondary_counts.append(0)
            primary_distances.append(0)
            secondary_distances.append(0)

            overlay = contour_image.copy()
            cv2.putText(overlay, 'No Cracks Found', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA)
            output_image_path = os.path.join(output_folder, f"{image_file[:5]}_processed_image.jpg")
            cv2.imwrite(output_image_path, overlay)

            continue

        points = np.array(crack_centers).reshape(-1, 1)
        clusters = fclusterdata(points, 10, criterion='distance')
        clustered_centers, types = [], []
        for cluster_id in np.unique(clusters):
            cluster_mask = clusters == cluster_id
            cluster_points = points[cluster_mask]
            cluster_indices = np.array(crack_indices)[cluster_mask]
            cluster_types = [contour_types[i] for i in cluster_indices]
            median_point = int(np.median(cluster_points))
            preferred_type = 'secondary' if 'secondary' in cluster_types else 'primary'
            clustered_centers.append(median_point)
            types.append(preferred_type)

        pixel_to_um_x = image_width_um / width
        crack_centers_um = [x * pixel_to_um_x for x in clustered_centers]

        primary_crack_centers = [crack_centers_um[i] for i in range(len(crack_centers_um)) if types[i] == 'primary']
        secondary_crack_centers = [crack_centers_um[i] for i in range(len(crack_centers_um)) if types[i] == 'secondary']

        primary_distances_um = [abs(primary_crack_centers[i + 1] - primary_crack_centers[i]) for i in range(len(primary_crack_centers) - 1)]
        secondary_distances_um = [abs(secondary_crack_centers[i + 1] - secondary_crack_centers[i]) for i in range(len(secondary_crack_centers) - 1)]

        primary_count = len(primary_crack_centers)
        secondary_count = len(secondary_crack_centers)
        primary_counts.append(primary_count)
        secondary_counts.append(secondary_count)

        primary_avg_distance = np.mean(primary_distances_um) if primary_distances_um else 0
        secondary_avg_distance = np.mean(secondary_distances_um) if secondary_distances_um else 0

        primary_distances.append(primary_avg_distance)
        secondary_distances.append(secondary_avg_distance)

        for j, x_um in enumerate(crack_centers_um):
            x = int(x_um / pixel_to_um_x)
            color = (0, 255, 0) if types[j] == 'primary' else (255, 255, 255)
            cv2.circle(contour_image, (x, line_y), 7, color, -1)

        overlay = contour_image.copy()
        alpha = 0.6
        scale_bar_text = f"{scale_bar_length_um} micrometer"

```

```
        if magnification == 10:
            font_scale = 0.6
            box_padding = 20
        elif magnification == 20:
            font_scale = 0.8
            box_padding = 30
        elif magnification == 50:
            font_scale = 1.0
            box_padding = 40
        elif magnification == 150:
            font_scale = 1.2
            box_padding = 50

        text_size = cv2.getTextSize(scale_bar_text, cv2.FONT_HERSHEY_SIMPLEX, font_scale, 2)[0]
        box_width = text_size[0] + box_padding
        box_height = text_size[1] + box_padding

        cv2.rectangle(overlay, (width - box_width - 10, height - box_height - 10), (width - 10, height - 10), (0, 0, 0), -1)
        contour_image = cv2.addWeighted(overlay, alpha, contour_image, 1 - alpha, 0)

        scale_bar_length_px = int(scale_bar_length_um / pixel_to_um_x)
        scale_bar_start = (width - box_width - 10, height - 25)
        scale_bar_end = (scale_bar_start[0] + scale_bar_length_px, height - 25)
        cv2.line(contour_image, scale_bar_start, scale_bar_end, (255, 255, 255), 2)
        cv2.putText(contour_image, scale_bar_text, (scale_bar_start[0], height - 35), cv2.FONT_HERSHEY_SIMPLEX, font_scale, (255, 255, 255), 2)

        output_image_path = os.path.join(output_folder, f"{image_file[:5]}_processed_image.jpg")
        cv2.imwrite(output_image_path, contour_image)

        if i == 0 or (i + 1) % 30 == 0 or i == len(image_files) - 1:
            plt.figure(figsize=(10, 10))
            plt.imshow(cv2.cvtColor(contour_image, cv2.COLOR_BGR2RGB))
            plt.title(f"Contours and Measurement Line with Midpoints for Image {i+1}")
            plt.xticks(np.linspace(0, width, num=11), [f"{int(x)} m" for x in np.linspace(0, image_width_um, num=11)])
            plt.yticks(np.linspace(0, height, num=11), [f"{int(y)} m" for y in np.linspace(0, image_height_um, num=11)])
            plt.show()

    min_length = min(len(image_files), len(von_mises_stresses), len(primary_counts), len(secondary_counts), len(primary_distances), len(secondary_distances), len(primary_fractions), len(secondary_fractions))

    data = {
        'Image': image_files[:min_length],
        'Primary Cracks': primary_counts[:min_length],
        'Secondary Cracks': secondary_counts[:min_length],
        'Avg Primary Distance (m)': primary_distances[:min_length],
        'Avg Secondary Distance (m)': secondary_distances[:min_length]
    }

    df = pd.DataFrame(data)
    df.to_excel(os.path.join(output_folder, 'crack_analysis.xlsx'), index=False)

    if plot_fractions:
        data['Primary Fraction'] = primary_fractions[:min_length]
        data['Secondary Fraction'] = secondary_fractions[:min_length]
        df = pd.DataFrame(data)
        df.to_excel(os.path.join(output_folder, 'crack_analysis_with_fractions.xlsx'), index=False)
```

```python

        x_data = range(min_length)
        x_labels = [f[:5] for f in image_files[:min_length]]

        fig3, ax3 = plt.subplots(figsize=(10, 5))
        ax3.plot(x_data, primary_fractions[:min_length], 'g--', label='Primary Fraction')
        ax3.plot(x_data, secondary_fractions[:min_length], 'r--', label='Secondary Fraction')

        for idx in no_crack_indices:
            if idx < min_length:
                ax3.annotate('No Cracks', (idx, 0), textcoords="offset points", xytext=(0,10), ha='center', color='red')
                ax3.plot(idx, 0, 'ro')

        ax3.set_xlabel('Image Files')
        ax3.set_ylabel('Fraction of Image Area')
        ax3.legend()
        ax3.set_xticks(x_data[::20])
        ax3.set_xticklabels(x_labels[::20], rotation=45, ha='right')
        plt.tight_layout()
        plt.savefig(os.path.join(output_folder, 'area_fractions_plot.png'))
        plt.show()

    x_data = range(min_length)
    x_labels = [f[:5] for f in image_files[:min_length]]

    fig1, ax1 = plt.subplots(figsize=(10, 5))
    ax1.plot(x_data, primary_counts[:min_length], 'g--', label='Primary Cracks')
    ax1.plot(x_data, secondary_counts[:min_length], 'r--', label='Secondary Cracks')

    for idx in no_crack_indices:
        if idx < min_length:
            ax1.annotate('No Cracks', (idx, 0), textcoords="offset points", xytext=(0,10), ha='center', color='red')
            ax1.plot(idx, 0, 'ro')

    ax1.set_xlabel('Image Files')
    ax1.set_ylabel('Number of Cracks')
    ax1.legend()
    ax1.set_xticks(x_data[::20])
    ax1.set_xticklabels(x_labels[::20], rotation=45, ha='right')
    plt.tight_layout()
    plt.savefig(os.path.join(output_folder, 'crack_counts_plot.png'))
    plt.show()

    fig2, ax2 = plt.subplots(figsize=(10, 5))
    ax2.plot(x_data, primary_distances[:min_length], 'g--', label='Avg Primary Distance (m)')
    ax2.plot(x_data, secondary_distances[:min_length], 'r--', label='Avg Secondary Distance (m)')

    for idx in no_crack_indices:
        if idx < min_length:
            ax2.annotate('No Cracks', (idx, 0), textcoords="offset points", xytext=(0,10), ha='center', color='red')
            ax2.plot(idx, 0, 'ro')

    ax2.set_xlabel('Image Files')
    ax2.set_ylabel('Average Distance (m)')
    ax2.legend()
    ax2.set_xticks(x_data[::20])
    ax2.set_xticklabels(x_labels[::20], rotation=45, ha='right')
```

```python
    plt.tight_layout()
    plt.grid(False)
    plt.savefig(os.path.join(output_folder, 'crack_distances_plot.png'))
    plt.show()

def find_newest_directory(base_path):
    date_pattern = re.compile(r"\d{4}-\d{2}-\d{2}_\d{2}-\d{2}")
    directories = [os.path.join(base_path, d) for d in os.listdir(base_path)
                   if os.path.isdir(os.path.join(base_path, d)) and date_pattern.match(d)]
    latest_dir = max(directories, key=os.path.getmtime, default=None)
    return latest_dir

def add_scales_to_image(image_path, magnification, output_path):
    dimensions = {
        10: (1700, 1418.6),
        20: (850, 709),
        50: (340, 283.7),
        150: (113.3, 94.6)
    }

    scale_bar_lengths = {
        10: 500,
        20: 200,
        50: 50,
        150: 20
    }

    if magnification not in dimensions or magnification not in scale_bar_lengths:
        raise ValueError("Unsupported magnification. Choose from 10, 20, 50, or 150.")

    image_width_um, image_height_um = dimensions[magnification]
    scale_bar_length_um = scale_bar_lengths[magnification]

    image = cv2.imread(image_path)
    height, width, _ = image.shape
    pixel_to_um_x = image_width_um / width

    overlay = image.copy()
    alpha = 0.6
    scale_bar_text = f"{scale_bar_length_um} micrometer"

    if magnification == 10:
        font_scale = 0.6
        box_padding = 20
    elif magnification == 20:
        font_scale = 0.8
        box_padding = 30
    elif magnification == 50:
        font_scale = 1.0
        box_padding = 40
    elif magnification == 150:
        font_scale = 1.2
        box_padding = 50

    text_size = cv2.getTextSize(scale_bar_text, cv2.FONT_HERSHEY_SIMPLEX, font_scale, 2)[0]
    box_width = text_size[0] + box_padding
    box_height = text_size[1] + box_padding

    cv2.rectangle(overlay, (width - box_width - 10, height - box_height - 10), (width - 10, height - 10),
         (0, 0, 0), -1)
```

```python
    image = cv2.addWeighted(overlay, alpha, image, 1 - alpha, 0)

    scale_bar_length_px = int(scale_bar_length_um / pixel_to_um_x)
    scale_bar_start = (width - box_width - 10, height - 25)
    scale_bar_end = (scale_bar_start[0] + scale_bar_length_px, height - 25)
    cv2.line(image, scale_bar_start, scale_bar_end, (255, 255, 255), 2)
    cv2.putText(image, scale_bar_text, (scale_bar_start[0], height - 35), cv2.FONT_HERSHEY_SIMPLEX, font_scale, (255, 255, 255), 2)

    plt.figure(figsize=(10, 10))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.xticks(np.linspace(0, width, num=11), [f"{int(x)} m" for x in np.linspace(0, image_width_um, num=11)])
    plt.yticks(np.linspace(0, height, num=11), [f"{int(y)} m" for y in np.linspace(0, image_height_um, num=11)])
    plt.savefig(output_path, bbox_inches='tight')
    plt.close()

def create_pdf_with_processed_images_and_plots(source_directory, output_directory, pressure_values, von_mises_stresses, strains, magnification, save_with_scales):
    image_types = ['plux_image.jpg', 'processed_image.jpg']
    files = {image_type: [] for image_type in image_types}
    processed_images_dir = os.path.join(output_directory, 'processed_images')

    for file_name in os.listdir(source_directory):
        if file_name.endswith('plux_image.jpg'):
            files['plux_image.jpg'].append(os.path.join(source_directory, file_name))

    for file_name in os.listdir(processed_images_dir):
        if file_name.endswith('.jpg'):
            files['processed_image.jpg'].append(os.path.join(processed_images_dir, file_name))

    for image_type, file_list in files.items():
        files[image_type] = sorted(file_list, key=lambda x: int(re.search(r'(\d+)', os.path.basename(x)).group(1)))

    output_path = os.path.join(output_directory, "processed_images_report.pdf")
    doc = SimpleDocTemplate(output_path, pagesize=letter, rightMargin=72, leftMargin=72, topMargin=18, bottomMargin=18)
    story = []
    styles = getSampleStyleSheet()

    logo_path = "H:\\01 Bulge Testing Versuche\\6mm_14mm\\logo.png"
    if os.path.exists(logo_path):
        logo = Image(logo_path, width=100, height=50)
    else:
        logo = Paragraph("<font color='red'><b>Empa Thun</b></font>", styles['Title'])

    title = Paragraph("<font size=12><b>Processed Images Report</b></font>", styles['Title'])
    datum = datetime.datetime.now().strftime("%Y-%m-%d %H-%M")

    header_data = [[logo, title, datum]]
    header_table = Table(header_data, colWidths=[108, 324, 108], rowHeights=60)
    header_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story.append(header_table)
    story.append(Spacer(1, 20))
```

