Hallvard Tangvik Tellefsen Relling

EBSP Indexer

Using an open-source dictionary indexing software for phase differentiation

Master's thesis in Materials Science and Engineering
Supervisor: Jarle Hjelen
June 2023

Hallvard Tangvik Tellefsen Relling

EBSP Indexer

Using an open-source dictionary indexing software for phase differentiation

Master's thesis in Materials Science and Engineering
Supervisor: Jarle Hjelen
June 2023

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Materials Science and Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This master's thesis covers the work conducted during the spring of 2023 at the Department of Materials Science and Engineering at the Norwegian University of Science and Technology. It is the continuation of the work done as part of a project thesis in the autumn of 2022, where a prototype of the software presented in this thesis was developed. The software has been developed by a team of three students, Erlend Mikkelsen Østvold, Olav Leth-Olsen and Hallvard Tangvik Tellefsen Relling, under the supervision of Prof. Jarle Hjelen, and has resulted in the release of a free software for EBSD indexing called **EBSP Indexer** now available for both Mac OS and Windows.

First of all, I would like to thank my supervisor Prof. Jarle Hjelen, for excellent guidance in the field of EBSD and for always being available for answering questions and for hours spent testing the software. Second of all I want to thank my fellow developers, Olav and Erlend, this has been a team effort and I'm grateful for all the time we've spent together developing this product.

I would also like to thank Håkon W. Ånses for the willingness to share his knowledge about both `kikuchipy` and experience in the field of EBSD and Dictionary indexing, as well as Berit V. Kramer for preparation of the AlSi sample.

Finally, a big thank you to all the wonderful people I've been so fortunate to spend these last five years with, and especially my fellow students for the countless hours spent both studying and, of course, drinking coffee. Of course, I also have to thank my family for always having my back and supporting me, I wouldn't have come this far without them.

Hallvard Tangvik Tellefsen Relling

NTNU, Trondheim 16th June 2023

Abstract

This work presents the development of a free software for analysing electron backscatter diffraction (EBSD) patterns called **EBSP Indexer**. The project is the result of a joint developing effort by a team of three students, E. Østvold [1], O. Leth-Olsen [2] and H. Relling, as part of their master theses, and this thesis covers the author's main contributions to version 0.1.0 distributed for download in May 2023.[1] The software is written in Python, using the `PySide6` library for building the user interface and the `kikuchipy` library for the EBSD analysis functionality. The project is open-source, and the complete source code and documentation have been made available on GitHub[2], with the goal of facilitating for future development of the software by interested parties.

This thesis focuses mainly on the implementation of the EBSD indexing approach Dictionary Indexing (DI), an indexing method that uses a dictionary of simulated patterns to determine the phase and orientation. Additionally, the implementation of tools for both cropping recorded datasets, as well as interactive investigation of EBSD datasets and the indexed results, is covered. As part of the development process, a comprehensive user guide has also been made available for download together with the software.

The DI implementation is tested for indexing of an Al-10wt%Si sample, demonstrating how DI can be used for phase differentiation of two similar crystal structures, something conventional Hough indexing (HI) has been found to struggle with. The same DI implementation was also used for phase differentiation in a Super duplex stainless steel (SDSS) with σ- and χ-precipitates, to demonstrate how the module scales well with several phases.

Finally, an approach for finding the pattern center (PC) from the working distance (WD) for a specific SEM/EBSD detector configuration is presented. The method relies on the observed linear relationship between the PC coordinates and the WD, a calibration line can then be calculated for all three PC coordinates (pc_x, pc_y, pc_z) by interpolation between two PCs determined at two reasonably spaced working distances. It is shown that the method provides reasonable values for the PC coordinate at intermediate WDs, and the value provided from the calibration line can in many situations be used directly for indexing. This method is implemented as a calibration tool in **EBSP Indexer**.

[1]https://sourceforge.net/projects/ebsp-indexer/
[2]https://github.com/EBSP-Indexer/EBSP-Indexer

Sammendrag

Denne oppgaven presenterer utviklingen av en gratis programvare for analyse av tilbakespredte elektron diffraksjonsmønster (EBSD) kalt **EBSP Indexer**. Prosjektet er resultatet av et felles utviklingsarbeid av et team på tre studenter, E. Østvold [1], O. Leth-Olsen [2] og H. Relling, som en del av deres masteroppgaver. Denne oppgaven dekker forfatterens hovedbidrag til versjon 0.1.0 som ble distribuert for nedlastning i mai 2023.[3] Programvaren er skrevet i Python og bruker `PySide6`-biblioteket for å bygge brukergrensesnittet og `kikuchipy`-biblioteket for EBSD-analysefunksjonaliteten. Prosjektet er åpen kildekode, og fullstendig kildekode og dokumentasjon er gjort tilgjengelig på GitHub[4], med mål om å legge til rette for fremtidig utvikling av programvaren av interesserte parter.

Denne oppgaven fokuserer hovedsakelig på implementeringen av indiseringsmetoden Dictionary Indexing (DI), en indiseringsmetode som baserer seg på bruken av et bibliotek med simulerte diffraksjonsmønser for å bestemme kornorientering og fase. I tillegg presenteres implementeringen av verktøy for å velge ut et mindre område av et EBSD skann samt et verktøy for interaktiv undersøkelse av EBSD-datasett både før og etter indisering. Som en del av utviklingsprosessen har det i tillegg blitt utarbeidet en omfattende brukermanual som kan lastes ned sammen med programvaren.

DI-implementeringen blir testet på en Al-10wt%Si-prøve og viser hvordan DI kan brukes til fasedifferensiering av to lignende krystallstrukturer, noe som konvensjonell Hough-indeksering (HI) har problemer med. Den samme DI-implementeringen blir også brukt for fasedifferensiering i en super-duplex rustfritt stål (SDSS)-prøve med σ- og χ-presipitater, og viser hvordan modulen fungerer godt med flere faser.

Til slutt presenteres en metode for å finne projeksjonssenteret (PC) fra arbeidsavstanden (WD) for et spesifikt SEM/EBSD-detektoroppsett. Metoden baserer seg på det observerte lineære forholdet mellom PC-koordinatene og WD, og en kalibreringslinje kan deretter beregnes for alle tre PC-koordinater (pc_x, pc_y, pc_z) ved interpolasjon mellom to PC-koordinater bestemt ved to arbeidsavstander. Det vises at PC-koordinatene gitt ved mellomliggende arbeidsavstander er relativt nøyaktige og i mange tilfeller kan brukes direkte for indisering. Denne metoden er implementert som et eget kalibreringsverktøy i **EBSP Indexer**.

[3]https://sourceforge.net/projects/ebsp-indexer/
[4]https://github.com/EBSP-Indexer/EBSP-Indexer

Table of Contents

List of Figures

List of Tables

List of code listings

Chapter 1

Introduction

1.1 Background

Electron backscatter diffraction (EBSD) and orientation indexing has proved itself as a powerful and versatile tool for the study of polycrystalline samples in the SEM and is widely used in both the field of materials science and geology. The characterization method relies on extracting crystallographic information from electron backscatter patterns (EBSPs), which can be used to determine eg. the crystallographic orientation and phase. The spatial resolution is typically at the sub-micron scale, meaning that even very fine structures can be investigated in great detail [3]. The conventional EBSD setup consists of a high-energy (20 kV) stationary electron beam, a highly tilted sample (70°) and an EBSD detector as shown in Figure 2.1. The stationary beam scans an area of the sample, and for every scan point, an EBSP is recorded. The diffraction pattern is the result of backscattered high-energy electrons undergoing elastic scattering with the crystal lattice. The diffraction pattern consists of bright bands, so-called Kikuchi bands, where each band corresponds to the gnomic projection of the diffracting lattice planes (hkl) for a given unit cell orientation. What lattice planes forming Kikuchi bands are governed by the extinction rules for the given crystal structure [4], meaning that the resulting EBSP is a unique identifier holding information about both the crystallographic orientation as well as the crystal lattice allowing not only for determining orientation but also phase identification.

The popularity of EBSD has grown since the introduction of the automated Hough indexing (HI) routine as proposed by N. K. Lassen in 1994. The method relies on extracting the position of individual Kikuchi bands using the Hough transform (eq. 2.3). The position of the bands and the angles between them are compared to a look-up table of theoretical values to determine the orientation and phase [5]. HI is a quick and efficient method for indexing EBSD datasets, but the conventional approach has some limitations, one of which is struggling to differentiate between phases belonging to the same or very similar crystallographic space groups, as the positions of the Kikuchi bands are the same for a given space group, with only variations in relative band intensity and width [6]. HI has also been shown to struggle when the patterns are noisy and arise from areas near grain boundaries where pattern overlapping occurs. This led to the development of a newer approach, Dictionary indexing (DI), first proposed in 2015 [7]. This method relies on comparing the experimental patterns with a dictionary of simulated EBSPs generated from a simulated master pattern (MP) [8]. The experimental patterns are compared with the simulated dictionary using a similarity metric like the normalized cross-correlation (NCC) or normalized dot-product (NDP) in order to identify the orientation and phase. As the method relies on comparing the full pattern, it has proven itself a very robust method for indexing, even when dealing with low-quality and noisy patterns [9]. It has also been used to successfully differentiate between phases belonging to the same crystallographic space groups, which has not been possible using conventional HI [10].

1.2 Motivation and scope of the work

DI has already been implemented as part of many commercial software suites, and there are also a few open-source and freely available implementations like the Fortran-90 package EMSoft [11] and the python library `kikuchipy` [12]. The free options are however only available through scripting, requiring the user to feel comfortable with programming and a text-based interaction with the indexing method, and as commercial software in general is expensive, not all academics will have access to the latest commercial alternatives. The aim of the work presented in this thesis has therefore been to create a free indexing software that provides easy access to the dictionary indexing approach and its capabilities, and which can be run on most ordinary laptops, including both Mac OS and Windows. The project is to be open-source in order to let anyone interested contribute to future development.

Chapter 2

Theory

This chapter will present the fundamental theory for Electron Backscatter Diffraction (EBSD), Hough indexing (HI) and Dictionary indexing (DI).

2.1 Electron backscatter diffraction

EBSD is a material characterisation method used for studying polycrystalline materials. It has proven itself a versatile and useful characterization method as it allows for studying a wide range of material characteristics from one single dataset. Among these are the grain size distribution, grain orientation, strain, grain boundary and even phase distribution of a sample [3, 13]. EBSD analysis is the study of Electron backscatter diffraction patterns (EBSP), often referred to as Kikcuhi backscatter patterns. The name's origin is from the first observation by Seishi Kikuchi in 1923, when studying the diffraction pattern from passing a cathode ray through a thin plate of mica [14]. In the same journal, a similar experiment was performed by S. Nishikawa and S. Kikuchi, studying similar diffraction patterns formed when directing the electron beam at a grazing index [15]. The latter pattern was later termed High-angle Kikuchi patterns [16] and is what is used in conventional EBSD analysis today. In this thesis, EBSP will refer to the high-angle Kikuchi pattern, whereas EBSD refers to the process of recording EBSPs and subsequent analysis.

2.1.1 Formation of EBSPs

The EBSP seen on the detector is formed by millions of backscattered electrons (BSE) from the sample surface, typically from an interaction depth of 20 nm depending on the density of the sample material, hitting the flat EBSD detector. The characteristic pattern of an EBSP can be explained from a purely geometrical point of view as the gnomic projection of the reflecting crystal planes onto the detector. When BSEs are diffracted by a single plane they form a diffracting cone, resulting in what is observed as a band on the detector screen, the Kikuchi band, with the center line being the intersection between the diffracting plane, with miller indices (hkl), and edges corresponding to the two diffracting cones. The expected diffracting planes for a given crystal structure, and thus the expected Kikuchi bands, can be calculated from the structure factor F_{hkl} and $|F_{hkl}|^2$, being proportional to the diffraction intensity [4, p. 109], can be used to determine the expected relative intensity of the Kikuchi bans in the resulting EBSP. The structure factor F_{hkl} is given in Equation 2.1:

$$F_{hkl} = \sum_{j=1}^{N} f_j e^{[-2\pi i(hx_j+ky_j+lz_j)]} \tag{2.1}$$

where N is the number of atoms in the unit cell, f_j the scattering factor and (x_j, y_j, z_j) the fractional position of atom j and hkl are the miller indices of the lattice plane. Planes with $F = 0$ are termed forbidden reflections and will not form Kikuchi bands. For a given space group this results in a set of extinction rules, namely forbidden hkl-combinations.

The geometrical approach described above explains what planes are expected to be visible on the detector for a given crystal orientation, given that the geometry of the experimental setup is known, but the actual intensity distribution of an EBSP is not described. This can be achieved by taking into account the effect of dynamic scattering effects when the BSEs leave the sample surface [17].

2.1.2 EBSD acquisition setup

A conventional setup for EBSD analysis is shown in Figure 2.1, the electron beam acceleration voltage is typically 20 kV, the working distance (WD) in the range of 15-20 mm, with a highly tilted sample, typically at an angle of 70°[18]. The pattern center (PC) is defined as the sample-to-detector projection of the incident point of the electron beam on the sample surface.

