Indentation Investigation of 304L Stainless Steel Friction Stir Weld Simulated Crack Repair and Software Development for Rapid Processing of Indentation Data

A Thesis

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Science

with a

Major in Mechanical Engineering

in the

College of Graduate Studies

University of Idaho

by

Nicolene van Rooyen

Approved by:

Major Professor: Michael Maughan, Ph.D., P.E.

Committee Members: Indrajit Charit, Ph.D., P.E.; Matthew Swenson, Ph.D., P.E.

Department Administrator: Gabriel Potirniche, Ph.D., P.E.

December 2021

Abstract

Simulated cracks were repaired in 304L stainless steel using low temperature friction stir welding. Indentation studies were carried out to understand the effect of microstructural features on the mechanical property variation across the weld and to measure the size of the weld zones with a quantitative technique. Microhardness and nanoindentation hardness profiles were constructed on a transverse section across the weld. The data obtained were correlated by extrapolating the nanoindentation hardness to greater depths which showed that the nanoindentation hardness closely reflects the microhardness values throughout the weld. Hardness contour maps, with adequate resolution, revealed that nanoindentation can detect material flow induced features, like: “lazy-S” and onion rings. Grain size in the stir zone (SZ) was found to vary with the tool temperature which, in turn, alters the nanoindentation modulus variability and higher tool temperature resulted in softening and widening of the SZ. Comparing nanoindentation elastic contour maps suggest that temperature significantly effects the crystallographic texture development across all zones of the FSW. Four data processing Graphical User Interfaces were developed, using Python, to process the raw data files of the microhardness and nanoindentation tests.

Acknowledgements

I would like to sincerely thank the following individuals for all the encouragement and help given throughout my graduate experience at the University of Idaho:

Dr. Michael Maughan, for exposing me to a variety of advanced manufacturing methods and research methods along with giving advice and guidance to complete this project.

Dr. Indrajit Charit, for inviting collaboration on this project and providing access to sample preparation facilities.

Dr. Matthew Swenson, reviewing my thesis and giving valuable insight into nanoindentation testing methods.

Dr. Madhumanti Bhattacharyya, for exposing me to advance manufacturing methods and the opportunity to take lead on this portion of the research project.

Mr. Nicholas Brubaker, for motivating me during stressful times, giving insight and help with machining, along with trouble shooting and system development of the metal 3D printers.

Mr. Bill Magnie, for improving my machining and fabrication skills, and teaching me how to use the CNC lathe.

Dedication

I want to thank and show sincere appreciation to my family for supporting and encouraging me to pursue a graduate degree.

Table of Contents

List of Tables

List of Figures

Statement of Contribution

The research presented in this thesis is a continuation on previous investigations of processing microstructure relationships of low temperature friction stir welding for crack repair of 304L stainless steel [1]. Dr. Indrajit Charit and Dr. Madhumanti Bhattacharyya provided the friction stir welded samples, conducted the initial microhardness tests, completed the uniaxial tensile tests, determined the average and nominal grain sizes within the stir zone of the friction stir weld, and aided in the microstructure analysis. This work is partly supported by the US DOE Office of Nuclear Energy under award number DE-NE0008776.

All nanoindentation testing and microhardness contour map investigations, microstructural-property-processing correlations, and development of data processing tools were completed by me. Please note that portions of this thesis are being considered for publication, of which I was the primary author.

Chapter 1: Motivation

Friction stir welding (FSW) is a solid-state joining technique used in numerous aerospace, automotive, and shipbuilding applications [2]. Material joining at joints or for crack repair is achieved by severe plastic deformation of the material and frictional heat generation [1]–[5]. This is accomplished by inserting a non-consumable high speed rotating tool (Figure 1.1.b), into the material and then moving it along the abutting material surfaces or crack (Figure 1.1.a) [2], [3], [5]. The joining process leaves the material with unique microstructural features, crystallographic texture, and several microstructural zones each possessing different properties due to their processing history. Detailed discussion of these zones, other microstructural features, and crystallographic texture evolution (of the weld seam) is provided in Chapter 2: Friction Stir Welding. Traditional microscopy and indentation methods used for microstructure-property characterization will be discussed. Chapter 1 outlines the importance and versatility nanoindentation offers microstructure-property characterization of FSWed parts.

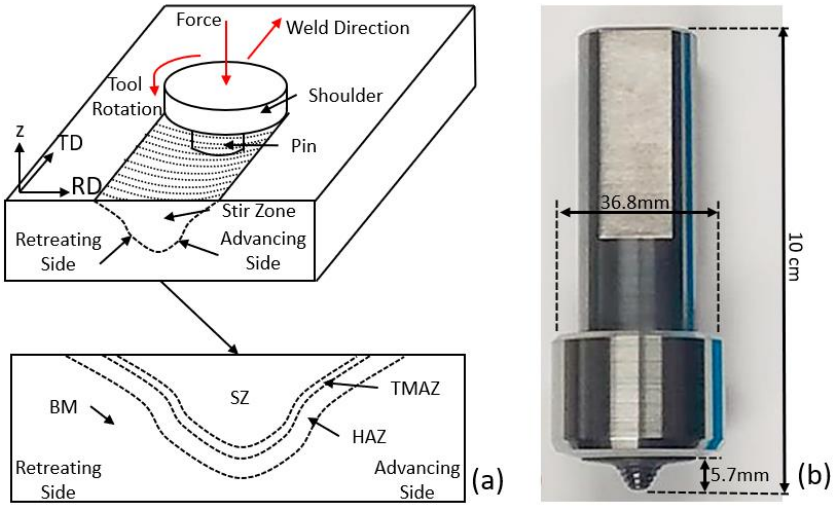


Figure 1.1 (a) Friction stir welding process and microstructural zone illustration, (b) polycrystalline cubic boron nitride tool used for repair welding.

FSW processing parameters significantly influence the average grain size distribution[1], [2], mechanical property variation [2], [4], [6], texture evolution [2], [5], [7], and degree of crystallographic misalignment within the welded material [7], [8]. Traditionally, electron backscatter diffraction (EBSD) is used in tandem with indentation testing to characterize crystallographic texture evolution of the welded region and the microstructure-property relationships of FSWed parts. EBSD is used to determine individual grain orientations, local textures, along with identifying phase and phase distributions of polycrystalline materials [5]. EBSD is a costly microscopy method that requires specialized equipment and careful sample preparation to produce good results. Microhardness is the

most common indentation method used with EBSD to assess the microstructure-property variations of FSWed parts [6], [9]–[11]. However, the length scale of microhardness testing is insufficient to capture slight microstructural changes, which has led to inconsistent reporting of mechanical behavior across the different microstructural zones of FSWed parts. Thus, indicating that microhardness testing lacks the resolution required to assess finer changes across the various FSW microstructural zones. The slight changes across microstructural zones are discussed more fully in Chapter 2: Friction Stir Welding.

The smaller length scale of nanoindentation testing provides higher resolution, thus making it a suitable alternative to microhardness [4], [12], [13]. It is a depth sensing method capable of providing the hardness and Young's (elastic) modulus from the indentation data [14]. The governing principles, differences, and hardness conversions between microhardness and nanoindentation methods are provided in the last three sections of Chapter 2. Depth sensing indentation techniques have been used to understand grain orientation dependent mechanical properties of mono-and-polycrystalline materials as early as 1994 [15]. It is well known that mechanical properties (elastic modulus) differ among grain orientations [12], [16], [17]. Anisotropic elastic behavior investigations, via nanoindentation, on polycrystalline materials have frequently been done on samples where the average grain size is much larger than the indent plastic zone [18] and EBSD is usually used to validate the findings [19], [20]. Variation in the elastic modulus of materials with a small average grain size with a predominant texture has not been explored.

Microstructure-property relationships and texture evolution of the microstructural zones have extensively been studied for aluminum alloys (covered in Chapter 2: Friction Stir Welding), but a significant knowledge gap exists for steels. In this work microhardness and nanoindentation testing method comparison studies will be performed for FSWed 304L stainless steel (SS) samples prepared with two different tool temperatures. Nanoindentation will also be used to assess the elastic modulus variation across the different microstructural zones of the FSW samples to estimate the dominant texture present. Custom Python user interfaces will also be developed for ease of nanoindentation data processing and will be available to the University of Idaho for future use.

Chapter 2: Background

Friction Stir Welding

FSW was invented in 1991 by The Welding Institute (TWI), in the United Kingdom [21], [22]. It is considered an energy efficient, environmentally friendly, and versatile solid state joining technique that produces welds, without filler material or shielding gas that are virtually defect free, and is capable of joining materials that are considered difficult for conventional fusion processes [2], [22]. FSW is considered a solid state joining technique because no material is melted, instead frictional heat produced due to the tool rotation in conjunction with downward force, and tool translation increases the local temperature and causes the plastic flow of the material around the tool pin, joining the material behind it [2], [3], [22]. Due to the mechanics of the FSW technique, its application it typically limited to the joint types outlined in Figure 2.1.

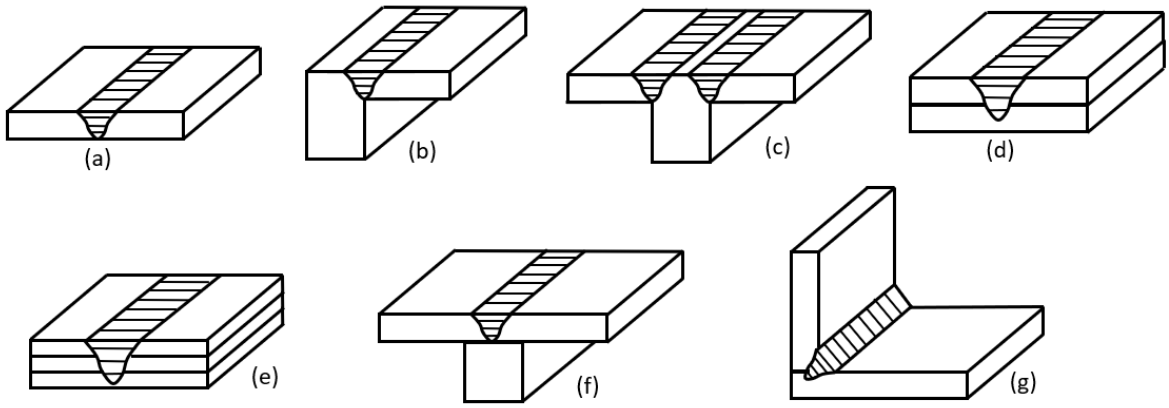


Figure 2.1 Joint configurations for FSW: (a) square butt, (b) edge butt, (c) T butt, (d) lap joint, (e) multiple lap joint, (f) T lap joint, and (g) fillet joint. Adapted from Mishra and Ma [3].

Like other welding processes, FSW leaves the material with several microstructural zones. The severe plastic deformation and frictional heat changes the microstructure of the base material (BM) in and around the weld, forming three distinct zone as shown in Figure 2.2. Each zone possesses different microstructure and properties due to their processing or thermo-mechanical history. The central stir zone (SZ) is enclosed by a thermomechanical affected zone (TMAZ), which is followed by a heat affected zone (HAZ) and then unaffected BM [2], [5]. The SZ is previously occupied by the tool pin which experiences the intense shearing, plastic deformation, and excessive frictional heat, results in a microstructure that is fine grained fully recrystallized. The TMAZ experiences both plastic deformation and thermal cycles but, insufficient plastic strain and heat gives rise to elongated grains. The HAZ experiences thermal cycles only, which are sufficient to modify the microstructure and mechanical properties.

These zones (Figure 2.2) are further classified as advancing or retreating depending on their position relative to the rotation of the tool with respect to its traverse direction. From microstructural investigations it has been observed that the retreating HAZ (HAZ_R) to SZ transition is usually more diffused due to the complex material flow around the tool pin [23].

Material flow line features (Figure 2.2) can be found within the SZ and are highly dependent on the processing parameters. One of these features is a set of metallurgical bands, commonly known as "onion rings," which manifest as a repeating pattern of second phase particle distribution, and/or grain orientation extending from where the tool contacts the material [2], [24], [25]. The joining line or "lazy-S" is the other material flow feature. It extends from the top surface to the bottom of the SZ [25]–[28]. The lazy-S region can have weak material bonding that adversely effects the mechanical performance of the weld [26]–[28].

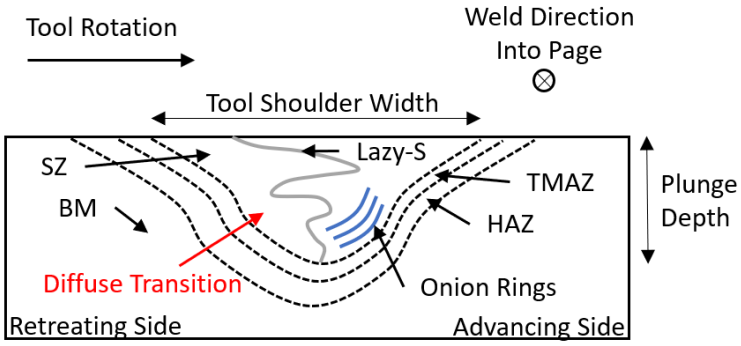


Figure 2.2 FSW microstructural zones and material flow line features.

FSW process parameters like: tool speed, travel speed, downward force, and tool temperature has a significant effect on the mechanical property, microstructure zone development, and texture evolution of the weld [2], [3], [11]. Higher local temperatures induced by the process parameters results in increased time required for heat dissipation to occur. Thus, giving ample time for grain growth to occur within the SZ [2], [11]. Average grain size within zones and zone widths are directly proportional to local temperatures induced by the processing parameters. Whereas this has an adverse effect on the mechanical properties of the FSW [11]. In other words, higher temperatures result in an increased average grain size and widening of the SZ, which decreases the mechanical performance of the FSW [2], [11], [12], [22]. The texture evolution of zones are heavily influenced by the travel and tool speed of the FSW process [7]. The degree of crystallographic misalignment is decreased with increasing tool speed, which creates a more uniform crystallographic orientation within the SZ [7], [8]. Jeon et al. [29] used high resolution EBSD to study the microstructural development of FSWed single-crystal austenitic stainless steel. They concluded that simple shear deformation refined the SZ

into a fine-grained polycrystalline aggregate [29]. Continuous and discontinuous recrystallization resulted in a final texture dominated by the ideal simple shear orientation [29].

Microhardness Testing

Hardness tests are inexpensive and simple nondestructive methods that measures a materials resistance to localized plastic deformation [17]. Various hardness tests exist across different length scales. The Vicker's hardness test method will be discussed in detail because it is the most used method for reporting hardness of FSWed parts. This method uses a sharp diamond pyramidal indenter which has equal sides with an applied load ranging from 1-1000 g [17]. If the sample surface has been prepared correctly, through grinding and polishing, the resulting indentation impression (Figure 2.3) should have diagonals equal in length. The length scale of these diagonals are within the μm length scale and this is the reason that Vicker's hardness is often referred to as microhardness [17].

The microhardness is defined as the applied load divided by the surface area of the impression and can be calculated using Eq. 1. Where F is the load applied in gf and d_{avg} is the averaged of the two diagonals, seen in Figure 2.3, in μm.

$$VH = 1.854 \cdot 10^3 \times \frac{F}{d_{avg}^2} \quad (1)$$

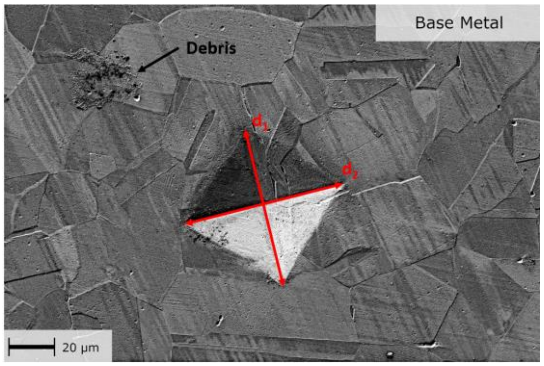


Figure 2.3 Vickers microhardness impression size relative average grain size of BM.

Nanoindentation

Nanoindentation is an indentation test where the penetration of the indenter tip into the surface is measured in nanometer (10^{-9} m). Nanoindentation testing is also a nondestructive testing technique and is more versatile than microhardness testing because nanoindentation (Meyer) hardness, Young's (elastic) modulus, strain-hardening exponent, fracture toughness of brittle materials, viscoelastic properties and residual stress can be determined from the test data [12], [30].

The main difference between calculating microhardness and nanoindentation hardness is the determination of contact area [30]. As described above, the contact area for microhardness is

determined by measuring the diagonals of the residual impression. Whereas the contact area of nanoindentation is the projected contact area and is determined indirectly from indentation test data because the residual indentation impression is too small to be measured conveniently [14], [30]. As shown in Figure 2.4.a , the material surface experiences both elastic and plastic deformation during loading. The projected contact area (contact area) of elastic contact is used for the Berkovich indenter tip geometry [30]. The contact area is related to the indenter geometry, applied load, and the materials elastic modulus [14], [30]. Eq. 2 shows that the contact depth, h_c, is determined by using the maximum applied load (P_{max}), material stiffness (S), and maximum penetration depth (h_{max}) [14]. ε is a geometric constant dependent on indenter tip geometry.

$$h_c = h_{max} - \epsilon \frac{P_{max}}{S} \tag{2}$$

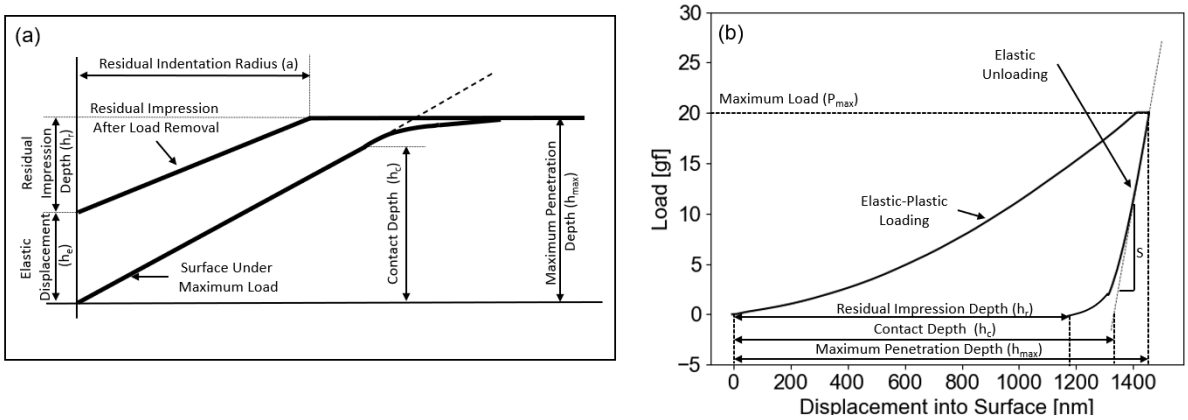


Figure 2.4 (a) Idealized representation of indenter contact interaction with sample surface under maximum load and no load. (b) Load versus indenter displacement into material surface. Adapted from Oliver and Pharr [14], and Fischer-Cripps [30].

Figure 2.4.b shows the typical load vs displacement curve for nanoindentation. The elastic modulus is related to the stiffness of the initial portion of the unloading curve. Eq. 3 shows stiffness is a function of the reduced modulus, E_r, and the contact area, A.

$$S = \frac{dP}{dh} = \frac{2}{\sqrt{\pi}} E_r \sqrt{A} \tag{3}$$

E_r represents the combined properties of the indenter and the material probed. Eq. 4 shows the E_r is related to the indenter elastic modulus, E_i, indenter Poison's ratio, υ_i. E and υ are the sample parameters. E_i, υ_i , and υ are assumed to be 1141 GPa, 0.07, and 0.29 respectively.

$$\frac{1}{E_r} = \frac{1-\upsilon^2}{E} + \frac{1-\upsilon_i^2}{E_i} \tag{4}$$

A is a function of the elastic contact depth, h_c, and is represented by Eq. 5. Where C_1, C_2, and C_3 are constants determined through calibration of the indenter itself. The specific calibration constants used during testing are outlined in Chapter 3.

$$A = 24.5h_c^2 + C_1h_c + C_2h_c^{1/2} + C_3h_c^{1/4} \quad (5)$$

Nanoindentation hardness, H, was determined by Eq. 6. Where P_{max} is the last recorded force before unloading.

$$H = \frac{P_{max}}{A} \quad (6)$$

The nanoindentation hardness was converted to microhardness hardness using Eq. 7 [30].

$$VH = \frac{1.8544}{2(9.81)} \times H = 0.094495 \times H \quad (7)$$

Indentation Size Effect

Comparing hardness measurements across length scales is complicated by the existence of the indentation size effect, which is an increase in hardness with decreasing indentation size [30]–[34]. This behavior was first explained by a simple dislocation density model in 1993 proposed by Shell De Guzman et al. [31]. The dislocation density model proposes that geometrically necessary dislocation are created under the indenter tip, for materials that can experience strain hardening [31]. Figure 2.5 shows an idealized model of geometrically necessary dislocation loops being created underneath a conical indenter, within the plastically strained region, inhibiting further penetration of the indenter into the material. High dislocation density is present at shallow penetration depths, resulting in an apparent increase in hardness. The dislocation density decreases with increasing penetration depth, thus the hardness measured at deeper indentations should represent the intrinsic/true hardness of the material. This hardness is often referred to as infinite depth hardness.

Further investigation showed that the geometrically necessary dislocations are believed to originate due to large strains and strain gradients present in small indentations [30], [32] along with other defect structures within the material [35]. The initial Shell De Guzman model was refined by Nix and Gao in 1997, and is referred to as the Nix-Gao model [32]. The Nix-Gao model incorporated the indenter geometry, slip step spacing (Figure 2.5), and deformation resistance to show that total dislocation density depends on the depth of penetration and average strain present during indentation [32]. The Nix-Gao also developed a new law for strain gradient plasticity that shows the importance of large strain gradients, at shallow penetration depths, on the observed increased hardness [32].

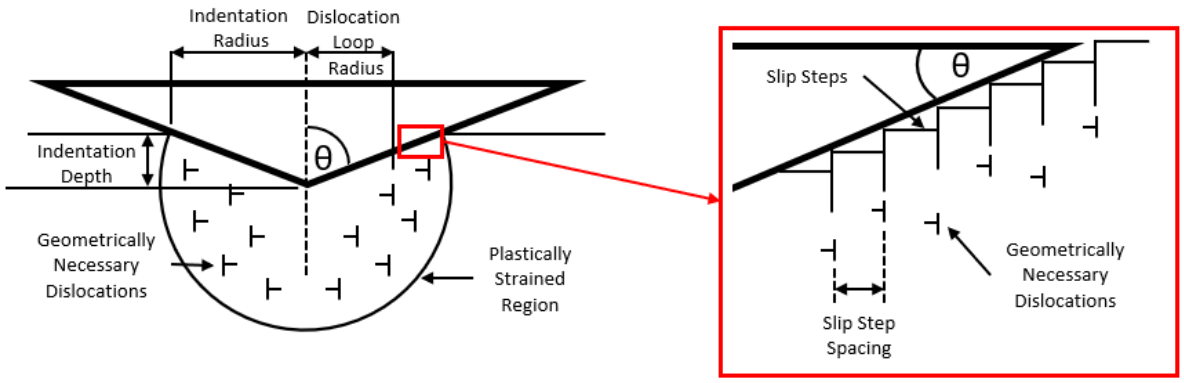


Figure 2.5 Idealized model illustrating geometrically necessary dislocations and plasticly strained region generated by conical indenter. Adapted from Nix and Gao [32], and De Guzman et al. [31].

To assess the ISE of a given material, it should be probbed at various depths in the same location. Figure 2.6.a shows the typical load vs indentation depth curves produced by cyclical loading nanoindentation. From this data (Figure 2.6.a) the Nix-Gao model (Eq. 8) can be used to determine the intrisic hardness of the material, as shown by H_0 in Figure 2.6.b

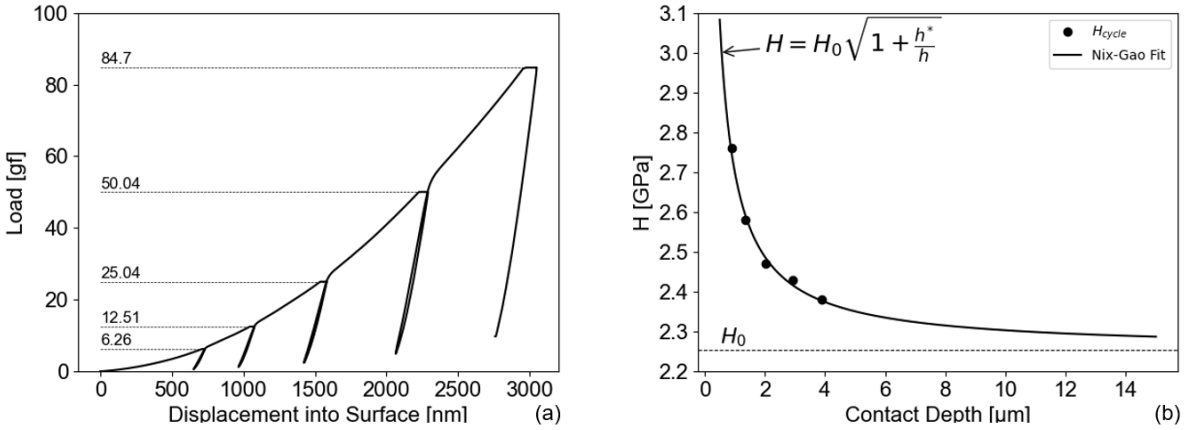


Figure 2.6 (a) Cyclical nanoindentation loading vs displacement profile. (b) Nix-Gao fit of cyclical nanoindentation loading.