```python

    available_width = letter[0] - 60
    first_column_width = 70
    image_width = (available_width - first_column_width) / len(image_types) - (0.1 * inch)
    headers = ['P// '] + [img_type.replace('.jpg', '').replace('_', ' ').title() for img_type in image_types]
    table_data = [headers]

    num_rows = max(len(files[image_type]) for image_type in image_types)
    for index in range(num_rows):
        if index < len(pressure_values):
            pressure = f"{pressure_values[index]:.2f} bar"
            stress = f"{von_mises_stresses[index]:.2f} MPa"
            strain = f"{strains[index] * 100:.2f} [%]"
            row = [Paragraph(f"{pressure}<br/>{stress}<br/>{strain}", styles['Normal'])]
            for image_type in image_types:
                try:
                    img_path = files[image_type][index]
                    output_img_path = img_path.replace('.jpg', '_with_scales.jpg')
                    if save_with_scales:
                        if not os.path.exists(output_img_path):
                            add_scales_to_image(img_path, magnification, output_img_path)
                    else:
                        output_img_path = img_path
                    img = Image(output_img_path, width=image_width, height=image_width * 0.75)
                    img_name = Paragraph(f"<font size=9>{os.path.basename(output_img_path)}</font>", styles["Normal"])
                    row.append([img, img_name])
                except (IndexError, FileNotFoundError):
                    row.append('')
            table_data.append(row)

    table = Table(table_data, colWidths=[first_column_width] + [image_width] * len(image_types), style=TableStyle([
        ('INNERGRID', (0,0), (-1,-1), 0.25, colors.black),
        ('BOX', (0,0), (-1,-1), 0.25, colors.black),
        ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
        ('ALIGN', (0,0), (-1,-1), 'CENTER')
    ]))
    story.append(table)
    story.append(Spacer(1, 20))

    plots_path = os.path.join(output_directory, "Plots")
    if os.path.exists(plots_path):
        for plot_file in os.listdir(plots_path):
            plot_full_path = os.path.join(plots_path, plot_file)
            if plot_full_path.endswith('.jpg'):
                new_width, new_height = resize_image(plot_full_path, available_width, 270)
                plot_image = Image(plot_full_path, width=new_width, height=new_height)
                story.append(plot_image)
                story.append(Spacer(1, 12))

    pressure_deflection_image_path = os.path.join(output_directory, "Pressure_vs_Deflection.png")
    if os.path.exists(pressure_deflection_image_path):
        new_width, new_height = resize_image(pressure_deflection_image_path, available_width, 270)
        deflection_image = Image(pressure_deflection_image_path, width=new_width, height=new_height)
        story.append(deflection_image)

    story.append(Spacer(1, 20))

```

```
    footer_data = [[logo, '', datum]]
    footer_table = Table(footer_data, colWidths=[108, 324, 108], rowHeights=60)
    footer_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story.append(footer_table)

    doc.build(story)
    print(f"PDF created with processed images and plots: {output_path}")

def create_pdf_with_table_and_strain_stress(source_directory, output_directory, pressure_values, von_mises_stresses, strains, magnification, save_with_scales):
    image_types = ['plux_image.jpg']
    files = {image_type: [] for image_type in image_types}
    strain_stress_plots_dir = os.path.join(output_directory, 'Plots', 'Succesive_Plots')
    pressure_deflection_plots_dir = os.path.join(output_directory, 'Plots', 'Succesive_Plots_p_vs_d')

    for file_name in os.listdir(source_directory):
        if file_name.endswith('plux_image.jpg'):
            files['plux_image.jpg'].append(os.path.join(source_directory, file_name))

    for image_type, file_list in files.items():
        files[image_type] = sorted(file_list, key=lambda x: int(re.search(r'(\d+)', os.path.basename(x)).group(1)))

    # Generate PDF with strain-stress plots
    output_path_strain_stress = os.path.join(output_directory, "output_with_strain_stress.pdf")
    doc_strain_stress = SimpleDocTemplate(output_path_strain_stress, pagesize=letter, rightMargin=72, leftMargin=72, topMargin=18, bottomMargin=18)
    story_strain_stress = []
    styles = getSampleStyleSheet()

    logo_path = "H:\\01 Bulge Testing Versuche\\6mm_14mm\\logo.png"
    if os.path.exists(logo_path):
        logo = Image(logo_path, width=100, height=50)
    else:
        logo = Paragraph("<font color='red'><b>Empa Thun</b></font>", styles['Title'])
    title_strain_stress = Paragraph("<font size=12><b>Pressure and Deflection Report with Strain-Stress Plots</b></font>", styles['Title'])
    datum = datetime.datetime.now().strftime("%Y-%m-%d %H-%M")

    header_data = [[logo, title_strain_stress, datum]]
    header_table = Table(header_data, colWidths=[108, 324, 108], rowHeights=60)
    header_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story_strain_stress.append(header_table)
    story_strain_stress.append(Spacer(1, 20))

    available_width = letter[0] - 60
    first_column_width = 70
    image_width = (available_width - first_column_width) / 2 - (0.1 * inch)
    headers = ['P//  ', 'Plux Image', 'Strain-Stress Curve']
    table_data_strain_stress = [headers]

    num_rows = len(files['plux_image.jpg'])
```

```
    for index in range(num_rows):
        if index < len(pressure_values):
            pressure = f"{pressure_values[index]:.3f} bar"
            stress = f"{von_mises_stresses[index]:.3f} MPa"
            strain = f"{strains[index] * 100:.3f} [%]"
            row = [Paragraph(f"{pressure}<br/>{stress}<br/>{strain}", styles['Normal'])]

            try:
                img_path = files['plux_image.jpg'][index]
                output_img_path = img_path.replace('.jpg', '_with_scales.jpg')
                if save_with_scales:
                    if not os.path.exists(output_img_path):
                        add_scales_to_image(img_path, magnification, output_img_path)
                else:
                    output_img_path = img_path
                img = Image(output_img_path, width=image_width, height=image_width * 0.75)
                img_name = Paragraph(f"<font size=9>{os.path.basename(output_img_path)}</font>", styles["Normal"])
                row.append([img, img_name])
            except (IndexError, FileNotFoundError):
                row.append('')

            try:
                strain_stress_plot_path = os.path.join(strain_stress_plots_dir, f'strain_stress_{index + 1}.png')
                strain_stress_img = Image(strain_stress_plot_path, width=image_width, height=image_width * 0.75)
                row.append(strain_stress_img)
            except (FileNotFoundError, IndexError):
                row.append('')

            table_data_strain_stress.append(row)

    table_strain_stress = Table(table_data_strain_stress, colWidths=[first_column_width] + [image_width] * 2, style=TableStyle([
        ('INNERGRID', (0,0), (-1,-1), 0.25, colors.black),
        ('BOX', (0,0), (-1,-1), 0.25, colors.black),
        ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
        ('ALIGN', (0,0), (-1,-1), 'CENTER')
    ]))
    story_strain_stress.append(table_strain_stress)
    story_strain_stress.append(Spacer(1, 20))

    # Generate PDF with pressure-deflection plots
    output_path_pressure_deflection = os.path.join(output_directory, "output_with_pressure_deflection.pdf")
    doc_pressure_deflection = SimpleDocTemplate(output_path_pressure_deflection, pagesize=letter, rightMargin=72, leftMargin=72, topMargin=18, bottomMargin=18)
    story_pressure_deflection = []

    title_pressure_deflection = Paragraph("<font size=12><b>Pressure and Deflection Report with Pressure-Deflection Plots</b></font>", styles['Title'])
    header_data_pressure_deflection = [[logo, title_pressure_deflection, datum]]
    header_table_pressure_deflection = Table(header_data_pressure_deflection, colWidths=[108, 324, 108], rowHeights=60)
    header_table_pressure_deflection.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story_pressure_deflection.append(header_table_pressure_deflection)
```

```python
    story_pressure_deflection.append(Spacer(1, 20))

    headers_pressure_deflection = ['P//', 'Plux Image', 'Pressure-Deflection Curve']
    table_data_pressure_deflection = [headers_pressure_deflection]

    for index in range(num_rows):
        if index < len(pressure_values):
            pressure = f"{pressure_values[index]:.3f} bar"
            stress = f"{von_mises_stresses[index]:.3f} MPa"
            strain = f"{strains[index] * 100:.3f} [%]"
            row = [Paragraph(f"{pressure}<br/>{stress}<br/>{strain}", styles['Normal'])]

            try:
                img_path = files['plux_image.jpg'][index]
                output_img_path = img_path.replace('.jpg', '_with_scales.jpg')
                if save_with_scales:
                    if not os.path.exists(output_img_path):
                        add_scales_to_image(img_path, magnification, output_img_path)
                else:
                    output_img_path = img_path
                img = Image(output_img_path, width=image_width, height=image_width * 0.75)
                img_name = Paragraph(f"<font size=9>{os.path.basename(output_img_path)}</font>", styles["Normal"])
                row.append([img, img_name])
            except (IndexError, FileNotFoundError):
                row.append('')

            try:
                pressure_deflection_plot_path = os.path.join(pressure_deflection_plots_dir, f'pressure_deflection_{index + 1}.png')
                pressure_deflection_img = Image(pressure_deflection_plot_path, width=image_width, height=image_width * 0.75)
                row.append(pressure_deflection_img)
            except (FileNotFoundError, IndexError):
                row.append('')

            table_data_pressure_deflection.append(row)

    table_pressure_deflection = Table(table_data_pressure_deflection, colWidths=[first_column_width] + [image_width] * 2, style=TableStyle([
        ('INNERGRID', (0,0), (-1,-1), 0.25, colors.black),
        ('BOX', (0,0), (-1,-1), 0.25, colors.black),
        ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
        ('ALIGN', (0,0), (-1,-1), 'CENTER')
    ]))
    story_pressure_deflection.append(table_pressure_deflection)
    story_pressure_deflection.append(Spacer(1, 20))

    # Finalizing both PDFs
    doc_strain_stress.build(story_strain_stress)
    doc_pressure_deflection.build(story_pressure_deflection)

    print(f"PDF with strain-stress plots created: {output_path_strain_stress}")
    print(f"PDF with pressure-deflection plots created: {output_path_pressure_deflection}")



def create_pdf_with_table_and_deflection_image(source_directory, output_directory, pressure_values, von_mises_stresses, strains, magnification, save_with_scales):
    image_types = ['plux_image.jpg', 'plux_stack.jpg', 'plux_z.jpg']
```