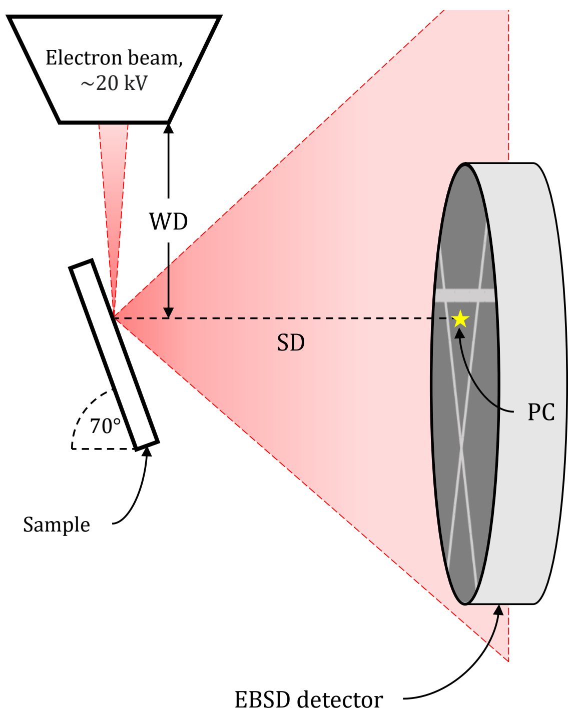


Figure 2.1: A standard EBSD setup. The electron beam source energy is typically 20 kV, the sample is tilted at a 70° angle, WD is the working distance, SD is the sample-to-detector distance and PC is the projection of the incident point of the electron beam on the sample.

The pattern center position is given by three coordinates, $(pc_x,\ pc_y,\ pc_z)$, and different indexing software and vendors use different reference frames. Oxford Instruments and EDAX TSL use the same convention where the pattern center coordinates are given in fractions of the EBSD signal width S_x with the origin in the lower-left corner of the signal grid. In the Bruker convention the coordinates $(pc_x,\ pc_y,\ pc_z)$ are given in fractions of S_x, S_y and S_y, respectively, with the origin in the upper left corner of the detector grid [12]. A comparison of the TSL and Bruker convention in an $\mathrm{S}_x \times \mathrm{S}_y$ grid is given in Figure 2.2.

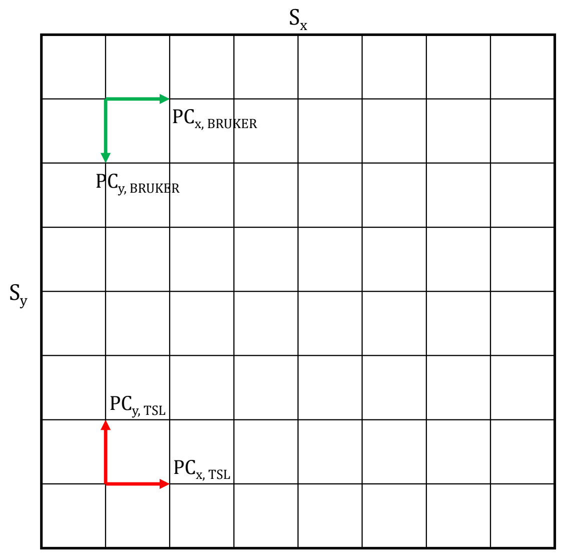


Figure 2.2: The Bruker and TSL PC-convention in a $S_x \times S_y$ detector grid.

The accuracy in determining the pattern center has been, and still is, a limiting factor in the accuracy of any EBSD indexing routine [19, 20]. One approach is to perform an optimization of the PC on a few calibration patterns from different positions and using the mean value as proposed in [20]. This optimization approach does however need a relatively good initial guess in order to converge. Fortunately, for a given EBSD acquisition setup, most components are relatively static with only a few parameters changing, one of which is the working distance. Figure 2.3 shows the relation between the working distance and the change in the PC-position, corresponding to the $\mathrm{pc_y}$ value. The relationship between the working distance and $\mathrm{pc_y}$ is linear, and a line on the form $\mathrm{pc_y} = a \cdot \mathrm{WD} + b$ can be calculated for a given acquisition setup by fitting a line between the PC value for two different working distances, preferably spaced by a minimum of 10-15 mm. Ideally both $\mathrm{pc_x}$ and $\mathrm{pc_z}$ stay relatively constant, but might shift depending on the relative movement of the sample stage, so the same linear fit can be done for $\mathrm{pc_x}$ and $\mathrm{pc_z}$ as well. The fitted lines will not provide a perfectly calibrated PC value but can be used as a good initial guess for further optimization of the PC.

Using a static mean value for the pattern center is sufficient for small area EBSD scans, as the shift in the pattern center is negligible as shown in [21], however for large area scans the change in the pattern center position must be taken into account.

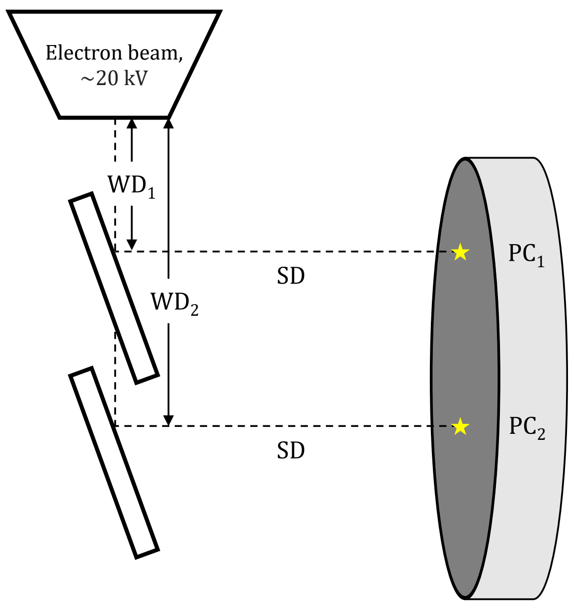


Figure 2.3: The relationship between pattern center position PC and the working distance WD in an EBSD detector setup.

The conventional detector in a standard EBSD setup consists of a phosphor screen and a high-sensitivity CCD camera, functioning as an indirect electron detector [3], but recent advances in detector technology now allow for direct electron detection, resulting in recorded patterns with higher image quality, less noise and finer details [22]. The first working prototype of the latter type was presented in 2013 by A. Wilkinson et. al. [22], with commercial detectors available for sale from 2020 [23]. The development of these newer detectors will allow for even higher quality acquisition of EBSD patterns going forward.

The EBSD dataset is acquired by scanning point-by-point of the sample, and for each grid point, an EBSP is stored. The BSEs from each point forms a Kikchui pattern on the EBSD detector, with characteristic lines as sketched in Figure 2.1. The resulting dataset can be described as a four-dimensional signal as presented in Figure 2.4, for each scanning point on the sample surface, an EBSP is stored. In `kikuchipy` nomenclature these axes are referred to as the *navigation axis* and *signal axis* respectively [12].

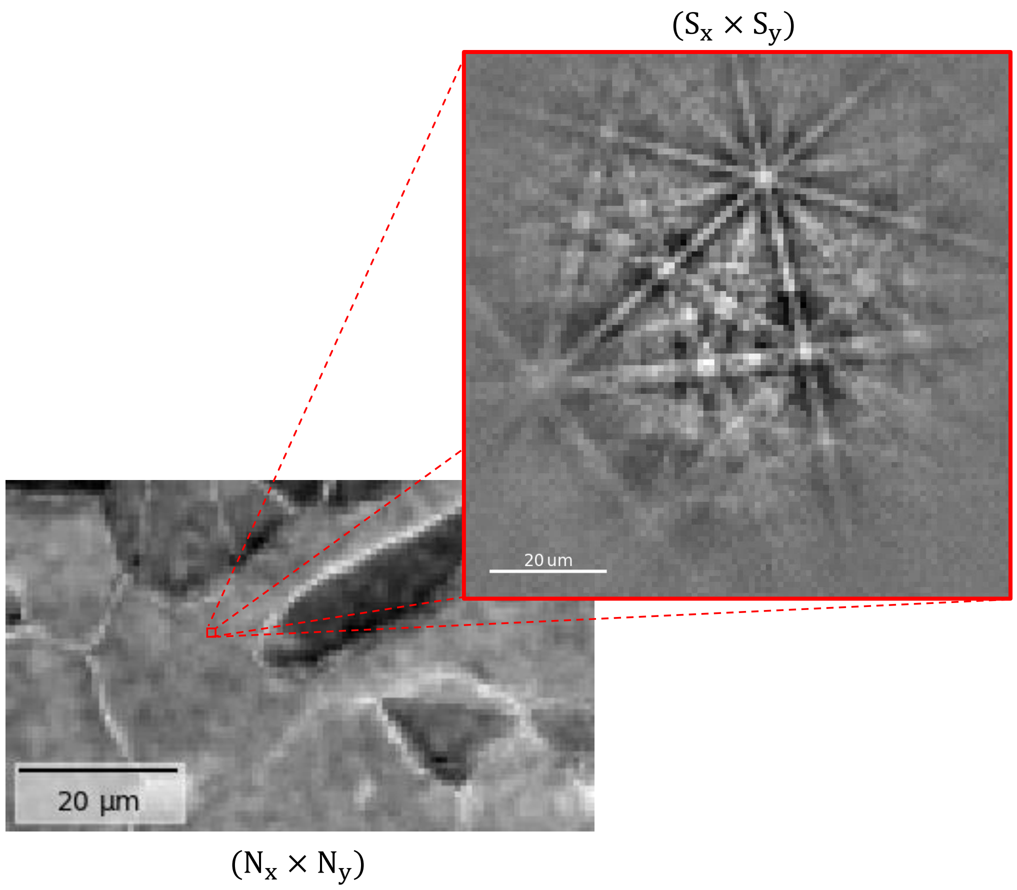


Figure 2.4: An EBSD recording can be treated as a four-dimensional signal where each pixel in the $(N_x \times N_y)$ region of interest stores a $(S_x \times S_y)$ EBSD pattern.

Interpretation of EBSPs

The intersection of Kikuchi bands forms zone axes corresponding to crystallographic directions of the unit cell, <uvw>. Identifying three zone axes in an EBSP is sufficient to provide a full description of the unit cell crystal orientation [24]. The lattice spacing d_{hkl} can also be calculated from an EBSP as the angular width of a Kikuchi band is twice the Bragg angle θ_{hkl} and d_{hkl} can be calculated directly using Braggs law [3] given in Equation 2.2:

$$2d_{hkl}\sin\theta_{hkl} = n\lambda \tag{2.2}$$

where n is an integer and λ is the wavelength.

2.2 EBSD Indexing

This section will cover the two most conventional approaches for automated EBSD indexing today. The traditional approach, Hough indexing (HI) (1994) [18], and the newer approach, Dictionary indexing (DI) (2015) [7].

2.2.1 Hough indexing

The traditional approach for EBSD indexing is Hough indexing, there are several commercial and open-source solutions available, all of which are based in some way on the framework first developed and presented by N. K. Lassen in 1994. The core of HI is the Hough transform, a technique for extracting geometrical features from images [5]. For the purpose of EBSD indexing the Hough transform maps the bright Kikuchi bands in the recorded EBSP, to bright peaks in Hough space with the parametrization given in Equation 2.3.

$$\rho = x \sin\theta + y \cos\theta \tag{2.3}$$

An example of the Hough transform on two lines is shown in Figure 2.5. As the figure shows, the transform of three points (x, y) along the same straight line in image space all intersect in the same point (ρ, θ) in parameter space, meaning that a line can be described by one point. When applied to an experimental EBSP the result is an image with bright spots often referred to as "butterfly peaks" due to their shape. The origin of this shape is described in detail by N. K Lassen in his original publication [5]. The high-intensity peaks can be easily identified and their position determined. Each individual peak corresponds to the center line of one Kikuchi band, and the identified Kikuchi bands can then be matched against a look-up table of theoretical bands for indexing.