Eq. 8 is the symbolic expression of the Nix-Gao model. Where H is the nanoindentation hardness, h is the indentation contact depth, h^* is a fitting parameter, and H_0, is the infinite depth hardness, which is assumed to reasonably represent the microhardness hardness. Once H_0 is determined it can be converted to microhardness using Eq. 7.

$$H(H_0, h) = H_0\sqrt{1+\frac{h^*}{h}} \tag{8}$$

Chapter 3: Methods

FSW was used to simulate crack repair in a 304L SS plate. Two sets of processing parameters were used to heal the simulated cracks. Samples machined from the section transverse to the welding direction were used for microscopy, indentation, and uniaxial tensile testing. Data processing software for indentation testing were developed using Python. Destructive and non-destructive tests performed are as follows:

1. Vickers hardness testing was used to construct microhardness profiles and contour maps across the weld zones.
2. Nanoindentation was used to construct similar profiles and contour maps, except these collected Meyer hardness and elastic modulus of the SZ, TMAZ, and HAZ at discrete locations.
3. ISE was investigated to correlate hardness across length scales.
4. Uniaxial tensile tests were completed for elastic modulus comparison to nanoindentation data.

Friction Stir Welding Sample Preparation & Microscopy

Electric discharge machining (EDM) was used to create a 5 mm deep and 0.33 mm wide crack along the length of an as-received 304L stainless steel plate (Rolled Alloys, Inc.). The as-received plate was hot-rolled and annealed with the following dimensions: 330 mm long, 149 mm wide, and 12.7 mm high. The chemical composition of the as-received plates was provided by the supplier and is shown in Table 3.1. Crack repair, along the length of the as-received plate, was done at Pacific Northwest National Laboratory (PNNL) using a TTI gantry FSW machine with a control algorithm which maintains constant tool temperature. The polycrystalline cubic boron nitride tool (MegaStir™) used during FSW is shown in Figure 1.1.b. The tool shoulder diameter and pin length were 36.8 mm and 5.7 mm respectively. Two different sets of FSW processing parameters were carried out to repair the simulated crack. Table 3.2 shows the processing parameters and associated sample names. Ultrasonic testing was done to assure the simulated crack was healed properly.

Table 3.1 As-received 304L SS Plates Chemical Composition.

Element	**C**	**Mn**	**P**	**S**	**Si**	**Cr**	**Ni**	**Mo**	**Cu**	**N**	**Nb**	**Ti**
wt%	0.016	1.53	0.06	<0.001	0.32	18.34	8.17	0.32	0.43	0.09	0.021	0.003

Table 3.2 Sample nomenclature and FSW process parameters

Sample	Tool Temperature [°C]	Weld Speed [mm/min]	Tool Speed [rev/min]	Vertical Load [kN]	Spindle Torque [Nm]	Weld Power [kW]	Tilt Angle [°]
1-C	825	25.4	95-130	51.1	199.3	2.0	-0.5
1-D	725	25.4	63-69	48.9	215.5	1.6	-0.5

Samples of the as-received plate and welded region were taken for microstructural characterization. All samples were taken from the center of the plate where the crack was simulated, and Table 3.3 shows the sample dimensions. Samples 1-C and 1-D were also used for the indentation investigations. The samples were sectioned using a Buehler Isomet® 1000 precision cutter. The lowest cutting speed and feed rate was used to minimize deformation induced phase transformations at the cutting surface. Each sample was prepared for microscopy by grinding with silicon carbide paper starting at 600 grit and successively working down to 1200 grit. After grinding, each sample was polished using 3 μm and 1 μm diamond suspension. Each sample was cleaned for 10 min with an ultrasonic vibratory bath while submerged in ethanol. Samples were then electrochemically etched to reveal the grain structure by submerging the sample surface in a 10% oxalic acid solution maintained at 80°C for 10 s with 10 V applied. Figure 3.1 shows the equipment and setup used during the etching procedure. After etching, each sample was thoroughly rinsed with alcohol and cleaned in an ultrasonic water bath for 10 min. Light optical microscopy and scanning electron microscopy (SEM) were used to examine the microstructure of the as received and FSW samples. An AmScope ME520TA was used to create montages of the microstructure of samples 1-C and 1-D. A Zeiss Supra 35VP field emission gun SEM was used to capture the nanoindentation impression size relative the grain size in different zones of the 1-D FSW. The SEM was operated at an acceleration voltage and working distance of 5 kV and 10.0 mm respectively. The average grain sizes were determined using the mean linear intercept method. Sample 1-C and 1-D were repolished using the same procedure prior to each indentation testing step.

Table 3.3 Dimensions of samples used for microscopy and indentation

Sample	Length [mm]	Width [mm]	Height [mm]
As-received	25.4	12.7	12.7
1-C	42.58		12.03
1-D	42.54		12.12

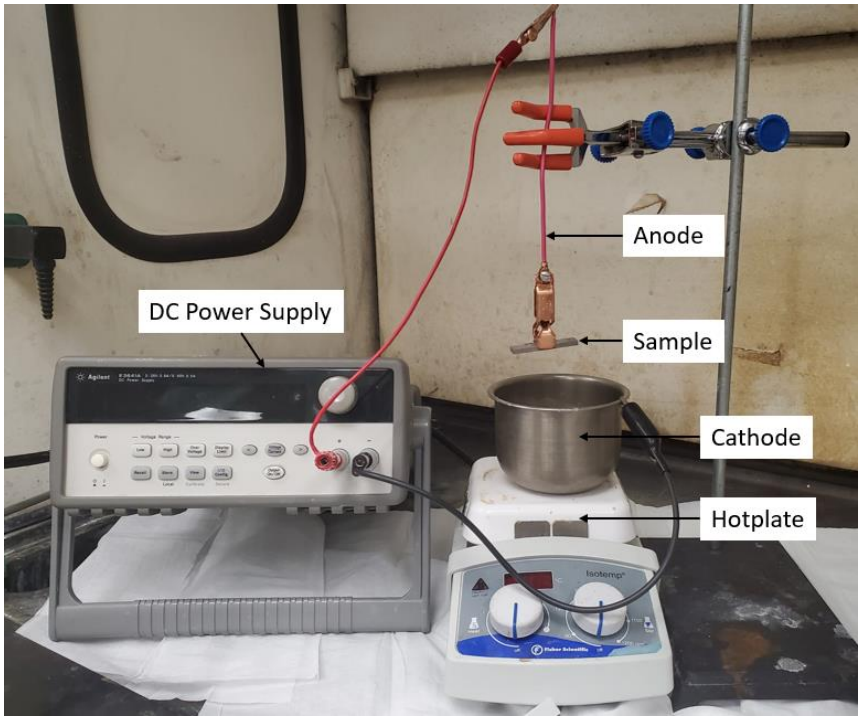


Figure 3.1 Electro-chemical etching setup.

Indentation Methods

Indentation Comparison Methods

The etched 1-D sample, seen in Figure 3.2.a, serves as reference where indentation comparison tests were conducted relative to the SZ. Two sets of indentation comparison tests were conducted. First, microhardness testing and nanoindentation was used to create line profiles across 1-D FSW. A schematic of indentation profile locations relative to the top surface can be found in Figure 3.2.b and Table 3.4 shows the distances from the top of the sample and gives the naming convention for various indentation tests. Secondly, microhardness and nanoindentation contour maps were created for sample 1-C and 1-D, at the areas shown in Figure 3.2.c. Microhardness testing was completed using a LECO LM-100 Vickers microhardness tester and is shown in Figure 3.3.a. Figure 3.3.b shows the KLA Nano Indenter G200 (Milpitas, CA) used to perform nanoindentation testing. The Nano Indenter G200 was equipped with a diamond Berkovich tip for all tests. Microhardness and nanoindentation testing procedures for both profile (1-D) and contour maps (1-C &1-D) comparisons are outlined below. All nanoindentation tests conform to ASTM 2546-07.

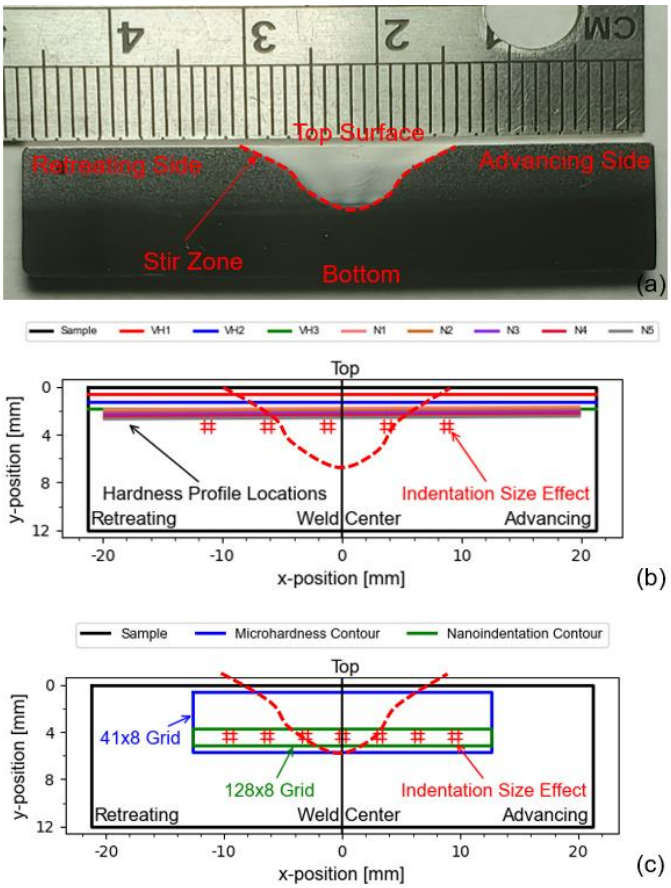


Figure 3.2 (a) Etched 1-D sample with advancing, retreating, SZ, and top surface annotated, (b) profile comparison indentation locations, (c) contour map indentation locations.

Table 3.4 Profile comparison indentation locations relative to top surface and associated acronyms.

Indentation Method	**Acronym**	**Distance from Top [mm]**
Microhardness Line 1	VH1	0.635
Microhardness Line 2	VH2	1.270
Microhardness Line 3	VH3	1.905
Nanoindentation Line 1	N1	1.867
Nanoindentation Line 2	N2	2.067
Nanoindentation Line 3	N3	2.280
Nanoindentation Line 4	N4	2.466
Nanoindentation Line 5	N5	2.666

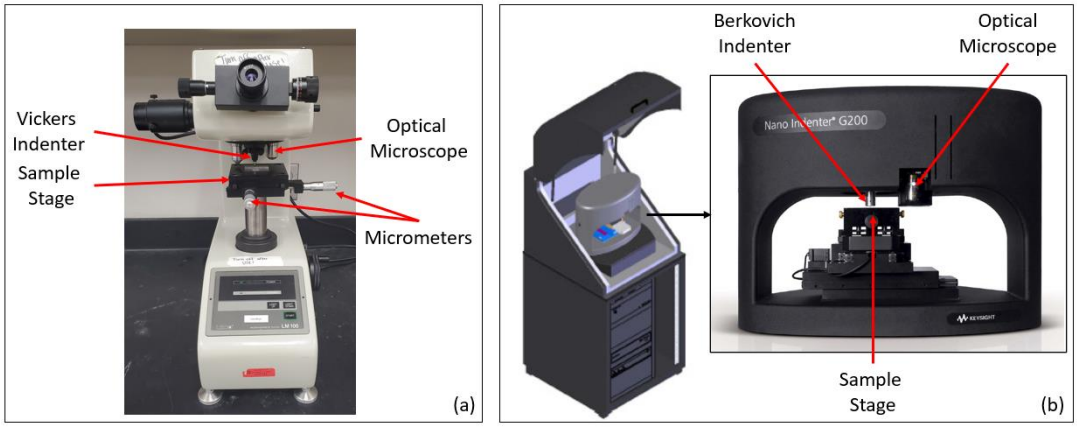


Figure 3.3 (a) LECO LM-100 Vickers microhardness tester and (b) KLA G200 Nanoindenter used for indentation testing.

Microhardness Testing

All microhardness testing was performed by applying a 200 gf (1.961 N) load. Microhardness lines consisted of 60 indents spaced 0.635 mm apart, and each line was spaced 0.635 mm from another. The lines are indicated with the red, blue, and green lines in Figure 3.2.b. The contour maps were constructed within the blue area marked in Figure 3.2.c. The microhardness indentation tests to create the contour map consisted of a 41 by 8 grid with square indent spacing of 0.635 mm. Slight deviations from parallelism among the lines were visible with the naked eye, however, these should not affect the results.

Nanoindentation Testing

Figure 3.4.a shows the size of sample 1-C relative to the standard Aluminum sample mounts used with the G200 nanoindenter. To obtain a continuous profile across the entire sample a custom sample mount had to be fabricated to ensure the sample remains square with the coordinate frame of the nanoindenter. Figure 3.4.b shows the CAD model of the custom mount and Figure 3.4.c shows how each mount fits into the sample stage. The set screw was tightened on the flat edge extending the length of the sample mount and the sample was mounted such that the top surface of the FSW is flush with the second flat face. This ensured that the nanoindentation profiles and contour maps were parallel to the top surface of the sample being tested, thus resulting in a very small parallax fault of 0.100 ± 0.035 mm from the retreating to advancing side of the sample.

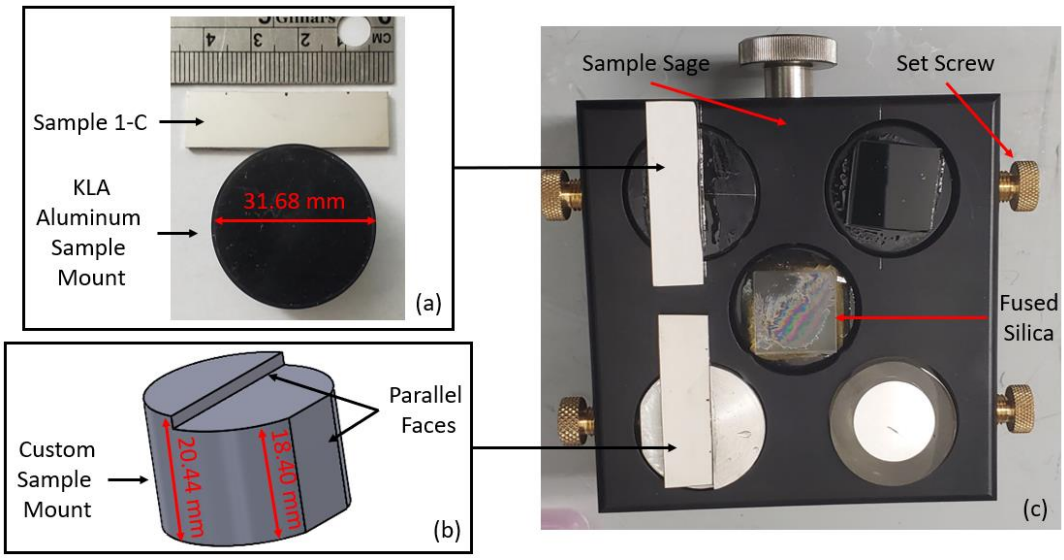


Figure 3.4 (a) Aluminum sample mount provided by KLA, (b) custom sample mount, and (c) samples mounted and secured in the sample stage.

The sample was mounted by heating the mount on a benchtop hot plate. Adhesive mounting wax was then applied to the top surface of the mount and allowed to fully melt. The mount was removed from the hotplate using the test tube clamps once the adhesive was completely melted. With gloves the sample was then placed onto the melted adhesive after which the sample was pressed and held until the adhesive was fully solidified. The equipment used for sample mounting is shown in Figure 3.5.

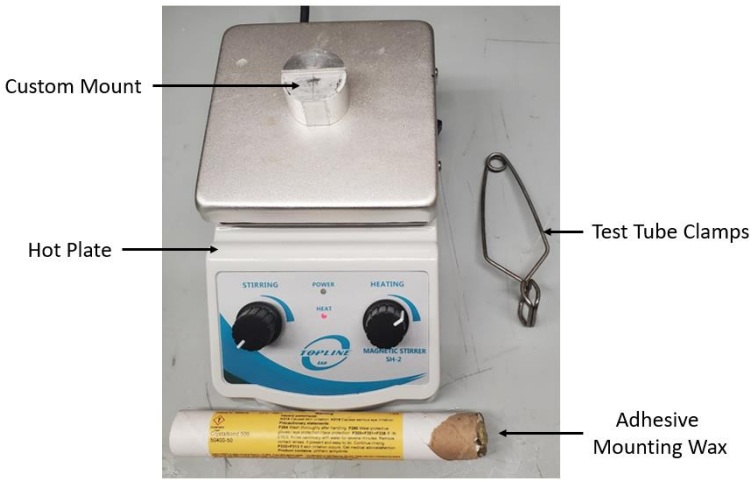


Figure 3.5 Equipment used to adhere the sample to the custom mount.

Five lines of 220 nanoindentations were constructed at the locations indicated by “hardness profile locations” in Figure 3.2.b. Each of the lines were separated into two segments of 110 indents each. The sample had to be shifted such that the indented region was adhered to the sample mount and not cantilevered off the edge onto the sample stage. The custom sample mount allowed for easy

realignment to continue generating the profile. Each nanoindentation line was spaced 200 µm from the next. Indents were spaced 200 µm apart which is approximately 10 times the indentation "diameter" to avoid testing the plastically deformed region created by the previous indent [36]. The green area shown in Figure 3.2.c was used to construct a contour map which has 128 by 8 grid with square spacing of 200 µm. For the contour maps the 1-C and 1-D FSWed samples were centered on the custom mount, as shown in Figure 3.6. This allowed for the entire grid to be completed in one single test. For both profile and contour mapping, the sample was loaded linearly to 20 gf (196.1 mN) within 15 s and held constant for 30 s before linear unloading in 10 s. The nanoindentation test loading profile is provided in Figure 3.7.a. The maximum indentation depth varied from 1450 nm to 1875 nm.

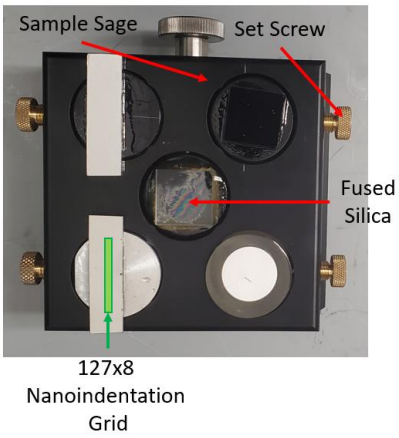


Figure 3.6 Sample mounted for nanoindentation contour mapping.

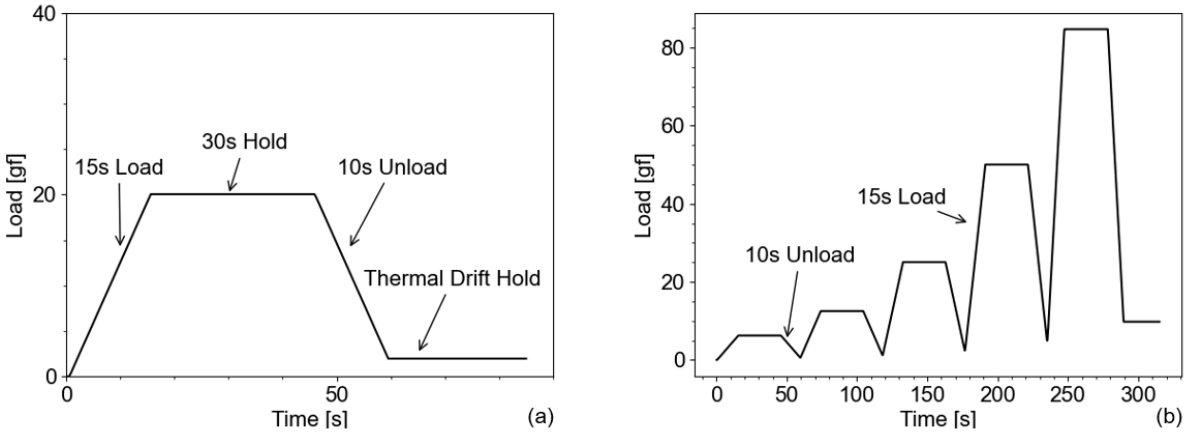


Figure 3.7 (a) Nanoindentation loading profile used to create profiles and contour maps of FSW samples, (b) Cyclical nanoindentation loading profile used for indentation size effect characterization.

Indentation Size Effect Tests

Hardness vs. depth data were captured at each location indicated by the # symbol in Figure 3.2.b and Figure 3.2.c. This was done to characterize the ISE [33] of the different regions comprising FSWed samples and facilitate comparisons between nanoindents and microhardness indents despite their size. Two ISE characterization was completed for the 1-D sample. The first approximately 3.0 mm from the top surface (Figure 3.2.b) and the other approximately 4.5 mm from the top surface (Figure 3.2.c). Only one ISE characterization was conducted for the 1-C sample, approximately 4.5 mm from the top surface. As shown in Figure 3.7.b, the sample was cyclically loaded to a maximum load of 85 gf (833.85 mN). A total of five load-unload cycles, evenly spaced between 0 gf and 85 gf (833.85 mN), were completed for each indent, with 500 µm spacing between each indent. Each cycle was linearly loaded within 15 seconds and held constant for 30 seconds and unloaded within 10 seconds. Four indentations were conducted at locations indicated by the # symbol in Figure 3.2.b and Figure 3.2.c.

Uniaxial Tensile Tests

Three tensile samples (Figure 3.8.a) from each temperature were prepared such that the stir zone was within the gauge length (25.4 mm). The thickness and width of the samples were 1.38 mm and 3.48 mm, respectively. Uniaxial tensile tests were conducted at room temperature with a strain rate of 10^{-3} s^{-1} using an Instron 5982 universal tester (Figure 3.8.b).

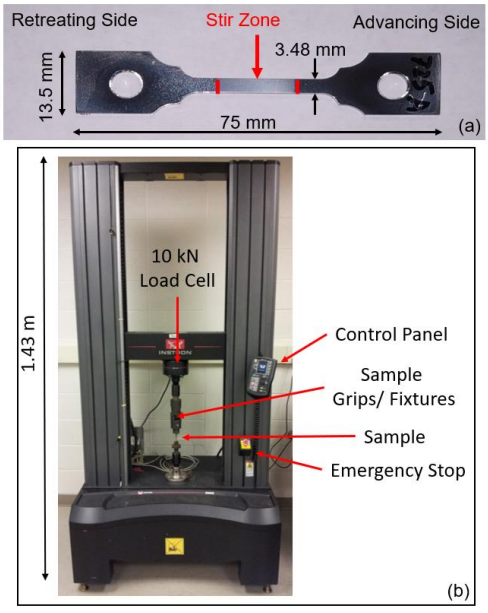


Figure 3.8 (a) Uniaxial tensile test specimen and (b) Instron 5982 universal tester.

Data Analysis Using Python

A custom graphical user interfaces (GUI's) were created using Python, where each GUI was dedicated to specific data analysis tasks. The first GUI was dedicated to processing the raw microhardness data to generate profile and contour plots. The second GUI processed the single loading profile nanoindentation data to discard outliers, smooth the data, compute the elastic modulus coefficient of variance, and generates profile plots of the hardness and elastic modulus. The third GUI was dedicated to processing the cyclic loading nanoindentation data using the standard methods outlined in Chapter 2: Nanoindentation, along with producing hardness and elastic modulus vs depth plots. Lastly, the fourth GUI evaluated the ISE (Chapter 2:Indentation Size Effect) from the output file of the third processing GUI. As shown in Figure 3.9, the system flow diagram outlines that each GUI was specifically designed to require minimal user inputs to process the data.