```
    files = {image_type: [] for image_type in image_types}

    for file_name in os.listdir(source_directory):
        for image_type in image_types:
            if file_name.endswith(image_type):
                files[image_type].append(os.path.join(source_directory, file_name))

    for image_type, file_list in files.items():
        files[image_type] = sorted(file_list, key=lambda x: int(re.search(r'(\d+)',
            os.path.basename(x)).group(1)))

    output_path = os.path.join(output_directory, "output.pdf")
    doc = SimpleDocTemplate(output_path, pagesize=letter, rightMargin=72, leftMargin=72, topMargin=18,
        bottomMargin=9)
    story = []
    styles = getSampleStyleSheet()

    logo_path = "H:\\01 Bulge Testing Versuche\\6mm_14mm\\logo.png"
    if os.path.exists(logo_path):
        logo = Image(logo_path, width=100, height=50)
    else:
        logo = Paragraph("<font color='red'><b>Empa Thun</b></font>", styles['Title'])
    title = Paragraph("<font size=12><b>Pressure and Deflection Report</b></font>", styles['Title'])
    datum = datetime.datetime.now().strftime("%Y-%m-%d %H-%M")

    header_data = [[logo, title, datum]]
    header_table = Table(header_data, colWidths=[108, 324, 108], rowHeights=60)
    header_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story.append(header_table)
    story.append(Spacer(1, 20))

    available_width = letter[0] - 60
    first_column_width = 70
    image_width = (available_width - first_column_width) / len(image_types) - (0.1 * inch)
    headers = ['P// '] + [img_type.replace('.jpg', '').replace('_', ' ').title() for img_type in
        image_types]
    table_data = [headers]

    num_rows = max(len(files[image_type]) for image_type in image_types)
    for index in range(num_rows):
        if index < len(pressure_values):
            pressure = f"{pressure_values[index]:.3f} bar"
            stress = f"{von_mises_stresses[index]:.3f} MPa"
            strain = f"{strains[index] * 100:.3f} [%]"
            row = [Paragraph(f"{pressure}<br/>{stress}<br/>{strain}", styles['Normal'])]
            for image_type in image_types:
                try:
                    img_path = files[image_type][index]
                    output_img_path = img_path.replace('.jpg', '_with_scales.jpg')
                    if save_with_scales:
                        if not os.path.exists(output_img_path):
                            add_scales_to_image(img_path, magnification, output_img_path)
                    else:
                        output_img_path = img_path
                    img = Image(output_img_path, width=image_width, height=image_width * 0.75)
```

```
                    img_name = Paragraph(f"<font size=9>{os.path.basename(output_img_path)}</font>", styles["Normal"])
                    row.append([img, img_name])
                except (IndexError, FileNotFoundError):
                    row.append('')
            table_data.append(row)

    table = Table(table_data, colWidths=[first_column_width] + [image_width] * len(image_types), style=TableStyle([
        ('INNERGRID', (0,0), (-1,-1), 0.25, colors.black),
        ('BOX', (0,0), (-1,-1), 0.25, colors.black),
        ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
        ('ALIGN', (0,0), (-1,-1), 'CENTER')
    ]))
    story.append(table)
    story.append(Spacer(1, 20))

    plots_path = os.path.join(output_directory, "Plots")
    if os.path.exists(plots_path):
        for plot_file in os.listdir(plots_path):
            plot_full_path = os.path.join(plots_path, plot_file)
            if plot_full_path.endswith('.jpg'):
                new_width, new_height = resize_image(plot_full_path, available_width, 270)
                plot_image = Image(plot_full_path, width=new_width, height=new_height)
                story.append(plot_image)
                story.append(Spacer(1, 12))

    pressure_deflection_image_path = os.path.join(output_directory, "Pressure_vs_Deflection.png")
    if os.path.exists(pressure_deflection_image_path):
        new_width, new_height = resize_image(pressure_deflection_image_path, available_width, 270)
        deflection_image = Image(pressure_deflection_image_path, width=new_width, height=new_height)
        story.append(deflection_image)

    story.append(Spacer(1, 20))

    footer_data = [[logo, '', datum]]
    footer_table = Table(footer_data, colWidths=[108, 324, 108], rowHeights=60)
    footer_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story.append(footer_table)

    doc.build(story)
    print(f"PDF created with table and Deflection image: {output_path}")


def create_pdf_with_processed_images_and_plots_with_strain_stress(source_directory, output_directory, pressure_values, von_mises_stresses, strains, magnification, save_with_scales):
    image_types = ['plux_image.jpg', 'processed_image.jpg']
    files = {image_type: [] for image_type in image_types}
    processed_images_dir = os.path.join(output_directory, 'processed_images')
    strain_stress_plots_dir = os.path.join(output_directory, 'Plots', 'Succesive_Plots')
    pressure_deflection_plots_dir = os.path.join(output_directory, 'Plots', 'Succesive_Plots_p_vs_d')

    for file_name in os.listdir(source_directory):
        if file_name.endswith('plux_image.jpg'):
            files['plux_image.jpg'].append(os.path.join(source_directory, file_name))

```

```python
    for file_name in os.listdir(processed_images_dir):
        if file_name.endswith('.jpg'):
            files['processed_image.jpg'].append(os.path.join(processed_images_dir, file_name))

    for image_type, file_list in files.items():
        files[image_type] = sorted(file_list, key=lambda x: int(re.search(r'(\d+)', os.path.basename(x)).group(1)))

    output_path = os.path.join(output_directory, "succesive_strain_stress_report.pdf")
    doc = SimpleDocTemplate(output_path, pagesize=letter, rightMargin=72, leftMargin=72, topMargin=18, bottomMargin=18)
    story = []
    styles = getSampleStyleSheet()

    logo_path = "H:\\01 Bulge Testing Versuche\\6mm_14mm\\logo.png"
    if os.path.exists(logo_path):
        logo = Image(logo_path, width=100, height=50)
    else:
        logo = Paragraph("<font color='red'><b>Empa Thun</b></font>", styles['Title'])
    title = Paragraph("<font size=12><b>Processed Images Report</b></font>", styles['Title'])
    datum = datetime.datetime.now().strftime("%Y-%m-%d %H-%M")

    header_data = [[logo, title, datum]]
    header_table = Table(header_data, colWidths=[108, 324, 108], rowHeights=60)
    header_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story.append(header_table)
    story.append(Spacer(1, 20))

    available_width = letter[0] - 60
    first_column_width = 70
    image_width = (available_width - first_column_width) / (len(image_types) + 1) - (0.1 * inch)
    headers = ['P// '] + [img_type.replace('.jpg', '').replace('_', ' ').title() for img_type in image_types] + ['Strain-Stress Curve']
    table_data = [headers]

    num_rows = max(len(files[image_type]) for image_type in image_types)
    for index in range(num_rows):
        if index < len(pressure_values):
            pressure = f"{pressure_values[index]:.2f} bar"
            stress = f"{von_mises_stresses[index]:.2f} MPa"
            strain = f"{strains[index] * 100:.2f} [%]"
            row = [Paragraph(f"{pressure}<br/>{stress}<br/>{strain}", styles['Normal'])]
            for image_type in image_types:
                try:
                    img_path = files[image_type][index]
                    output_img_path = img_path.replace('.jpg', '_with_scales.jpg')
                    if save_with_scales:
                        if not os.path.exists(output_img_path):
                            add_scales_to_image(img_path, magnification, output_img_path)
                    else:
                        output_img_path = img_path
                    img = Image(output_img_path, width=image_width, height=image_width * 0.75)
                    img_name = Paragraph(f"<font size=9>{os.path.basename(output_img_path)}</font>", styles["Normal"])
                    row.append([img, img_name])
                except (IndexError, FileNotFoundError):
```

```python
                    row.append('')
            try:
                strain_stress_plot_path = os.path.join(strain_stress_plots_dir, f'strain_stress_{index + 1}.png')
                strain_stress_img = Image(strain_stress_plot_path, width=image_width, height=image_width * 0.75)
                row.append(strain_stress_img)
            except (FileNotFoundError, IndexError):
                row.append('')

            table_data.append(row)

    table = Table(table_data, colWidths=[first_column_width] + [image_width] * (len(image_types) + 1), style=TableStyle([
        ('INNERGRID', (0,0), (-1,-1), 0.25, colors.black),
        ('BOX', (0,0), (-1,-1), 0.25, colors.black),
        ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
        ('ALIGN', (0,0), (-1,-1), 'CENTER')
    ]))
    story.append(table)
    story.append(Spacer(1, 20))

    plots_path = os.path.join(output_directory, "Plots")
    if os.path.exists(plots_path):
        for plot_file in os.listdir(plots_path):
            plot_full_path = os.path.join(plots_path, plot_file)
            if plot_full_path.endswith('.jpg'):
                new_width, new_height = resize_image(plot_full_path, available_width, 270)
                plot_image = Image(plot_full_path, width=new_width, height=new_height)
                story.append(plot_image)
                story.append(Spacer(1, 12))

    pressure_deflection_image_path = os.path.join(output_directory, "Pressure_vs_Deflection.png")
    if os.path.exists(pressure_deflection_image_path):
        new_width, new_height = resize_image(pressure_deflection_image_path, available_width, 270)
        deflection_image = Image(pressure_deflection_image_path, width=new_width, height=new_height)
        story.append(deflection_image)

    story.append(Spacer(1, 20))

    footer_data = [[logo, '', datum]]
    footer_table = Table(footer_data, colWidths=[108, 324, 108], rowHeights=60)
    footer_table.setStyle(TableStyle([
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
        ('SPAN', (1, 0), (1, 0))
    ]))
    story.append(footer_table)

    doc.build(story)
    print(f"PDF created with processed images, plots, and strain-stress curves: {output_path}")

    # Create second PDF with processed images and pressure-deflection plots
    output_path_deflection = os.path.join(output_directory, "processed_and_pressure_deflection_report.pdf")
    doc_deflection = SimpleDocTemplate(output_path_deflection, pagesize=letter, rightMargin=72, leftMargin=72, topMargin=18, bottomMargin=18)
    story_deflection = []

    title_deflection = Paragraph("<font size=12><b>Processed Images and Pressure-Deflection Plots Report</b></font>", styles['Title'])
```

```python
header_data_deflection = [[logo, title_deflection, datum]]
header_table_deflection = Table(header_data_deflection, colWidths=[108, 324, 108], rowHeights=60)
header_table_deflection.setStyle(TableStyle([
    ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
    ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
    ('SPAN', (1, 0), (1, 0))
]))
story_deflection.append(header_table_deflection)
story_deflection.append(Spacer(1, 20))

headers_deflection = ['P//', 'Processed Image', 'Pressure-Deflection Curve']
table_data_deflection = [headers_deflection]

for index in range(num_rows):
    if index < len(pressure_values):
        pressure = f"{pressure_values[index]:.2f} bar"
        stress = f"{von_mises_stresses[index]:.2f} MPa"
        strain = f"{strains[index] * 100:.2f} [%]"
        row = [Paragraph(f"{pressure}<br/>{stress}<br/>{strain}", styles['Normal'])]

        try:
            img_path = files['processed_image.jpg'][index]
            img = Image(img_path, width=image_width, height=image_width * 0.75)
            img_name = Paragraph(f"<font size=9>{os.path.basename(img_path)}</font>", styles["Normal"])
            row.append([img, img_name])
        except (IndexError, FileNotFoundError):
            row.append('')

        try:
            pressure_deflection_plot_path = os.path.join(pressure_deflection_plots_dir,
                 f'pressure_deflection_{index + 1}.png')
            pressure_deflection_img = Image(pressure_deflection_plot_path, width=image_width,
                 height=image_width * 0.75)
            row.append(pressure_deflection_img)
        except (FileNotFoundError, IndexError):
            row.append('')

        table_data_deflection.append(row)

table_deflection = Table(table_data_deflection, colWidths=[first_column_width] + [image_width] * 2,
     style=TableStyle([
    ('INNERGRID', (0,0), (-1,-1), 0.25, colors.black),
    ('BOX', (0,0), (-1,-1), 0.25, colors.black),
    ('VALIGN', (0,0), (-1,-1), 'MIDDLE'),
    ('ALIGN', (0,0), (-1,-1), 'CENTER')
]))
story_deflection.append(table_deflection)
story_deflection.append(Spacer(1, 20))

footer_data_deflection = [[logo, '', datum]]
footer_table_deflection = Table(footer_data_deflection, colWidths=[108, 324, 108], rowHeights=60)
footer_table_deflection.setStyle(TableStyle([
    ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
    ('VALIGN', (0, 0), (-1, -1), 'MIDDLE'),
    ('SPAN', (1, 0), (1, 0))
]))
story_deflection.append(footer_table_deflection)

doc_deflection.build(story_deflection)
print(f"PDF created with processed images and pressure-deflection plots: {output_path_deflection}")
```