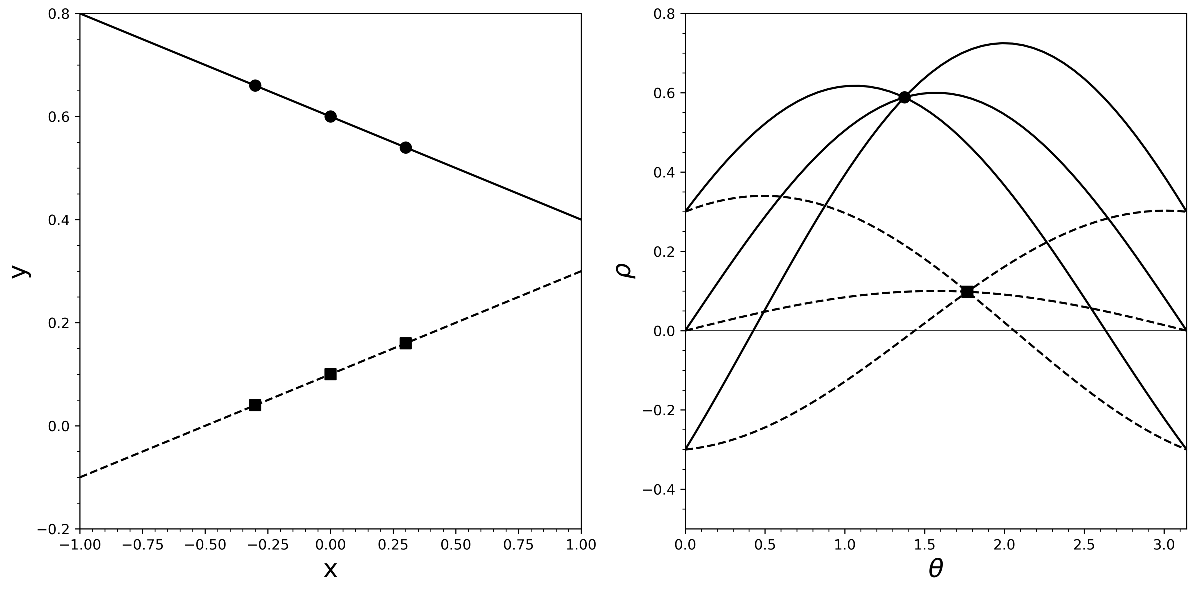


Figure 2.5: The Hough transform, transforming two lines to two points in parameter space using Equation 2.3.

2.2.2 Dictionary indexing

The newer approach, referred to as Dictionary indexing or pattern matching, differs from HI in the sense that instead of extracting the geometrical features of the EBSP and using these values in indexing, the full experimental patterns are compared to a dictionary of dynamically simulated EBSPs [8, 17]. A full framework for the DI approach was presented by Chen, Y. et al. in 2015 [7] and to date, there are several implementations available, both commercial and open-source, where notable open-source implementations are EMsoft [11] written in Fortran and the Python implementation `kikuchipy` [12]. The work in this thesis is based on the latter. The DI framework can be divided into four main parts: dynamical simulation of a master pattern (MP) [8], orientation sampling and interpolation [25], indexing [7] and subsequent orientation refinement [21].

The master pattern

The master pattern (MP) is a dynamical simulation that combines the dynamic scattering effects as described in [17] with a Monte Carlo simulation of the incident electrons to account for the stochastic nature of the energy distribution of BSEs [8]. The MP can be described as the result of enclosing a crystal unit cell in a sphere, firing a large number of electrons (in order of $10^9 - 10^{10}$) and capturing the BSEs in all directions. The sphere holds the electron yield in all directions and thus also all possible EBSPs that can be generated from the given crystal structure.

The two hemispheres of the MP sphere can be mapped to a 2D square using the modified equal-area Lambert projection as presented by Rosça [26], providing a full 2D description of the spherical MP. For crystal structures with inversion symmetry, only one hemisphere is enough to describe all possible EBSPs from the given crystal structure.

Sampling

The dictionary of simulated EBSPs is generated in a two-step process. First, the continuous Kikuchi sphere described by the MP is discretized by a uniform orientation sampling of 3D space, described in detail in [25]. By taking into account the symmetry of the crystallographic point group, only orientations that are unique within the Rodrigues fundamental zone (RFZ) are kept [25]. The number of orientations, and thus the size of the simulated dictionary for a given crystal structure, will vary depending on the degree of symmetry. A comparison of the cubic point group m$\overline{3}$m and the tetragonal point group 4/mmm is found in Table 2.1.

Table 2.1: Number of unique orientations after sampling of orientation space for two space groups. θ is the average angular sampling step size. Orientation sampling was done using `orix` [27].

θ [°]	Cubic (m$\overline{3}$m)	Tetragonal (4/mmm)
1°	857 973	2 571 073
2°	100 347	301 055
3°	30 443	90 979

For each orientation an interpolation between the MP Kikuchi sphere and a digital twin of the experimental detector, as described in section 2.1.2, is performed. The result is a dictionary of theoretical EBSPs "as seen" by the detector if the phase was present in the sample.

Indexing

Indexing is done by comparing every experimental EBSP with the dictionary of simulated patterns using a suitable comparison metric. Two common similarity metrics are the normalized dot-product (NDP), used in the first paper presenting the DI framework [7], and the normalized cross-correlation (NCC) [28] which is used as the default similarity metric in the `kikuchipy` implementation. The best matching experimental pattern is used for determining the orientation of the pixel.

Refinement

The discrete nature of the orientation sampling means that the "true" orientation of each individual pixel probably lies somewhere between the available orientations in the simulated dictionary. This can be accounted for by refining the orientations, the refinement takes the initial orientation from the indexing result and tries to optimize the orientation using a similarity metric like the NDP or NCC by varying the orientation around the initial "guess". The refinement stops when it has reached an optimum or a maximum number of iterations [21]. This approach removes to a large extent the discrete artefacts introduced by the DI approach and should in general be applied when using discrete indexing methods.

Indexing of multi-phase samples

The DI routine as described above can be extended to multi-phase samples simply by repeating the indexing process for the expected phases present in the sample. The resulting datasets with orientations and corresponding "match" can then be compared with each other and merged into one dataset with information about unit cell orientation and phase.

2.2.3 Hough indexing vs. Dictionary indexing

The influence of noise

Automated Hough indexing has proven itself as a reliable orientation indexing method since it was introduced, but reaches its limits for certain use cases, one being when dealing with datasets with poor pattern quality.

HI relies on detecting and extracting the Kikuchi band edges from the diffraction pattern in order to give reliable indexing results. For patterns of poor quality, either due to a high degree of noise or samples with a low yield of BSEs, the reliability of the Hough transform quickly drops [29, 30]. This is one of the biggest strengths of DI, where its robustness against noisy patterns has been demonstrated in several publications [9, 31]. A comparison of successfully indexed points between the two indexing approaches for increasing levels of noise is presented in figure 2.6.

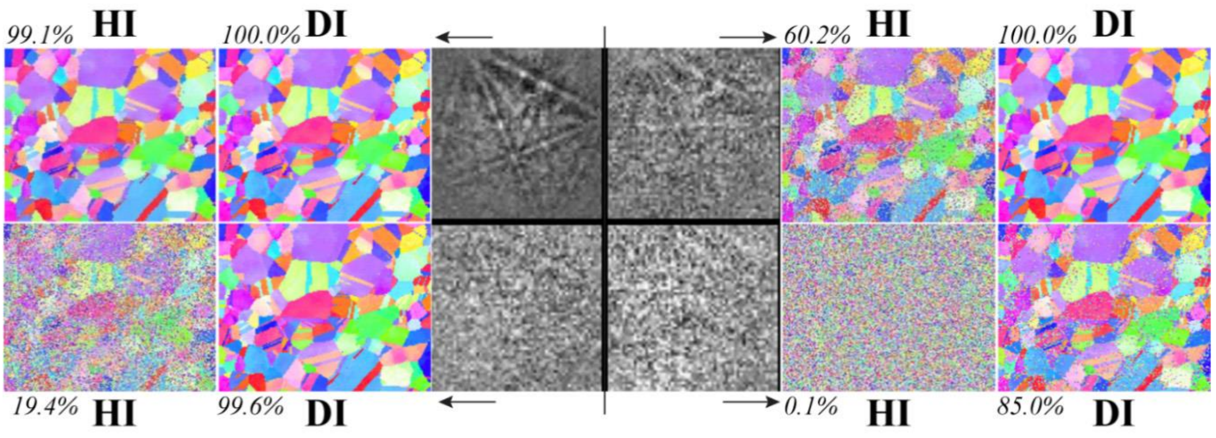


Figure 2.6: Comparison of indexing result using HI and DI on patterns with four different levels of noise. The percentage represents the number of successfully indexed points. Adapted from [32].

Phase differentiation

EBSD can be used for phase differentiation with spatial resolution at the sub-micron level, and for many samples, conventional HI is more than able to provide useful results, although with some limitations [33–35]. For samples with phases belonging to different crystal systems, eg. cubic and hexagonal, the differentiation can be fully based on a crystallographic criterion as the strongest diffracting planes are different and thus also what crystallographic planes will form bright Kikuchi bands on the detector.

The Hough transform can also be used for distinguishing between phases belonging to the same crystal system, but from different space groups, eg. the cubic space groups FCC and BCC, since the expected high-intensity diffraction planes, and thus brightest Kikuchi bands, are different [10]. In the case of samples belonging to the same, or very similar, crystal structures conventional HI is no longer able to reliably distinguish between the phases as the EBSPs for each phase will be very similar, with only subtle differences in relative intensity and width of the Kikuchi bands [6, 36].

One approach for overcoming this problem is to combine EBSD with X-ray energy-dispersive spectroscopy (EDS), where chemical information is recorded for each scanning point and used as an aid for the subsequent phase differentiation using EBSD. This allows for phase differentiation of samples even for phases with similar crystal structures, as long as the phases have distinct chemical composition [6]. The combined approach will however negatively impact the spatial resolution, as EDS typically has a resolution of 1 μm compared to the resolution of an EBSD scan which can typically be in the range of 10-50 nm [6, 37]. This means that the combined method cannot be used for samples with structural features at the nanometer scale.

Since DI uses the full EBSP in the indexing routine, it is able to detect the minor variations in intensity and bandwidth between EBSPs originating from phases with very similar crystal structures, allowing for reliable phase differentiation for the situations where conventional HI will struggle, still keeping the spatial resolution of EBSD, unlike the combined approach mentioned above [10].

2.3 Post-processing of EBSD datasets

Prior to indexing an EBSD dataset, it is necessary to apply some basic image processing in order to improve pattern quality. The goal of which is to increase the contrast and visibility of the Kikuchi bands. This applies to both indexing methods.

The first measure is to subtract the diffuse background readily apparent in Figure 2.8a. This is termed the static background and is the intensity distribution resulting from the inelastic scattering effects the BSEs undergo from the sample to the detector. The intensity fades moving outward from the center of the detector due to the increased travel distance between the sample and the detector. The static background is in effect the intensity distribution when all crystallographic information is removed. In conventional indirect EBSD detectors the CCD camera also captures the detector itself, this information is also stored in the static background. After removing the static background, only the intensity variations from point to point are left, increasing the visibility of the Kikuchi bands [5] as seen in Figure 2.8b.

There are still unwanted intensity variations in the pattern after removing the static background, the shading as seen in Figure 2.8b is a result of local defects and texture on the sample surface. These intensity variations can be removed by passing a high-pass filter, removing low-frequency variations in the image. A process often referred to as dynamical background removal [7]. The effect of dynamic background removal is shown in Figure 2.8c.

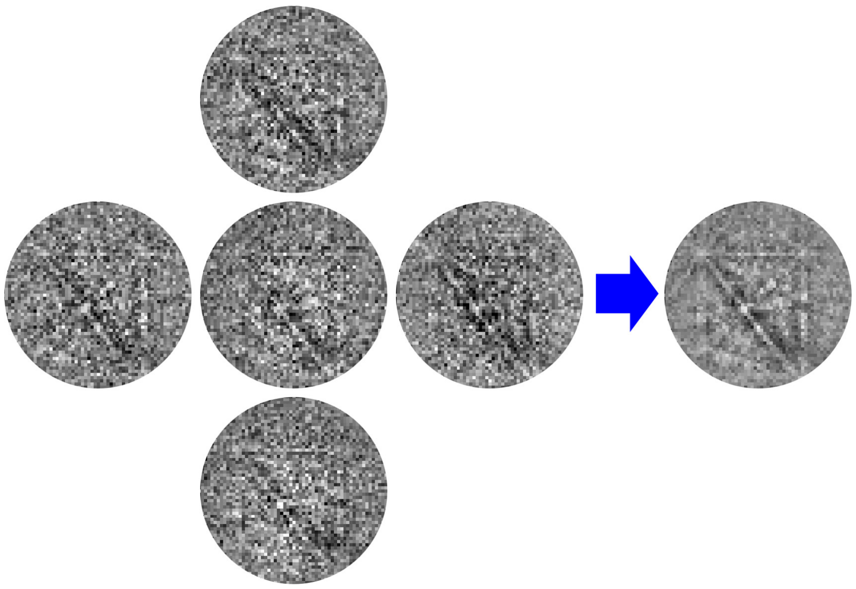


Figure 2.7: Schematic representation of averaging of a pattern with the four closest neighbouring patterns. Adapted from [9].