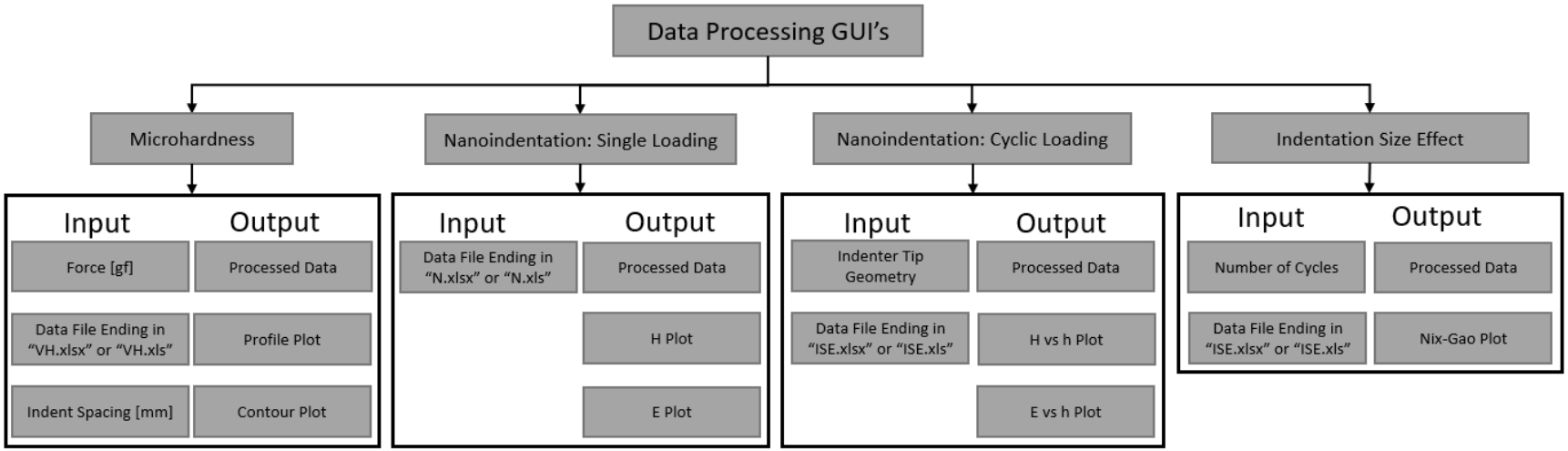


Figure 3.9 Data analysis GUI system flow diagram.

Microhardness Data Processing

As shown in Figure 3.9, the microhardness processing GUI required three user inputs; force applied in gf, the data file, and the indent spacing, in mm, used during testing. Figure 3.10 shows the raw microhardness data file template formatting. Python is case sensitive; thus, formatting of the raw data file should be precise. The excel template file ends with the extension "VH.xlsx." All data analysis GUI's are compatible with ".xlsx" files only. The sheets should be formatted as shown in Figure 3.10, where the first sheet must have the label "Overall." This sheet contains only the index values of indents for each profile line created. The calculated hardness values and x-y coordinates for each profile line is auto populated in the columns following the index. The sheets following "Overall" contains the measured diagonals (d1, d2) of each indent matching indices. Columns "d" and "VH" were then auto-populated after processing was concluded.

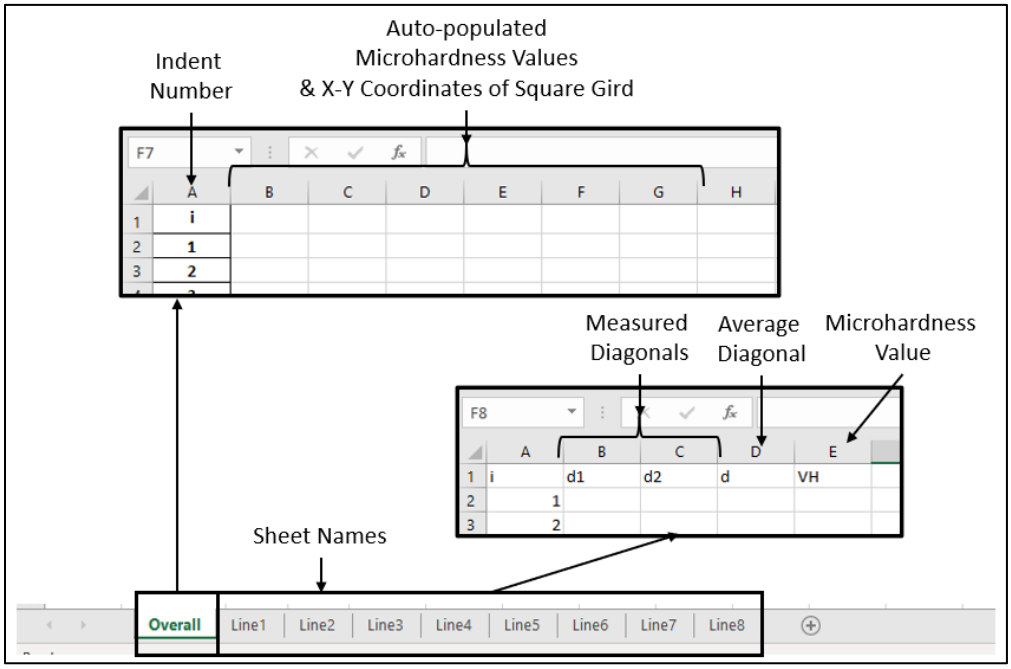


Figure 3.10 Microhardness testing data file template.

Nanoindentation Data Processing

This section outlines the methods used to develop three GUI's used to process various nanoindentation experiment data. The first GUI developed was dedicated to processing the single loading profile (Figure 3.7.a) data, the second to cyclic loading profile (Figure 3.7.b) data, and the third to characterizing the ISE from the cyclic loading profile processed data.

The nanoindenter was calibrated prior to all indentation testing. Table 3.5 shows the tip area calibration constants used for profile and contour plot comparison testing. These constants were used

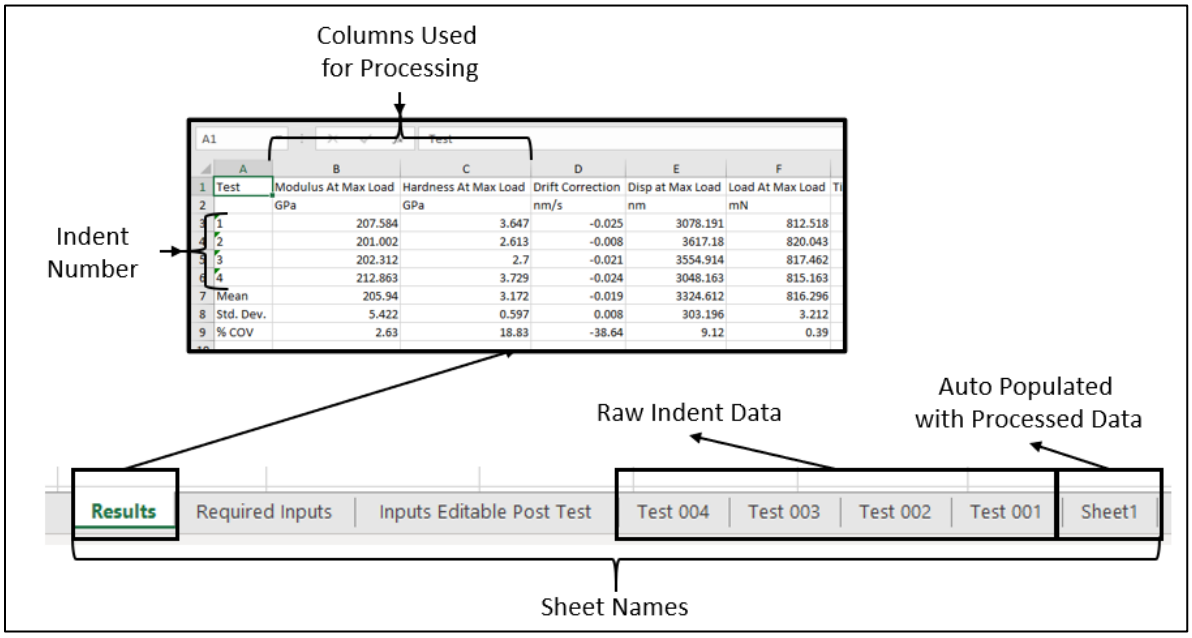
in conjunction with Eq.5 to calculate the projected contact are at specific indentation depths. The tip area constants were incorporated into the appropriate GUI's to complete data processing.

Table 3.5 Tip area constants used for profile and contour map comparison testing.

Testing Type	Tip Calibration Name	C_1	C_2	C_3
Profile Comparison	TB27416 09102020	2445.74	120842	-330337
Contour Comparison	TB27416 06222021	2613.31	254653	-775669

Single Loading Profile

The fist nanoindentation data processing GUI developed was the single loading profile GUI. It was determined that the single loading profile GUI required the raw data file as an input. This was determined by examining the output file exported by the indenter, the standard load, *P*, vs *h* curve, and considering the equations outlined above. Figure 3.11Figure 3.11 shows the exported data file format. The "Results" sheet reports *E*, *H*, drift correction, h_{max}, and P_{max} for each indent. Sheets starting with "Test" holds the instantaneous data points collected during the indentation process. From the exported data file, it was concluded that the indent number, *E*, and *H* columns on the "Results" sheet were to be used for profile comparison analysis. However, cyclical loading (Figure 2.6.a and Figure 3.7.b) would require analysis of individual unloading segments to determine *E* and *H*, because the exported "Results" only reflect the last loading segment.



Test	Modulus At Max Load	Hardness At Max Load	Drift Correction	Disp at Max Load	Load At Max Load
	GPa	GPa	nm/s	nm	mN
1	207.584	3.647	-0.025	3078.191	812.518
2	201.002	2.613	-0.008	3617.18	820.043
3	202.312	2.7	-0.021	3554.914	817.462
4	212.863	3.729	-0.024	3048.163	815.163
Mean	205.94	3.172	-0.019	3324.612	816.296
Std. Dev.	5.422	0.597	0.008	303.196	3.212
% COV	2.63	18.83	-38.64	9.12	0.39

Figure 3.11 Exported nanoindentation data file format.

The single loading profile GUI lists all files ending in "N.xlsx" extension. Once the file was selected the indent number, *E*, and *H* from the "Results" sheet were retrieved for processing. First, *H* was

processed to minimize the grain boundary effect [19], [37]. This was done by discarding data that deviated more that 8% from the average microhardness of a moving average. The moving average window consisted of a total of 10 indents, five forward-looking and five backward-looking compared to the current indent. After removing the outliers, the data were smoothed using a Fast Fourier Transform (FFT) filter. More information about the FFT filter is provided in Chapter 4:Data Analysis Using Python, Single Loading Profile GUI. Smoothing was necessary to obtain a continuous profile from the large number of indents. Lastly, the coefficient of variance (COV) was calculated for *H* and *E* across all indents. The COV was calculated by determining the standard deviation of each point, then dividing it by the mean of the moving-average window of 10 indents. The smoothed *E* and *H* data and associated COV for each indent was then saved in "Sheet1".

Cyclic Loading Profile

The cyclic loading analysis GUI required two inputs: indenter calibration constants and the raw data file. This GUI only lists files ending in the extension "ISE.xlsx", and automatically lists all calibration constants for various calibrations (Figure 3.12).

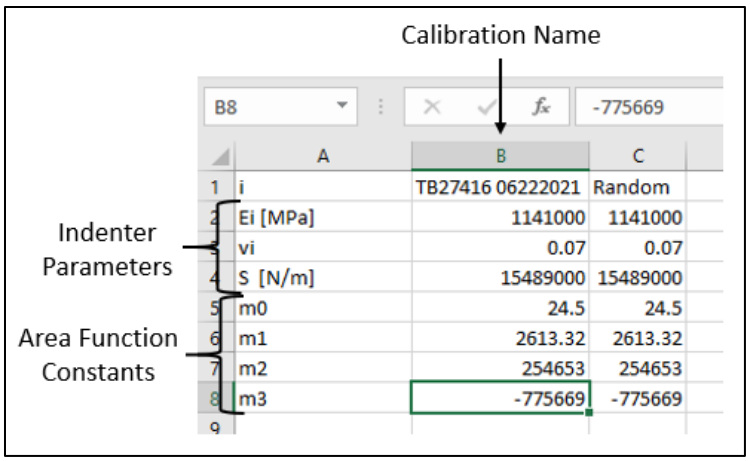


Figure 3.12 Tip geometry calibration constants and indent parameters for specific calibrations.

Once the tip geometry and raw data file were selected, the drift correction and associated indent number from the "Results" sheet were extracted (Figure 3.13) for cyclic loading analysis. The "Segment", *h*, *P*, and time, *t* in seconds, columns from sheets starting with "Test" were extracted to determine *E* and *H* for each unloading segment of the cyclic loading profile. First, *h* is corrected by subtracting the drift correction and *t* product from *h*. After this correction the number of cycles and unloading start index is logged by parsing through the "Segment" column until the tag matches "Unload From Peak Segment Type." Next the unload start indices were used to perform the standard nanoindentation analysis as outlined Chapter 2:Nanoindentation. Once processing was finished the *h*,

H, *VH*, and *E* for each cycle was saved on "Sheet1" along with *H* vs *h* and *E* vs *h* plots are generated and displayed in the GUI.

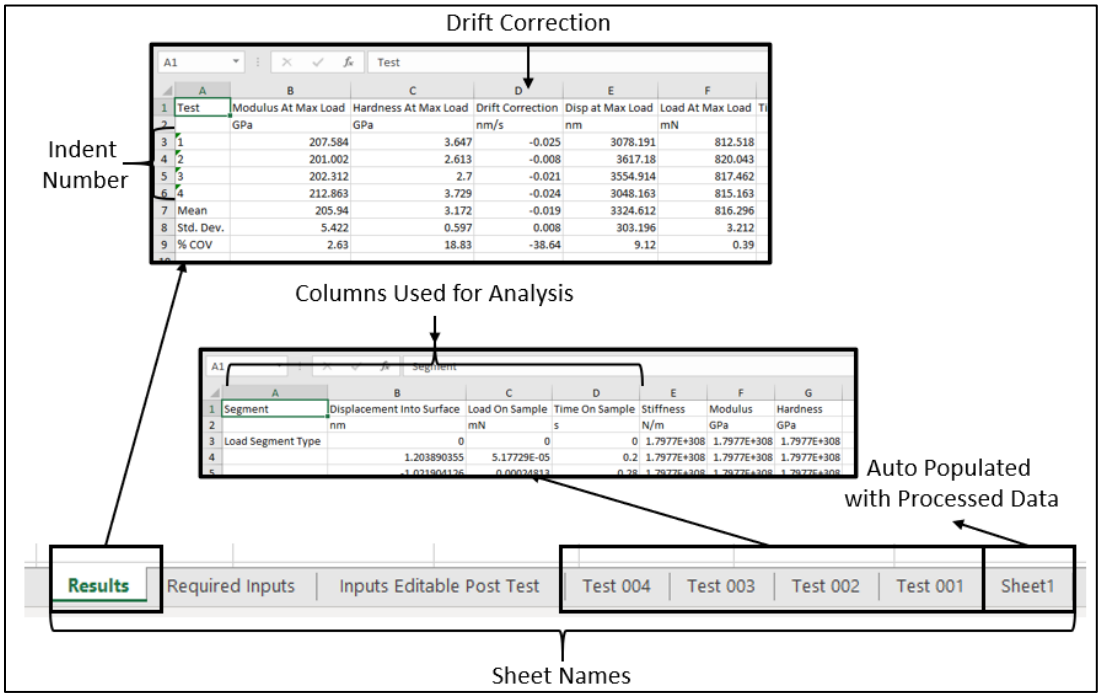


Figure 3.13 Information retrieved from exported nanoindentation file for cyclic loading data analysis.

Indentation Size Effect Analysis

The indentation size effect analysis GUI required two user inputs; the number of cycles completed per indent and the data processed by the cyclical loading GUI (Figure 3.14). This GUI will also automatically list all files ending in the extension "ISE.xlsx" or "ISE.xls." As indicated in Figure 3.14, *h* and *H* data for each indent was extracted from "Sheet1." H_0 and h^* was then determined by implementing a curve fitting algorithm. After which these parameters were appended into "Sheet1" for each test and the Nix-Gao model for each indent was displayed. The specific curve fitting algorithm employed is discussed further in Chapter 4:Data Analysis Using Python, Cyclic Loading Profile GUI.

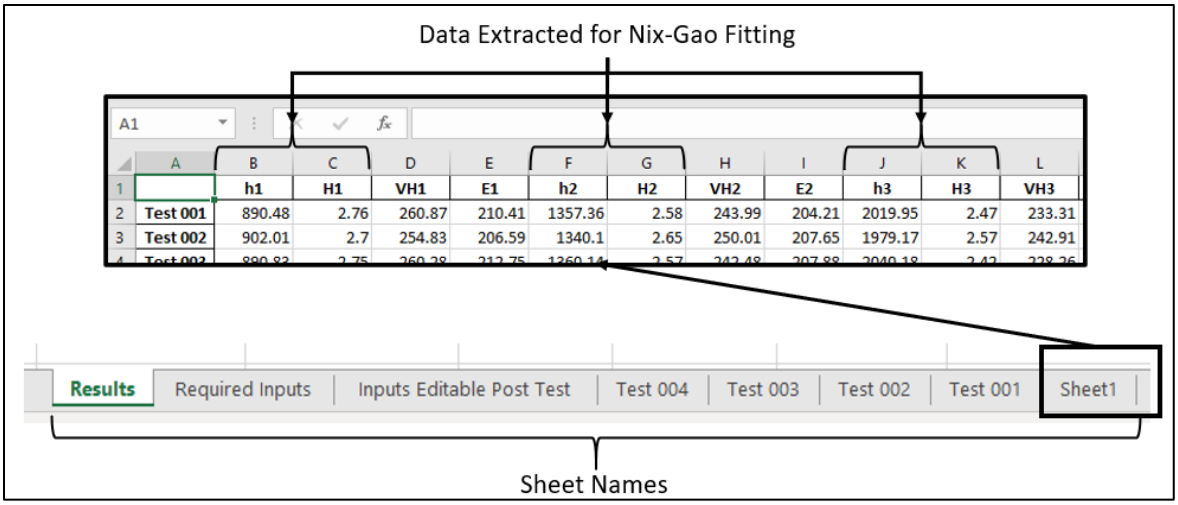


Figure 3.14 Indentation size effect data extracted from processed cyclic loading data file.

Chapter 4: Results

Microstructure Analysis

Figure 4.1 shows a composite micrograph of the 725 °C FSWed sample from a transverse section. The BM microstructure had fully recrystallized equiaxed gains with an average grain size of 47±16 µm. The SZ had a basin shape that was widest at the top surface, followed by a sharp decrease in width towards the center, and a gradual decrease in width until the bottom of the weld nugget. The base of the weld nugget was approximately 5.5 mm from the top surface. As expected, the SZ was enclosed by the TMAZ and then the HAZ on either side. In Figure 4.1 the transition between the different zones appears more diffused on the retreating side of the weld. The microstructure changed from small equiaxed grains, with average grain size 2.3±0.3 µm within the SZ, to severely deformed elongated grains in the TMAZ. The same observations are present in the 825 °C sample shown in Figure 4.2, except for a wider SZ and larger SZ average grain size of 4.6±0.6 µm, compared to the 725 °C sample. It is also apparent that the SZ widened in the 825 °C sample compared to the 725 °C

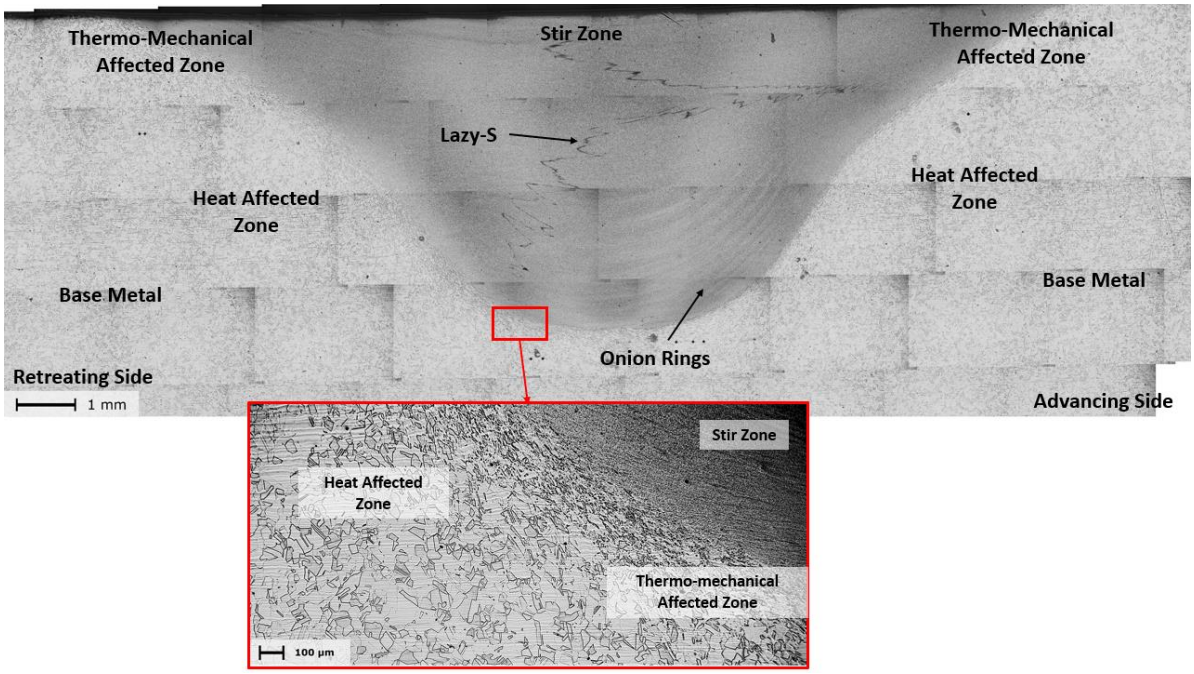


Figure 4.1 Composite micrograph of various zones in 304L FSW sample 1-D prepared with 725 °C tool temperature.

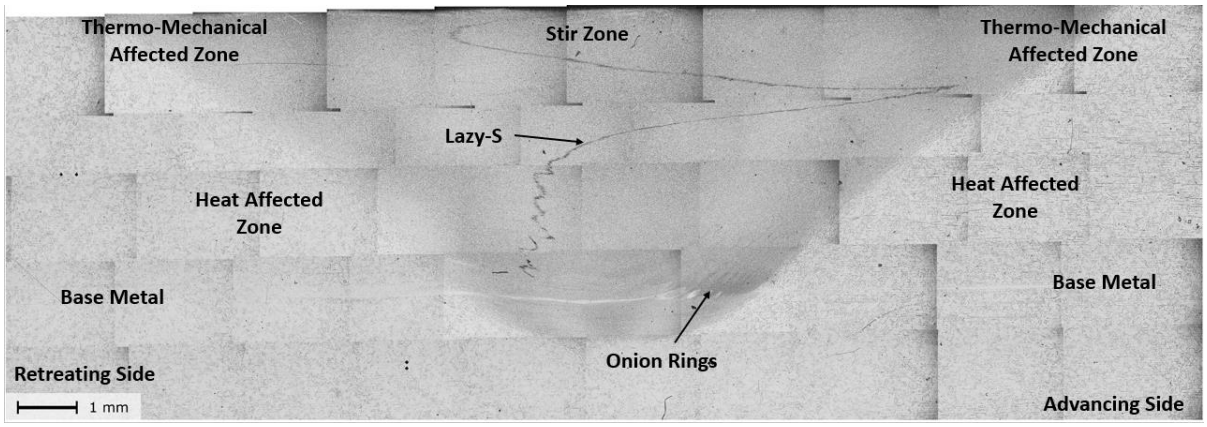


Figure 4.2 Composite micrograph of various zones in 304L FSW sample 1-C prepared with 825 °C tool temperature.

Onion rings and the 'lazy-S' were the two material-flow-induced features found within the SZ of both samples (Figure 4.1 and Figure 4.2). The onion rings appear under microscope as light-colored bands that reflect the basin shape of the SZ. These flow lines were only present in the advancing side of the SZ, stretching from ~1 mm from the top surface to the base of the weld nugget. The onion ring spacing decreased from the center of the SZ toward the TMAZ. Fewer onion rings were present in the 825 °C sample compared to the 725 °C sample. The 'lazy-S' was a nonuniform spiral feature that stretched from the top surface to roughly 0.4 mm from the base of the weld nugget.