```python
def resize_image(image_path, max_width, max_height):
    img = cv2.imread(image_path)
    original_width, original_height = img.shape[1], img.shape[0]
    ratio = min(max_width / original_width, max_height / original_height)
    new_width = int(original_width * ratio)
    new_height = int(original_height * ratio)
    return new_width, new_height

def find_nearest_time(df, target_time):
    idx = (np.abs(df['Time'] - target_time)).idxmin()
    return df.loc[idx, 'Time']

def plot_pressure_and_statistics(file_path, save_path, start_time=60, interval=180):
    plots_directory = os.path.join(save_path, 'Pressure_Handling')
    if not os.path.exists(plots_directory):
        os.makedirs(plots_directory)

    df = pd.read_csv(file_path, skiprows=2, header=None, names=['Time', 'Set pressure', 'Get pressure'])
    df['Difference'] = df['Set pressure'] - df['Get pressure']
    mean_diff = df['Difference'].mean()
    std_diff = df['Difference'].std()
    max_diff = df['Difference'].max()
    min_diff = df['Difference'].min()

    stats_text = (
        f"Mean difference: {mean_diff:.6f}\n"
        f"Standard deviation: {std_diff:.6f}\n"
        f"Maximum difference: {max_diff}\n"
        f"Minimum difference: {min_diff}"
    )

    print("Statistics for the difference between Set pressure and Get pressure:")
    print(stats_text)

    plt.figure(figsize=(12, 6))
    plt.plot(df['Time'], df['Set pressure'], label='Soll Druck (Set pressure)', color='blue', linestyle='-')
    plt.plot(df['Time'], df['Get pressure'], label='Ist Druck (Get pressure)', color='orange',
        linestyle='--')
    plt.xlabel('Zeit (Time) in s')
    plt.ylabel('Druck (Pressure) in bar')
    plt.title('Soll und Ist Druck ber Zeit')
    plt.legend()
    plt.grid(False)
    plt.text(0.98, 0.02, stats_text, transform=plt.gca().transAxes, fontsize=10, verticalalignment='bottom',
        horizontalalignment='right')
    plt.savefig(os.path.join(plots_directory, 'Soll_Ist_Druck_ueber_Zeit.png'))
    plt.show()

    df_short = df[df['Time'] <= 7200]

    plt.figure(figsize=(12, 6))
    plt.plot(df_short['Time'], df_short['Set pressure'], label='Soll Druck (Set pressure)', color='blue',
        linestyle='-')
    plt.plot(df_short['Time'], df_short['Get pressure'], label='Ist Druck (Get pressure)', color='orange',
        linestyle='--')
    plt.xlabel('Zeit (Time) in s')
```

```
    plt.ylabel('Druck (Pressure) in bar')
    plt.title('Soll und Ist Druck ber Zeit (bis 7200 s)')
    plt.legend()
    plt.grid(False)
    plt.text(0.98, 0.02, stats_text, transform=plt.gca().transAxes, fontsize=10, verticalalignment='bottom', horizontalalignment='right')
    plt.grid(False)
    plt.savefig(os.path.join(plots_directory, 'Soll_Ist_Druck_ueber_Zeit_bis_7200s.png'))
    plt.show()

    plt.figure(figsize=(12, 6))
    plt.plot(df['Time'], df['Difference'], label='Difference (Set - Get)', color='green', linestyle='-')
    plt.axhline(0, color='black', linewidth=0.5)
    plt.xlabel('Zeit (Time) in s')
    plt.ylabel('Differenz (Difference) in bar')
    plt.title('Unterschied zwischen Soll und Ist Druck ber Zeit')
    plt.legend()
    plt.grid(False)
    plt.text(0.98, 0.02, stats_text, transform=plt.gca().transAxes, fontsize=10, verticalalignment='bottom', horizontalalignment='right')
    plt.savefig(os.path.join(plots_directory, 'Unterschied_Soll_Ist_Druck_ueber_Zeit.png'))
    plt.show()

    time_points = np.arange(start_time, df['Time'].max(), interval)
    nearest_times = [find_nearest_time(df, t) for t in time_points]
    bar_values_time = df[df['Time'].isin(nearest_times)][['Time', 'Get pressure']]
    bar_values_time.to_csv(os.path.join(plots_directory, 'bar_values_time.csv'), index=False)
    print(f"Extracted values with time saved to {save_path}/bar_values_time.csv")

    bar_values = bar_values_time[['Get pressure']]
    bar_values.to_csv(os.path.join(plots_directory, 'bar_values.csv'), header=False, index=False)
    print(f"Extracted values saved to {save_path}/bar_values.csv")

    extended_values_time = []
    for t in time_points:
        nearest_time = find_nearest_time(df, t)
        avg_pressure = df[(df['Time'] >= nearest_time) & (df['Time'] < nearest_time + 5)]['Get pressure'].mean()
        extended_values_time.append({'Time': nearest_time, 'Avg Get pressure': avg_pressure})
    extended_df_time = pd.DataFrame(extended_values_time)
    extended_df_time.to_csv(os.path.join(plots_directory, 'bar_values_extended_time.csv'), index=False)
    print(f"Extended values with time saved to {save_path}/bar_values_extended_time.csv")

    extended_values = extended_df_time[['Avg Get pressure']]
    extended_values.to_csv(os.path.join(plots_directory, 'bar_values_extended.csv'), header=False, index=False)
    print(f"Extended values saved to {save_path}/bar_values_extended.csv")


def create_video_with_combined_plots(image_directory, analysis_directory, output_directory, pressure_values, von_mises_stresses, strains, fps=1):
    combined_images_dir = os.path.join(output_directory, 'combined_images')
    os.makedirs(combined_images_dir, exist_ok=True)

    original_files = sorted([f for f in os.listdir(image_directory) if f.endswith('plux_image.jpg')])
    combined_images = []

    for idx, original_file in enumerate(original_files):
        img_path = os.path.join(image_directory, original_file)
        if idx < len(pressure_values):
```

```
            img = cv2.imread(img_path)
            if img is None:
                continue

            # Add overlay text
            font = cv2.FONT_HERSHEY_SIMPLEX
            bar_text = f"{pressure_values[idx]:.5f} bar"
            stress_text = f"{von_mises_stresses[idx]:.2f} MPa"
            strain_text = f"{strains[idx] * 100:.2f} [%]"
            overlay_text = f"{bar_text} | {stress_text} | {strain_text}"
            cv2.putText(img, overlay_text, (50, 50), font, 1, (255, 255, 255), 2, cv2.LINE_AA)

            # Read corresponding strain-stress image
            strain_stress_path = os.path.join(analysis_directory, 'Plots', 'Succesive_Plots',
                 f'strain_stress_{idx + 1}.png')
            if not os.path.exists(strain_stress_path):
                continue
            strain_stress_img = cv2.imread(strain_stress_path)
            if strain_stress_img is None:
                continue

            # Read corresponding pressure-deflection image
            pressure_deflection_path = os.path.join(analysis_directory, 'Plots', 'Succesive_Plots_p_vs_d',
                 f'pressure_deflection_{idx + 1}.png')
            if not os.path.exists(pressure_deflection_path):
                continue
            pressure_deflection_img = cv2.imread(pressure_deflection_path)
            if pressure_deflection_img is None:
                continue

            # Resize images while maintaining aspect ratio
            height, width, _ = img.shape
            strain_stress_img = cv2.resize(strain_stress_img, (int(strain_stress_img.shape[1] * height /
                 strain_stress_img.shape[0]), height))
            pressure_deflection_img = cv2.resize(pressure_deflection_img,
                 (int(pressure_deflection_img.shape[1] * height / pressure_deflection_img.shape[0]), height))

            # Layout: Pressure-Deflection on Left, Plux Image in the Middle, Strain-Stress on Right
            total_width = pressure_deflection_img.shape[1] + width + strain_stress_img.shape[1]
            canvas = np.ones((height, total_width, 3), dtype=np.uint8) * 255

            # Place images on the canvas
            canvas[:, :pressure_deflection_img.shape[1], :] = pressure_deflection_img
            canvas[:, pressure_deflection_img.shape[1]:pressure_deflection_img.shape[1] + width, :] = img
            canvas[:, pressure_deflection_img.shape[1] + width:, :] = strain_stress_img

            combined_img_path = os.path.join(combined_images_dir, f"combined_{idx + 1}.jpg")
            cv2.imwrite(combined_img_path, canvas)
            combined_images.append(combined_img_path)

            print(f"Processed combined image {idx + 1} with overlay: {overlay_text}")

    if not combined_images:
        shutil.rmtree(combined_images_dir)
        print("No valid combined images were created.")
        return

    print(f"Combined images saved at: {combined_images_dir}")

    # Create a video from the combined images
```

```
    video_output_path = os.path.join(output_directory, 'output_video_combined.mp4')
    clip = ImageSequenceClip(combined_images, fps=fps)
    clip.write_videofile(video_output_path, codec='libx264')

    print(f"Video created at {video_output_path}")


def create_video_with_bar_overlay(image_directory, output_video_path, pressure_values, von_mises_stresses, strains, fps=1):
    temp_dir = os.path.join(image_directory, 'temp_images')
    os.makedirs(temp_dir, exist_ok=True)

    original_files = sorted([f for f in os.listdir(image_directory) if f.endswith('plux_image.jpg')])
    renamed_files = []
    for idx, original_file in enumerate(original_files):
        new_filename = f"{idx + 1}_image.jpg"
        src_path = os.path.join(image_directory, original_file)
        dst_path = os.path.join(temp_dir, new_filename)
        shutil.copy(src_path, dst_path)
        renamed_files.append(dst_path)

    print("Total images found:", len(original_files))
    print("Total renamed files:", len(renamed_files))
    print("Total pressure values:", len(pressure_values))

    clips = []
    for idx, img_path in enumerate(renamed_files):
        if idx < len(pressure_values):
            img = cv2.imread(img_path)
            if img is None:
                print(f"Failed to read image: {img_path}")
                continue
            font = cv2.FONT_HERSHEY_SIMPLEX
            bar_text = f"{pressure_values[idx]:.5f} bar"
            stress_text = f"{von_mises_stresses[idx]:.2f} MPa"
            strain_text = f"{strains[idx] * 100:.2f} [%]"
            overlay_text = f"{bar_text} | {stress_text} | {strain_text}"
            cv2.putText(img, overlay_text, (50, 50), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
            cv2.imwrite(img_path, img)
            clips.append(img_path)
            print(f"Processed image {idx+1} with overlay: {overlay_text}")

    if not clips:
        print("No valid clips were created.")
        return

    clip = ImageSequenceClip(clips, fps=fps)
    clip.write_videofile(output_video_path, codec='libx264')
    shutil.rmtree(temp_dir)
    print(f"Video created at {output_video_path}")



def extract_bar_values_from_filenames(source_directory):
    bar_values = []
    regex = re.compile(r'(\d+)_([\d.]+)mbar\.plux$')

    for file_name in os.listdir(source_directory):
        match = regex.search(file_name)
        if match:
```

```
            bar_value = float(match.group(2)) / 1000 # Convert mbar to bar
            bar_values.append(bar_value)

    bar_values = sorted(bar_values)
    return bar_values


def main():
    source_directory = input("Please enter the path to the source directory: ")
    automatic_mode = input("Was the measurement done automatically? (y/n): ").lower() == 'y'
    radius_of_window = int(input("Please enter the radius of window in m: "))
    thickness = float(input("Please enter the thickness (t) in micrometers: "))
    nu = float(input("Please enter Poisson's ratio (): "))
    magnification = int(input("Please enter the magnification (10, 20, 50, 150): "))
    save_with_scales = input("Do you want to save images with scales? (y/n): ").lower() == 'y'

    nullpunkt = None
    if automatic_mode:
        set_nullpunkt = input("Do you want to set the NULLPUNKT? (y/n): ")
        if set_nullpunkt.lower() == 'y':
            nullpunkt = float(input("Please enter the NULLPUNKT value in mm: ")) * 1000

    create_video = input("Do you want to create a video from the images? (y/n): ").lower() == 'y'
    if create_video:
        fps = int(input("Please enter the desired FPS for the video: "))

    do_crack_analysis = input("Do you want to do a crack analysis? (y/n): ").lower() == 'y'

    max_bar_value_elastic = float(input("Please enter the maximum bar value for the elastic regime: "))

    current_time = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M")
    output_directory = os.path.join(source_directory, f'analysis_{current_time}')
    os.makedirs(output_directory, exist_ok=True)

    max_values_csv = os.path.join(source_directory, 'addmax.csv')

    z_values_original, missing_indices = extract_and_save_z_values(source_directory, automatic_mode)

    if nullpunkt is not None:
        z_values_original.insert(0, nullpunkt)
        missing_indices = [i + 1 for i in missing_indices]
    print(f"Missing indices are {missing_indices}")

    csv_file_path = None
    for file_name in os.listdir(source_directory):
        if file_name.endswith('.csv'):
            with open(os.path.join(source_directory, file_name), 'r') as file:
                first_line = file.readline().strip()
                if first_line.startswith('Bulge'):
                    csv_file_path = os.path.join(source_directory, file_name)
                    break

    use_txt_data = False

    if csv_file_path:
        start_time = int(input("Please enter the start time for pressure intervals in seconds: "))
        interval = int(input("Please enter the interval for pressure extraction in seconds: "))

        plot_pressure_and_statistics(csv_file_path, output_directory, start_time, interval)
        bar_values_path = os.path.join(os.path.join(output_directory, 'Pressure_Handling'), 'bar_values.csv')
```