A third measure is to reduce the amount of noise in the pattern. As noise is a randomly distributed signal, it can be removed by averaging. Averaging can be done both online, meaning at the instrument, and offline, in post-processing. Online averaging involves recording n EBSPs from the same point and taking the mean value of the recorded patterns, resulting in patterns with little noise, but at the cost of the scan taking n-times longer. In post-processing, de-noising can be done by averaging each EBSP with its neighbours as schematically presented in Figure 2.7. For EBSPs in the interior of a grain, this is a simple and effective way of reducing the noise level in the signal as neighbouring patterns are virtually the same, but this is not necessarily the case at grain boundaries or structures with very fine features, as the averaging in effect increases the interaction volume and in such cases, neighbour averaging should be used with care [9]. Figure 2.8d shows the effect of averaging an EBSP with 8 neighbouring patterns using a Gaussian kernel with $\sigma = 1$. The noise level is clearly reduced, and the contrast of the Kikuchi bands is increased.

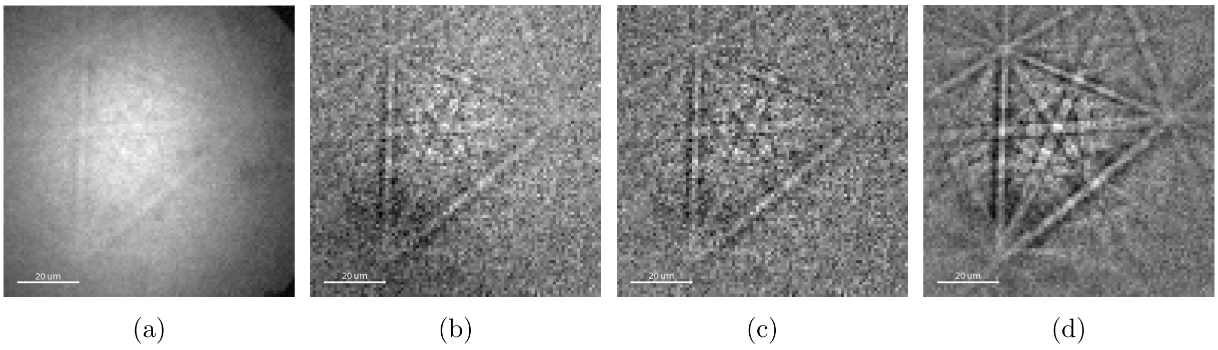


Figure 2.8: Comparison of EBSD patterns when applying different pre-processing steps. **(a)** the raw pattern, **(b)** static background removed, **(c)** dynamic background removed and **(d)** averaging with neighbouring patterns with a (3×3) Gaussian kernel [31].

2.4 EBSD imaging

Prior to indexing, a set of images can be generated from the EBSD dataset directly. This involves the Image quality map, Average dot product map and also the use of the EBSD signal as a virtual BSE detector.

2.4.1 The image quality factor

As it is convenient to quantitatively determine the quality of the EBSPs in an EBSD dataset prior to indexing, the image quality (IQ) metric was formulated. This is typically calculated as proposed by N. K. Lassen in [5] and the calculation measures how far away or close to the recorded EBSP is to pure white noise. The metric is usually given on a scale from 0 to 1, where 1 is a perfect pattern with no noise. A typical EBSD dataset will have patterns with an image quality in the range of 0.3-0.6 depending on the sensitivity of the detector and sample preparation. Patterns with an IQ value below 0.3 will to the naked eye be perceived as an image with no recognizable Kikuchi bands. A so-called Image quality map can be generated by calculating the image quality for all patterns in an EBSD dataset, which provides a quick way of evaluating the overall quality of the recording.

2.4.2 Average dot product map

An average dot-product (ADP) map captures the relative differences between neighbouring diffraction patterns. The dot product between a pattern and its neighbours is calculated and the mean value gives a measure of how similar pixels are to each other, providing information about the relative orientation gradient moving from pixel to pixel. This can be used for highlighting areas subject to large changes in orientation, like grain boundaries and sample artefacts. The mapping says nothing about the unit cell rotation, only point-to-point orientation differences.

2.4.3 Vrtual BSE imageing

The EBSD detector can be used as a virtual BSE detector, and it has been shown that it is possible to produce BSE images from EBSD recording similar to slow scan SEM imaging by extreme binning of the detector pixels [38]. By using the full EBSP as a single pixel it is possible to generate what is termed a mean intensity map, capturing the total intensity of the EBSP for each pixel, this can be used for generating an image of the scanned sample area. It is also possible to assign different areas of the detector to specific color channels, which can be used to generate RGB virtual BSE (VBSE) images.

Chapter 3

Software tools

This chapter provides an overview of the main software tools and packages used in the development of **EBSP Indexer**.

3.1 Kikuchipy

The software is built around the open-source Python library `kikuchipy` which provides tools for handling and analysing, as well as both geometrical and kinematic simulation of EBSD patterns. The library can read EBSD datasets from a wide range of vendors including NORDIF, TSL OIM, Bruker and Oxford. The documentation also provides in-depth guides in the form of `jupyter` notebooks for visualization and indexing EBSD patterns as well as interpretation of the resulting orientation maps [12]. The library builds on the multi-dimensional data handling from the Python library `hyperspy` [39].

Figure 3.1: The `kikuchipy` logo.

The library offers methods for both Dictionary indexing as described in [7] and conventional Hough indexing. The latter functionality is provided by the Python library `PyEBSDIndex` [40]. Version 0.8.4 of `kikuchipy` is used in **EBSP Indexer v0.1.0**.

3.2 Orix

`Orix` is a Python library for studying crystallographic orientations and symmetry. It provides a wide range of tools for both visualisations and handling datasets with information about unit cell rotations and phases with the `crystalMap`-class.

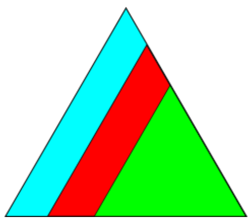


Figure 3.2: The `orix` logo

In **EBSP Indexer** `orix` is primarily used for the uniform sampling of orientation space when creating simulation dictionaries from master patterns as well as imaging of the resulting orientation maps generated from either DI or HI. Version 0.11.1 is used in **EBSP Indexer v0.1.0**.

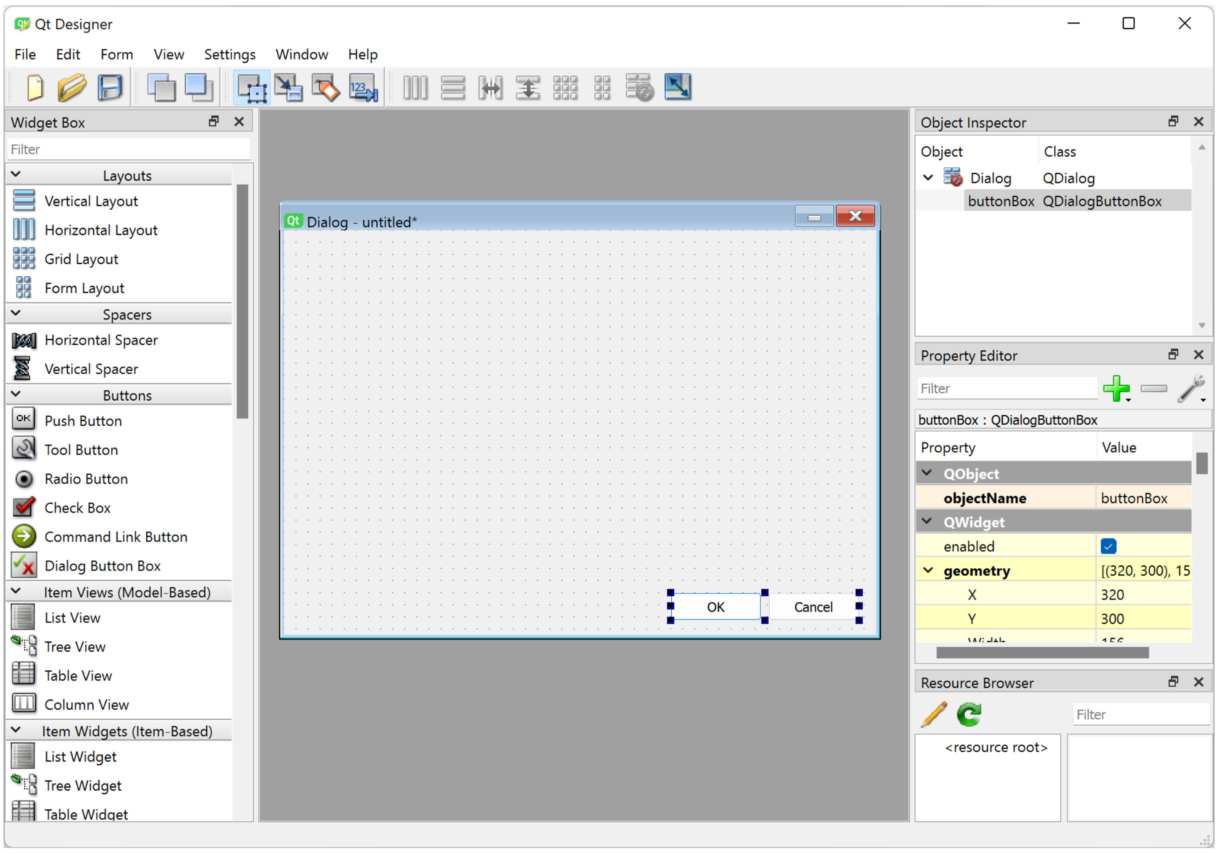
3.3 Diffsim

The open-source Python library `diffsims` provides tools for the simulation of diffraction [41]. It is primarily used for calculating reciprocal lattice vectors and the corresponding structure factors used in the geometrical simulations provided in the software. Version 0.5.0 is used in **EBSP Indexer v0.1.0**

3.4 Qt and PySide6

Qt is a cross-platform framework for the development of applications and user interfaces. Qt, and the latest version Qt 6, is distributed both under a commercial licence as well as a GNU General Public License (GPL) and can be used freely for non-commercial software. The framework is written in C++, but almost complete access to the Qt framework is achieved with the Qt binding `PySide6`, developed and maintained by the **Qt for Python**-project.

UI elements can be designed both pragmatically and in a drag-and-drop fashion using the *Qt Designer* which is part of the `PySide6` distribution. An example of an empty UI project in Qt Designer is given in Figure 3.3.



Object	Class
Dialog	QDialog
buttonBox	QDialogButtonBox

Property Editor

buttonBox : QDialogButtonBox

Property	Value
QObject	
objectName	buttonBox
QWidget	
enabled	✓
geometry	[(320, 300), 15
X	320
Y	300

Figure 3.3: An empty Qt Designer window where the layout of the GUI can be designed by simple drag-and-drop.

Chapter 4

Functionality and development

This chapter will give an overview of the authors' main contributions to the development of the software functionality. This chapter can be considered as the first part of the results from this master thesis project, with chapter 5 presenting results from the application of the final version of **EBSP Indexer**. Table 4.1 summarizes the available features in **EBSP Indexer v0.1.0**. Features in ***bold italic*** are covered in detail in the following section. The main developer for each module is given in the last column, but all three developers have contributed in some way to most parts of the software. To get a full overview of the available functionality the reader is referred to the master theses written by Østvold [42] and Leth-Olsen [2], as well as the software itself [1].

Table 4.1: Current functionality available in EBSP Indexer v0.1.0.

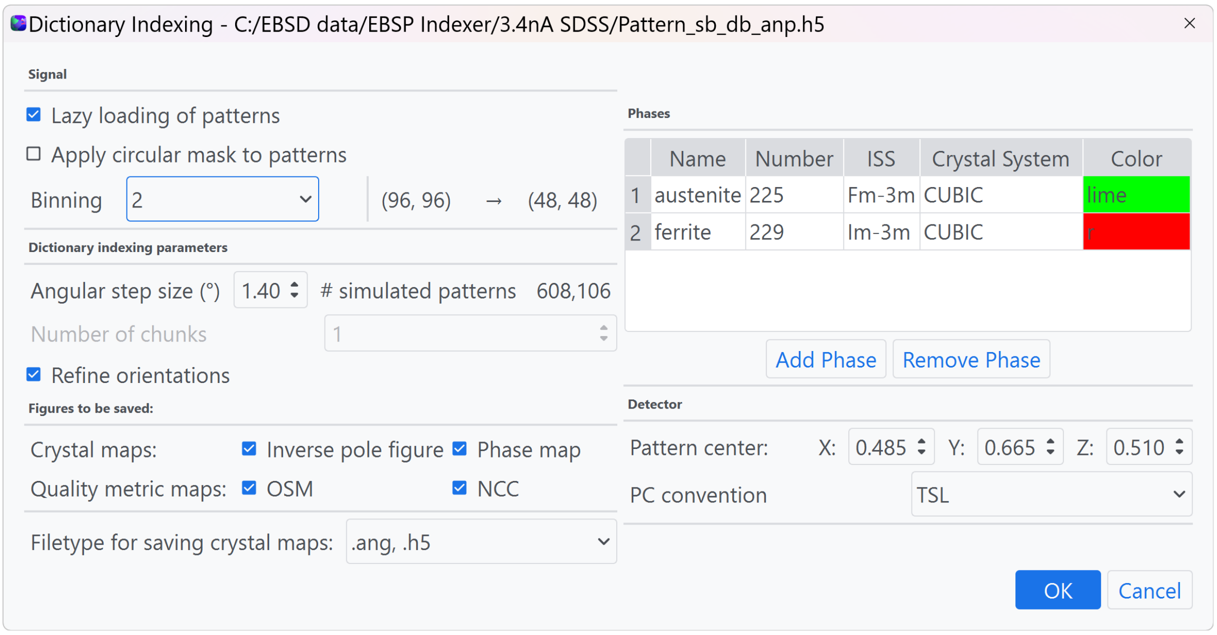
Pattern processing		**Developer**
S/N-improvement	Static background removal	*Relling*
	Dynamic background removal	
	Pattern averaging	
ROI selector	Selection of sub-ROI of the EBSD dataset	*Relling*
PC calibration		
Calibration patterns	Optimization of PC from calibration patterns	*Leth-Olsen*
Pattern selection	Optimization of PC from EBSPs from the dataset	*Leth-Olsen*
WD calibration	Calibration curve for added microscopes	*Relling*
EBSD Indexing		
Hough indexing	Hough indexing of using `PyEBSDIndex`[1]	*Østvold*
Dictionary indexing	Dictionary indexing using `kikuchipy`	*Relling*
Post-processing		
Orientation refinement	Orientation refinement using `kikuchipy`	*Østvold*
Visualisation		
Signal navigation	EBSD dataset	*Relling*
	Indexing results with geometrical simulation	
Feature maps	Mean intensity map	*Relling*
	Image quality map	
	Average dot-product map	
	Virtual BSE map	
Additional tools		
Terminal	Python interpreter with access to the `kikuchipy` library	*Østvold*
Job manager	Display pending, active and finished processes	*Østvold*
System viewer	File system viewer	*Østvold*

[1]`PyEBSDIndex` currently supports indexing of FCC and BCC phases.