Indentation Profiles

Microhardness and Nanoindentation Hardness Profile Comparison

Figure 4.3.a shows the nanoindentation hardness vs. indent depth of four indents within the 725 °C SZ, close to the weld center. The hardness decreased from 320 Hv to 280 Hv over a depth range of 2 µm within the SZ, typical of the indentation size effect. The initial hardness decrease was similar for the TMAZ and HAZ, but the overall profiles were shifted slightly downward (softer). The H_0 values for all regions determined were then plotted with the nanoindentation and microhardness profiles as shown in Figure 4.3.b. This plot shows the hardness across the entire weld section starting from the BM on the retreating side and ending in the BM on the advancing side of the 725 °C sample. The nanoindentation hardness profiles were shifted upward (harder) on the plot due to the indentation size effect. It is evident from Table 4.1 that the Nix-Gao fitting parameters are unique for each zone. It is important to note that the thermal drift for all nanoindentation tests was below 0.1 nm/s and all indent data were corrected with the individually recoded drift rate. H_0 was highest within the SZ then successively lowers moving outward from the weld center. The H_0 profile closely matches the VH3 microhardness profile. There is one exception that is on the cusp between the SZ and TMAZ on the advancing side where H_0 is lower by 41 Hv compared to the microhardness value at that location.

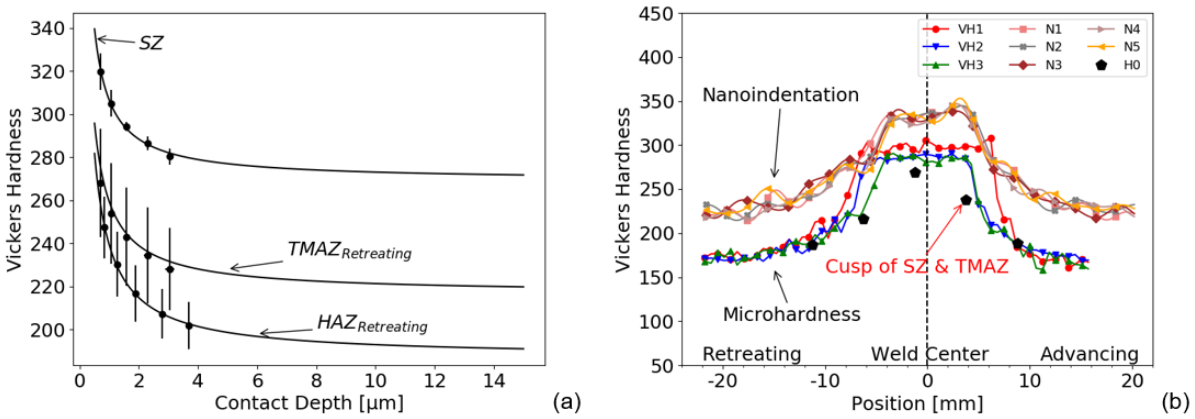


Figure 4.3 (a) Nix-Gao fit of cyclical nanoindentation indents in the SZ. (b) Microhardness and nanoindentation hardness profiles across the 725 °C 304L SS FSW sample, and infinite depth hardness (H0).

The VH3 and N1 profiles (Figure 4.4) were chosen to compare methods for determining the widths of the TMAZ and HAZ due to their proximity to one another. The width of each region was estimated by evaluating the slope changes across each hardness profile and microstructure correlations. Plotting the linear slope between two consecutive hardness values vs. position produced a varying profile, where significant changes in slope or inflection points indicate possible transition points between different zones. The different zone widths estimated for VH3 and N1 are presented in Table 4.1.

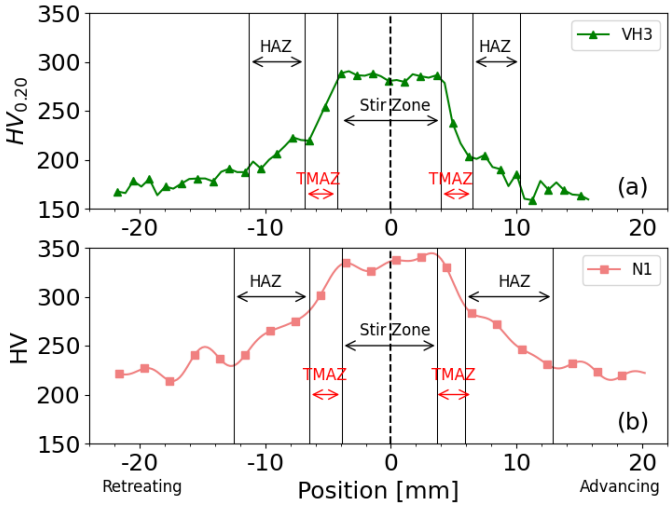


Figure 4.4 (a) Microhardness profile 1.905 mm from the top surface, (b) nanoindentation hardness profile 1.867 mm from the top surface of the 725 °C sample.

Table 4.1 Infinite depth microhardness, microhardness widths, and nanoindentation widths of each zone of the 725 °C approximately 1.905 mm from the top surface.

	HAZ_R	$TMAZ_R$	SZ	$TMAZ_A$	HAZ_A
VH3 [mm]	4.4	2.5	8.3	2.5	3.8
N1 [mm]	6.0	2.6	7.6	2.2	7.0
H_0 [HV]	187.08	216.64	269.07	237.76	188.10
h* [nm]	633.36	431.46	296.03	322.65	662.80

Uniaxial Tensile Test – Nanoindentation Comparison of Elastic Modulus

Figure 4.5 shows the elastic modulus profile generated by nanoindentation for line N1 (725 °C sample). In this plot the coefficient of variance (COV) of the elastic modulus is indicated with the black bars. The COV was calculated by determining the standard deviation of each point, then dividing it by the mean of the moving-average window. The length of the bars indicates that the variability of the elastic modulus is much lower in the SZ compared to other regions. A cluster of consecutive indents that measure the same elastic modulus is present in the base metal. Figure 4.5also shows that the uniaxial tensile test elastic modulus is slightly lower than the average indentation modulus across all zones. The tensile test and nanoindentation elastic moduli for the 725 °C sample are 201 GPa and 214 GPa, respectively.

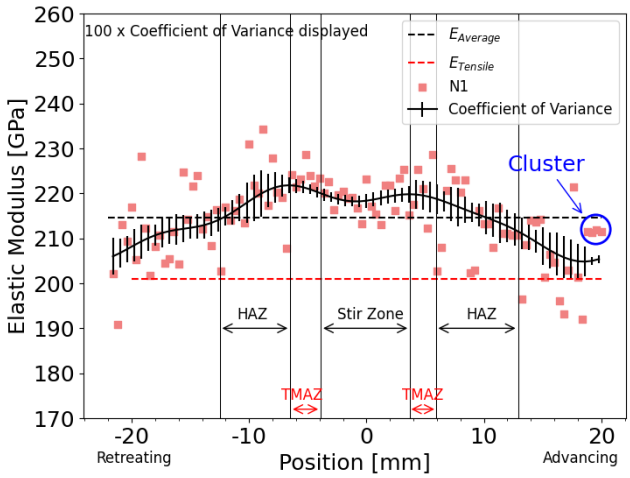


Figure 4.5 Nanoindentation elastic modulus profile of the 725 °C sample 1.867mm from the top surface.

The tensile test results for BM and FSWed samples are outlined in Table 4.2. The most notable differences between the as-received BM and the FSWed samples are with respect to the yield strength (YS), uniform elongation, and elongation to fracture. The YS of the FSWed samples are ~116 MPa higher than the as-received BM. Whereas both the uniform elongation and elongation to fracture are

less than half of the as-received BM. The 825 °C specimen had slightly higher yield and ultimate tensile strengths compared to the 725 °C.

Table 4.2 Tensile properties of the BM and FSW 304L SS (± indicate one standard deviation).

Sample	Elastic Modulus [GPa]	Yield Strength [MPa]	Ultimate Tensile Strength [MPa]	Uniform Elongation [%]	Elongation to Fracture [%]
BM	196±1.5	312±2.0	678±7.0	62.5±1.0	70.3±2.0
725 °C FSW	201±4.2	428±6.0	690±3.5	22.8±0.1	28.3±0.1
825 °C FSW	202±2.8	444±5.0	698±7.0	25.6±0.2	30.5±0.7

Indentation Contour Maps

Microhardness - Nanoindentation Hardness Contour Map Comparison

Figure 4.6 shows the microhardness and nanoindentation hardness maps for the 725 °C sample. The highest hardness is found within the SZ and has heterogeneous distribution from the top surface to the base of the weld. Where the highest hardness values are found closest to the top surface and gradually softens to the center of the SZ, which is then followed by a gradual hardening toward the base of the weld. The TMAZ and HAZ are clearly encapsulates the SZ. The transition from BM to HAZ to TMAZ is more diffused on the retreating side compared to the advancing side.

The regions outlined in red represents the overlap between the microhardness and nanoindentation hardness contour maps. The nanoindentation hardness map appears to have an overall increased hardness due to the ISE. The nanoindentation map also reflects the gradual increases in hardness from the SZ center to the base of the weld. The diffused transition from BM to TMAZ on the retreating side is also reflected in the nanoindentation contour map.

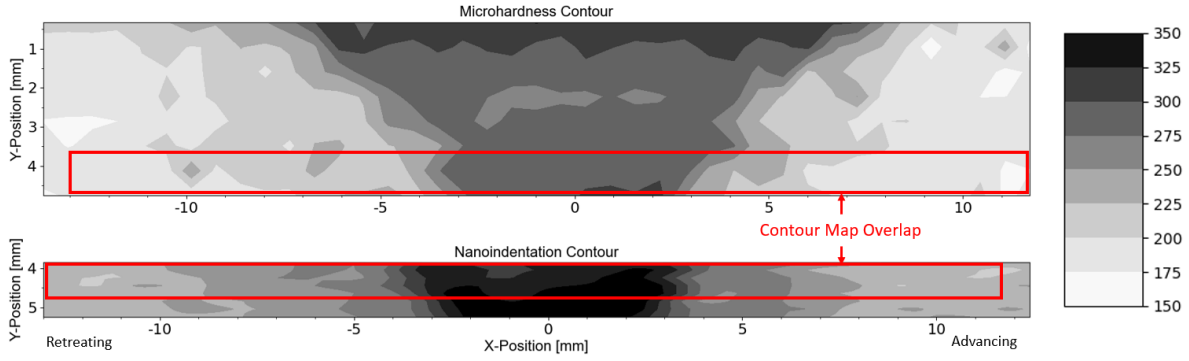


Figure 4.6 Indentation contour map of the 725 °C FSW sample. The graded scale represents the Hv.

The microhardness contour map for the 825 °C FSW sample (Figure 4.7) also reflects the basin shape like the 725 °C sample. However, the SZ appears wider closer to the top surface and the overall hardness within the SZ in lower compared to the 725 °C sample. The hardness within the 825 °C SZ is also heterogeneous, where the softest region is shifted closer to the base of the weld. The transitions from BM to HAZ to TMAZ are less distinct compared to the 725 °C sample. Again, the red outlined regions represent the overlap between the microhardness and nanoindentation hardness contour maps.

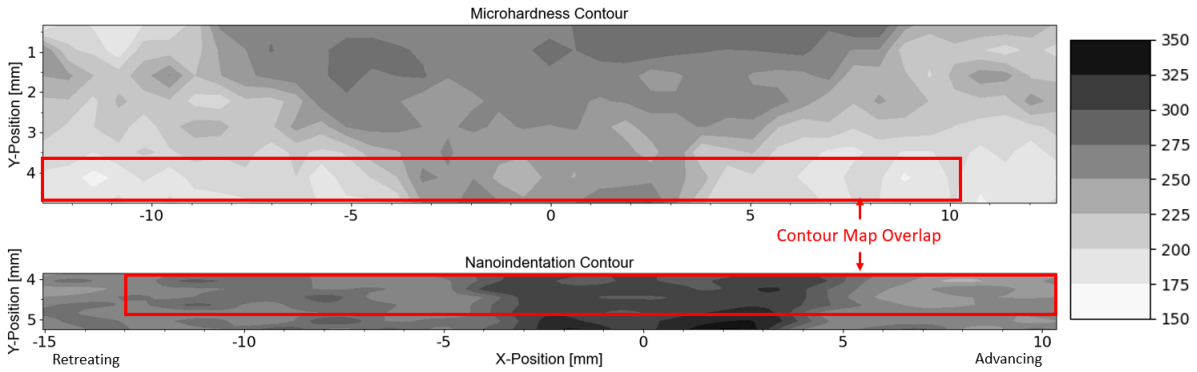


Figure 4.7 Indentation contour map of the 825 °C FSW sample. The graded scale represents the Hv.

Figure 4.8 shows the microhardness, nanoindentation hardness, and Nix-Gao H_0 for both 725 °Cand 825 °C FSWed samples approximately 4.445 mm from the top surface. As expected, the nanoindentation hardness profiles are shifted slightly upwards compared to the microhardness profiles and the H_0 profiles closely matches the microhardness profiles. Table 4.3 shows the Nix-Gao fitting parameters are unique for each zone. Comparing Figure 4.8.a and Figure 4.8.b it is clear higher tool temperature results in softening in all zones. This behavior was also observed when comparing H_0 of various zones (Table 4.3) for both samples.

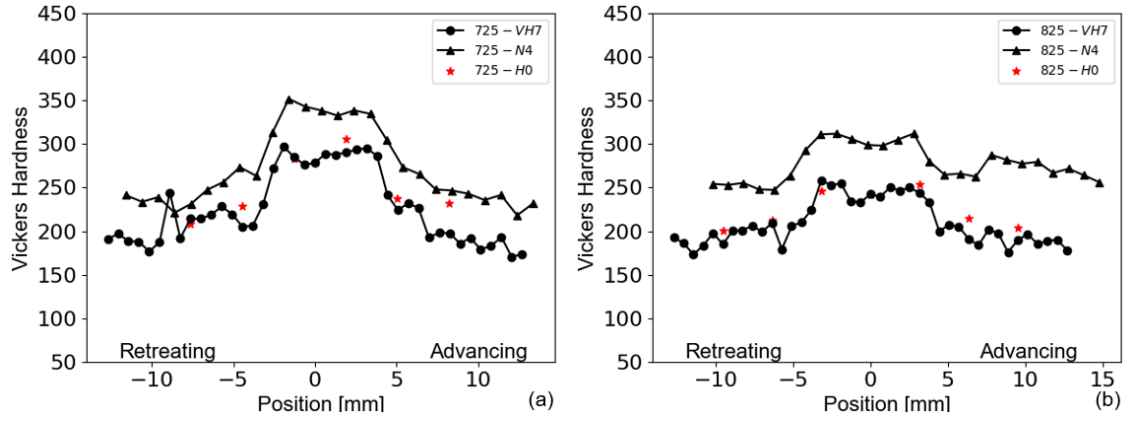


Figure 4.8 Microhardness and Nanoindentation profiles approximately 4.445mm from the top surface for (a) 725 °C and (b) 825 °C FSWed sample.

Table 4.3 Nix-Gao fitting parameters for both 725 °C and 825 °C FSWed samples approximately 4.445 mm from the top surface.

		HAZ_R	$TMAZ_R$	SZ	$TMAZ_A$	HAZ_A
725 °C	H_0 [HV]	228.5	283.1	305.2	237.4	231.5
	h* [nm]	403.2	116.1	146.8	130.3	342.2
825 °C	H_0 [HV]	215.2	253.4	241.5	246.6	212.2
	h* [nm]	491.5	279.6	244.6	246.0	411.51

Elastic Modulus Contour Maps

Figure 4.9 compares the elastic modulus contour maps, generated by nanoindentation, for the 725 °C and 825 °C FSW samples. The different zone widths were estimated from the microhardness and nanoindentation hardness contour maps. For both samples the SZ had the highest observed *E* compared to all other zones of the FSW. *E* within the 725 °C SZ was highest on the retreating side and lowest in the center. This behavior was also reflected in the 825 °C sample; however, it was on average ~10GPa lower than the 725 °C sample.

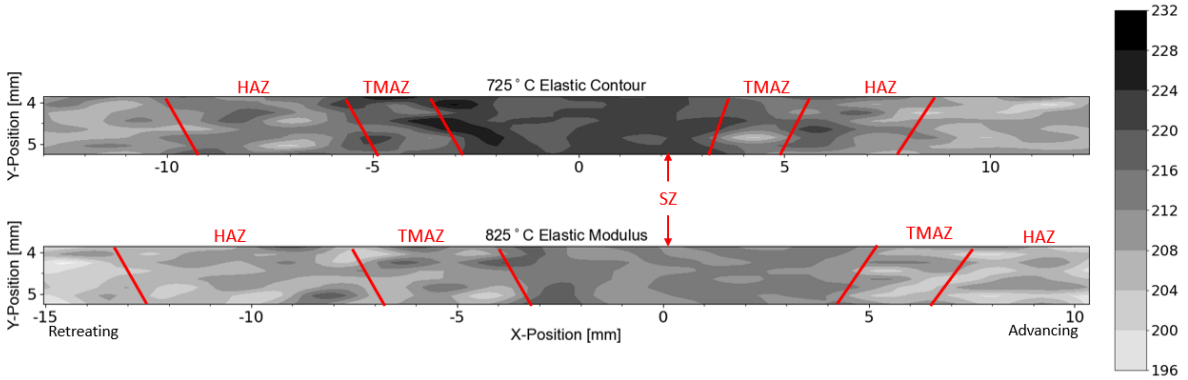


Figure 4.9 Nanoindentation elastic modulus contour map of 725 °C (top) and 825 °C (bottom) FSW sample. The graded scale to the right is in GPa.

Data Analysis Using Python

Python is an versatile object-oriented scripting programming language used for systems programming, GUI's, internet scripting, component integration, database programming, numeric and scientific programming, to name a few [38]. For this application, python was used to develop custom GUI's for data extraction, analysis, and visualization of microhardness along with nanoindentation data. The libraries used to create the GUI's can be found in

Table 4.4, which outlines the library name, short description of the function, what the libraries were used for, as well as library documentation reference.

Table 4.4 Python libraries used for GUI development with short library descriptions.

Library	Description	Use	Reference
os	Easy use of operating system dependent functionality.	Retrieve and list files in the current directory	[39]
pandas	Data analysis toolkit ideal for tabular data.	Data manipulation	[40]
numpy	Advanced numeric programming tools used for scientific computing.	Data analysis and smoothing	[38], [41]
openpyxl	Library to read and write Excell files.	Append processed data to existing Excell file	[42]
PyQt5	Platform independent abstractions for GUI, networking, SQL databases, 3D animation, etc.	GUI development	[43]
matplotlib	Data visualization utility.	Generate data plots	[44]
scipy	Collection of numerical algorithms and domain-specific toolboxes for scientific computing.	Curve fitting	[45]

A systematic process was followed to develop each GUI. First, the mechanics of data extraction, analysis, and visualization was developed in separate scripts to ensure that the code functions correctly. Secondly, user-defined classes (python object) were generated to encapsulate unique GUI functionality, like; text entry boxes and slider displays, in user-defined functions. Next, the user-defined classes (UDC) were incorporated into a custom GUI template and tested for proper functionality. Finally, the GUI template was modified for each unique application and the data extraction, analysis, and visualization scripts were integrated to produce the final GUI ready for future use. In total four GUIs were developed independently: (1) microhardness processing, (2) nanoindentation processing of single loading profile, (3) nanoindentation processing of cyclic loading profiles, and (4) ISE characterization.

Microhardness Processing GUI

The microhardness processing GUI, shown in Figure 4.10, was developed to have two separate functionalities. The first was dedicated to data processing (Figure 4.10, top left) and the second to data visualization (Figure 4.10, bottom left). The python script had six UDC, of which four were dedicated to generating objects that appear on the GUI like: drop down displays, text entry boxes, select buttons, and canvas tabs for data visualization. These UDC contained user-defined functions

(UDF) for retrieving the information currently displayed and stores them to unique variables when the button is clicked. The remaining two UDC were used to analyze the data and to create the sub-layout and assign functionality of the main GUI.

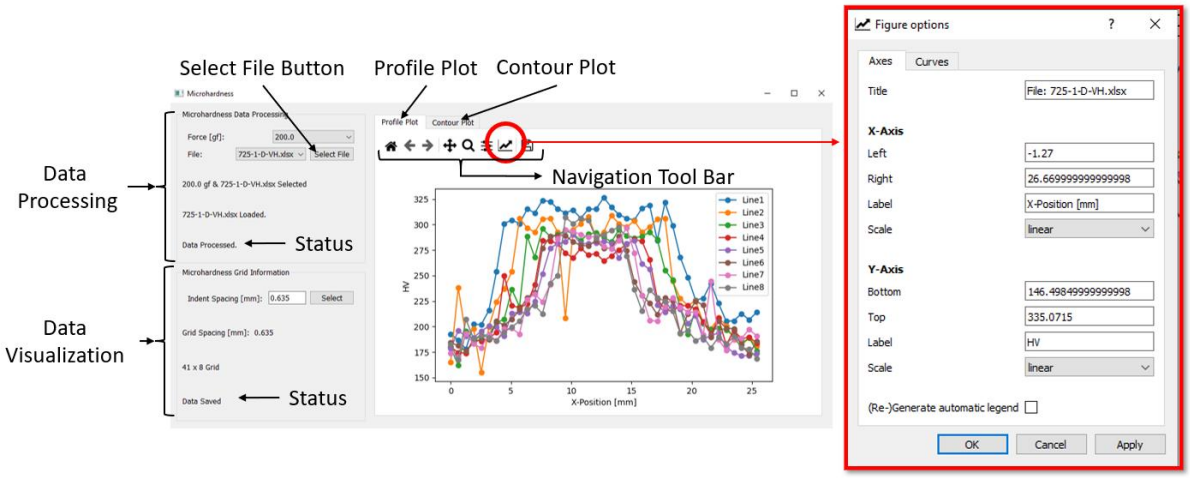


Figure 4.10 Microhardness processing GUI.

The first sub-layout was the data processing functionality (Figure 4.10, top left) and contained two drop down displays, one selector button, and three status message text boxes. The first dropdown display lists the applied force, in increments of 100 gf, stating from 0.0 gf to 1000.0 gf. The second dropdown display listed all Excell files ending in extension “VH.xlsx” that were in the current directory. Note that the file formatting is outlined in Chapter 3: Microhardness Data Processing is crucial for the function of the script and the data contained within the file are the measured diagonals of the microhardness impressions in µm. Once the force and file were selected by clicking the button and were assigned to unique variables for data processing use and displayed on the first status message. The Excell file selected is loaded and checked for formatting prior to data processing. If the file can be loaded its’ status message was then displayed on the second status message, after which the average microhardness diagonal was calculated in µm and then converted into the Vickers hardness number. Once the data has been processed the status is displayed in the third status message, which means it is ready for saving and data visualization.

The second sub-layout (Figure 4.10, bottom left) was used to save the processed data and to generate figures of the processed data. This sub-layout had one text entry box, one selector button and three status messages. The text entry box only accepts a “float” data type and represents the “x” and “y” grid spacing required to generate the profile and contour plots. If any other data type is entered the first status message would prompt the user to enter the appropriate data type. The profile plots and contour plot were automatically displayed to the right of the sub-layouts in separate tabs (Figure 4.10)

if the processed microhardness data is available and the gird spacing was entered. The “plot” and “contourf” functions were used from the matplotlib library to generate the profile and contour plots respectively. The maximum and minimum values used to create the contour scale are hard coded to be 330 and 150, which represents the Hv number. The axis limits and labels of the profile plot can be adjusted by clicking the “image parameters” button, on Figure 4.10 highlighted in red, on the plotting navigation tool bar.

Single Loading Profile GUI

The GUI shown in Figure 4.11 was developed to process the nanoindentation data of a single loading profile (Figure 2.4.b and Figure 3.7.a). The python script had four UDC’s, two created objects that appear on the GUI like: dropdown displays and canvas tabs for data visualization. The remaining two UDC were used to analyze the data and to create the GUI layout and assign functionality.

The single loading profile GUI (Figure 4.11) required one user input, the nanoindentation data file ending in extension “N.xlsx”, which was listed in the dropdown display next to the “Select File” button. It is very important to check that all indents were completed properly or that the user has removed any incomplete indent data from the nanoindentation file. Otherwise, the application will crash. The nanoindentation data file was loaded once the “Select File” button was clicked and the first status message was updated to reflect this. Once the file was selected the *E* and *H* data from the “Results” sheet (Figure 3.11) was retrieved from processing. Any outliers within the data were removed and the COV were calculated using the methods mentioned in Chapter 3: Data Analysis Using Python, Single Loading Profile. *H* and *E* data were smoothed independently using a one-dimensional discrete FFT algorithm for real-valued arrays within the numpy library (Table 4.4). An Fourier transformation is a mathematical expression that converts the data from the time and space domain to the frequency domain and vice versa [46]. An FFT is commonly used used to filter out any noise present in a discrete data set to get a better representation of trends present [46]. The specific FFT used was a numpy function called fft.rfft(*a*, *n*=None, *axis*=-1, *norm*=None). Where *a* represents the real-valued array, *n* is the number of points along the transformation axis to use, *axis* is the axis over which the to compute the FFT, and *norm* is the normalization mode. For this application the *axis* and the *norm* inputs are left as default and *a* was 8 and 5 for *H* and *E* smoothing respectively. These values dictate the amount of distortion the smoothed profile will exhibit with respect to the raw data and were determined through trial and error. After smoothing the processed data was then appended to the Excell file and displayed in the plotting tabs (Figure 4.11).