```
        bar_values = pd.read_csv(bar_values_path, header=None).squeeze().tolist()
        if nullpunkt is not None:
            bar_values.insert(0, 0)

        use_txt_data = input("Do you want to use the set bar values from the fix-ramp-TXT file even though the CSV file exists? (y/n): ").lower() == 'y'

    if use_txt_data and automatic_mode or ((not csv_file_path) and automatic_mode):
        text_file_path = None
        for file_name in os.listdir(source_directory):
            if file_name.endswith('.txt'):
                with open(os.path.join(source_directory, file_name), 'r') as file:
                    first_line = file.readline().strip()
                    if first_line.startswith('fix'):
                        text_file_path = os.path.join(source_directory, file_name)
                        break

        if text_file_path:
            try:
                bar_values = parse_fix_entries(text_file_path)
                if nullpunkt is not None:
                    bar_values.insert(0, 0)
            except FileNotFoundError:
                print("Data file not found. Please check the file path and try again.")
                return

            pressure_handling_directory = os.path.join(output_directory, 'Pressure_Handling')
            os.makedirs(pressure_handling_directory, exist_ok=True)
            bar_values_extracted = np.array(bar_values)
            df = pd.DataFrame(bar_values_extracted, columns=['Avg Get pressure'])
            bar_values_path = os.path.join(pressure_handling_directory, 'bar_values_set.csv')
            df.to_csv(bar_values_path, header=False, index=False)
            print(f"Extracted bar_values saved to {bar_values_path}")
        else:
            print("No suitable data file found. Please check the directory and try again.")
            return
    elif not automatic_mode:
        bar_values = extract_bar_values_from_filenames(source_directory)
        print("Extracted bar values:", bar_values) # Debug print

    for index in sorted(missing_indices, reverse=True):
        del bar_values[index]
        #if index < len(bar_values):
            #del bar_values[index]

    if max_bar_value_elastic is not None:
        for i, value in enumerate(bar_values):
            if value > max_bar_value_elastic:
                bar_values_elastic = bar_values[:i]
                break
        else:
            bar_values_elastic = bar_values # Ensure it is assigned even if all values are less than max_bar_value_elastic
    else:
        bar_values_elastic = bar_values

    if os.path.exists(max_values_csv):
        df_max_values = pd.read_csv(max_values_csv, delimiter=';')
        max_values = df_max_values['Value 0'].fillna(0).tolist()
        if nullpunkt is not None and max_values:
```

```python
            max_values.insert(0, 0)
        max_values = (max_values + [0] * len(z_values_original))[:len(z_values_original)]
        z_values_modified = [z + m for z, m in zip(z_values_original, max_values)]
        pd.DataFrame(z_values_original, columns=['Z Positions']).to_csv(os.path.join(output_directory, 'z_positions.csv'), index=False)
        pd.DataFrame(z_values_modified, columns=['Z Positions']).to_csv(os.path.join(output_directory, 'z_positions_modified.csv'), index=False)
    else:
        z_values_modified = z_values_original.copy()
        pd.DataFrame(z_values_original, columns=['Z Positions']).to_csv(os.path.join(output_directory, 'z_positions.csv'), index=False)

    if not bar_values:
        print("No bar values found. Exiting.")
        return



    max_bar_value_analysis = input("Do you want to set a maximum bar value for the analysis of the strain-stress-curve? (y/n): ").lower() == 'y'
    if max_bar_value_analysis:
        max_bar_value = float(input("Please enter the maximum bar value to include: "))
        bar_values_analysis = [value for value in bar_values if value <= max_bar_value]
    else:
        bar_values_analysis = bar_values

    bar_values_extracted = np.array(bar_values)
    pressure_handling_directory = os.path.join(output_directory, 'Pressure_Handling')
    os.makedirs(pressure_handling_directory, exist_ok=True) # Ensure the directory exists
    df = pd.DataFrame(bar_values_extracted, columns=['Avg Get pressure'])
    bar_values_path = os.path.join(pressure_handling_directory, 'bar_values_missing_indices.csv')
    df.to_csv(bar_values_path, header=False, index=False)
    print(f"Extracted bar_values saved to {bar_values_path}")

    while True:
        sigma_0, E_modul_Nix = analyze_fitting(z_values_modified, bar_values_elastic, radius_of_window, output_directory, thickness, nu, max_bar_value_elastic)
        analyze_dynamic_check(z_values_modified, bar_values_analysis, radius_of_window, output_directory, thickness, E_modul_Nix, sigma_0, missing_indices)

        # Ask user if they are happy with the fitting plot
        happy = input("Are you satisfied with the fitting plot? (y/n): ").lower()
        if happy == 'y':
            break
        else:
            max_bar_value_elastic = float(input("Please enter a new maximum bar value for the elastic regime: "))
            for i, value in enumerate(bar_values):
                if value > max_bar_value_elastic:
                    bar_values_elastic = bar_values[:i]
                    break
            else:
                bar_values_elastic = bar_values


    analyze_dynamic_with_uncertainties_and_sympy(z_values_modified, bar_values_analysis, radius_of_window, output_directory, thickness, E_modul_Nix, sigma_0)
    von_mises_stresses_unfitted, strains_unfitted = analyze_dynamic(z_values_modified, bar_values_analysis, radius_of_window, output_directory, thickness, E_modul_Nix, sigma_0, missing_indices)

```

```
    if nullpunkt is not None:
        create_pdf_with_table_and_strain_stress(source_directory, output_directory, bar_values_analysis[1:],
            von_mises_stresses_unfitted[1:], strains_unfitted[1:], magnification, save_with_scales)
        create_pdf_with_table_and_deflection_image(source_directory, output_directory,
            bar_values_analysis[1:], von_mises_stresses_unfitted[1:], strains_unfitted[1:], magnification,
            save_with_scales)
    else:
        create_pdf_with_table_and_strain_stress(source_directory, output_directory, bar_values_analysis,
            von_mises_stresses_unfitted, strains_unfitted, magnification, save_with_scales)
        create_pdf_with_table_and_deflection_image(source_directory, output_directory, bar_values_analysis,
            von_mises_stresses_unfitted, strains_unfitted, magnification, save_with_scales)

    if do_crack_analysis:
        processed_images_path = os.path.join(output_directory, 'processed_images')
        if not os.path.exists(processed_images_path):
            process_images(source_directory, output_directory, von_mises_stresses_unfitted, magnification,
                plot_fractions=True)
        if nullpunkt is not None:
            create_pdf_with_processed_images_and_plots(source_directory, output_directory,
                bar_values_analysis[1:], von_mises_stresses_unfitted[1:], strains_unfitted[1:],
                magnification, save_with_scales)
            create_pdf_with_processed_images_and_plots_with_strain_stress(source_directory, output_directory,
                bar_values_analysis[1:], von_mises_stresses_unfitted[1:], strains_unfitted[1:],
                magnification, save_with_scales)
        else:
            create_pdf_with_processed_images_and_plots(source_directory, output_directory,
                bar_values_analysis, von_mises_stresses_unfitted, strains_unfitted, magnification,
                save_with_scales)
            create_pdf_with_processed_images_and_plots_with_strain_stress(source_directory, output_directory,
                bar_values_analysis, von_mises_stresses_unfitted, strains_unfitted, magnification,
                save_with_scales)

    if create_video:
        if nullpunkt is not None:
            create_video_with_bar_overlay(source_directory, os.path.join(output_directory,
                'output_video.mp4'), bar_values_analysis[1:], von_mises_stresses_unfitted[1:],
                strains_unfitted[1:], fps)
            create_video_with_combined_plots(source_directory, output_directory,
                os.path.join(output_directory, 'output_video_strain_stress.mp4'), bar_values_analysis[1:],
                von_mises_stresses_unfitted[1:], strains_unfitted[1:], fps)
            print("Video has been created.")
        else:
            create_video_with_bar_overlay(source_directory, os.path.join(output_directory,
                'output_video.mp4'), bar_values_analysis, von_mises_stresses_unfitted, strains_unfitted, fps)
            create_video_with_combined_plots(source_directory, output_directory,
                os.path.join(output_directory, 'output_video_strain_stress.mp4'), bar_values_analysis,
                von_mises_stresses_unfitted, strains_unfitted, fps)
            print("Video has been created.")