4.1 Modules

4.1.1 Dictionary indexing

The Dictionary indexing routine used in the software is based on the `Jupyter` notebook tutorials available in the `kikuchipy` documentation [12]. It is initiated in the dialog window shown in Figure 4.1 where the indexing parameters can be set. The example setup is for indexing a super duplex stainless steel (SDSS) sample. Full documentation for the dictionary indexing module is found in Appendix A.


Dictionary Indexing - C:/EBSD data/EBSP Indexer/3.4nA SDSS/Pattern_sb_db_anp.h5

Signal

☑ Lazy loading of patterns
☐ Apply circular mask to patterns
Binning: 2 (96, 96) → (48, 48)

Dictionary indexing parameters

Angular step size (°): 1.40
# simulated patterns: 608,106
Number of chunks: 1
☑ Refine orientations

Figures to be saved:

Crystal maps: ☑ Inverse pole figure ☑ Phase map
Quality metric maps: ☑ OSM ☑ NCC
Filetype for saving crystal maps: .ang, .h5

Phases

	Name	Number	ISS	Crystal System	Color
1	austenite	225	Fm-3m	CUBIC	lime
2	ferrite	229	Im-3m	CUBIC	r

Add Phase | Remove Phase

Detector

Pattern center: X: 0.485 Y: 0.665 Z: 0.510
PC convention: TSL

OK | Cancel

Figure 4.1: The dictionary indexing dialog window in **EBSP Indexer v0.1.0**. The example setup is for indexing an SDSS sample.

Signal parameters

The parameters set in the *Signal* section are applied to the EBSD dataset. *Lazy loading of patterns* specifies if the dataset should be loaded into memory (RAM) in smaller chunks. When checked the dataset is loaded as the `kikuchipy` signal `LazyEBSD`, allowing handling of datasets larger than available memory on the computer. If not checked, the dataset is loaded as the `kikuchipy` signal `EBSD` with the full dataset loaded into memory. The *Apply circular mask to patterns* does as it says, apply a circular mask to every EBSP, removing the recorded signal in the outer edges of the pattern, meaning that only the signal from the center of the pattern is used in indexing. An example of applying a circular mask is shown in Figure 4.2, where the outer region pixels are set to a value of 0.

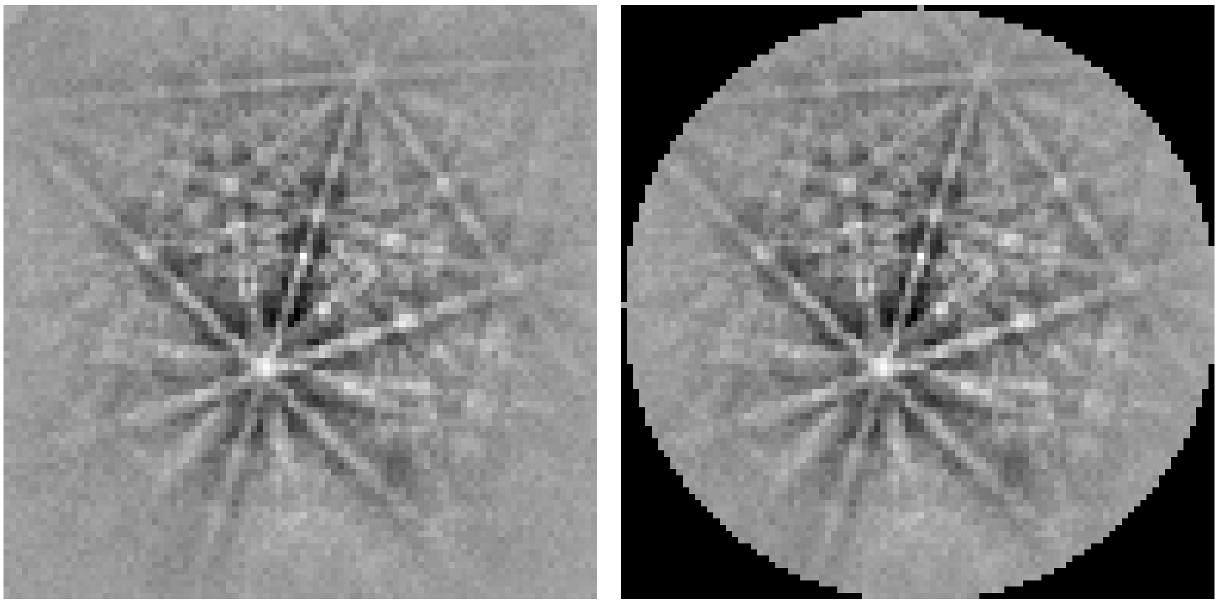


Figure 4.2: Applying a circular mask to the EBSP. The mask sets the pixel value in the areas to be excluded from indexing to 0.

The last signal parameter is *Binning*, this means combining pixels into bigger ones, reducing the size of the EBSP. As DI is an image comparison method, the reduced size of each pattern will reduce the computation time, but at the cost of accuracy. The signal can only be binned down by its divisors, which are calculated upon initialization of the DI dialog window. The available binning factors are added to the drop-down menu. The resulting signal shape is displayed next to the binning factor as seen in Figure 4.1. A binning factor of 2 means to combine an array of (2×2) pixels into one, reducing an EBSP of size (96×96) to (48×48). The effect of binning with factors 2 and 4 is shown in Figure 4.3.

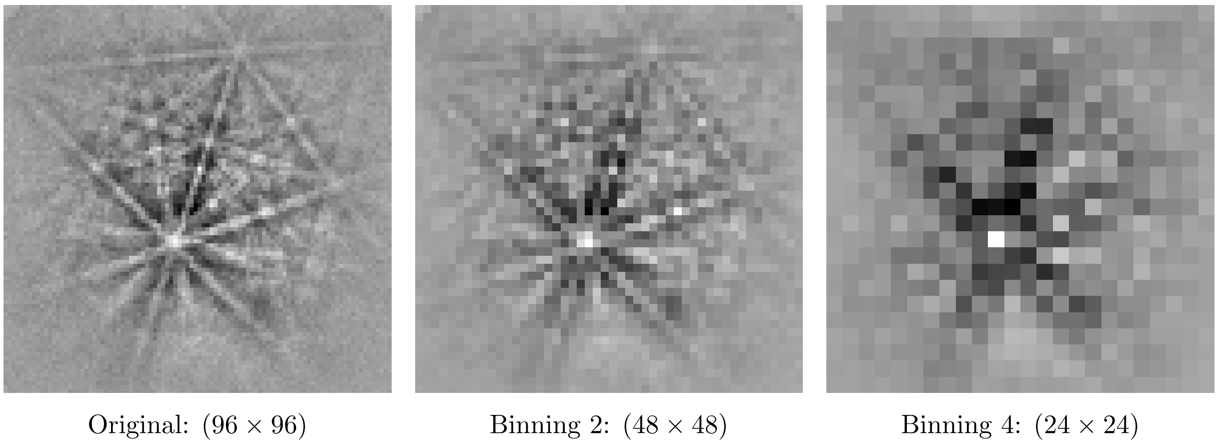

Original: (96×96)
Binning 2: (48×48)
Binning 4: (24×24)

Figure 4.3: The effect of signal binning with a binning factor of respectively 2 and 4. The binning and resulting EBSP shape is given under each image.

Dictionary indexing parameters

This section defines the parameters specific to the dictionary of simulated EBSPs and the indexing method. The *Angular step size (°)* defines the orientation sampling step size and thus the number of simulated patterns contained in the dictionary. A preview of the resulting size of the total simulation dictionary is given, which is the sum of the expected size of the dictionary for each phase given in the phase list. This preview is only available for the cubic point group m-3m and the tetragonal point group 4/mmm. For other phases, the preview will show N/A. The preview calculation is given in Code listing 4.1.

```python
SAMPLE_ROTATIONS = {
    "3.0": {"4/mmm": 90979, "m-3m": 30443},
    "2.9": {"4/mmm": 97627, "m-3m": 32363},
    "2.8": {"4/mmm": 110709, "m-3m": 36829},
    "2.7": {"4/mmm": 124885, "m-3m": 41625},
    "2.6": {"4/mmm": 132893, "m-3m": 44073},
    "2.5": {"4/mmm": 157151, "m-3m": 52607},
    "2.4": {"4/mmm": 175689, "m-3m": 58453},
    "2.2": {"4/mmm": 226787, "m-3m": 75539},
    "2.3": {"4/mmm": 195441, "m-3m": 65097},
    "2.1": {"4/mmm": 262101, "m-3m": 87529},
    "2.0": {"4/mmm": 301055, "m-3m": 100347},
    "1.9": {"4/mmm": 357801, "m-3m": 119077},
    "1.8": {"4/mmm": 421899, "m-3m": 141267},
    "1.7": {"4/mmm": 492855, "m-3m": 164179},
    "1.6": {"4/mmm": 592249, "m-3m": 197225},
    "1.5": {"4/mmm": 729573, "m-3m": 243129},
    "1.4": {"4/mmm": 913161, "m-3m": 304053},
    "1.3": {"4/mmm": 1157317, "m-3m": 385545},
    "1.2": {"4/mmm": 1481139, "m-3m": 494039},
    "1.1": {"4/mmm": 1906537, "m-3m": 635777},
    "1.0": {"4/mmm": 2571073, "m-3m": 857973},
}
```

⋮

```python
def estimateSimulationSize(self):
    """
    Estimates the size of the simulation dictionary based on the angular step-size.

    If the phases are CUBIC or TETRAGONAL the number of orientations are collected
    from the SAMPLE_ROTATIONS look-up dictionary.
    If any other point groups are given, the method sets the dictionary size to N/A.
    """
    angle = float(self.ui.doubleSpinBoxStepSize.value())
    total_sim = 0
    if len(self.phaseList.ids) > 0:
        try:
            for i, ph in self.phaseList:
                total_sim += SAMPLE_ROTATIONS[f"{round(angle, 1)}"][
                    ph.point_group.name
                ]
            self.ui.numSimPatterns.setText(f"{total_sim:,}")
        except:
            self.ui.numSimPatterns.setText("N/A")
    else:
        self.ui.numSimPatterns.setText("N/A")
```

Code listing 4.1: Code for calculating the expected number of simulated EBSPs. A look-up table of the dictionary size for the point groups m-3m and 4/mmm for angular step sizes between 1° and 3° is defined in the code. For any other phases, the method returns N/A. From `dictionary_indexing.py`

The option *Number of chunks* sets how many chunks the simulated dictionary will be split up into if the EBSD dataset is not *lazily loaded* as explained in the section above. If *Lazy loading* is checked, the software automatically takes care of dividing the simulated dictionary into suitable chunks, but when not checked this chunking has to be manually set by the user. The number of chunks defines how many simulated patterns are loaded into memory at once during indexing, and can to some degree control the speed of the indexing. It does however require some knowledge from the user about the size of both the experimental dataset, the simulated dictionary and available memory. Non-lazy loading of patterns is in general an option only recommended for experienced users.

The last option *Refine orientations* tells the indexer if an orientation refinement should be performed on the resulting crystal map from indexing. The orientation refinement is done with a local optimization where the orientation, stored as a Bunge-Euler triplet $[\phi_1, \Phi, \phi_2]$, is varied in a trust region of $\pm1°$ for each orientation to find the orientation that gives the highest NCC value. Orientation refinement can also be done as a separate step in another dialog window, which is described in detail by Østvold in [42].