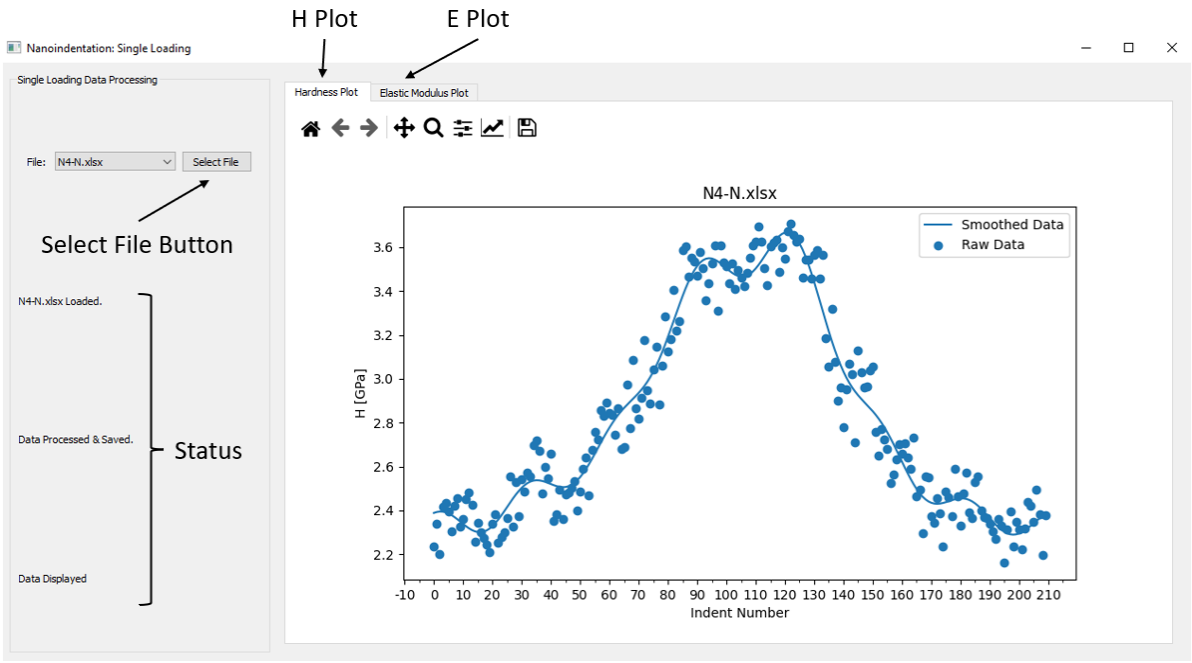


Figure 4.11 Nanoindentation single loading profile GUI.

Cyclic Loading Profile GUI

The cyclic loading profile GUI, shown in Figure 4.12, was developed to process the nanoindentation data with "n" number of cycles (Figure 2.6.a) to extract the elastic modulus along with nanoindentation hardness and *VH* at each cycle depth. The python script had four UDC, of which two were dedicated to generating objects that appear on the GUI like: drop-down displays, select buttons, and canvas tabs for data visualization. The UDF within these UDC's were created to retrieve selected information and assigning them to unique variables when the select buttons were clicked. The other two UDC's were dedicated to creating the GUI interface, assign functionality and to process the nanoindentation selected.

Figure 4.12 Nanoindentation cyclic loading profile GUI.

As seen on the left of Figure 4.12, the GUI has two dropdown displays with selector buttons, three status message text boxes, plotting button, followed by a plotting status message text box. Two figure tabs were on the right-hand side of the GUI. The first displayed the *H* vs *h* plot of the processed data and the second showed the *E* vs *h* plot. The first drop-down display showed the different nanoindenter tip calibrations entered in the Excell file named "TipGeometry.xlsx" and the first status message text box is updated when the desired tip was selected by the clicking the "Select Tip" button. The next drop-down display listed all Excell files ending in extension "ISE.xlsx". The raw nanoindentation data was processed once the tip geometry and raw data file was selected by clicking the "Select File" button. It is very important to check that all cyclical indents were completed properly or removing partial indents before selecting the raw data file. Otherwise, the data processing won't be completed, and the application would be terminated be for completion. It is important to note that this GUI was specifically developed to process indentation data for 304L SS, thus υ was hardcoded to be 0.29 within the data processing UDC.

The UDC dedicated to processing the raw nanoindentation data uses the procedure outlined in Chapter 2:Nanoindentation and methods discussed in Chapter 3:Data Analysis Using Python, Cyclic Loading Profile to determine the *H, VH,* and *E* at each *h.* To reiterate, the data is first corrected for the recorded drift rate of each indent, then *H* is determined parsing through the corrected raw data and retrieving the recorded *P* and *h* values at the very end of the 30 sec hold (Figure 2.6.a and Figure

3.7.b) for each cycle. *E* of each cycle was determined by first determining the system *S* (slope of linear fit) using the first 15 data points of the cycle unloading curve, by implementing the "curve_fit" function within "scipy" library. The "curve_fit" function implements a non-linear least square regression algorithm to determine the best fitting parameters. From *S*, h_c was determined which in turn allowed *A* to be calculated. Next E_r was also determined from *S* and then used to find *E* of each cycle. The third status message box was updated to show the selected file has been processed and saved once *H, VH,* and *E* were determined. Finally, the data was visually represented by pressing the "Plot" button. Plotting was only done if the current tip geometry and file displayed in the drop-down displays match those of the processed raw data.

Indentation Size Effect GUI

Figure 4.13 shows the ISE GUI and it was developed to use the processed data file from the cyclic loading profile GUI (Figure 4.12) to determine the Nix-Gao parameters of each indent. The Nix-Gao parameters of interest are outlined in Chapter 2:Indentation Size Effect and the methods implemented are found in Chapter 3:Data Analysis Using Python, Indentation Size Effect Analysis. The ISE GUI python script used four UDC's, where three were dedicated to generating and imbedding objects on the GUI interface, like: slider selectors, drop-down displays, select buttons, and canvas tabs for data visualization. The remaining UDC was dedicated to assigning GUI functionality and extracting the processed cyclic loading profile data, determining and saving the Nix-Gao parameters, along with displaying the characteristic ISE curve.

The ISE GUI was divided into two sections. The first (left of Figure 4.13) was dedicated to selecting the number of cycles and data file of interest. The second (right of Figure 4.13) was created for visualizing the Nix-Gao fit generated for the given data. The slider display was hard coded to select a minimum of 2 cycles and a maximum of 10 cycles. Like the cyclic loading GUI, the dropdown display only lists the Excell files ending in extension "ISE.xlsx". Be sure to select the files that has been processed by the cyclic loading GUI, otherwise the ISE GUI will not function. The Nix-Gao parameters was determined by using the "curve_fit" function within "scipy" library. Two status message text boxes indicate which file was selected and if the file was processed and saved. The Nix-Gao model and raw data points was then automatically displayed in the plotting tab, which is shown on the right-hand side of Figure 4.13.

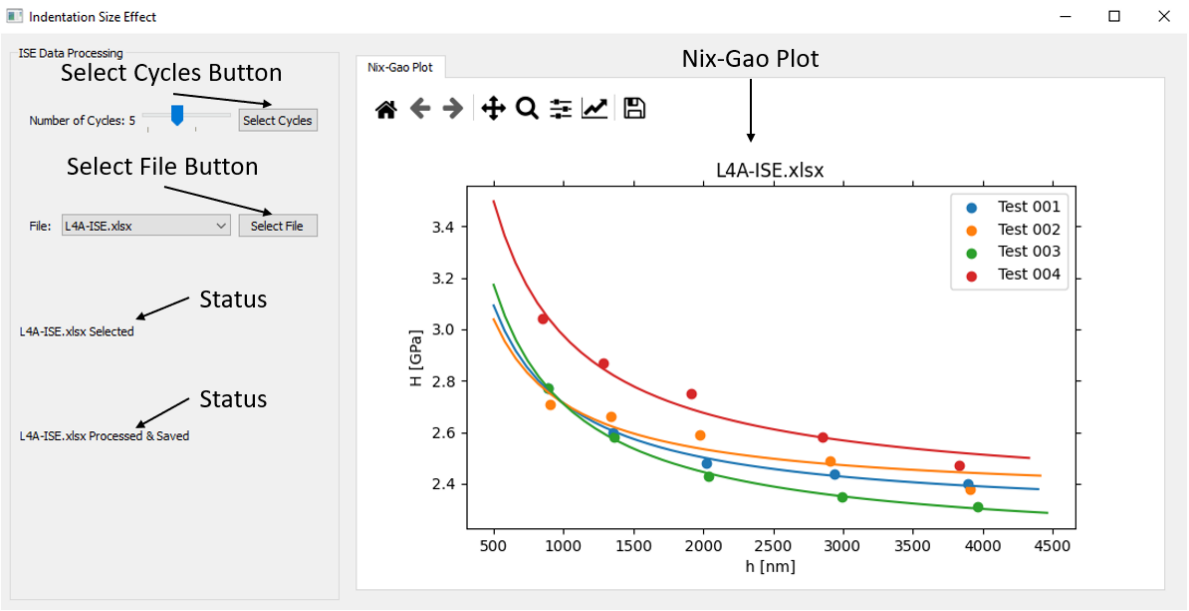


Figure 4.13 GUI using the Nix-Gao model to process ISE data.

Chapter 5: Discussion

Microstructural Analysis

The fine equiaxed grains within the SZ result mostly from dynamic recrystallization [10], [47]. The distinct basin shape of the SZ is caused by severe plastic deformation and frictional heating between the material and the tool profile [3]. Differences in material flow on the advancing and retreating side for the rotating tool affect the transition sharpness from the SZ to HAZ seen in Figure 4.1 and Figure 4.2 [47], [48]. Increased mixing on the retreating side promotes a wider and more diffuse transition between the different zones [23]. Mixing is promoted on the retreating side because the weld travel speed and tool rotation are in opposite directions [23]. These differences in material flow on the advancing and retreating side for the rotating tool affect the transition sharpness from the SZ to HAZ seen in Figure 4.1 and Figure 4.2. The elongated grains within the TMAZ originate due to plastic deformation and insufficient deformation strain, which result in little recrystallization [3].

The onion rings reflect localized differences in grain size and particulate density within the SZ. These localized differences are caused by deformation differences resulting from shear layer flow [3], [23], [49]. The ‘lazy-S’ appears because of impaired mixing, promoted by the presence of second phase particles, along the vortex of the material flow [1], [26]. As reported by Bhattacharyya et al., SEM-EDS analysis of this sample showed indications of second phase particles within the onion rings and the ‘lazy-S’[1].

Hardness Comparison of Indentation Methods

Profile Comparisons

The indentation size effect was characterized for different regions of the 725 °C sample to validate that nanoindentation hardness can be correlated with microhardness values. This characterization showed that each zone has a unique indentation size effect. The average grain size is smallest within the SZ then successively grows outward from the weld center. The H_0 profile in Figure 4.3.b closely reflects that of VH3, except for one outlier at the cusp between the SZ and TMAZ. At this location three of the four indents were within the TMAZ with one in the SZ which inflates H_0 at this point. The consistency of H_0 with the microhardness profile VH3 indicates that nanoindentation can be quantitatively compared with Vickers when the appropriate data corrections are used.

The bell-shaped nanoindentation hardness and microhardness profiles in Figure 4.3.b results from the process of grain refinement within the SZ, TMAZ, and HAZ [10]. The BM and SZ grain size relative to nanoindentation impression is shown in Figure 5.1.a and Figure 5.1.b, respectively. These profiles for FSWed steels are usually characterized by a diffuse increase in hardness from the retreating BM

to a plateau within the SZ followed by a sharp decrease to the BM on the advancing side [3], [50]. The microstructural relationship across the different zones is highlighted in Figure 5.2. Figure 5.2 shows enlarged optical images of each zone relative to the nanoindentation profile zones. The defuse transition on the retreating side is captured by nanoindentation and reflects the microstructure. This is also a representation of the Hall-Petch relationship between material hardness and grain size [10], [11]. The Hall-Petch relationship describes the inverse relationship between material strength and grain size. The relationship exists because dislocation motion is prohibited/delayed due to the increased number of grain boundaries encountered in fine grained material [10], [17].

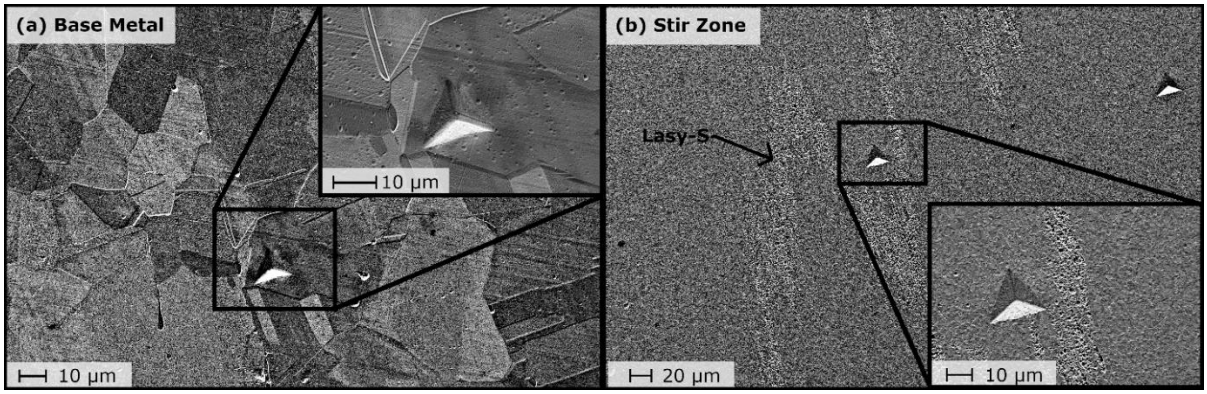


Figure 5.1 SEM image of nanoindentation size relative to grain size in (a) base metal and (b) stir zone of the 725 °C sample.

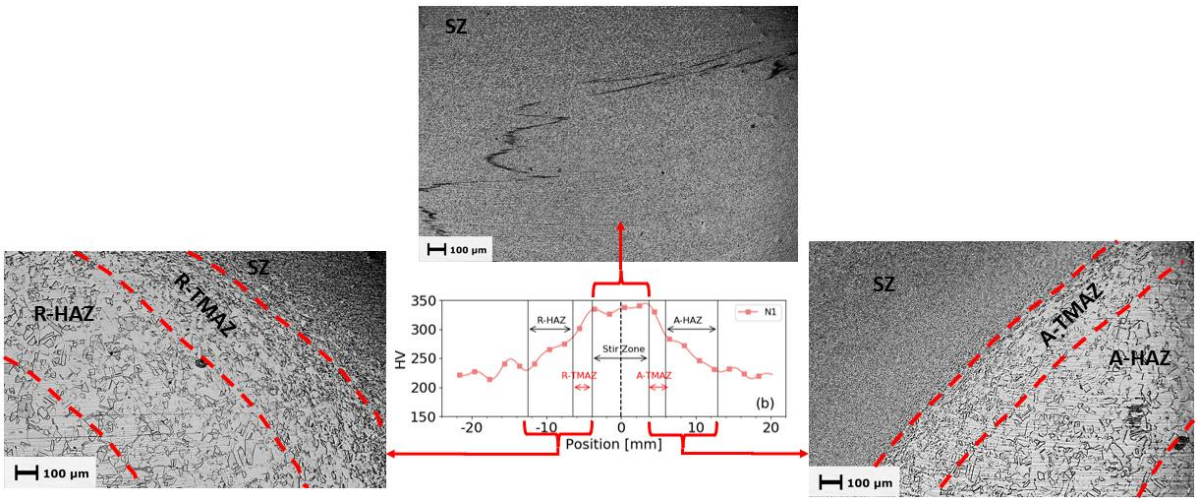


Figure 5.2 Microstructural zones relative to the 725 °C nanoindentation hardness profile approximately 1.867 mm from the top surface

The VH1 plateau is significantly wider than VH2 and VH3 (Figure 4.3.b). This is expected because VH1, VH2, and VH3 are successively further from the top surface of the sample. The nanoindentation profiles indicate no significant differences in the width of the plateau within the SZ

and have a similar width to VH3. This is because the nanoindentation profiles are spaced much closer together and over the same region that is represented by VH3.

The indentation method comparison of the 725 °C sample shown in Figure 4.4 indicates that nanoindentation was able to detect the slight differences between $TMAZ_R$ and $TMAZ_A$ widths, whereas microhardness testing does not. The TMAZ widths measured using microhardness testing were the same, 2.54 mm, on both the advancing and retreating side. Using nanoindentation, however, $TMAZ_R$ and $TMAZ_A$ were found to be 2.6 mm and 2.2 mm respectively. Nanoindentation also captured the slight microstructural changes within the HAZ by detecting small spatial changes in hardness. These changes correlate to the gradual grain size increase from the innermost zone within the HAZ (closest to TMAZ) to the outermost region (closest to BM). In this case, the HAZ_R width measured using nanoindentation was 6.0 mm, which is 1.56 mm wider than the width determined by microhardness testing. The difference in width between nanoindentation and microhardness for HAZ_A is even larger. Here the nanoindentation-measured HAZ_A is 7 mm wide, 3.19 mm wider than the microhardness width. The nanoindentation method, due to its smaller length scale, is more sensitive to spatial microstructural changes within the HAZ and provides greater resolution compared to microhardness testing; thus, making nanoindentation more suitable for estimating the HAZ width of FSWed materials.

Difference in average grain size between the 725 °C and 825 °C samples indicates that more grain growth occurred in the 825 °C sample following recrystallization. This results in the downward shift of hardness profiles and lower H_0 values reported for the 825 °C sample (Figure 4.8 and Table 4.3). The widening of the SZ with increased tool temperature has been documented for 316L SS and is attributed to discontinuous dynamic recrystallization and grain growth within samples prepared with higher tool temperature[11].

The 725 °C hardness profiles and H_0 values closest to the top surface (Table 4.1), where the grains are larger, are shifted lower compared to those closest to the base of the weld (Table 4.3), where the grains are smaller. This shows that microhardness and nanoindentation can detect the heterogeneous grain size distribution within the SZ, which has been documented in previous work using EBSD [1].

Hardness Contour Map Comparisons

Figure 5.3 shows the EBSD determined nominal grain size relative to microhardness within the SZ of both FSWed samples. Further investigation into the heterogeneous SZ microstructure revealed that the nominal grain size is smallest closest to the base of the weld and largest in the center of the SZ [1]. From Figure 4.8 and Figure 5.3 it is evident that grain size, grain size distribution, and resulting

hardness are highly dependent on FSW processing temperature. Higher FSW processing temperature results in increased grain growth producing larger nominal and average grain size which results lower hardness and widening the observed SZ [11], [12]. The less distinct transition from BM to HAZ to TMAZ observed on the 825 °C FSWed sample is attributed to the increased thermal cycles experienced during processing.

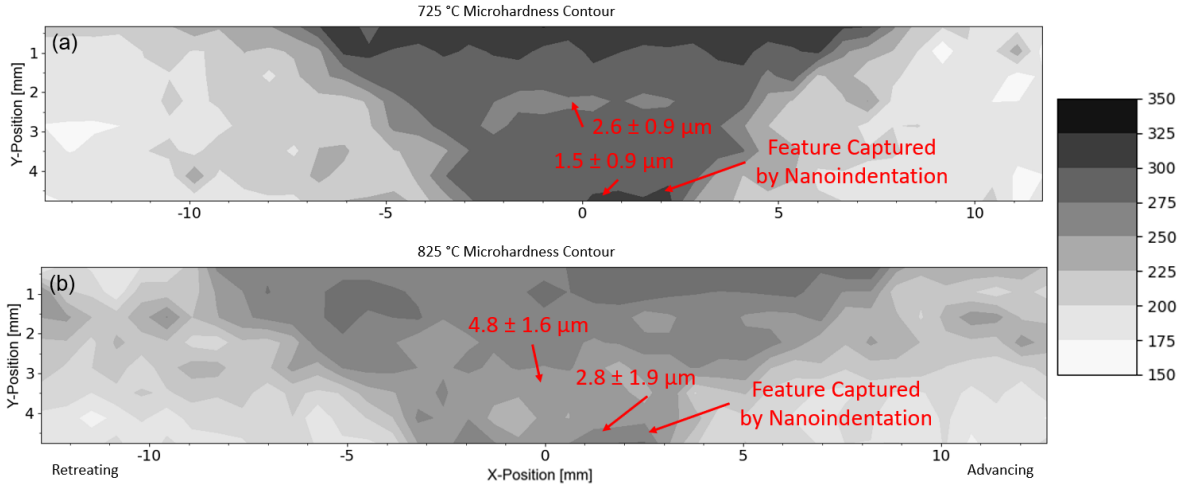


Figure 5.3 Nominal grain size determined by EBSD [1] within the SZ of (a) 725 °C and (b) 825 °C FSWed sample.

The features reflecting higher hardness at the base of the microhardness contour maps (Figure 5.3) were captured in higher resolution with nanoindentation (Figure 4.6 and Figure 4.7). These features overlap the regions where onion rings and “lazy-S” microstructural features are present within the SZ and are shown in Figure 5.4 and Figure 5.5. These microstructural features are larger and more prominent within the 725 °C FSWed sample (Figure 5.4) compared to the 825 °C sample (Figure 5.5). Nanoindentation hardness contour maps successfully captured the size and shape of these microstructural features, which is attributed to the higher resolution (smaller indent spacing) this indentation method provides. The nanoindentation contour maps encapsulates 1024 indents spaced 0.200 mm apart, whereas the microhardness contour map encompasses 328 indents spaced 0.635mm apart. Thus, indicating that nanoindentation is more appropriate for microstructure correlation of FSW material flow features like onion rings and “lazy-S”.

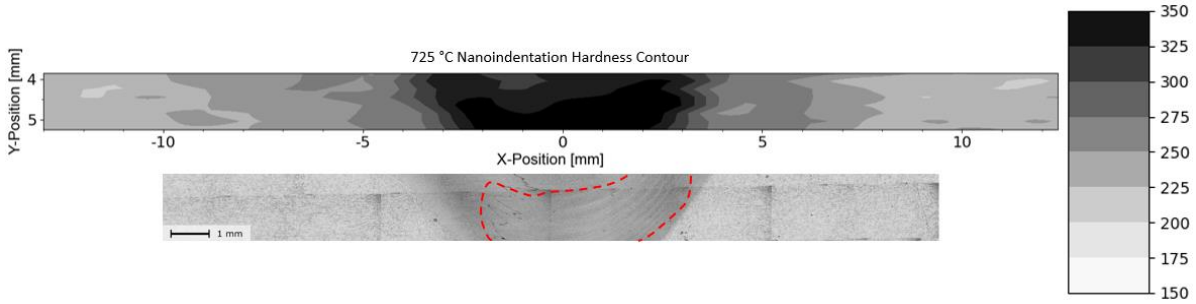


Figure 5.4 725 °C FSWed sample nanoindentation hardness contour map correlation to microstructural features.

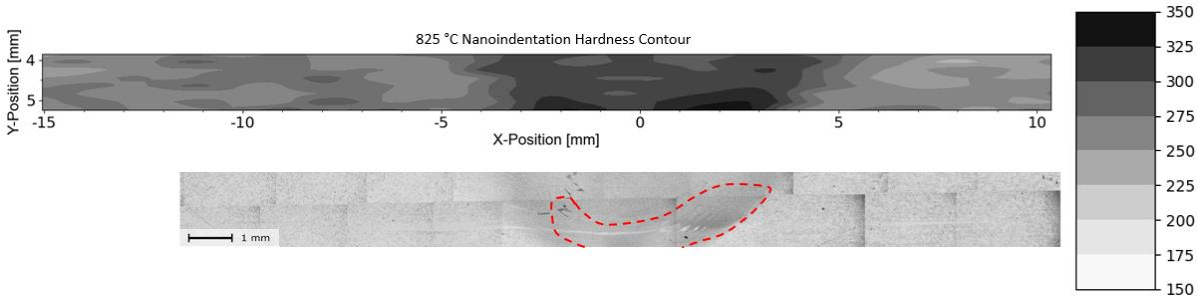


Figure 5.5 825 °C FSWed sample nanoindentation hardness contour map correlation to microstructural features.