if __name__ == "__main__":
    main()
```

D Bulge Test Manual

The following manual serves as a practical guide, detailing the necessary equipment setup, system configurations, and procedural steps for conducting a bulge test with accuracy and consistency.

This manual assumes familiarity with the Sensofar Confocal Microscope, as well as the theoretical background provided in earlier sections of the thesis. For specific operational details, please refer to the relevant chapters.

Preparation

1. **Start Sensofar 6.7:**
   - Switch on the Sensofar device
   - Open Sensofar 6.7
   - Use the password: Adm1234
   - Set to 3D Automode (See Fig. 1)
2. **Install the Fit:** Place the Aluminium-Plate covered with a 3D-printed template without the bulge setup

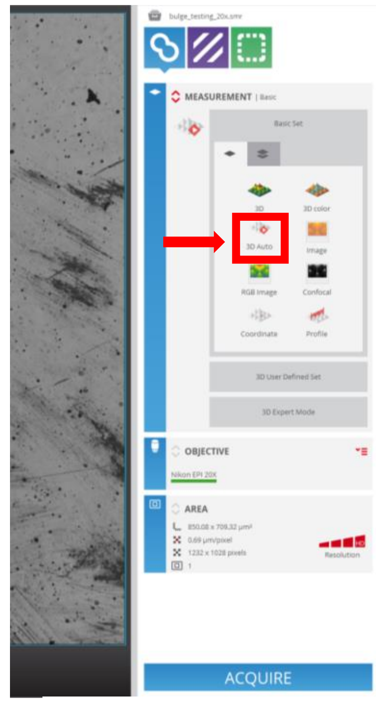
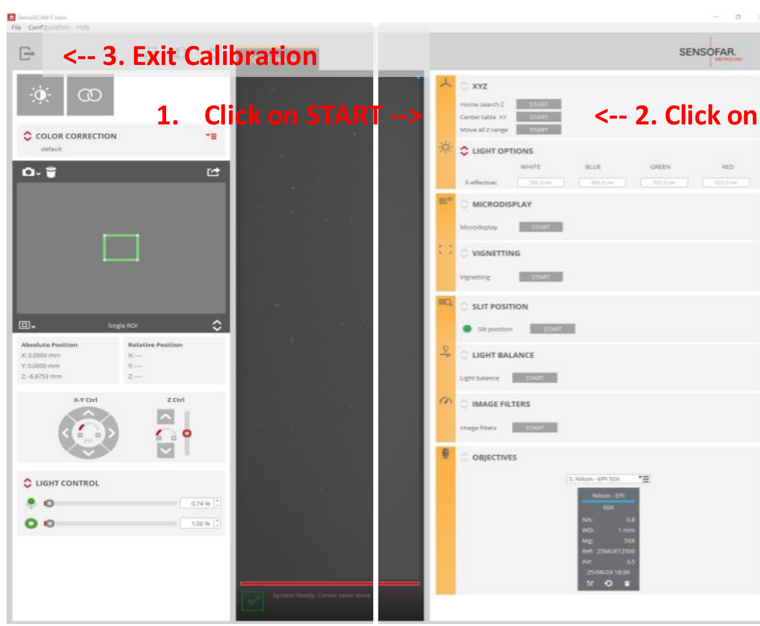


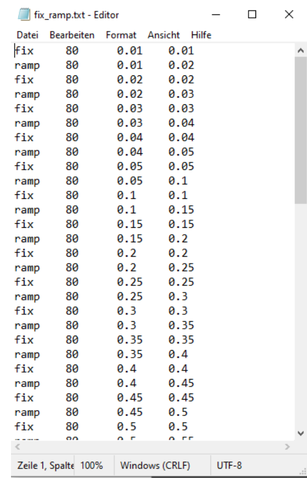
Figure 1: Setting Automode

OPTIONAL:
- Open Configuration menu
- Select Center Table XY Start (1)
- After calibration, press Finish button (2)
- Exit the Calibration screen (3)



3. **Create a Folder for Measurements:** For example, name it 2024-07-18-150x (Date-Objective type)

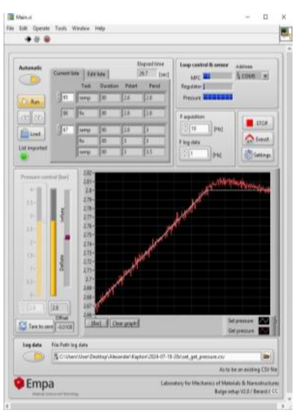
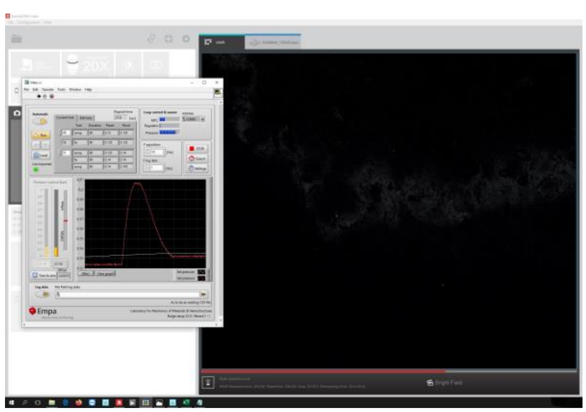
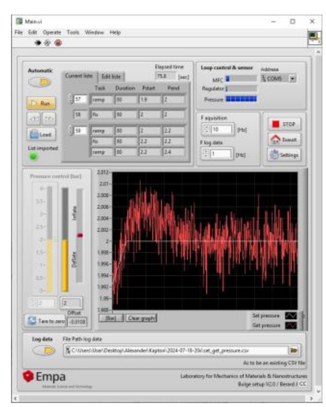
4. **Insert Fix-Ramp.txt File in Folder:**



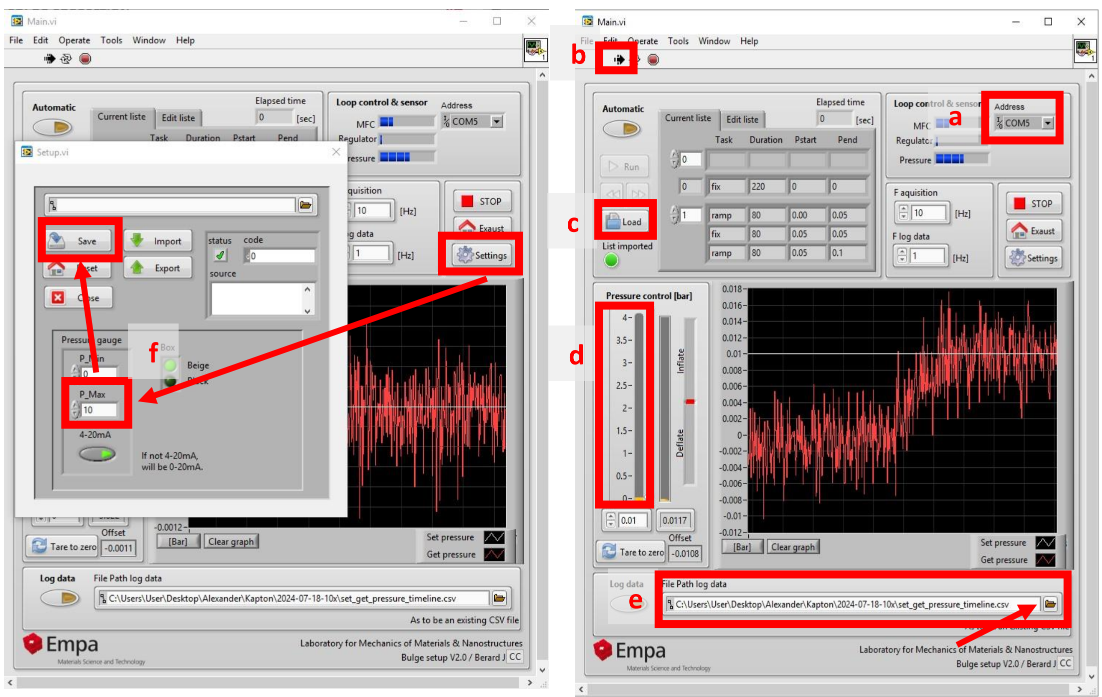
- Create the file manually or use the Python Script 'fix_ramp_creator.py' This file defines the intervals for pressure build-up
- Sum and record the duration of the Fix (hold phase) and Ramp (pressure increase phase). E.g., if Fix is 100 sec and Ramp is 20 sec, record 120 sec in the experiment protocol sheet
- Note the number of hold phases in the experiment sheet. The number of hold phases (fix-phases) will be noted in the file-name if the 'fix_ramp_creator.py' was used

5. **Create a .csv File in the Folder:**

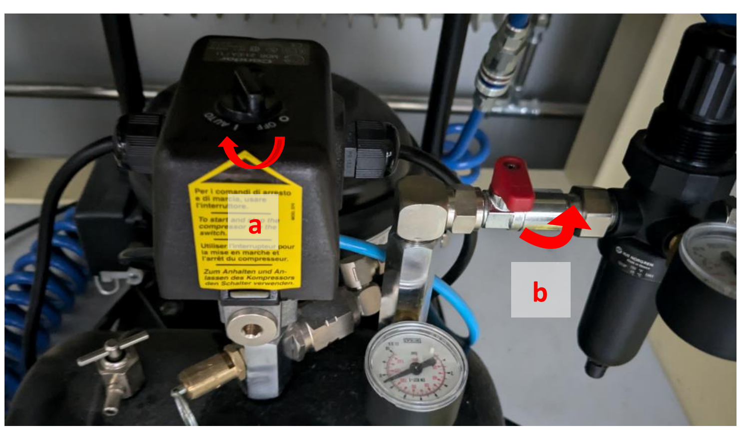
- Create a text file and rename it to .csv as needed, e.g., set_get_pressure_timeline.csv
- The fix-ramp text file sets the target pressure. Due to possible deviations up to 0.02 bar and overshooting, this file is needed to know the actual pressure during the measurement

6. **Start Inficon Control Program:**
   a. Select 'COM5'
   b. click on the 'Start' Icon 
   c. Load the .txt –file (fix-ramp)
   d. Adjust the bar scale for desired experimental range by clicking on the upper value and modifying it
   e. Load the .csv file
   f. Set Settings to 10 bar if using a 0-10 bar sensor and save adjustment

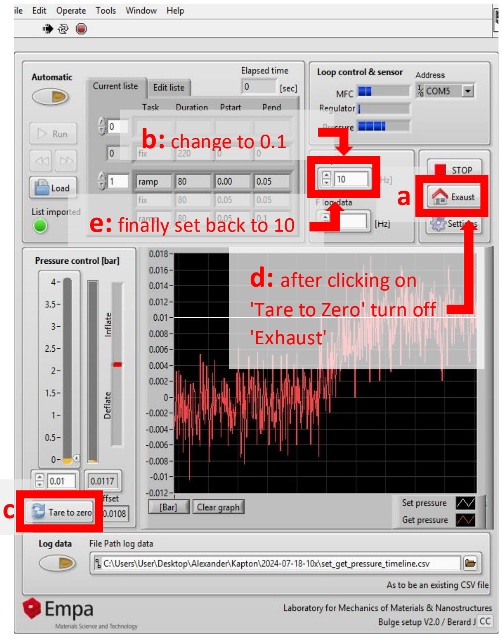


7. **Turn On Compressor:** Switch from OFF to AUTO (a) and open the valve (b)



8. **Set Exhaust and Frequency:**

    a. Press Exhaust button
    b. Change Frequency from 10 to 0.1
    c. Wait briefly, then press Tare to Zero
    d. Ensure Exhaust button is off
    e. Set Frequency back to 10

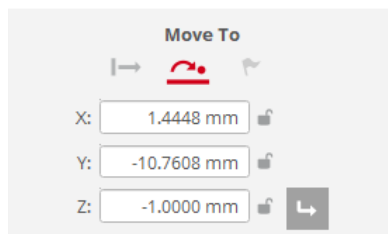


9. **Place Sample on Bulge Setup:**

    - Use a 20mm-diameter sample for the 14mm-diameter bulge setup and a 13mm-diameter sample for the 6mm-diameter bulge setup)
    - Place the lid on without tightening screws (ensure screws are in place but not touching the objectives)

10. **Focus Using 10x Objective:**