Figures and saving

The last section of the dialog specifies what images to generate from the indexed results as well as what file type the indexed results are stored in. The **Inverse pole figure (IPF)** map is the conventional way of displaying grain orientations with respect to a given axis. For IPF-maps from DI in **EBSP Indexer** the z-axis is perpendicular to the sample. The choice for generating a **phase map** is available for multi-phase samples, and the color of the phase in the resulting phase map is the same as shown in the *Phases*-table in the dialog window.

The two quality metric maps that can be generated is an NCC map, showing the score of the match between the experimental EBSP and the best matching simulated EBSP. This can be used to evaluate the quality of the indexing results, where a high NCC score corresponds to a good match. The other quality metric map is the orientation similarity metric (OSM)-map, which calculates how similar the indexed orientation in one point is to its neighbours.

The indexing results can be saved to two file formats, `.h5` and `.ang`. For further study of the indexing results within **EBSP Indexer** the results have to be stored in the `.h5`-format. The latter, `.ang`, is supported by other analysis software like ATEX [43].

Phases and detector geometry

The phases that the indexer will look for are loaded in the **Phase**-table. If PC-calibration, as described in detail by Leth-Olsen in [2], has been done, the file paths of the relevant master patterns are stored in the `project_setting.txt`-file and the phases are already loaded upon opening the dialog window as shown in Code listing 4.2. Master patterns can also be loaded/removed by pressing *Load phase* or *Remove phase* respectively.

```python
# Paths for master patterns and phases
self.mpPaths = {}
self.phaseList = PhaseList()
i = 1
while True:
    try:
        mp_path = self.setting_file.read(f"Master pattern {i}")
        mp = kp.load(mp_path, lazy=True)
        if mp.phase.name == "":
            mp.phase.name = path.dirname(mp_path).split("/").pop()
        phase = mp.phase.name
        self.mpPaths[phase] = mp_path
        self.phaseList.add(mp.phase)
        i += 1
    except:
        break

self.updatePhaseTable()
```

Code listing 4.2: Loading of phases from master pattern stored in the directory given in `project_settings.txt`. The phase is added to the `orix`-class `PhaseList`. From `dictionary_indexing.py`

The detector geometry is defined in the **Detector** section, only the PC and PC-convention are set here, the remaining detector and EBSD acquisition information, working distance (WD), sample tilt and the acceleration voltage is read from the EBSD dataset metadata. The PC is read from `project_setting.txt` if any PC-calibration [2] has been done prior to indexing. Alternatively, the PC is calculated based on the WD if a calibration curve for the EBSD setups has been added. WD calibration is described in detail in the next section. If neither options are available the PC is set to (0.5, 0.8, 0.5) upon loading and has to be set manually. The loading of PC values upon initialization of the dialog is shown in Code listing 4.3.

```python
# Update pattern center to be displayed in UI
try:
    self.pc = np.array(eval(self.setting_file.read("PC")))
except:
    try:
        microscope = ebsd_signal.metadata.Acquisition_instrument.SEM.microscope
        working_distance = (
            ebsd_signal.metadata.Acquisition_instrument.SEM.working_distance
        )
        self.pc: list[float] = pc_from_wd(
            microscope, working_distance, self.convention
        )
    except:
        self.pc = np.array((0.500, 0.800, 0.500))

self.ui.patternCenterX.setValue(self.pc[0])
self.ui.patternCenterY.setValue(self.pc[1])
self.ui.patternCenterZ.setValue(self.pc[2])
```

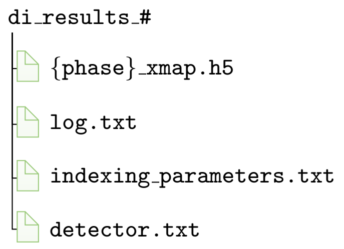
Code listing 4.3: Loading of PC values. The PC can be set from different sources, in the priority: the value stored in `project_settings.txt"`, the value from WD and lastly a fixed default value of (0.5, 0.8, 0.5). From `dictionary_indexing.py`

When *Ok* is pressed, the runner function `run_dictionary_indexing()` is triggered as shown in Code listing 4.4. The function creates a new directory for storing results and loads the full EBSD dataset. Both are passed as arguments to the indexer function `DiSetupDialog.dictionary_indexing` which is run in a separate computational thread with the `utils.sendToJobManager`-function. Threading and how parallel jobs and operations are handled by the software are covered in detail by Østvold in [42].

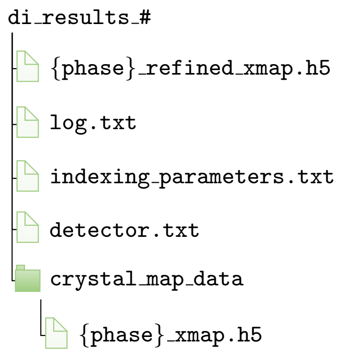
```python
# Call worker to start DI in separate thread
def run_dictionary_indexing(self):
    """
    Runner function, passes dictionary_indexing() to a new worker which is shown
    in the job manager list.

    Folder for storing results is created outside thread and ebsd signal is
    loaded before it is passed as an argument to dictionary_indexing().
    """
    # Create folder for storing DI results in working directory
    i = 1
    while True:
        try:
            self.results_dir = path.join(
                self.working_dir, f"di_results_{i}")
            mkdir(self.results_dir)
            break
        except FileExistsError:
            pass
        i += 1

    # load pattern

    ebsd_pattern = kp.load(
        self.pattern_path, lazy=self.ui.checkBoxLazy.isChecked())

    # Pass the function to execute
    sendToJobManager(
        job_title=f"DI {self.pattern_name}",
        output_path=self.results_dir,
        listview=self.parentWidget().ui.jobList,
        func=self.dictionary_indexing,
        allow_cleanup=True,
        allow_logging=True,
        ebsd=ebsd_pattern,
    )

```

Code listing 4.4: Initializing dictionary indexing and starting the `dictionary_indexing`-function in a separate thread. From `dictionary_indexing.py`

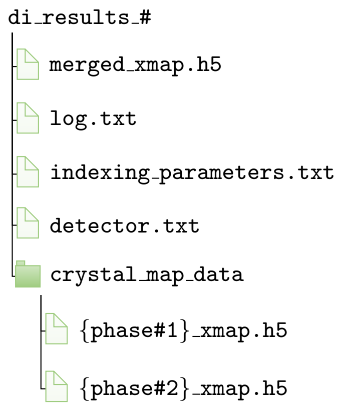
The output from DI depends on how many phases are indexed and if refinement is performed or not. The file output for four different indexing cases is presented in Figure 4.4.



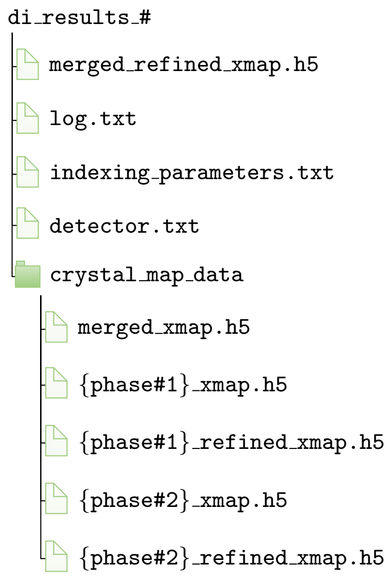
(a) Single phase, no refinement.



(b) Single phase, with refinement.



(c) Two phases, no refinement.



(d) Two phases, with refinement.

Figure 4.4: File structure of results from DI for four different indexing cases.

For single-phase indexing without refinement, the crystal map generated from indexing is stored in the results directory. When orientation refinement is applied, the refined crystal map is stored in the results directory and the raw indexing result is stored in the sub-folder `crystal_map_data`. For multi-phase samples, the merged result is stored in the results directory, with the individual indexing results for each phase stored in the sub-folder `crystal_map_data`. If orientation refinement is performed, the refined result is stored in the results directory and the un-refined merged result is stored in the sub-folder.

4.1.2 Signal navigation

The motivation for creating a custom signal navigation module was to have a tool that lets the user open and interactively investigate the pattern quality in EBSD datasets within the application. The signal navigator supports investigating EBSD datasets prior to indexing and investigation of the indexed results. The basic layout of the signal navigator widget is a navigator view to the left and a signal view to the right as shown in Figure 4.5. The current view, meaning the navigator and the currently displayed EBSP, can at any time be saved as a `.png` image by pressing *Save*. The signal navigator relies on two Python scripts, `signal_loader.py` and `signal_navigation_widget.py`, and the documentation for both scripts is found in Appendix B and C, respectively.

When a supported dataset is selected by the user a check is performed to determine if the dataset is an EBSD recording or a crystal map from Hough or Dictionary indexing. This is done by a simple `try-except`-statement, as the two loaders, respectively from `kikuchipy` and `orix`, will emit an error if the dataset is not supported. For each case the dataset is then passed as an argument to the correct custom class: `EBSDDataset` or `crystalMap` defined in `signal_loader.py`. The code can be found below in Code listing 4.5.

```
def load_dataset(self, file_path):
    self.file_dir = path.dirname(file_path)
    try:
        signal = kp.load(file_path, lazy=True)
    except:
        signal = orix.io.load(file_path)

    if isinstance(signal, kp.signals.ebsd.EBSD):
        self.dataset = EBSDDataset(signal)

    if isinstance(signal, orix.crystal_map.crystal_map.CrystalMap):
        self.dataset = crystalMap(signal, file_path, compute_all=True)
```

Code listing 4.5: `try/except`-statement to determine the type of dataset. The dataset is then passed as an argument to the correct custom class defined in `signal_loader.py`. From `signal_navigation_widget.py`

Investigating an EBSD dataset

Upon loading an EBSD recording, the dataset is passed as an argument to the `init()`-function of the `EBSDDataset`-class. A navigator is generated from the data and displayed in the navigator view to the left. By default, this is the **Mean intensity map**, with two other navigators available, **Image quality map** and **VBSE image**, which can be selected from the drop-down menu. Upon loading a dataset the first EBSP in the dataset is displayed in the signal view, corresponding to the navigation index [0, 0].

The navigation index is constantly updated when hovering over the navigation view, whereas the signal viewer is only updated when the user clicks on a point in the navigator. The corresponding diffraction pattern is then displayed in the signal viewer and the index of the current EBSP. An example of the signal navigation with an EBSD dataset from an SDSS sample is shown in Figure 4.5.