Elastic Modulus Comparison: Uniaxial Tensile Test and Nanoindentation

The increase in YS (Table 4.2) of the FSWed samples are caused by the grain refinement within the SZ. Necking occurred in the transition region between the BM and the retreating HAZ and resulted in fracture within the BM for both 725 °C and 825 °C samples. Most of the elongation occurred outside the SZ and is attributed to the microstructural change observed from the center of the weld to the BM. This microstructural variation between different regions of the FSWed samples are the main contributor to the reduced ductility compared to the as-received BM [1].

The average elastic modulus determined by nanoindentation, for the 725 °C sample, is 15 GPa higher than uniaxial tensile tests (Figure 4.5). Comparing the physical sample size of the tensile test to that of nanoindentation, it is apparent that the amount of material tested by nanoindentation is less. The average indentation elastic modulus is biased by the number of indents within the SZ compared to the BM. If more indents were conducted in the BM, the average would shift lower, closer to that measure by the tensile tests. The gradual increase and then decrease in modulus from the retreating to advancing side of the weld indicates that crystallographic texture is also influencing the elastic modulus. The relationship between crystallographic texture and elastic modulus are discussed in detail in the next section.

Indentation Variability and Relationship to Crystallographic Texture

Elastic Profiles

Face centered cubic (FCC) materials like austenitic steels exhibit highly anisotropic behavior with crystallographic orientation [51]. Evident from Figure 5.1.b, the nanoindentation elastic modulus within the SZ reflects the average of many small equiaxed grains. Thus, assessing the elastic modulus COV in different regions of the FSWed sample should identify crystallographic texture changes.

In regions where grain size is larger than the indent size, increased variability will result. This is because individual indents will return only the modulus of that large grain. In regions where the grain size is much smaller than the indent size, the average modulus of many grains with unique orientations will be returned resulting in a low COV. Figure 5.1.a shows that the nanoindentation impressions are smaller than the grain size in the BM zone, where a single indentation is more likely to probe a specific crystallographic orientation rather than that of many grains. This is further substantiated by the clustering of elastic modulus values within the advancing BM, which would result from several indents in a single large grain (Figure 4.5). The COV within the SZ is significantly smaller than the other regions within the FSWed sample (Figure 4.5). This is due to the average grain size being significantly smaller than the nanoindentation impression in this zone [18]. The resulting elastic modulus within the SZ is thus representative of the average of many grains.

EBSD analysis using orientation distribution mapping and orientation distribution function in 304 SS from Hajizadeh et al. [8] indicates a simple shear texture within the SZ, predominantly aligned in the <110> direction. The BM of Hajizadeh et al. was equiaxed and randomly oriented. Since the processing parameters were similar, it can be assumed that the dominant orientations in the weld zones of the present study are similar. Eq. 9 can be used to determine the modulus in any crystallographic direction based on the modulus in the <100> and <111> directions [51].

$$\frac{1}{E_{[hkl]}} = \frac{1}{E_{<100>}} - 3\left(\frac{1}{E_{<100>}} - \frac{1}{E_{<111>}}\right)(\alpha^2\beta^2 + \alpha^2\gamma^2 + \beta^2\gamma^2) \qquad (9)$$

In Eq. 9, α, β, and γ represent the direction cosines of the crystallographic directions and E is the elastic modulus in the direction indicated by the subscript. For AISI 304L stainless steel $E_{<100>}$ and $E_{<111>}$ are 179 GPa and 208 GPa, respectively [20]. The elastic modulus within the SZ (~$E_{<110>}$) was calculated to be 200.0 GPa by substituting the values for $E_{<100>}$, $E_{<111>}$, and the direction cosines for the <110> direction into Equation 3. This is in reasonable agreement with measured indention values within the SZ, considering the variable nature of individual grain orientations within textures, slight off-axis character of the sample texture from the <110> direction, and hemispherical plastic zone sampled by the indenter [20]. The $E_{<110>}$ value was verified using the elastic compliance method,

shown in Eq. 10, where three independent stiffnesses are required to describe the anisotropic behavior of cubic crystals [16].

$$\frac{1}{E_{[hkl]}} = \frac{1}{S_{11}} - 2(S_{11} - S_{12} - \frac{1}{2}S_{44})(\alpha^2\beta^2 + \alpha^2\gamma^2 + \beta^2\gamma^2) \quad (10)$$

In Eq. 10, S_{11}, S_{12}, and S_{44} represent the single crystal elastic compliance constants of a cubic crystal oriented in the <*hkl*> direction. AISI 304 SS compliances from three different sources were used for validation and are provided in Table 5.1.

Table 5.1 AISI 304 SS compliance constants and calculated $E_{<110>}$

Reference	S_{11} **[10^{-3} GPa^{-1}]**	S_{12} **[10^{-3} GPa^{-1}]**	S_{44} **[10^{-3} GPa^{-1}]**	$E_{<110>}$ **[GPa]**
[52]	9.47	-3.68	8.27	201.5
[53]	9.43	-3.66	8.26	202.0
[54]	11.63	-4.76	7.25	190.6

The profile in Figure 4.5 shows the change in modulus across the weld, indicating that nanoindentation can be used to estimate changes of the dominant crystallographic texture within the metallurgical zones, once calibrated.

Elastic Contour Maps

The elastic modulus contour maps generated by nanoindentation (Figure 4.9) shows that FSW processing temperature has a significant effect on the local *E* observed within the SZ. Suggesting that crystallographic texture development is influenced by the thermal history and that the 725 °C and 825 °C FSWed samples each have unique dominant textures within the SZ. This behavior has been observed, using EBSD, in 316L and 304 SS FSWed parts [8], [11]. Literature also revealed that the SZ_A, SZ_{center}, and SZ_R each have unique textural components [11]. Bhattacharyya et al. [1] completed localized EBSD analysis of SZ_{center} close to the top surface, middle, and close to base of both 725 C and 825 C FSWed samples. However, no conclusions were made regarding the dominant crystalline orientation. Jeon et al. [29] also reported the heterogeneous texture development within the SZ along with unique texture development within the HAZ and TMAZ of single crystal FSWed 316 SS. From Figure 4.9 it is evident that the TMAZ and HAZ *E* distributions are distinctly different from the SZ and BM for both 725 °C and 825 °C FSWed sample. The overlap in literature texture observations and behavior seen in the elastic modulus contour maps indicate that nanoindentation can be used to estimate texture changes across the FSW zones, once appropriately calibrated with EBSD analysis.

Python Data Analysis Tools

The Python data analysis GUI development came with unique challenges. These challenges were minimized by implementing a systematic approach to developing each GUI. As seen throughout Chapter 4:Data Analysis Using Python all GUI's have similar layouts. The template script used for GUI development was developed separately from data extraction, analysis, and visualization scripts. This allowed to debug and integrate scripts or functionality more easily. Three observations were notable while developing data processing GUI's: (1) appending data files with the processed data takes a long time, (2) cyclic loading data processing is computationally expensive, and (3) the data visualization plots within the nanoindentation GUI's does not update consistently.

The addition of status message(s) to each GUI was strategic to signal the user which file is being processed and if the processed data has been saved. Processing time is defined to be from the time the "Select File" button has been clicked until the user is signaled that the processed data has been saved. For the single loading profile GUI, most of the processing time is designated to saving (appending) the processed data to the Excell file. This is because the raw data file has a significantly larger size (~1500 KB) and number of sheets (>130) compared to microhardness raw data files (~22 KB, 9 sheets). Processing the raw data takes ~ 0.7 sec while saving the processed data takes ~3.8 sec. While saving the data in the raw data file is copied, then the processes data is appended to the designated sheet and then the whole file is then rewritten. Unfortunately, this is an attribute of the "pandas" and "openpyxl" libraries which cannot be addressed. For the cyclic loading profile GUI, data processing time is increased due to the nested loops designated to calculating H and E for each indentation depth. This processing time cannot be reduced because there is no other way of determining the distinct markers for determining the number of cycles within the raw data files.

The saving/appending of the processed data and the plot visualization of large files causes a bottle neck within the CPU of the PC being used. Resulting in inconsistent updating of plots of subsequent raw data files being processed. An example of this is shown in Figure 5.6, where there is a mismatch between the file name displayed on the status message and that displayed within the plotting tabs. The user must force the plot to update by pressing the "pan" button on the navigation tool bar and then on the plot. Please note this is necessary for all GUI's other than the "Microhardness Processing GUI". No bottle neck is created within the CPU when processing and plotting the microhardness data.

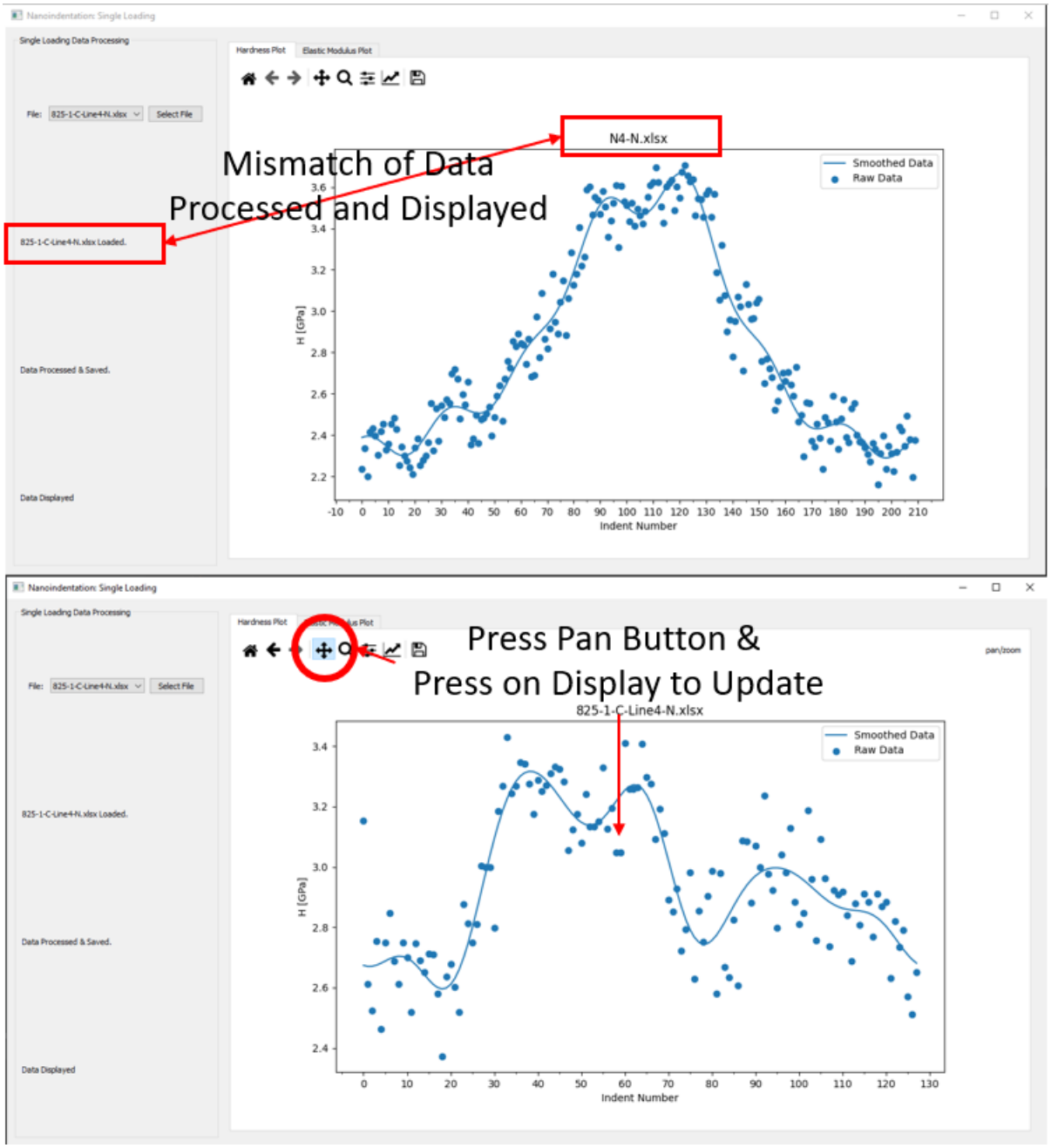


Figure 5.6 Manually updating plot tab images for all nanoindentation data processing GUI's.

Chapter 6: Conclusions

The indentation size effect was characterized for the SZ, TMAZ, and HAZ of 304L FSWed samples prepared with 725 °C and 825 °C tool temperatures. H_0 determined from the indentation size effect was used in conjunction with the microstructure composite micrographs to estimate zone widths between nanoindentation and microhardness. Using nanoindentation the TMAZ width is estimated to be slightly larger and the HAZ is significantly larger than that determined by microhardness for both FSWed processing parameters. The hardness profiles and contour maps showed that higher processing temperature results the SZ to soften and widen due to increased grain growth. Elastic modulus profiles generated by nanoindentation showed increased variability within the BM compared to the SZ. Elastic modulus contour maps generated with nanoindentation showed that FSW processing temperature has a significant effect on the elastic modulus magnitude and distribution within the SZ. From these observations the following conclusions can be drawn:

- Higher tool temperature results in widening of the SZ, increased grain growth, and reduced overall hardness.
- Changes in average grain size between each zone results in unique indentation size effects within each zone.
- H_0 values determined by nanoindentation closely reflect the microhardness values.
- Nanoindentation has adequate resolution to capture slight microstructural changes across the different zones.
- Nanoindentation hardness contour maps with adequate resolution are capable of detecting material flow line features, like: “lazy-S” and onion rings.
- Variations of elastic modulus across the weld are due to texture.
- The significantly smaller average grain size within the SZ caused the elastic modulus COV in this zone to be substantially smaller than TMAZ, HAZ, and BM COV.
- Comparison of elastic modulus contour maps, generated by nanoindentation, suggest that nanoindentation can capture crystallographic texture changes across FSW zones.
- EBSD analysis is required to calibrate elastic modulus contour map texture prediction capabilities.

Four data processing Graphical User Interfaces (GUI’s) were developed to process the raw data files of the microhardness and nanoindentation tests. All GUI’s displayed graphical representation of the processed data.

Chapter 7: Future Work

Four recommendations are to be considered for future work. First, it is recommended that larger nanoindentation maps be constructed to capture the entire FSW region along with decreasing the indent spacing to 100 μm to capture an even higher resolution indentation map. Secondly, the larger and higher resolution nanoindentation maps should then be correlated to a composite EBSD map of the FSW to validate that nanoindentation can be used to capture the dominant texture within each zone of the FSW. Thirdly, X-ray diffraction (XRD) analysis should be completed at strategic point within the SZ, like the lazy-S and onion rings, to see if nanoindentation can capture specific phases present within the microstructure. Lastly, the Hall-Petch relationship between hardness and average grain size of the different microstructural zones should be investigated fully through correlation to known literature and experimentation.

References

[1] M. Bhattacharyya, A. Kundu, K. S. Raja, J. Darsell, S. Jana, and I. Charit, "Processing-microstructure-property correlations for temperature-controlled friction stir welding of 304L SS plates," *Mater. Sci. Eng. A*, vol. 804, p. 140635, Feb. 2021, doi: 10.1016/j.msea.2020.140635.

[2] R. S. Mishra, P. S. De, and N. Kumar, *Friction Stir Welding and Processing: Science and Engineering*, 1st ed. 2014. Cham: Springer International Publishing : Imprint: Springer, 2014. doi: 10.1007/978-3-319-07043-8.

[3] R. S. Mishra and Z. Y. Ma, "Friction stir welding and processing," *Mater. Sci. Eng. R Rep.*, vol. 50, no. 1–2, pp. 1–78, Aug. 2005, doi: 10.1016/j.mser.2005.07.001.

[4] G. Peng *et al.*, "Nanoindentation Hardness Distribution and Strain Field and Fracture Evolution in Dissimilar Friction Stir-Welded AA 6061-AA 5A06 Aluminum Alloy Joints," *Adv. Mater. Sci. Eng.*, vol. 2018, pp. 1–11, Oct. 2018, doi: 10.1155/2018/4873571.

[5] A. J. Schwartz, M. Kumar, B. L. Adams, and D. P. Field, Eds., *Electron Backscatter Diffraction in Materials Science*. Boston, MA: Springer US, 2009. doi: 10.1007/978-0-387-88136-2.

[6] O. M. Barabash, R. I. Barabash, G. E. Ice, Z. Feng, and D. Gandy, "X-ray microdiffraction and EBSD study of FSP induced structural/phase transitions in a Ni-based superalloy," *Mater. Sci. Eng. A*, vol. 524, no. 1–2, pp. 10–19, Oct. 2009, doi: 10.1016/j.msea.2009.03.086.

[7] J.-H. Cho and P. R. Dawson, "Investigation on Texture Evolution during Friction Stir Welding of Stainless Steels," *Metall. Mater. Trans. A*, vol. 37A, pp. 1147–1164, 2006.

[8] M. Hajizadeh, S. Emami, and T. Saeid, "Influence of welding speed on microstructure formation in friction-stir-welded 304 austenitic stainless steels," *Int. J. Miner. Metall. Mater.*, vol. 27, no. 11, pp. 1517–1524, Nov. 2020, doi: 10.1007/s12613-020-2001-8.

[9] G. P. Dinda and A. Ramakrishnan, "Friction stir welding of high-strength steel," *Int. J. Adv. Manuf. Technol.*, vol. 103, no. 9–12, pp. 4763–4769, Aug. 2019, doi: 10.1007/s00170-019-04003-7.

[10] R. Ramesh, I. Dinaharan, R. Kumar, and E. T. Akinlabi, "Microstructure and Mechanical Characterization of Friction-Stir-Welded 316L Austenitic Stainless Steels," *J. Mater. Eng. Perform.*, vol. 28, no. 1, pp. 498–511, Jan. 2019, doi: 10.1007/s11665-018-3802-z.

[11] S. Shashi Kumar, N. Murugan, and K. K. Ramachandran, "Effect of friction stir welding on mechanical and microstructural properties of AISI 316L stainless steel butt joints," *Weld. World*, vol. 63, no. 1, pp. 137–150, Jan. 2019, doi: 10.1007/s40194-018-0621-7.

[12] M. Haghshenas, M. A. Gharghouri, V. Bhakhri, R. J. Klassen, and A. P. Gerlich, "Assessing residual stresses in friction stir welding: neutron diffraction and nanoindentation methods," *Int. J. Adv. Manuf. Technol.*, vol. 93, no. 9–12, pp. 3733–3747, Dec. 2017, doi: 10.1007/s00170-017-0759-2.

[13] C. A. Charitidis and D. A. Dragatogiannis, "Finite element analysis, stress-strain distribution and size effects rise during nanoindentation of welded aluminum alloy," *Int. J. Struct. Integr.*, vol. 4, no. 1, pp. 78–90, Mar. 2013, doi: 10.1108/17579861311303645.

[14] W. C. Oliver and G. M. Pharr, "An improved technique for determining hardness and elastic modulus using load and displacement sensing indentation experiments," *J. Mater. Res.*, vol. 7, no. 06, pp. 1564–1583, 1992.

[15] J. J. Vlassak and W. D. Nix, "Measuring the elastic properties of anisotropic materials by means of indentation experiments," *J. Mech. Phys. Solids*, vol. 42, no. 8, pp. 1223–1245, Aug. 1994, doi: 10.1016/0022-5096(94)90033-7.

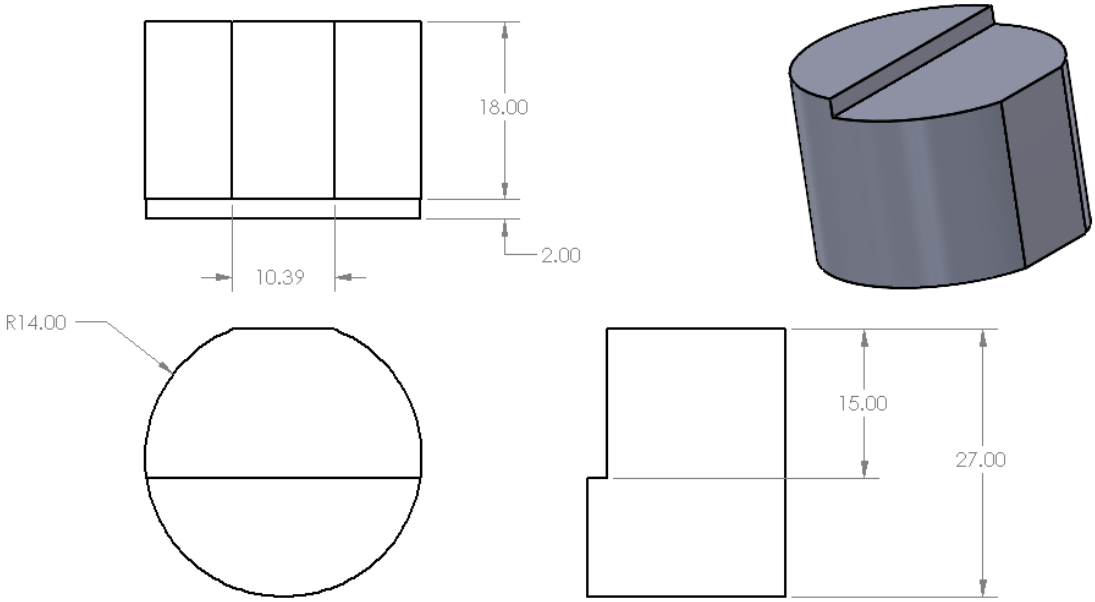
[16] T. H. Courtney, *Mechanical behavior of materials*. Long Grove, IL: Waveland Press, 2005.

[17] W. D. Callister, Jr. and D. G. Rethwisch, "Chapter 6: Mechanical Properties of Metals," in *Material Science and Engineering: An Introduction*, 9th ed., Wiley, 2014, pp. 191–197.

[18] J. J. Roa, G. Fargas, A. Mateo, and E. Jiménez-Piqué, “Dependence of nanoindentation hardness with crystallographic orientation of austenite grains in metastable stainless steels,” *Mater. Sci. Eng. A*, vol. 645, pp. 188–195, Oct. 2015, doi: 10.1016/j.msea.2015.07.096.
[19] C. Tromas, “Hardness and elastic modulus gradients in plasma-nitrided 316L polycrystalline stainless steel investigated by nanoindentation tomography,” *Acta Mater.*, p. 9, 2012.
[20] K. S. Mao *et al.*, “Grain orientation dependence of nanoindentation and deformation-induced martensitic phase transformation in neutron irradiated AISI 304L stainless steel,” *Materialia*, vol. 5, p. 100208, Mar. 2019, doi: 10.1016/j.mtla.2019.100208.
[21] W. M. Thomas, E. D. Nicholas, J. C. Needham, M. G. Murch, P. Temple-Smith, and C. J. Dawes, “Friction Welding,” 5,460,317, Oct. 24, 1995
[22] W. M. Thomas, K. I. Johnson, and C. S. Wiesner, “Friction Stir Welding – Recent Developments in Tool and Process Technologies,” *Adv. Eng. Mater.*, vol. 5, no. 7, pp. 485–490, Jul. 2003, doi: 10.1002/adem.200300355.
[23] A. Polar, F. Rumiche, M. Pareek, and J. E. Indacochea, “Friction stir welding of copper: metallurgical characterization and corrosion resistance.,” in *Trends in Welding Research: Proceedings of the 7th anual International Conference*, 2005, pp. 431–436.
[24] Z. W. Chen and S. Cui, “On the forming mechanism of banded structures in aluminium alloy friction stir welds,” *Scr. Mater.*, vol. 58, no. 5, pp. 417–420, Mar. 2008, doi: 10.1016/j.scriptamat.2007.10.026.
[25] S. Xu, “A study of texture patterns in friction stir welds,” *Acta Mater.*, vol. 56, no. 6, pp. 1326–1341, Apr. 2008, doi: 10.1016/j.actamat.2007.11.016.
[26] T. Le Jolu *et al.*, “Microstructural Characterization of Internal Welding Defects and Their Effect on the Tensile Behavior of FSW Joints of AA2198 Al-Cu-Li Alloy,” *Metall. Mater. Trans. A*, vol. 45, no. 12, pp. 5531–5544, Nov. 2014, doi: 10.1007/s11661-014-2537-1.
[27] Y. S. Sato, H. Takauchi, S. H. C. Park, and H. Kokawa, “Characteristics of the kissing-bond in friction stir welded Al alloy 1050,” *Mater. Sci. Eng. A*, vol. 405, no. 1–2, pp. 333–338, Sep. 2005, doi: 10.1016/j.msea.2005.06.008.
[28] H.-B. Chen, K. Yan, T. Lin, S.-B. Chen, C.-Y. Jiang, and Y. Zhao, “The investigation of typical welding defects for 5456 aluminum alloy friction stir welds,” *Mater. Sci. Eng. A*, vol. 433, no. 1–2, pp. 64–69, Oct. 2006, doi: 10.1016/j.msea.2006.06.056.
[29] J. J. Jeon, S. Mironov, Y. S. Sato, H. Kokawa, S. H. C. Park, and S. Hirano, “Grain Structure Development During Friction Stir Welding of Single-Crystal Austenitic Stainless Steel,” *Metall. Mater. Trans. A*, vol. 44, no. 7, pp. 3157–3166, Jul. 2013, doi: 10.1007/s11661-013-1692-0.
[30] A. C. Fischer-Cripps, *Nanoindentation*. New York, NY: Springer New York, 2011. doi: 10.1007/978-1-4419-9872-9.
[31] M. Shell De Guzman, G. Neubauer, P. Flinn, and W. D. Nix, “The Role of Indentation Depth on the Measured Hardness of Materials,” *MRS Proc.*, vol. 308, p. 613, 1993, doi: 10.1557/PROC-308-613.
[32] W. D. Nix and H. Gao, “Indentation size effect in crystalline materials: A law for strain gradient plasticity,” *J. Mech. Phys. Solids*, vol. 46, no. 3, pp. 411–425, 1998.
[33] G. Farges and D. Degout, “Interpretation of the indentation size effect in vickers microhardness measurements-absolute hardness of materials,” *Thin Solid Films*, vol. 181, no. 1, pp. 365–374, 1989.
[34] Q. Ma and D. R. Clarke, “Size dependent hardness of silver single crystals,” *J. Mater. Res.*, vol. 10, no. 4, pp. 853–863, Apr. 1995, doi: 10.1557/JMR.1995.0853.
[35] M. R. Maughan, A. A. Leonard, D. D. Stauffer, and D. F. Bahr, “The effects of intrinsic properties and defect structures on the indentation size effect in metals,” *Philos. Mag.*, vol. 97, no. 22, pp. 1902–1920, Aug. 2017, doi: 10.1080/14786435.2017.1322725.