    - Use the 'Move To' Function of Sensofar to move to center. Use the absolute coordinates by clicking on and unlock the fields
    - Input absolute x, y and z positions:
        - x: 1.4448
        - y: -10.7608
        - z: -1.000



11. **Focus and Record Zero Point**:

    - Focus the sample and note z-value in the experiment sheet under "Zero Point" at "Lid unstressed 0 bar"
    - Save the image, e.g., 10x_unstressed_0bar.plux and afterwards export images.
    - If measurements with 50x or 150x are desired, move up in z-direction so that the longer 50x and 150x objectives can be selected without touching the lid
    - Refocus and save the image, e.g., 150x_unstressed_0bar.plux

12. **Tighten Screws on Bulge Setup:**

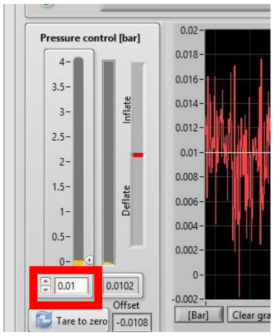
    - Remove setup and tighten screws with a torque wrench: 8-10 Nm (preferably 10 Nm to prevent leaking, even though it induces additional stress to the sample)
    - Tighten screws alternately in a cross pattern for uniform tension

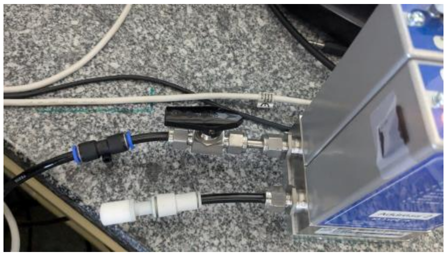
13. **Replace Bulge Setup and Record Zero Point**:

    - Place setup back and refocus
    - Note z-value in the experiment sheet under "Zero Point" at "Lid stressed 0 bar"
    - Optionally, save the image, e.g., 10x_stressed_0bar.plux

14. **Set Inflow Valve**:

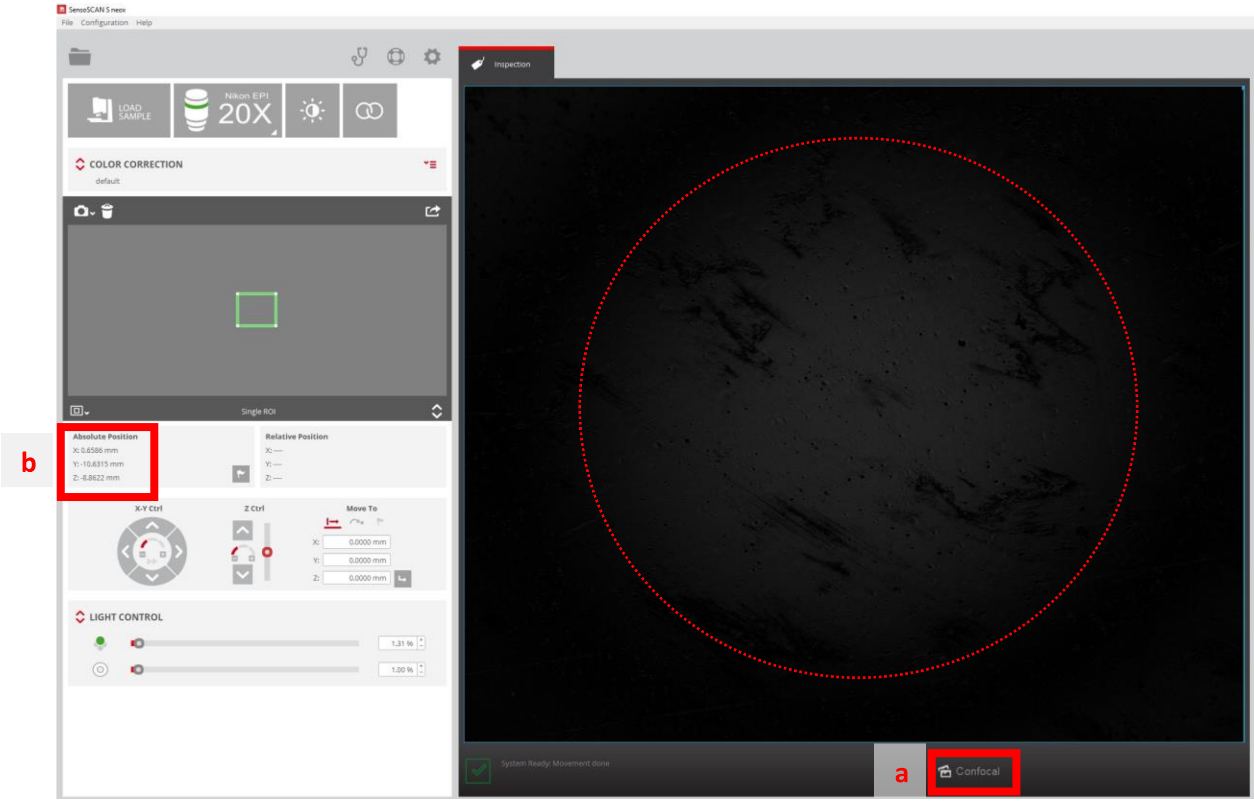
- Ensure the inflow valve is open (See Fig. 5)
- Set the bar to 0.01 bar in Inficon (See Fig. 6); for 0-1 bar pressure sensor, set a lower pressure, e.g., 0.005 bar





15. **Switch to Confocal Mode**:

a. Change from Bright Field Mode to Confocal Mode (See Fig. 7a)
b. Ensure the circle is centered (See Fig. 7b) and record coordinates in the experiment sheet

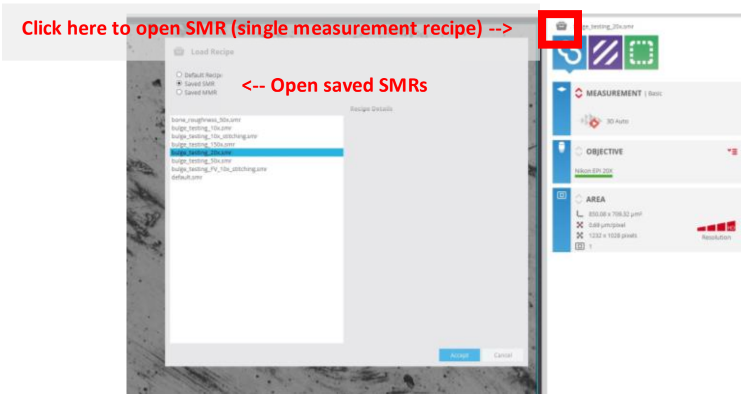


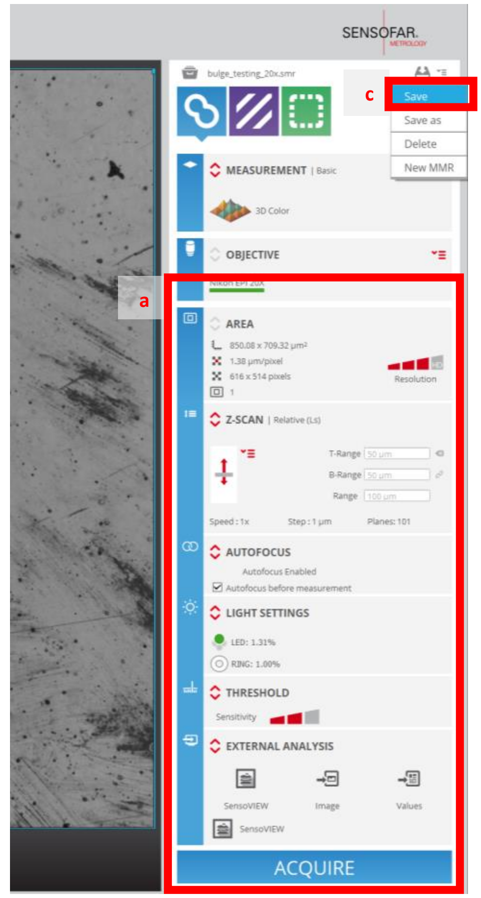
16. **Focus and Record Zero Point at Pressure**:

- Refocus and note z-value in the experiment sheet under "Zero Point" at "Lid stressed [blank] bar"
- Save the image, e.g., 10x_stressed_10mbar.plux

Measurement

1. **Switch from Automode to SMR Recipe**: Select the desired SMR recipe and objective (See Fig. 8)

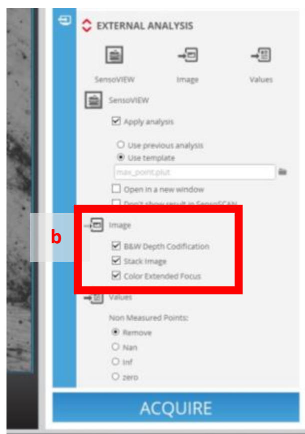




   a. Adjust Parameters if needed

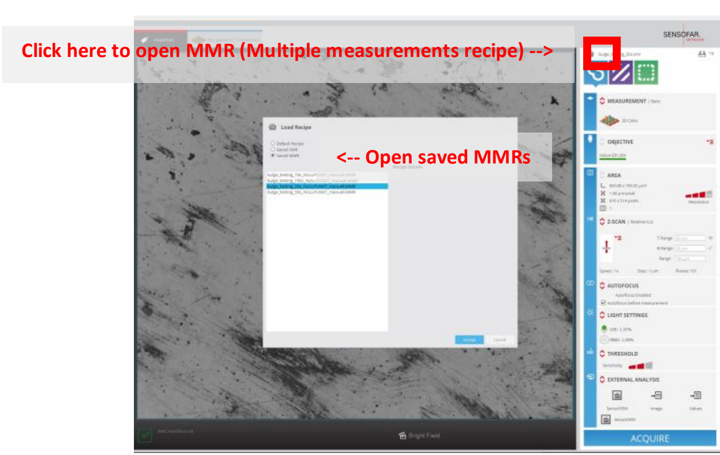
   b. Make sure images are exported

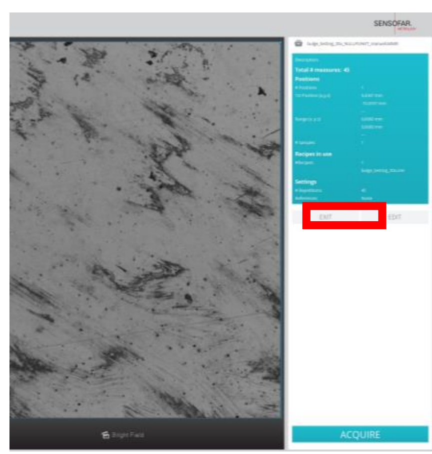
   c. Save the settings



2. **Open and Define MMR**:

   Open the MMR (Image left) and define it (Image right)





a. Enter x and y center values determined from the Confocal mode and recorded in the experiment sheet

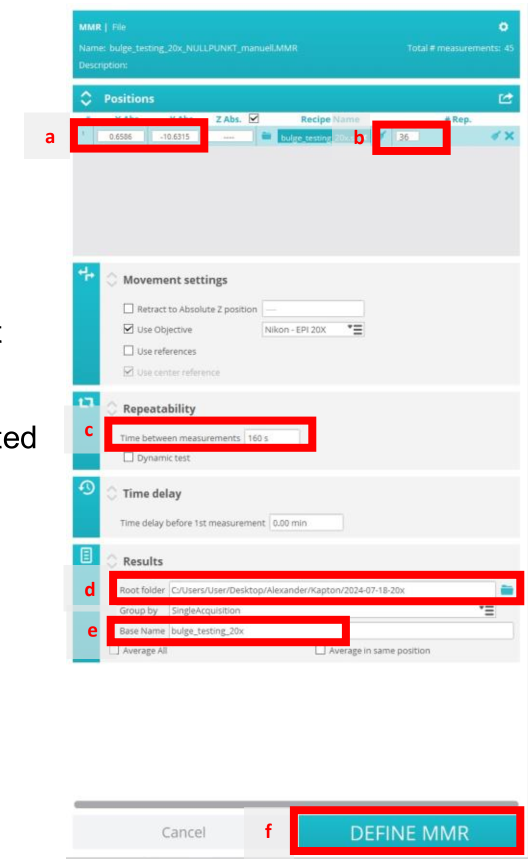
b. Enter the number of repetitions

c. Set Repeatability to the sum of Fix and Ramp phases recorded in the experiment sheet

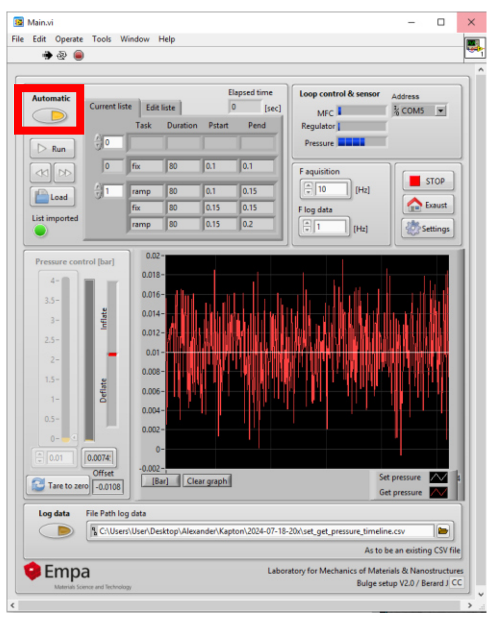
d. Select the save location (the initially created folder)

e. Change Base Name if whished

f. Define the MMR



3. **Activate Automatic Mode in Inficon**: Enable Automatic Mode (button lights up and manual pressure input field disappears)



4. **Log Data and Run**:

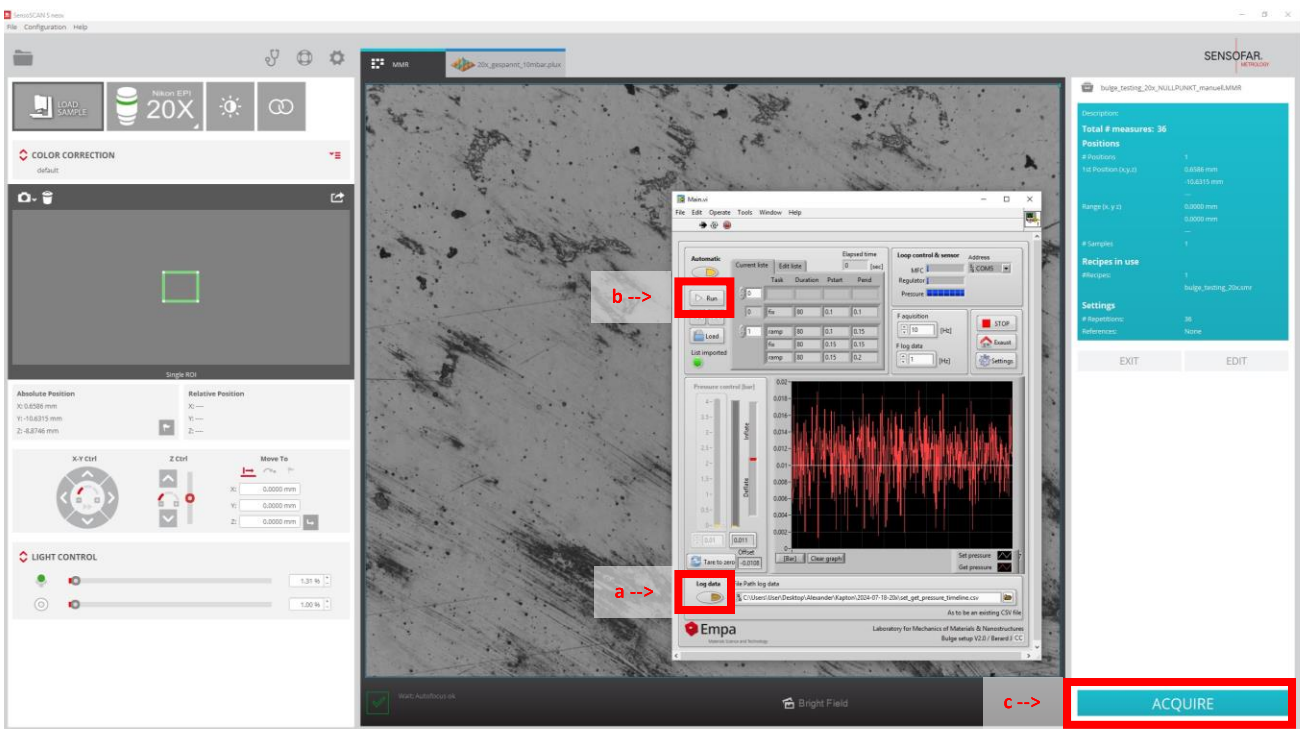
   Critical: The next two steps (a and b) must be carried out immediately one after the other, then click on Aquire (c) after 60 seconds

   a. Click 'Log Data'

   b. and 'Run' consecutively