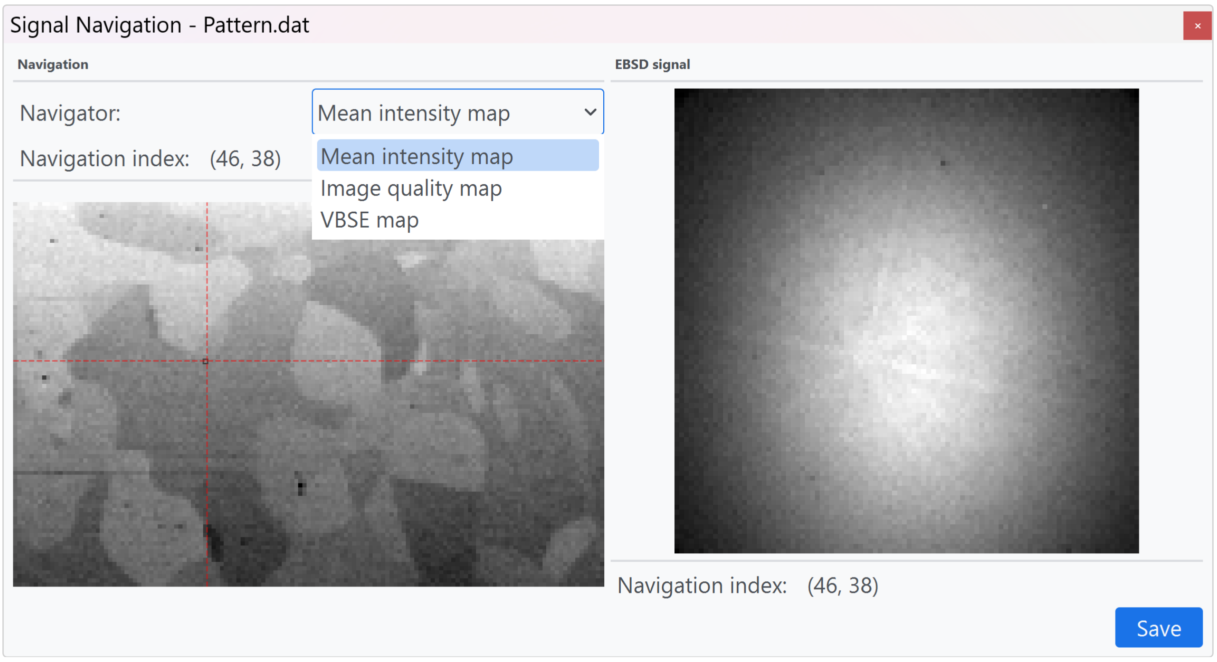
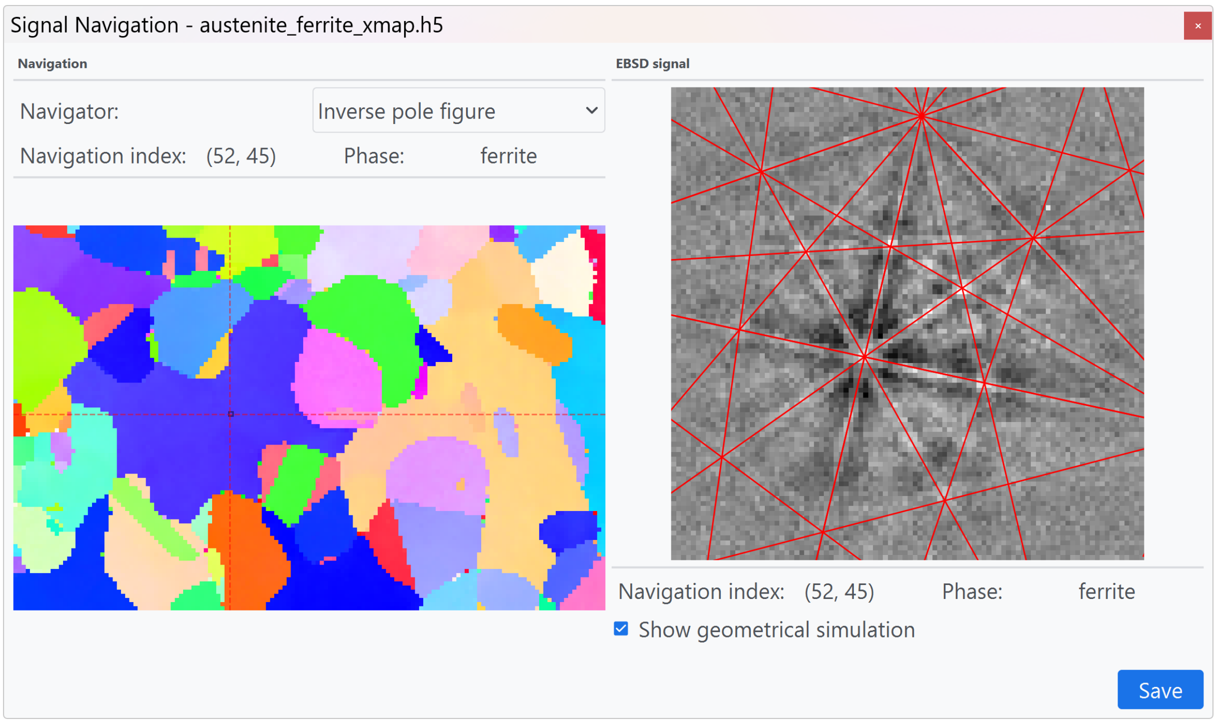


Figure 4.5: EBSD dataset from an SDSS sample opened in the signal navigator. The navigator can be changed in the drop-down menu, the index of the mouse hover is displayed to the left, whereas the index of the currently displayed EBSP is shown to the right.

Investigating indexed results

If the dataset is a crystal map from either Hough indexing or Dictionary indexing, it is passed to the custom class `crystalMap`. Again a navigator is calculated and displayed in the navigation view, and the EBSP with navigation index [0, 0] is shown in the signal view. The default navigator for a crystal map is the Inverse pole figure map. An example of a `crystalMap` from an SDSS sample opened in the signal navigator is given in Figure 4.6.

The indexed phase of each data point is displayed next to the *Navigation index* and is constantly updated when hovering over the navigation view. The indexing result can be visually investigated by toggling *Show geometrical simulation*. This takes the orientation of the given point and performs a geometrical simulation of the Kikuchi diffraction pattern, overlaying the center lines of the four strongest diffracting planes on the experimental pattern. A good match can then be verified by checking if the center lines are in agreement with the experimental diffraction pattern.



Navigator: Inverse pole figure

Navigation index: (52, 45) Phase: ferrite

Navigation index: (52, 45) Phase: ferrite

☑ Show geometrical simulation

Figure 4.6: Indexing result for an SDSS sample opened in the signal navigator. The index and phase of the mouse hover is displayed to the left, and the phase and index of the current EBSP is shown to the right. The geometrical simulation can be toggled by the checkbox and can be used to verify the indexing result.

Geometrical simulation

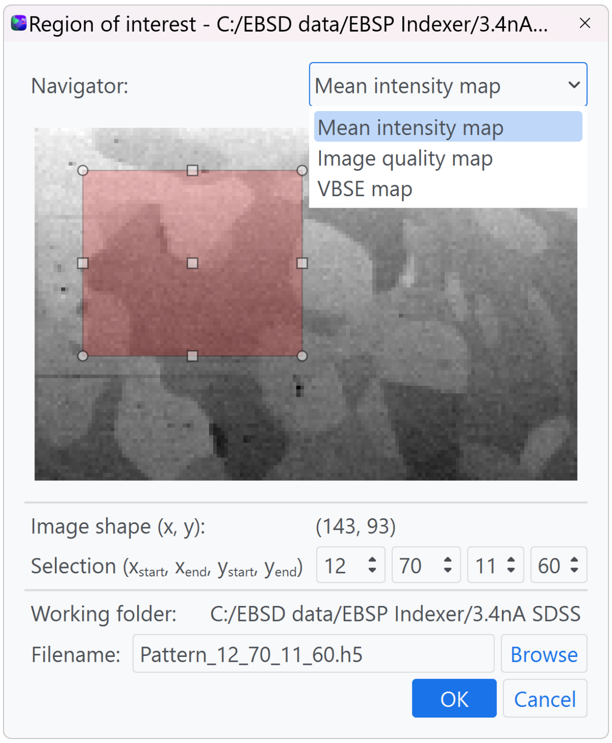
The diffracting planes used for the geometrical EBSD simulation are selected based on the value of their respective structure factor. Reciprocal lattice vectors are calculated for lattice planes with a minimum distance of 0.7 Å and the structure factor is calculated for all symmetrically unique planes. The four planes with the highest structure factor are kept, and symmetrically equivalent reciprocal lattice vectors are re-calculated before they are added to the list `hkl_simulations` which is stored as an attribute for the `crystalMap` object. This list is later accessed when the bands are simulated and projected onto the experimental diffraction pattern. The calculation of reciprocal lattice vectors and structure factors are carried out by the python library `diffsim` [41].

```python
def hkl_simulation(self):
    hkl_simulations = []

    for i, ph in self.crystal_map.phases:
        phase_lattice = ph.structure.lattice
        phase_lattice.setLatPar(
            phase_lattice.a * 10, phase_lattice.b * 10, phase_lattice.c * 10
        )  # diffsim uses ångstrøm and not nm for lattice parameters

        if i != -1:
            rlv = ReciprocalLatticeVector.from_min_dspacing(ph, 0.7)

            rlv.sanitise_phase()
            rlv = rlv.unique(use_symmetry=True)
            rlv.calculate_structure_factor("lobato")

            structure_factor = abs(rlv.structure_factor)
            order = np.argsort(structure_factor)[::-1]
            rlv = rlv[order[0:4]]
            rlv = rlv.symmetrise()
            hkl_simulations.append(rlv)

    return hkl_simulations
```

Code listing 4.6: Caluclation of the four strongest diffracting planes for the phases present in a crystal map. From `signal_loader.py`.

4.1.3 Region of interest selector

An EBSD dataset can be cropped to a smaller region of interest (ROI) using the **Region of interest** selection tool as shown in Figure 4.7. This can be done both before and after any pattern processing as the ROI selection makes a copy of the selected dataset. The documentation for the region of interest selection tool is found in Appendix D.


Region of interest - C:/EBSD data/EBSP Indexer/3.4nA...

Navigator: Mean intensity map

Image shape (x, y): (143, 93)

Selection (x_{start}, x_{end}, y_{start}, y_{end}) 12 70 11 60

Working folder: C:/EBSD data/EBSP Indexer/3.4nA SDSS

Filename: Pattern_12_70_11_60.h5

Figure 4.7: Region of interest selection dialog. The navigator can be changed in the drop-down menu in the upper right corner. The selection can either be set by changing the size of the red rectangle or using the spin-boxes.

Upon loading of the dialog box, the *Navigators* are calculated from the `EBSDDataset`-class defined in `signal_laoder.py` and added as options to the drop-down menu as shown in Code listing 4.7. The original EBSD scan area shape is given by *Image shape (x, y)*. A new ROI can be specified either by using the red selection rectangle or using the spin-boxes manually. The rectangle selection is updated with a call-back function as shown in Code listing 4.8. A press-and-release-event form the mouse inside the navigator area calls the function `line_select_callback()`-function and the start and end coordinates of the event are stored and set as the new ROI values.

```
try:
    ebsd_signal = kp.load(self.pattern_path, lazy=True)
except Exception as e:
    raise e

self.dataset = EBSDDataset(ebsd_signal)

for key in self.dataset.navigator.keys():
    self.ui.comboBoxNavigator.addItem(key)
```

Code listing 4.7: Loading of the EBSD dataset and adding navigators to the drop-down menu. From `region_of_interest.py`.

When the ROI indexes are changed using the spin-boxes the `textChanged`-event calls the `update_selection()` method and stores the new ROI values as shown in Code listing 4.9.

```
def line_select_callback(self, eclick, erelease):
    """
    Callback for line selection.

    *eclick* and *erelease* are the press and release events.

    """

    self.spinBoxBlockSignal(True)

    self.ui.spinBoxXstart.setValue(int(eclick.xdata))
    self.ui.spinBoxXend.setValue(int(erelease.xdata))
    self.ui.spinBoxYstart.setValue(int(eclick.ydata))
    self.ui.spinBoxYend.setValue(int(erelease.ydata))

    self.spinBoxBlockSignal(False)

    selection = self.getSelection()
    x0, x1, y0, y1 = selection
    if x1-x0 > 0 and y1-y0 > 0:
        pattern_name = f"{self.filenamebase}_{x0}_{x1}_{y0}_{y1}.h5"
    else:
        pattern_name = path.basename(self.save_path)
    self.ui.filenameEdit.setText(pattern_name)
```

Code listing 4.8: Updating the ROI with the rectangle selector. From `region_of_interest.py`.

The new ROI is stored as a new EBSD dataset file with the same name as its origin but with the ROI coordinates added at the end.

4.1.4 Pattern center coordinates from working distance

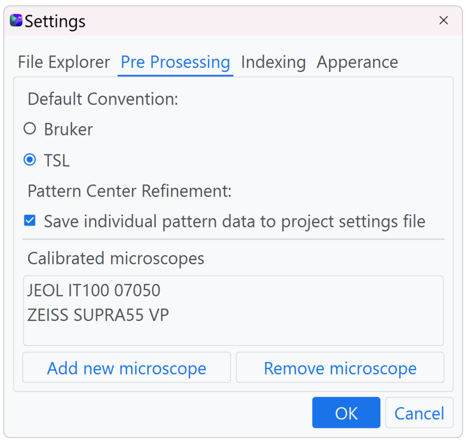
As described in section 2.1.2 it is possible to calculate a reasonable calibration curve for the PC as a function of the working distance by fitting a line between two PCs for two reasonably spaced working distances. It is possible to add this type of calibration in **EBSP Indexer**, allowing for easier determination of the PC for EBSD datasets acquired on the same microscope/EBSD detector setup. The calibration for a given setup is added from the **Settings** dialog as shown in Figure 4.8a where information about already added calibrations is available. From here it is possible to both delete old entries and add new ones. The documentation for the working distance calibration tool is found in Appendix E.

```
    self.ui.spinBoxXstart.textChanged.connect(
        lambda: self.update_selection())
    self.ui.spinBoxXend.textChanged.connect(
        lambda: self.update_selection())
    self.ui.spinBoxYstart.textChanged.connect(
        lambda: self.update_selection())
    self.ui.spinBoxYend.textChanged.connect(
        lambda: self.update_selection())
```

⋮

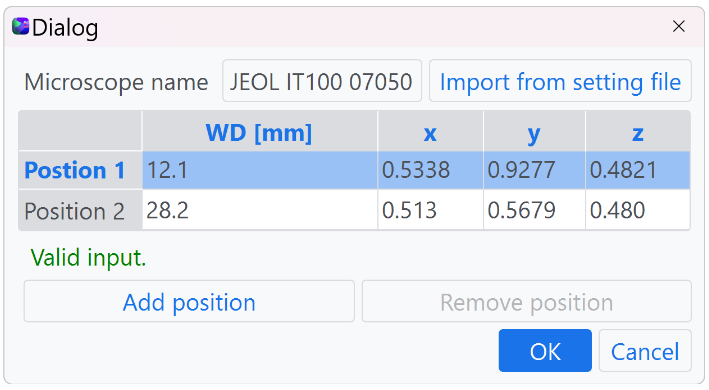
```
def update_selection(self):
    """Update selection rectangle when spinbox values are changed"""
    selection = self.getSelection()
    x0, x1, y0, y1 = selection
    if x1-x0 > 0 and y1-y0 > 0:
        pattern_name = f"{self.filenamebase}_{x0}_{x1}_{y0}_{y1}.h5"
    else:
        pattern_name = path.basename(self.save_path)
    self.ui.filenameEdit.setText(pattern_name)

    self.rs.extents = selection
```

Code listing 4.9: Updating the ROI using the spin-boxes. From `region_of_interest.py`.