[36] D. J. Morris and Bahr, David F., “Nanoindentation: Localized Probes of Mechanical Behavior of Materials,” in *Springer Handbook of Experimental Solid Mechanics*, Springer, 2008, pp. 389–408. Accessed: Apr. 15, 2014. [Online]. Available: http://link.springer.com/10.1007/978-0-387-30877-7_16
[37] T. Chen, L. Tan, Z. Lu, and H. Xu, “The effect of grain orientation on nanoindentation behavior of model austenitic alloy Fe-20Cr-25Ni,” *Acta Mater.*, vol. 138, pp. 83–91, Oct. 2017, doi: 10.1016/j.actamat.2017.07.028.
[38] M. Lutz, *Learning Python*, Fifth edition. Beijing: O’Reilly, 2013.
[39] “os — Miscellaneous operating system interfaces — Python 3.9.7 documentation.” https://docs.python.org/3/library/os.html (accessed Sep. 28, 2021).
[40] W. McKinney, “pandas: powerful Python data analysis toolkit,” p. 3603.
[41] “What is NumPy? — NumPy v1.21 Manual.” https://numpy.org/doc/stable/user/whatisnumpy.html (accessed Sep. 28, 2021).
[42] “openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files — openpyxl 3.0.9 documentation.” https://openpyxl.readthedocs.io/en/stable/ (accessed Sep. 28, 2021).
[43] “Introduction — PyQt v5.15 Reference Guide.” https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html (accessed Sep. 28, 2021).
[44] “Overview — Matplotlib 3.4.3 documentation.” https://matplotlib.org/stable/contents.html (accessed Sep. 28, 2021).
[45] “Scientific computing tools for Python — SciPy.org.” https://www.scipy.org/about.html (accessed Sep. 30, 2021).
[46] M. Diem, *Modern Vibrational Spectroscopy and Micro-Spectroscopy: Theory, Instrumentation and Biomedical Applications*. Chicester, UNITED KINGDOM: John Wiley & Sons, Incorporated, 2015. Accessed: Oct. 07, 2021. [Online]. Available: http://ebookcentral.proquest.com/lib/uidaho/detail.action?docID=4038970
[47] A. L. Biro, B. F. Chenelle, and D. A. Lados, “Processing, Microstructure, and Residual Stress Effects on Strength and Fatigue Crack Growth Properties in Friction Stir Welding: A Review,” *Metall. Mater. Trans. B*, vol. 43, no. 6, pp. 1622–1637, Dec. 2012, doi: 10.1007/s11663-012-9716-5.
[48] N. Dialami, M. Chiumenti, M. Cervera, C. Agelet de Saracibar, and J. P. Ponthot, “Material flow visualization in Friction Stir Welding via particle tracing,” *Int. J. Mater. Form.*, vol. 8, no. 2, pp. 167–181, Apr. 2015, doi: 10.1007/s12289-013-1157-4.
[49] K. Kumar and S. V. Kailas, “The role of friction stir welding tool on material flow and weld formation,” *Mater. Sci. Eng. A*, vol. 485, no. 1–2, pp. 367–374, Jun. 2008, doi: 10.1016/j.msea.2007.08.013.
[50] M. Matsushita, Y. Kitani, R. Ikeda, M. Ono, H. Fujii, and Y. -D. Chung, “Development of friction stir welding of high strength steel sheet,” *Sci. Technol. Weld. Join.*, vol. 16, no. 2, pp. 181–187, Feb. 2011, doi: 10.1179/1362171810Y.0000000026.
[51] Thomas H. Courtney, *Mechanical Behaviour of Materials*, 2nd ed. Waveland Press, Inc.
[52] P. Haušild, A. Materna, and J. Nohava, “Characterization of Anisotropy in Hardness and Indentation Modulus by Nanoindentation,” *Metallogr. Microstruct. Anal.*, vol. 3, no. 1, pp. 5–10, Feb. 2014, doi: 10.1007/s13632-013-0110-8.
[53] A. Teklu, H. Ledbetter, S. Kim, L. A. Boatner, M. McGuire, and V. Keppens, “Single-crystal elastic constants of Fe-15Ni-15Cr alloy,” *Metall. Mater. Trans. A*, vol. 35, no. 10, pp. 3149–3154, Oct. 2004, doi: 10.1007/s11661-004-0059-y.
[54] H. M. Ledbetter, “Predicted monocrystal elastic constants of 304-type stainless steel,” *Phys. BC*, vol. 128, no. 1, pp. 1–4, Jan. 1985, doi: 10.1016/0378-4363(85)90076-2.

Appendix A - Dimensioned Drawings of Custom Aluminum Sample Mount



* All dimensions are in mm.

Appendix B - Microhardness Data Processing GUI Python Code

```
import os
import pandas as pd
import numpy as np
from openpyxl import load_workbook
from PyQt5.QtWidgets import QSlider, QComboBox, QHBoxLayout, QApplication,
QWidget, QPushButton, QVBoxLayout, QLabel, QGroupBox, QLineEdit,
QTabWidget
from PyQt5.QtCore import Qt

from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as
NavigationToolbar
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
from matplotlib.colors import BoundaryNorm
from matplotlib.ticker import MaxNLocator
import matplotlib
matplotlib.use('QT5Agg')


class MicrohardnessTab(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.setWindowTitle("Microhardness")
        self.force = float
        self.file_name = str
        self.x_grid = int
        self.y_grid = int
        self.spacing = float
        self.overall = None

        # microhardness raw data processing.
        file_select = QGroupBox("Microhardness Data Processing")
        file_select_layout = QVBoxLayout()
        self.file = RawData()
        self.file.button.clicked.connect(self.select_file)
        file_select_layout.addWidget(self.file)
        self.file_mgs1 = QLabel()
        file_select_layout.addWidget(self.file_mgs1)
        self.file_mgs2 = QLabel()
        file_select_layout.addWidget(self.file_mgs2)
        self.file_mgs3 = QLabel()
        file_select_layout.addWidget(self.file_mgs3)
        file_select.setLayout(file_select_layout)

        # microhardness grid
        grid_info = QGroupBox("Microhardness Grid Information")
        grid_info_layout = QVBoxLayout()
        self.grid_spacing = TextEntryButton(name="Indent Spacing [mm]")
        grid_info_layout.addWidget(self.grid_spacing)
        self.grid_spacing.button.clicked.connect(self.select_spacing)
        self.grid_mgs1 = QLabel()
```

```
        grid_info_layout.addWidget(self.grid_mgs1)
        self.grid_mgs2 = QLabel()
        grid_info_layout.addWidget(self.grid_mgs2)
        self.grid_mgs3 = QLabel()
        grid_info_layout.addWidget(self.grid_mgs3)
        grid_info.setLayout(grid_info_layout)

        # plotting tab
        self.tab_widget = PlotTabs()
        self.profile_figure = Figure(figsize=(1000, 1000), dpi=100)
        self.profile_canvas = FigureCanvas(self.profile_figure)
        self.profile_ax = self.profile_figure.add_subplot(111)
        toolbar = NavigationToolbar(self.profile_canvas, self)
        self.tab_widget.profile_layout.addWidget(toolbar)
        self.contour_figure = Figure(figsize=(1000, 1000), dpi=100)
        self.contour_canvas = FigureCanvas(self.contour_figure)
        self.contour_ax = self.contour_figure.add_subplot(111)
        toolbar = NavigationToolbar(self.contour_canvas, self)
        self.tab_widget.contour_layout.addWidget(toolbar)

        # Add sub_layouts to main layout
        main_layout = QHBoxLayout()
        self.setLayout(main_layout)
        sub_layout = QVBoxLayout()
        sub_layout.addWidget(file_select)
        sub_layout.addWidget(grid_info)
        main_layout.addLayout(sub_layout)
        main_layout.addWidget(self.tab_widget)

    def profile_plot(self):
        self.profile_figure.clear()
        self.profile_ax = self.profile_figure.add_subplot(111)
        columns = self.overall.columns
        for i in columns[:-2]:
            self.profile_ax.plot(self.overall["x"], self.overall[i], "-o",
label=i)
        self.profile_ax.set_ylabel("HV")
        self.profile_ax.set_xlabel("X-Position [mm]")
        self.profile_ax.set_title("File: {0}".format(self.file_name))
        self.profile_ax.legend()
        self.tab_widget.profile_layout.addWidget(self.profile_canvas)

    def contour_plot(self):
        self.contour_figure.clear()
        self.contour_ax = self.contour_figure.add_subplot(111)
        #self.contour_ax.cla()
        columns = self.overall.columns
        y, x = np.mgrid[slice(0, self.spacing*self.y_grid, self.spacing),
slice(0, self.spacing*self.x_grid, self.spacing)]

        z = []
        for i in columns[:-2]:
            z.append(list(self.overall[i]))
        z = np.flipud(z) # this line is specifically for FSW remove for
other samples
```

```
        levels = MaxNLocator(nbins=200).tick_values(150, 330)
        cmap = plt.get_cmap('Greys')
        norm = BoundaryNorm(levels, ncolors=cmap.N, clip=True)
        im = self.contour_ax.contourf(x[:, :] + self.spacing/2., y[:, :] +
self.spacing/2., z, cmap=cmap, norm=norm)
        self.contour_figure.colorbar(im, ax=self.contour_ax)
        self.contour_ax.set_aspect("equal")
        self.contour_ax.set_ylabel("Y-Position [mm]")
        self.contour_ax.set_xlabel("X-Position [mm]")
        self.contour_ax.set_title("File: {0}".format(self.file_name))
        self.tab_widget.contour_layout.addWidget(self.contour_canvas)

    def plot(self):
        if self.file_name == self.file.current_file():
            try:
                self.profile_plot()
                self.contour_plot()
                self.grid_mgs3.setText("Data Saved")
            except:
                self.grid_mgs3.setText("Data Not Saved")

    def select_file(self):
        self.file_name = self.file.current_file()
        self.force = self.file.current_force()
        self.file_mgs1.setText("{0} gf & {1} Selected".format(self.force,
self.file_name))
        file = MicrohardnessProcessing(file_name=self.file_name,
force=self.force)
        msg = file.load_file()
        self.file_mgs2.setText(msg)
        msg = file.process_data()
        self.file_mgs3.setText(msg)

    def calc_grid(self):
        if self.file_name == self.file.current_file():
            data = pd.ExcelFile(self.file_name)
            self.overall = data.parse("Overall")
            self.overall.set_index("i", inplace=True)
            for i in range(1, len(data.sheet_names)):
                if np.mod(i, 2) == 0:
                    self.overall.loc[:, data.sheet_names[i]] =
list(reversed(list(self.overall[data.sheet_names[i]])))
                else:
                    print("")
            self.x_grid = len(self.overall.index)
            self.y_grid = len(self.overall.columns[:-2])
            self.overall.loc[:, "x"] = np.arange(0,
float(self.x_grid)*self.spacing, self.spacing)
            self.overall.loc[1:self.y_grid, "y"] = np.arange(0,
float(self.y_grid)*self.spacing, self.spacing)
            self.grid_mgs2.setText("{0} x {1} Grid". format(self.x_grid,
self.y_grid))

            with pd.ExcelWriter(self.file_name, engine='openpyxl',
mode='a') as writer:
```

```
                writer.book = load_workbook(self.file_name)
                writer.sheets = dict((ws.title, ws) for ws in
writer.book.worksheets)
                self.overall.to_excel(writer, sheet_name="Overall")
        else:
            self.grid_mgs2.setText("Data Not Available. \nGrid Not
Calculated. ")

    def select_spacing(self):
        try:
            self.spacing = float(self.grid_spacing.value())
            self.grid_mgs1.setText("Grid Spacing [mm]:  {0}".
format(self.spacing))
            self.calc_grid()
            self.plot()
        except ValueError:
            self.grid_mgs1.setText("Please Enter Number")


class PlotTabs(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        # setup tab widget
        self.layout = QHBoxLayout()
        self.setLayout(self.layout)
        self.tab = QTabWidget()
        # add profile plot tab to widget
        self.profile_tab = QWidget()
        self.profile_layout = QVBoxLayout()
        self.profile_tab.setLayout(self.profile_layout)
        self.tab.addTab(self.profile_tab, "Profile Plot")
        # add contour plot tab to widget
        self.contour_tab = QWidget()
        self.contour_layout = QVBoxLayout()
        self.contour_tab.setLayout(self.contour_layout)
        self.tab.addTab(self.contour_tab, "Contour Plot")
        self.layout.addWidget(self.tab)


class TextEntryButton(QWidget):
    def __init__(self, name):
        QWidget.__init__(self)

        layout = QHBoxLayout()
        self.setLayout(layout)
        self.text = QLineEdit()
        self.button = QPushButton("Select")
        self.display = QLabel()
        self.display.setText("{0}: ".format(name))
        layout.addWidget(self.display)
        layout.addWidget(self.text)
        layout.addWidget(self.button)

    def value(self):
        #print(self.text.text())
```

```
        return self.text.text()


class RawData(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        cur_dir = os.curdir     # retrieves current directory
        file_list = []
        for file in os.listdir(cur_dir):
            if file.endswith("VH.xlsx"):
                file_list.append(file)

        # create layout
        force_layout = QHBoxLayout()
        # create dropdown
        number_list = np.linspace(0, 1000, 11)
        number_list = number_list.astype("str")
        self.cb1 = QComboBox()
        self.cb1.addItems(number_list)                          # add files
extracted to the dropdown
        self.cb1.activated.connect(self.current_force)              # get
current file name
        # add description label
        self.display1 = QLabel()
        self.display1.setText("Force [gf]: ")       # add label describing
dropdown
        # add widgets to layout
        force_layout.addWidget(self.display1)
        force_layout.addWidget(self.cb1)

        file_layout = QHBoxLayout()
        # create dropdown
        self.cb = QComboBox()
        self.cb.addItems(file_list)                             # add files
extracted to the dropdown
        self.cb.activated.connect(self.current_file)              # get
current file name
        # add description label
        self.display = QLabel()
        self.display.setText("File: ")       # add label describing
dropdown
        self.button = QPushButton("Select File")              # add file
select button
        # add widgets to layout
        file_layout.addWidget(self.display)
        file_layout.addWidget(self.cb)
        file_layout.addWidget(self.button)

        layout = QVBoxLayout()
        self.setLayout(layout)
        layout.addLayout(force_layout)
        layout.addLayout(file_layout)

    # return current file name displayed/selected
    def current_file(self):
```

```
        return self.cb.currentText()

    def current_force(self):
        return float(self.cb1.currentText())


class MicrohardnessProcessing:
    def __init__(self, file_name=None, force=200.00, data=None,
sheet_names=None, return_msg=None):
        self.file_name = file_name
        self.force = force
        self.data = data
        self.sheet_names = sheet_names
        self.return_msg = return_msg

    def load_file(self):
        try:
            self.data = pd.ExcelFile(self.file_name)
            self.sheet_names = list(self.data.sheet_names)
            self.return_msg = "{0} Loaded.".format(self.file_name)
        except:
            self.return_msg = "{0} NOT Loaded.".format(self.file_name)
        return self.return_msg

    def process_data(self):
        c = 1.854*10**3
        try:
            overall = self.data.parse("Overall")
            overall.set_index("i", inplace=True)
            #print(overall)
            for j in range(1, len(self.sheet_names)):
                vh = []
                d = []
                line = self.data.parse(sheet_name=self.sheet_names[j])
                line.set_index("i", inplace=True)
                line = line.fillna(0)
                line["d"] = round(line["d"], 1)
                for i in line.index:
                    d.append(round(np.average(np.array([line.loc[i, "d1"],
line.loc[i, "d2"]])), 1))

vh.append(round(c*self.force/(np.average(np.array([line.loc[i, "d1"],
line.loc[i, "d2"]]))**2), 2))
                overall.loc[:, self.sheet_names[j]] = vh
                line.loc[:, "VH"] = vh
                line.loc[:, "d"] = d

                #for i in line.index[2:-3]:
                #    average = np.average(line["VH"][i-2:i+2])
                #    if (line["VH"][i]-average)/average >= 0.08:
                #        line["VH"][i] = average
                #        print(average)
                #    elif (line["VH"][i]-average)/average <= -0.08:
                #        line["VH"][i] = average
                #        print(average)
```

```
                with pd.ExcelWriter(self.file_name, engine='openpyxl',
mode='a') as writer:
                    writer.book = load_workbook(self.file_name)
                    writer.sheets = dict((ws.title, ws) for ws in
writer.book.worksheets)
                    line.to_excel(writer, sheet_name=self.sheet_names[j])
                    overall.to_excel(writer, sheet_name="Overall")
                #print(overall)
            self.return_msg = "Data Processed.".format(self.file_name)
        except:
            self.return_msg = "Data NOT Processed. \nResubmit File for
Processing.".format(self.file_name)
        return self.return_msg


if __name__ == '__main__':
    app = QApplication([])
    interface = MicrohardnessTab()
    interface.show()
    app.exec_()
```

Appendix C - Single Loading Profile GUI Python Code

```
import os
import pandas as pd
import numpy as np

from openpyxl import load_workbook
from PyQt5.QtWidgets import QComboBox, QHBoxLayout, QApplication, QWidget,
QPushButton, QVBoxLayout, QLabel, QGroupBox, QTabWidget

from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as
NavigationToolbar
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure
import matplotlib
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter)
matplotlib.use('QT5Agg')


class SingleLoadingTab(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.setWindowTitle("Nanoindentation: Single Loading")
        # declare global variables
        self.file_name = str
        self.data = None
        self.processed_data = None

        # Single Loading data processing.
        file_select = QGroupBox("Single Loading Data Processing")
        file_select_layout = QVBoxLayout()
        self.file = RawData()
        self.file.button.clicked.connect(self.select_file)
        file_select_layout.addWidget(self.file)
        self.file_mgs1 = QLabel()
        file_select_layout.addWidget(self.file_mgs1)
        self.file_mgs2 = QLabel()
        file_select_layout.addWidget(self.file_mgs2)
        #self.plot_button = QPushButton("Plot Data")
        #file_select_layout.addWidget(self.plot_button)
        #self.plot_button.clicked.connect(self.plot)
        #self.clear_button = QPushButton("Clear Plot Data")
        #file_select_layout.addWidget(self.clear_button)
        #self.clear_button.clicked.connect(self.clear_plots)
        self.file_mgs3 = QLabel()
        file_select_layout.addWidget(self.file_mgs3)
        file_select.setLayout(file_select_layout)

        # plotting tab
        self.tab_widget = PlotTabs()
        self.H_figure = Figure(figsize=(1000, 1000), dpi=100)
        self.H_canvas = FigureCanvas(self.H_figure)
        self.H_ax = self.H_figure.add_subplot(111)
        toolbar = NavigationToolbar(self.H_canvas, self)
```

```
        self.tab_widget.H_layout.addWidget(toolbar)
        self.E_figure = Figure(figsize=(1000, 1000), dpi=100)
        self.E_canvas = FigureCanvas(self.E_figure)
        self.E_ax = self.E_figure.add_subplot(111)
        toolbar = NavigationToolbar(self.E_canvas, self)
        self.tab_widget.E_layout.addWidget(toolbar)

        # Add sub_layouts to main layout
        main_layout = QHBoxLayout()
        self.setLayout(main_layout)
        sub_layout = QVBoxLayout()
        sub_layout.addWidget(file_select)
        main_layout.addLayout(sub_layout)
        main_layout.addWidget(self.tab_widget)

    def hardness_plot(self):
        self.H_figure.clear(True)
        self.H_ax = self.H_figure.add_subplot(111)
        self.H_ax.plot(self.processed_data.index,
self.processed_data["H"])#, yerr=self.processed_data["Hcov"][4:-5])
        self.H_ax.scatter(self.processed_data.index, self.data["Hardness
At Max Load"])
        self.H_ax.set_title("{0}".format(self.file_name))
        self.H_ax.set_ylabel("H [GPa]")
        self.H_ax.set_xlabel("Indent Number")
        self.H_ax.xaxis.set_major_locator(MultipleLocator(10))
        self.H_ax.xaxis.set_major_formatter(FormatStrFormatter('%d'))
        self.H_ax.xaxis.set_minor_locator(MultipleLocator(5))
        self.H_ax.legend(["Smoothed Data", "Raw Data"])
        self.tab_widget.H_layout.addWidget(self.H_canvas)

    def elastic_plot(self):
        self.E_figure.clear(True)
        self.E_ax = self.E_figure.add_subplot(111)
        self.E_ax.errorbar(self.processed_data.index[4:-5],
self.processed_data["E"][4:-5], yerr=self.processed_data["Ecov"][4:-
5]*100)
        self.E_ax.scatter(self.processed_data.index, self.data["Modulus At
Max Load"])
        self.E_ax.set_title("{0}".format(self.file_name))
        self.E_ax.set_ylabel("E [GPa]")
        self.E_ax.set_xlabel("Indent Number")
        self.E_ax.annotate("100 x Coefficient of Variance displayed",
xy=(0, max(self.data["Modulus At Max Load"])))
        self.E_ax.xaxis.set_major_locator(MultipleLocator(10))
        self.E_ax.xaxis.set_major_formatter(FormatStrFormatter('%d'))
        self.E_ax.xaxis.set_minor_locator(MultipleLocator(5))
        self.E_ax.legend(["Raw Data", "Smoothed Data"])
        self.tab_widget.E_layout.addWidget(self.E_canvas)

    def clear_plots(self):
        self.H_figure.clear(True)
        self.H_ax = self.H_figure.add_subplot(111)
        self.tab_widget.H_layout.addWidget(self.H_canvas)
        self.E_figure.clear(True)
```

```
        self.E_ax = self.E_figure.add_subplot(111)
        self.tab_widget.E_layout.addWidget(self.E_canvas)
        print("cleared")

    def plot(self):
        if self.file_name == self.file.current_file():
            if self.processed_data is not None:
                try:
                    self.hardness_plot()
                    self.elastic_plot()
                    self.file_mgs3.setText("Data Displayed")
                except:
                    self.file_mgs3.setText("Data Not Displayed")
            else:
                self.file_mgs3.setText("Data Not Displayed")
        else:
            self.file_mgs3.setText("Data Not Displayed")

    def select_file(self):
        self.file_name = self.file.current_file()
        file = SingleLoadingProcessing(file_name=self.file_name)
        self.file_mgs1.setText(file.load_file())
        msg, self.processed_data, self.data = file.process_data()
        self.file_mgs2.setText(msg)
        if msg == "Data Processed & Saved.":
            self.hardness_plot()
            self.elastic_plot()
            self.file_mgs3.setText("Data Displayed")
        else:
            self.file_mgs3.setText("Data Not Displayed")
        #print(self.processed_data)
        #self.plot()


class SingleLoadingProcessing:
    def __init__(self, file_name=None, data=None, processed_data=None,
sheet_names=None, return_msg=None):
        self.file_name = file_name
        self.data = data
        self.sheet_names = sheet_names
        self.return_msg = return_msg
        self.processed_data = processed_data

    def load_file(self):
        try:
            self.data = pd.ExcelFile(self.file_name)
            self.return_msg = "{0} Loaded.".format(self.file_name)
        except:
            self.return_msg = "{0} NOT Loaded.".format(self.file_name)
        return self.return_msg

    def smooth(self, y, n):
        rft1 = np.fft.rfft(y)
        abs_rft1 = abs(rft1)
        x1 = np.linspace(0, len(abs_rft1), len(abs_rft1))
```

```
        rft1[n:] = 0  # Note, rft.shape = 21
        n1_smooth = np.fft.irfft(rft1)
        return x1, abs_rft1, n1_smooth

    def process_data(self):
        try:
            self.data = self.data.parse("Results")
            self.data.set_index("Test", inplace=True)
            self.data = self.data[self.data.columns[0:2]][1:-3]
            index = self.data.index.astype(int)

            self.processed_data = pd.DataFrame(data=None, columns=["E",
"Ecov", "H", "Hcov", "VH", "x", "y"], index=self.data.index[:])
            # process raw data to remove outliers
            for i in index[4:-5]:
                average = np.average(self.data["Hardness At Max Load"][i-
5:i+5])
                if (self.data["Hardness At Max Load"][i]-average)/average
>= 0.08:
                    self.data["Hardness At Max Load"][i] = average
                    self.data["Modulus At Max Load"][i] =
np.average(self.data["Modulus At Max Load"][i-5:i+5])
            # Smooth hardness data and convert to VH
            x1, abs_rft1, cof1_smooth = self.smooth(self.data["Hardness At
Max Load"][:], 8)
            self.processed_data.loc[:, "H"] = cof1_smooth
            self.processed_data.loc[:, "VH"] =
self.processed_data["H"][:]*1000*0.094495
            # smooth elastic modulus data
            x2, abs_rft2, cof2_smooth = self.smooth(self.data["Modulus At
Max Load"][:], 5)
            self.processed_data.loc[:, "E"] = cof2_smooth
            for i in index[4:-5]:
                self.processed_data.loc[str(i), "Ecov"] =
np.std(self.data["Modulus At Max Load"][i-
5:i+5])/np.average(self.data["Modulus At Max Load"][i-5:i+5])
                self.processed_data.loc[str(i), "Hcov"] =
np.std(self.data["Hardness At Max Load"][i-
5:i+5])/np.average(self.data["Hardness At Max Load"][i-5:i+5])
            #print(self.processed_data)
            with pd.ExcelWriter(self.file_name, engine='openpyxl',
mode='a') as writer:
                writer.book = load_workbook(self.file_name)
                writer.sheets = dict((ws.title, ws) for ws in
writer.book.worksheets)
                self.processed_data.to_excel(writer, sheet_name="Sheet1")
            self.return_msg = "Data Processed &
Saved.".format(self.file_name)
        except:
            self.return_msg = "Data NOT Processed. \nResubmit File for
Processing.".format(self.file_name)
            self.processed_data = None
            self.data =None
        return self.return_msg, self.processed_data, self.data
```

```
class PlotTabs(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        # setup tab widget
        self.layout = QHBoxLayout()
        self.setLayout(self.layout)
        self.tab = QTabWidget()
        # add hardness plot tab to widget
        self.H_tab = QWidget()
        self.H_layout = QVBoxLayout()
        self.H_tab.setLayout(self.H_layout)
        self.tab.addTab(self.H_tab, "Hardness Plot")
        # add elastic modulus tab to widget
        self.E_tab = QWidget()
        self.E_layout = QVBoxLayout()
        self.E_tab.setLayout(self.E_layout)
        self.tab.addTab(self.E_tab, "Elastic Modulus Plot")
        self.layout.addWidget(self.tab)


class RawData(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        cur_dir = os.curdir     # retrieves current directory
        # create list of Excell files with appropriate extension
        file_list = []
        for file in os.listdir(cur_dir):
            if file.endswith("N.xlsx"):
                file_list.append(file)

        file_layout = QHBoxLayout()
        # create dropdown
        self.cb = QComboBox()
        self.cb.addItems(file_list)                             # add files
extracted to the dropdown
        self.cb.activated.connect(self.current_file)              # get
current file name
        # add description label
        self.display = QLabel()
        self.display.setText("File: ")       # add label describing
dropdown
        # add file select button
        self.button = QPushButton("Select File")
        # add widgets to layout
        file_layout.addWidget(self.display)
        file_layout.addWidget(self.cb)
        file_layout.addWidget(self.button)

        layout = QVBoxLayout()
        self.setLayout(layout)
        layout.addLayout(file_layout)

    # return current file name displayed/selected
    def current_file(self):
```

```
        return self.cb.currentText()


if __name__ == '__main__':
    app = QApplication([])
    interface = SingleLoadingTab()
    interface.show()
    app.exec_()
```