   c. then <u>after at least 30 seconds</u> click 'Aquire' (during the first measurement, several windows will open, always click Accept). It is recommended to start after 60 seconds because of the before-mentioned problem with pressure overshooting. Record the time, the interval between 'Run' and 'Aquire' in the experiment sheet!

   Important: Do not open the .csv file (e.g., set_get_pressure_timeline.csv) while the measurement is running



Post-Measurement

1. **Move .txt and .csv file to folder with all recordings**: Copy both fix-ramp.txt and .csv files to the folder with all recordings (.plux files)
2. **Create Template to read out max**-value:

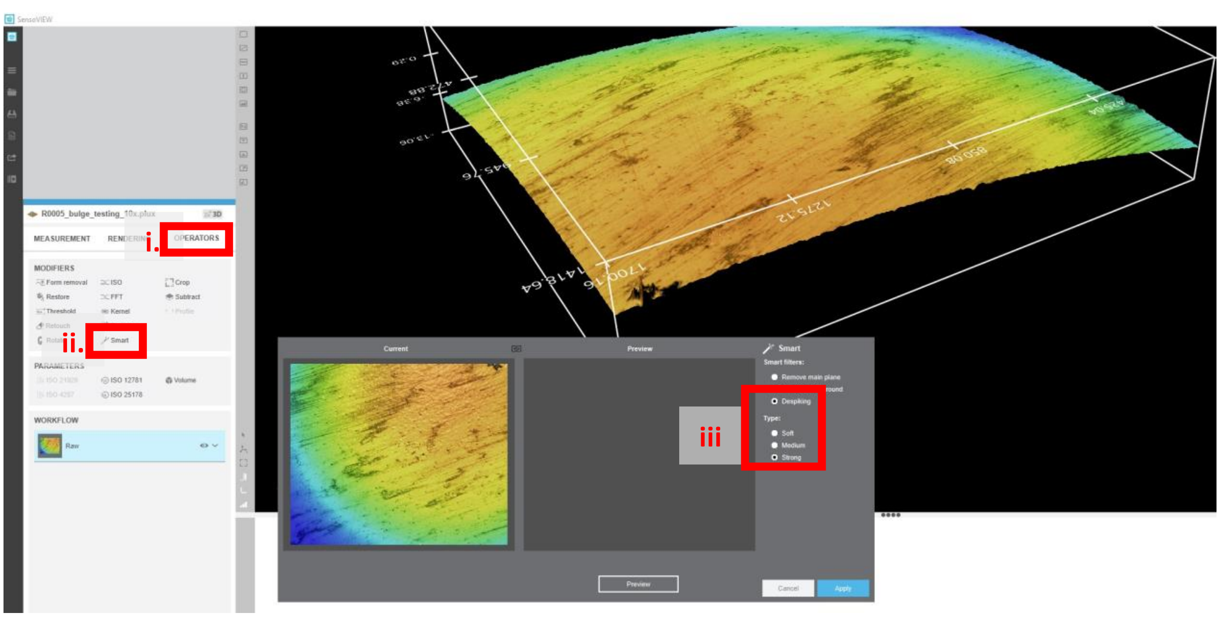
If the measurement was done with 150x, this step can be skipped, but if the user wants extra precision, it should be included. The deviation from the maximum value is under 600 nm (valid for a material with a stiffness of 2.7 GPa up to an applied pressure of 3.5 bar. For pressures smaller than 3.5 bar the deviation is even smaller, for pressure higher than 3.5 bar it is getting higher).

Important Note: The z-value in the XML file is not the max-value. However, the null line within a .plux file can help identify the point with the maximum deflection. To get this point, a template needs to be created that reads out the max value and then applies this template to all recordings (all saved .plux files). Depending on the material, Sensofar may create artificial spikes (artifacts), which must first be removed to avoid getting an artificial max point.

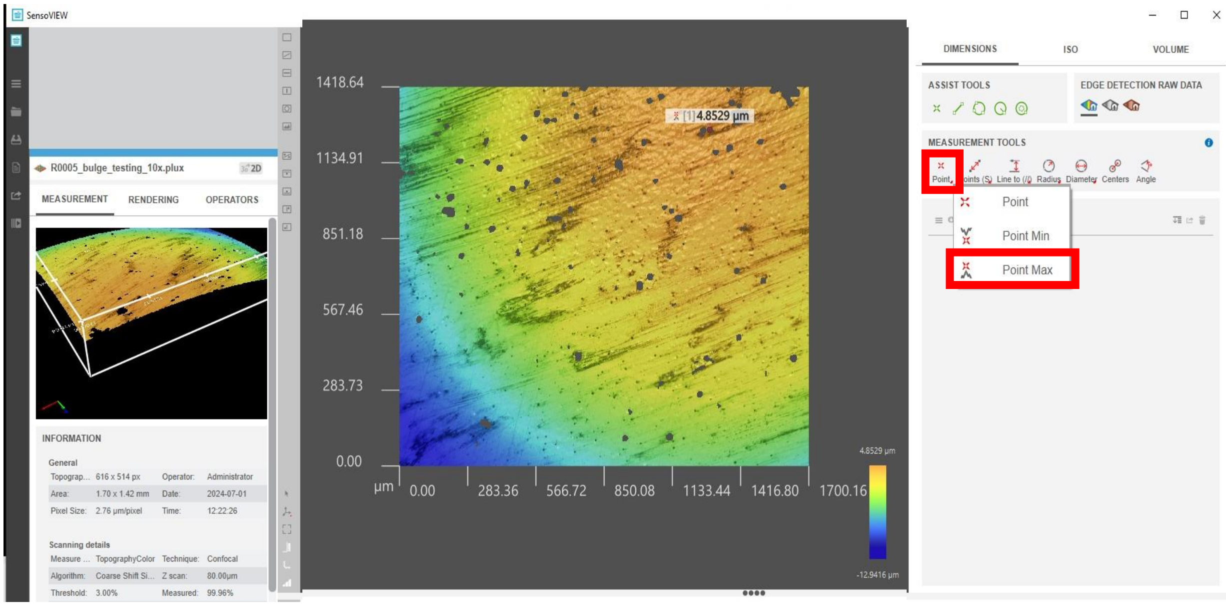
Instructions:

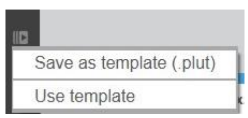
1. Open any .plux recording:
2. Optional: Despiking (if extra precision is desired):
   i. Go to Operators
   ii. Choose Modifier 'Smart' 

   then select 'Despiking'
   iii. Depending on Severity of Spiking choose either 'Soft', 'Medium' or 'Strong' and click 'Apply'

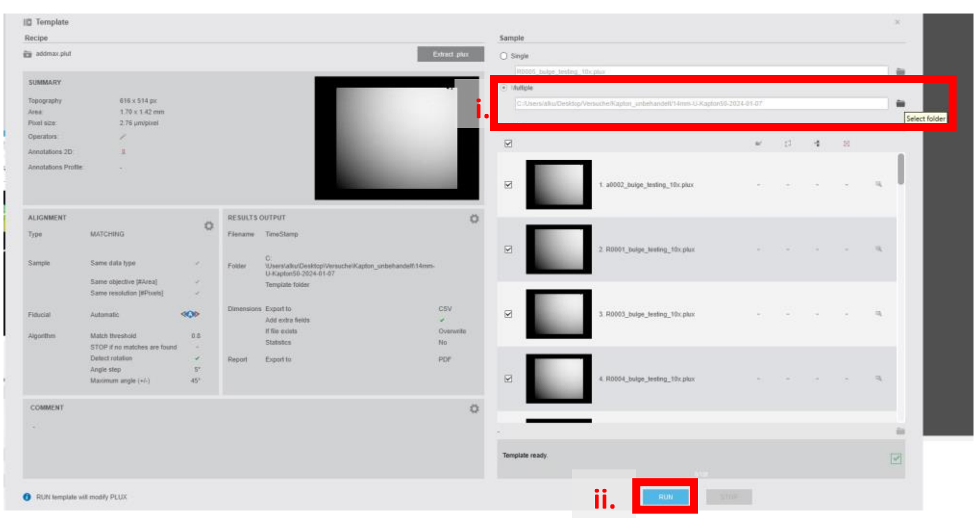


3. Go to Dimensions, **right-click** on Point and choose 'Point Max'



4. Save Template: 

5. Save the template as addmax.plut
6. Run the template and in the Sample (Multiple) field
   i. choose the folder where all the recordings are saved
   ii. Click on 'Run' to execute the template over all files



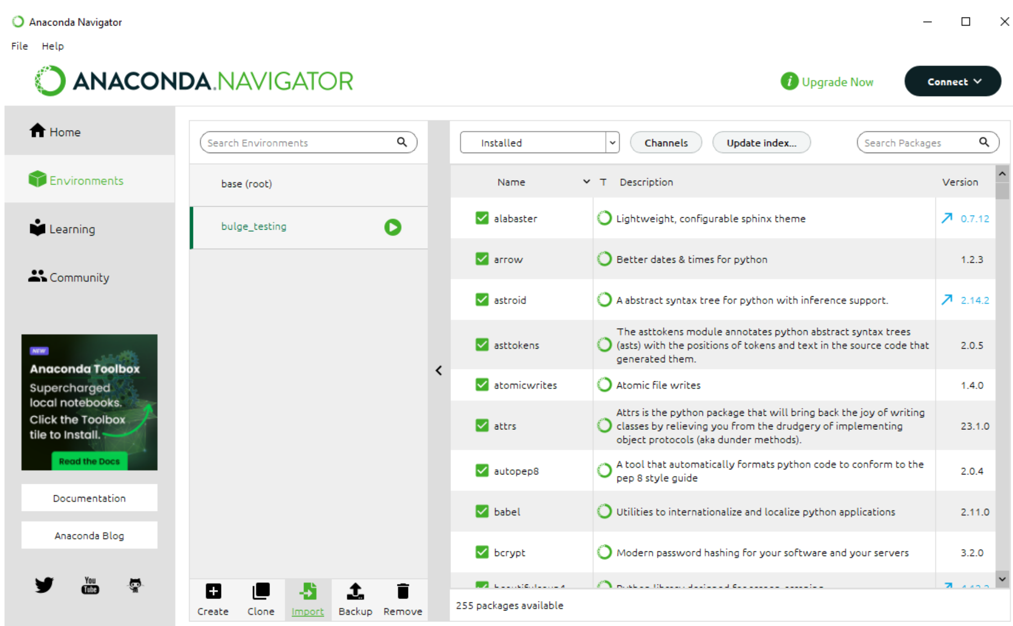
7. A CSV file with the max values will be stored in the folder. Rename this CSV file to **addmax.csv**. Do not choose another filename for this CSV file!

Common Error:

Sensofar can sometimes not handle both steps at once. If issues occur, first run a despiking template over all .plux files, and then create and run an addmax template on the now modified (despiked) .plux files.

3. Transfer Data for Post-Processing:

- Move the data to your personal laptop if the post-processing script cannot be run on the Sensofar room computer
- Install Anaconda if not already installed
- Open the Environment window and import bulge_testing_libraries.yaml located at G:\Limit\Alexander\01 Bulge Testing Versuche\Env_Libraries_Backup



The .yaml-file contains all necessary libraries to execute the bulge-testing script in Spyder.

4. Run the Python Script:

- Open Spyder and the script 'all_step_at_once_nix_modulus_vX_reportPlots_video.py'
- Run the script and follow instructions with the experiment sheet on hand