Appendix D - Cyclic Loading Profile GUI Python Code

```
import os
import pandas as pd
import numpy as np
from openpyxl import load_workbook
from PyQt5.QtWidgets import QSlider, QComboBox, QHBoxLayout, QApplication,
QWidget, QPushButton, QVBoxLayout, QLabel, QGroupBox, QLineEdit,
QTabWidget
from PyQt5.QtCore import Qt
from scipy.optimize import curve_fit
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as
NavigationToolbar
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
from matplotlib.colors import BoundaryNorm
from matplotlib.ticker import MaxNLocator
import matplotlib
matplotlib.use('QT5Agg')


class CyclicLoadingTab(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.setWindowTitle("Nanoindentation: Cyclic Loading")
        self.tip_name = str
        self.file_name = str
        self.data = None
        self.cycle_num = int

        # indentation size effect raw data processing.
        file_select = QGroupBox("Cyclic Loading Data Processing")
        file_select_layout = QVBoxLayout()
        # create drop down displays
        self.file = RawData()
        # assign functionality to file select push button
        self.file.file_button.clicked.connect(self.select_file)
        # assign functionality to tip geometry select push button
        self.file.tip_button.clicked.connect(self.select_tip)
        # add drop down displays to group
        file_select_layout.addWidget(self.file)
        # create status messages and add to group
        self.file_mgs1 = QLabel()
        file_select_layout.addWidget(self.file_mgs1)
        self.file_mgs2 = QLabel()
        file_select_layout.addWidget(self.file_mgs2)
        self.file_mgs3 = QLabel()
        file_select_layout.addWidget(self.file_mgs3)
        # create plot button, assign functionality and add to group
        self.plot_button = QPushButton("Plot")
        file_select_layout.addWidget(self.plot_button)
        self.plot_button.clicked.connect(self.plot)
        # create status messages and add to group
```

```
        self.file_mgs4 = QLabel()
        file_select_layout.addWidget(self.file_mgs4)
        # add all created object to the sub-layout
        file_select.setLayout(file_select_layout)

        # create plotting tabs
        self.tab_widget = PlotTabs()
        # create hardness vs depth canvas to display data
        self.hardness_figure = Figure(figsize=(1000, 1000), dpi=100)
        self.hardness_canvas = FigureCanvas(self.hardness_figure)
        self.hardness_ax = self.hardness_figure.add_subplot(111)
        toolbar = NavigationToolbar(self.hardness_canvas, self)
        self.tab_widget.hardness_layout.addWidget(toolbar)
        # create elastic modulus vs depth canvas to display data
        self.elastic_figure = Figure(figsize=(1000, 1000), dpi=100)
        self.elastic_canvas = FigureCanvas(self.elastic_figure)
        self.elastic_ax = self.elastic_figure.add_subplot(111)
        toolbar = NavigationToolbar(self.elastic_canvas, self)
        self.tab_widget.elastic_layout.addWidget(toolbar)

        # add sub-layouts to main layout
        main_layout = QHBoxLayout()
        self.setLayout(main_layout)
        sub_layout = QVBoxLayout()
        sub_layout.addWidget(file_select)
        main_layout.addLayout(sub_layout)
        main_layout.addWidget(self.tab_widget)

    def select_tip(self):
        self.tip_name = self.file.current_tip()
        self.file_mgs1.setText("Tip:    {0} ".format(self.tip_name))

    def select_file(self):
        self.file_name = self.file.current_file()
        self.file_mgs2.setText("File:   {0}".format(self.file_name))
        if self.tip_name == self.file.current_tip():
            file = ISEProcessing(file_name=self.file_name,
tip_name=self.tip_name)
            msg = file.load_file()
            self.data = file.processed_data
            print(self.data)
            self.cycle_num = file.cycle_num
            self.file_mgs3.setText(msg)
        else:
            self.file_mgs3.setText("{0} Not Loaded. \nCheck Tip
Selection.".format(self.file_name))

    def hardness_plot(self):
        #print(self.cycle_num)
        #self.hardness_figure.clear()
        #self.hardness_ax = self.hardness_figure.add_subplot(111)
        self.hardness_ax.cla()
        for i in self.data.index:
            #print(i)
            h = []
```

```
            H = []
            for j in range(1, self.cycle_num+1):
                h.append(self.data["h{0}".format(j)][i])
                H.append(self.data["H{0}".format(j)][i])

            #print(h, H)
            self.hardness_ax.plot(h, H, "-o", label=i)
        self.hardness_ax.set_title("Tip: {0}, File:
{1}".format(self.tip_name, self.file_name))
        self.hardness_ax.set_ylabel("Hardness [GPa]")
        self.hardness_ax.set_xlabel("Penetration Depth [nm]")
        self.hardness_ax.legend()
        self.tab_widget.hardness_layout.addWidget(self.hardness_canvas)

    def elastic_plot(self):
        self.elastic_ax.cla()
        #self.elastic_figure.clear()
        #self.elastic_ax = self.elastic_figure.add_subplot(111)
        for i in self.data.index:
            #print(i)
            h = []
            E = []
            for j in range(1, self.cycle_num+1):
                h.append(self.data["h{0}".format(j)][i])
                E.append(self.data["E{0}".format(j)][i])
            self.elastic_ax.plot(h, E, "-o", label=i)
        self.elastic_ax.set_title("Tip: {0}, File:
{1}".format(self.tip_name, self.file_name))
        self.elastic_ax.set_ylabel("Elastic Modulus [GPa]")
        self.elastic_ax.set_xlabel("Penetration Depth [nm]")
        self.elastic_ax.legend()
        self.tab_widget.elastic_layout.addWidget(self.elastic_canvas)

    def plot(self):
        if self.file_name == self.file.current_file():
            if self.tip_name == self.file.current_tip():
                self.hardness_plot()
                self.elastic_plot()
                self.file_mgs4.setText("Plotting")
            else:
                self.file_mgs4.setText("Not Plotting")
        else:
            self.file_mgs4.setText("Not Plotting")


class ISEProcessing:
    def __init__(self, file_name=None, tip_name=None):
        self.file_name = file_name
        self.tip_name = tip_name
        self.return_msg = str
        self.data = None
        self.tip = None
        self.tip_info = None
        self.sheet_names = None
        self.results = None
```

```
        self.test_names = None
        self.cycle_num = int
        self.processed_data = None

    def load_file(self):
        try:
            self.data = pd.ExcelFile(self.file_name)
            self.tip = pd.ExcelFile("TipGeometry.xlsx")
            self.process_data()
            self.return_msg = "{0} Loaded.".format(self.file_name)
        except:
            self.return_msg = "{0} NOT Loaded.".format(self.file_name)
        return self.return_msg

    def process_data(self):
        self.tip_info = self.tip.parse("Sheet1")
        self.tip_info.set_index("i", inplace=True)
        self.tip_info = self.tip_info[self.tip_name]

        self.sheet_names = list(self.data.sheet_names)
        self.test_names = self.sheet_names[3:-1]
        self.test_names = list(reversed(self.test_names))

        # calculate the number of cycles
        test = self.data.parse(self.test_names[0])
        self.cycle_num = 0
        for i in range(0, len(test["Segment"])):
            if test["Segment"][i] == "Unload From Peak Segment Type":
                self.cycle_num += 1

        columns = []
        for i in range(1, self.cycle_num+1):
            columns.append("h{0}".format(i))
            columns.append("H{0}".format(i))
            columns.append("VH{0}".format(i))
            columns.append("E{0}".format(i))
        # create dataframe to hold processed data
        self.processed_data = pd.DataFrame(data=None, columns=columns,
index=self.test_names)
        # retrieve total number of test indents completed and add as index
to the processed data dataframe
        self.results = self.data.parse("Results")
        self.results.loc[1:len(self.test_names), "Test"] = self.test_names
        self.results.set_index("Test", inplace=True)

        def unload_sfit(x, slope, b):
            return slope*x+b

        def area(h):
            return self.tip_info["m0"]*h**2 + self.tip_info["m1"]*h +
self.tip_info["m2"]*h**(1/2) + self.tip_info["m3"]*h**(1/4)

        def modulus(reduced_modulus):
            Ei = self.tip_info["Ei [MPa]"]*10**6
            vi = self.tip_info["vi"]
```

```
            v = 0.29 # make this editable
            a = 1/reduced_modulus - (1-vi**2)/Ei
            return (1-v**2)/a

        m = 15  # number of data points used for linear fit of unloading
curve
        for i in self.test_names:
            data = self.data.parse(i)
            loc = []
            for j in range(0, len(data["Segment"])):
                if data["Segment"][j] == "Unload From Peak Segment Type":
                    loc.append(j)
            info = []
            for j in loc:
                corrected_displacement = data["Displacement Into
Surface"][j:j+m] - data["Time On Sample"][j:j+m]*self.results["Drift
Correction"][i]
                popt, pcov = curve_fit(unload_sfit,
corrected_displacement, data["Load On Sample"][j:j+m])
                S = popt[0]
                hc = -popt[1]/S
                A = area(hc)
                Er = np.sqrt(np.pi)*S/(2 * np.sqrt(A))*10**15
                E = modulus(Er)*10**(-9)
                H = data["Load On Sample"][j]/A*10**6
                VH = H*1000*0.094495
                info.append(round(corrected_displacement[j], 2))
                info.append(round(H, 2))
                info.append(round(VH, 2))
                info.append(round(E, 2))

            self.processed_data.loc[i, :] = info
        print(self.processed_data)
        with pd.ExcelWriter(self.file_name, engine="openpyxl", mode="a")
as writer:
            writer.book = load_workbook(self.file_name)
            writer.sheets = dict((ws.title, ws) for ws in
writer.book.worksheets)
            self.processed_data.to_excel(writer, sheet_name="Sheet1")


class PlotTabs(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        # setup tab widget
        self.layout = QHBoxLayout()
        self.setLayout(self.layout)
        self.tab = QTabWidget()
        # add hardness vs depth tab to widget
        self.hardness_tab = QWidget()
        self.hardness_layout = QVBoxLayout()
        self.hardness_tab.setLayout(self.hardness_layout)
        self.tab.addTab(self.hardness_tab, "H vs h Plot")
        # add elastic modulus vs depth tab to widget
        self.elastic_tab = QWidget()
```

```
        self.elastic_layout = QVBoxLayout()
        self.elastic_tab.setLayout(self.elastic_layout)
        self.tab.addTab(self.elastic_tab, "E vs h Plot")
        self.layout.addWidget(self.tab)


class RawData(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        cur_dir = os.curdir     # retrieves current directory

        # create list of of files with extension "ISE.xlsx"
        file_list = []
        for file in os.listdir(cur_dir):
            if file.endswith("ISE.xlsx"):
                file_list.append(file)

        # import tip geometry list
        data = pd.ExcelFile("TipGeometry.xlsx")
        sheet1 = data.parse("Sheet1")
        sheet1.set_index("i", inplace=True)
        tip_list = list(sheet1.columns)

        # create tip geometry dropdown
        tip_layout = QHBoxLayout()
        self.tip_cb = QComboBox()
        self.tip_cb.addItems(tip_list)
        self.tip_cb.activated.connect(self.current_tip)
        self.tip_display = QLabel()
        self.tip_display.setText("Tip: ")
        self.tip_button = QPushButton("Select Tip")
        tip_layout.addWidget(self.tip_display)
        tip_layout.addWidget(self.tip_cb)
        tip_layout.addWidget(self.tip_button)

        # create Excell file dropdown
        file_layout = QHBoxLayout()
        self.file_cb = QComboBox()
        self.file_cb.addItems(file_list)                                # add
files extracted to the dropdown
        self.file_cb.activated.connect(self.current_file)               #
get current file name
        self.file_display = QLabel()
        self.file_display.setText("File: ")       # add label describing
dropdown
        self.file_button = QPushButton("Select File")               # add
file select button
        file_layout.addWidget(self.file_display)
        file_layout.addWidget(self.file_cb)
        file_layout.addWidget(self.file_button)

        # add tip geometry and file dropdowns to main display
        layout = QVBoxLayout()
        self.setLayout(layout)
        layout.addLayout(tip_layout)
```

```
        layout.addLayout(file_layout)

    def current_file(self):
        # return current file name displayed/selected
        return self.file_cb.currentText()

    def current_tip(self):
        # return current tip geometry displayed/selected
        return self.tip_cb.currentText()


if __name__ == '__main__':
    app = QApplication([])
    interface = CyclicLoadingTab()
    interface.show()
    app.exec_()
```

Appendix E - Indentation Size Effect GUI Python Code

```
import os
import pandas as pd
from scipy.optimize import curve_fit
import numpy as np

from openpyxl import load_workbook
from PyQt5.QtWidgets import QSlider, QComboBox, QHBoxLayout, QApplication,
QWidget, QPushButton, QVBoxLayout, QLabel, QGroupBox, QTabWidget
from PyQt5.QtCore import Qt

from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as
NavigationToolbar
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure
import matplotlib
matplotlib.use('QT5Agg')


class NixGaoTab(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.setWindowTitle("Indentation Size Effect")
        self.file_name = str
        self.num_cycles = int
        self.data = None
        # Nix Gao data processing.
        file_select = QGroupBox("ISE Data Processing")
        file_select_layout = QVBoxLayout()
        self.cycles = SliderDisplay1(name="Number of Cycles")
        self.cycles.slider_button.clicked.connect(self.select_cycles)
        file_select_layout.addWidget(self.cycles)
        self.file = RawData()
        self.file.button.clicked.connect(self.select_file)
        file_select_layout.addWidget(self.file)
        self.file_mgs1 = QLabel()
        file_select_layout.addWidget(self.file_mgs1)
        self.file_mgs2 = QLabel()
        file_select_layout.addWidget(self.file_mgs2)
        self.file_mgs3 = QLabel()
        file_select_layout.addWidget(self.file_mgs3)
        file_select.setLayout(file_select_layout)

        # plotting tab
        self.tab_widget = PlotTabs()
        self.profile_figure = Figure(figsize=(1000, 1000), dpi=100)
        self.profile_canvas = FigureCanvas(self.profile_figure)
        self.profile_ax = self.profile_figure.add_subplot(111)
        toolbar = NavigationToolbar(self.profile_canvas, self)
        self.tab_widget.profile_layout.addWidget(toolbar)

        # Add sub_layouts to main layout
        main_layout = QHBoxLayout()
```

```
        self.setLayout(main_layout)
        sub_layout = QVBoxLayout()
        sub_layout.addWidget(file_select)
        main_layout.addLayout(sub_layout)
        main_layout.addWidget(self.tab_widget)

    def select_cycles(self):
        self.num_cycles = self.cycles.value()
        #print(self.num_cycles, type(self.num_cycles))

    def select_file(self):
        self.file_name = self.file.current_file()
        self.file_mgs1.setText("{0} Selected".format(self.file_name))
        msg = self.load_file()
        if msg is True:
            if self.num_cycles == self.cycles.value():
                self.process_data()
                self.file_mgs2.setText("{0} Processed &
Saved".format(self.file_name))
            else:
                self.file_mgs2.setText("Select Number of
Cycles".format(self.file_name))
        else:
            self.file_mgs2.setText("{0} NOT
Loaded".format(self.file_name))

    def load_file(self):
        try:
            self.data = pd.ExcelFile(self.file_name)
            return True
        except:
            return False

    def process_data(self):
        def NixGao_fit(x, h0, g):
            return h0 * (1 + g / x) ** (1 / 2)
        # clear figure for plotting
        #self.profile_figure.clear()
        #self.profile_ax = self.profile_figure.add_subplot(111)
        self.profile_ax.cla()
        # process data
        self.data = self.data.parse("Sheet1")
        self.data.set_index("Unnamed: 0", inplace=True)
        H0 = []
        h_star = []
        VH0 = []
        Vh_star = []
        for i in self.data.index:
            h = []
            H = []
            VH = []
            for j in range(1, self.num_cycles+1):
                h.append(self.data["h{0}".format(j)][i])
                H.append(self.data["H{0}".format(j)][i])
                VH.append(self.data["VH{0}".format(j)][i])
```

```
            self.profile_ax.scatter(h, H, label="{0}".format(i))
            popt, pcov = curve_fit(NixGao_fit, h, H)
            h_fit = np.linspace(500, max(h)+500)
            H_fit = NixGao_fit(h_fit,*popt)
            self.profile_ax.plot(h_fit, H_fit)
            H0.append(popt[0])
            h_star.append(popt[1])
            popt, pcov = curve_fit(NixGao_fit, h, VH)
            VH0.append(popt[0])
            Vh_star.append(popt[1])
        self.data.loc[:, "H0"] = H0
        self.data.loc[:, "h*"] = h_star
        self.data.loc[:, "VH0"] = VH0
        self.data.loc[:, "Vh*"] = Vh_star
        self.profile_ax.set_title("{0}".format(self.file_name))
        self.profile_ax.set_ylabel("H [GPa]")
        self.profile_ax.set_xlabel("h [nm]")
        self.profile_ax.legend()
        self.tab_widget.profile_layout.addWidget(self.profile_canvas)
        with pd.ExcelWriter(self.file_name, engine='openpyxl', mode='a')
as writer:
            writer.book = load_workbook(self.file_name)
            writer.sheets = dict((ws.title, ws) for ws in
writer.book.worksheets)
            self.data.to_excel(writer, sheet_name="Sheet1")


class PlotTabs(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.layout = QHBoxLayout()
        self.setLayout(self.layout)
        self.tab = QTabWidget()
        self.profile_tab = QWidget()
        self.profile_layout = QVBoxLayout()
        self.profile_tab.setLayout(self.profile_layout)
        self.tab.addTab(self.profile_tab, "Nix-Gao Plot")
        self.layout.addWidget(self.tab)


class SliderDisplay1(QWidget):
    def __init__(self, name, low=2, high=10):
        QWidget.__init__(self)
        self.name = name
        layout = QHBoxLayout()
        self.setLayout(layout)
        self.slider = QSlider(Qt.Horizontal)
        self.slider.setMinimum(low)
        self.slider.setMaximum(high)
        self.slider.setTickPosition(QSlider.TicksBelow)
        self.slider.setTickInterval(5)
        self.slider.valueChanged.connect(self.number)
        self.display = QLabel()
        self.number()
        #self.slider_display = QLabel()
```

```
        #self.slider_display.setText("Number of Cycles: ")       # add
label describing dropdown
        self.slider_button = QPushButton("Select Cycles")
        layout.addWidget(self.display)
        layout.addWidget(self.slider)
        layout.addWidget(self.slider_button)

        # retrieve current slider value
    def value(self):
        #print(self.slider.value())
        return self.slider.value()

    # display current slider value on widget
    def number(self):
        self.display.setText('{0}: {1}'.format(self.name, self.value()))
        return self.name, self.value()


class RawData(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        cur_dir = os.curdir     # retrieves current directory
        file_list = []
        for file in os.listdir(cur_dir):
            if file.endswith("ISE.xlsx"):
                file_list.append(file)
            elif file.endswith("ISE.xls"):
                file_list.append(file)

        file_layout = QHBoxLayout()
        # create dropdown
        self.cb = QComboBox()
        self.cb.addItems(file_list)                             # add files
extracted to the dropdown
        self.cb.activated.connect(self.current_file)              # get
current file name
        # add description label
        self.display = QLabel()
        self.display.setText("File: ")       # add label describing
dropdown
        self.button = QPushButton("Select File")              # add file
select button
        # add widgets to layout
        file_layout.addWidget(self.display)
        file_layout.addWidget(self.cb)
        file_layout.addWidget(self.button)

        layout = QVBoxLayout()
        self.setLayout(layout)
        layout.addLayout(file_layout)

    # return current file name displayed/selected
    def current_file(self):
        return self.cb.currentText()
```

```
    def current_force(self):
        return float(self.cb1.currentText())


if __name__ == '__main__':
    app = QApplication([])
    interface = NixGaoTab()
    interface.show()
    app.exec_()
```