Default Convention:
○ Bruker
◉ TSL
Pattern Center Refinement:
☑ Save individual pattern data to project settings file
Calibrated microscopes
JEOL IT100 07050
ZEISS SUPRA55 VP

(a) List of calibrated microscopes/EBSD detector setups in the settings dialog.



Microscope name: JEOL IT100 07050

	WD [mm]	x	y	z
Postion 1	12.1	0.5338	0.9277	0.4821
Position 2	28.2	0.513	0.5679	0.480

Valid input.

(b) Adding a calibration for a new setup.

Figure 4.8: The settings dialog displays the calibrated setups and the dialog window for adding a new calibration.

Adding a new microscope/EBSD detector setup is done in the dialog shown in Figure 4.8b. This requires that the user already has two EBSD datasets acquired at different working distances and that the PC is determined accurately for both. The microscope name is added in the text field at the top, this can either be done manually or for NORDIF EBSD datasets read from the `Setting.txt` stored together with one of the EBSD datasets. An example of the microscope metadata stored in a `Setting.txt` is shown in Code listing 4.10.

```
[Microscope]
Manufacturer        JEOL
Model        IT100 07050
Magnification        500        #
Scan direction        Direct
Accelerating voltage        20        kV
Working distance        14.9        mm
Tilt angle        70        °
```

Code listing 4.10: Microscope metadata in `Setting.txt` stored together with the EBSD dataset.

The microscope name is made up of the microscope *manufacturer* and the *model*, as is also the convention in `kikuchipy`. This ensures that the microscope name is correctly read and accessed at a later point.

```
def get_microscope_name(self):
"""
Gets the microscope name from the setting file Settings.txt stored in the same
folder as the EBSD dataset.

"""

if self.fileBrowserOD.getFile():
    setting_file_path = self.fileBrowserOD.getPaths()[0]
    N = 10
    microscope = {}
    try:
        with open(setting_file_path, "r") as f:
            for _ in range(N):
                item = next(f).strip().split("\t")
                if item[0].lower() in ["manufacturer", "model"]:
                    microscope[item[0]] = item[1]
        microscope_name = microscope["Manufacturer"] + \
            " " + microscope["Model"]
        self.ui.lineEdit.setText(microscope_name)
    except:
        QMessageBox(self).warning(
            self,
            "Could not set microscope name",
            "The file does not contain the keys 'Manufacturer' and 'Model'",
            QMessageBox.Ok,
            QMessageBox.Ok,
        )
        return
```

Code listing 4.11: The microscope name is read from the keys *manufacturer* and *model* in `Setting.txt`. From `pc_from_wd.py`.

The WD and corresponding pc_x, pc_y, pc_z are added for at least two working distances in the table. If wanted, it is possible to add additional calibration points by pressing **Add position**. All values in the table are constantly validated checking that the input values are numbers, that the input WDs are different and that the microscope name text field is not empty. If all checks are passed, the **Ok**-button is enabled and the calibration can be saved. The validation code snippet is found in Code listing 4.12.

```python
def checkValidInput(self):
    """
    Checks if the input is valid. The input is valid if the table contains
    only numbers and the microscope name is not empty.
    """
    columns = self.ui.tableWidget.columnCount()
    rows = self.ui.tableWidget.rowCount()
    checksum = columns * rows
    for column in range(self.ui.tableWidget.columnCount()):
        col = []
        for row in range(self.ui.tableWidget.rowCount()):
            try:
                float(self.ui.tableWidget.item(row, column).text())
                col.append(
                    float(self.ui.tableWidget.item(row, column).text()))
            except:
                checksum -= 1
        if column == 0 and len(col) > len(set(col)):
            checksum -= 1

    if checksum == columns * rows:
        self.ui.labelValidInput.setStyleSheet("QLabel{color:green}")
        self.ui.labelValidInput.setText("Valid input.")

    else:
        self.ui.labelValidInput.setStyleSheet("QLabel{color:red}")
        self.ui.labelValidInput.setText("Invalid input.")

    if checksum == columns * rows and self.ui.lineEdit.text():
        self.ui.buttonBox.button(QDialogButtonBox.Ok).setEnabled(True)
        self.ui.buttonBox.button(QDialogButtonBox.Ok).setToolTip(
            "Save microscope calibration."
        )
    else:
        self.ui.buttonBox.button(QDialogButtonBox.Ok).setEnabled(False)
        self.ui.buttonBox.button(QDialogButtonBox.Ok).setToolTip(
            "Missing valid microscope name and/or calibration values."
        )
```

Code listing 4.12: Validating the table values and checking if a microscope name is specified. If all checks are passed, the **Ok**-button is enabled and the calibration can be saved. From `pc_from_wd.py`.

The actual calibration is just a simple linear regressing for each PC-axis. The slope and intercept of the three fitted curves are stored as a tuple of tuples named `pc_curve`. This is again written to the program settings file with the microscope name as the identifier.

```python
def run_calibration(self):
    """
    Runs the calibration and saves the calibration values to the advanced settings file.
    The calibration values are saved as a tuple of tuples with microscope name as identifier.
    """
    self.setting_file = SettingFile("advanced_settings.txt")
    self.microscope_name = self.ui.lineEdit.text()
    calibration_values = [[], [], [], []]

    columns = self.ui.tableWidget.columnCount()
    rows = self.ui.tableWidget.rowCount()
    for column in range(columns):
        for row in range(rows):
            calibration_values[column].append(
                float(self.ui.tableWidget.item(row, column).text())
            )

    x_slope, x_intersept, * \
        _ = linreg(calibration_values[0], calibration_values[1])
    y_slope, y_intersept, * \
        _ = linreg(calibration_values[0], calibration_values[2])
    z_slope, z_intersept, * \
        _ = linreg(calibration_values[0], calibration_values[3])

    self.pc_curve = (
        (round(x_slope, 5), round(x_intersept, 5)),
        (round(y_slope, 5), round(y_intersept, 5)),
        (round(z_slope, 5), round(z_intersept, 5)),
    )
```

Code listing 4.13: Running the linear interpolation and saving the calibration curves to `advanced_settings.txt` with the microscope name as the identifier. From `pc_from_wd.py`.

4.2 User testing

A pre-release of the software was made available for a group of 10 students in April 2023 to be used in their project report on material characterization with the SEM as part of the course *TMT4166 - Experimental Materials Chemistry and Electrochemistry*. The software was used for indexing an EBSD dataset the students had recorded themselves and the results were included in their final reports.

The testing was split up into two parts, an introduction session and a lab session. The introductory session was used to give the students an introduction to the software and the core functionality, as well as to demonstrate both indexing methods. Prior to the introduction session, the students had received a copy of the pre-release version, both for Mac and Windows, so the session was also used to assist the students with any problems encountered during installation and getting the software up and running.

For the lab sessions, the group was split up into groups of 2-3 students, resulting in a total of four lab sessions. The sessions were dedicated to letting the students interact with the software themselves to process and index an EBSD dataset. The sessions were conducted under the supervision of Relling and Østvold, which were present both to give guidance but also to observe how the students interacted with the software in order to identify possible issues and bugs. A quick-start guide was made available to the students to give a few pointers on how to use the software beforehand, any additional guidance was kept to a minimum, as one of the aims of the software is to be relatively self-explanatory even for inexperienced users.

After the two sessions, the students were asked to give feedback, both on any bugs encountered during use as well as overall feedback on the interaction with the software. This was used going forward into the final developing stage resulting in version 0.1.0. The results from user testing and feedback are covered and discussed by Østvold in [42].

4.3 User guide

As part of the development and release of version 0.1.0 of **EBSP Indexer** a user guide for the software has been made. A draft of the guide was made available for the student during user testing, and revised before the release of the final version of the software. The guide aims to guide the user through a typical workflow for both HI and DI using **EBSP Indexer**, providing tips for indexing parameters and explaining what the different tools do. The user guide is specifically written for indexing EBSD datasets recorded on SEMs equipped with a NORDIF detector and is available together with the source code on GitHub. The full user guide is attached in Appendix F.

Chapter 5

Application of EBSP Indexer

This chapter presents the results from some specific applications of the final software and is considered part 2 of the results for this master's thesis.

5.1 Phase differentiation of aluminium and silicon

EBSP Indexer has been used for phase differentiation in an Al-10wt%Si with silicon particles at the sub-micron scale using DI. The results in this section have been summarized into an abstract which has been submitted to the 20th International Microscopy Congress (IMC20) taking place in Korea in September 2023. The abstract is attached in Appendix G.

Sample preparation and EBSD acquisition

The sample was prepared with the final polishing performed in a Buehler Vibromet 2 and the EBSD dataset was recorded from a scan area of 3.68μm $\times$ 2.34μm on a Zeiss Ultra 55 FESEM with a NORDIF UF-1100 EBSD detector with a step size of 0.02μm. The acceleration voltage, tilt angle and working distance were 20 kV, 70°and 22.5 mm respectively. Diffraction patterns with a resolution of 80×80 px were streamed to HDD at a rate of 300 fps. Pattern noise was reduced by applying a 2x online averaging. The static and dynamic background were removed before averaging with a Gaussian kernel with $\sigma = 1$ to improve the SNR.

Dictionary indexing

The pattern center was determined by optimizing the PC for five calibration patterns acquired prior to the EBSD recording and subsequent fine-tuning of the PC by fitting the PC for a 4×4 grid of patterns from the EBSD dataset. The mean of the latter optimization was used as the static PC for Dictionary indexing. The MPs for silicon and aluminium were simulated with EMSoft v5.0.0 [11] and the dictionary indexing and orientation refinement parameters are summarized in Table 5.1. The *Euler angle trust region* defines the search region of the orientation refinement algorithm and $NCC_{tolerance}$ defines the convergence limit of the optimization algorithm.

Table 5.1: Parameters for DI and orientation refinement.

Pattern center (x, y, z)	(0.524, 0.875, 0.555)
Angular step size [°]	1.4°
Binning	None
Optimization algorithm	Nelder-Mead
Euler angle trust region	(1°, 1°, 1°)
$\mathrm{NCC}_{tolerance}$	1×10^{-4}

Hough indexing

The same EBSD dataset from the Al-10%Si specimen was also indexed with HI using EDAX TSL OIM Data collection. The pattern center $(\mathrm{pc_x}, \mathrm{pc_y}, \mathrm{pc_z})$ was determined using TSL to a value of (0.539, 0.864, 0.560).

Indexing results from both indexing methods are represented by the inverse pole figure map and phase map for HI and DI in Figure 5.1 and 5.2, respectively. The average confidence index (CI) for HI was 0.02 and the average NCC match score for DI was 0.39. The phase fraction from HI was determined to be 43.3 % Al and 56.7 % Si and for DI 89.4% Al and 10.6% Si.

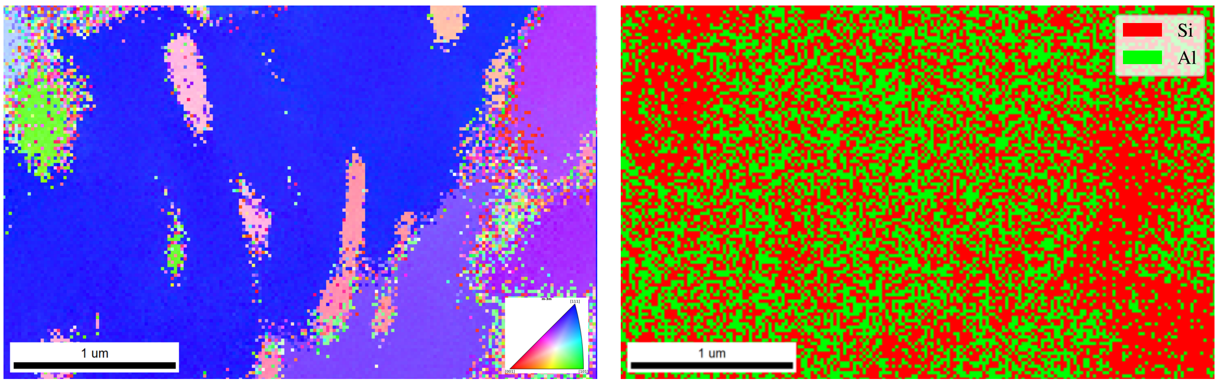


Figure 5.1: Indexing results from HI width EDAX/TSL OIM Data collection. The unit cell orientations are given in the inverse pole figure map (left) and the phase map (right). The phase fraction from HI was determined to be 43.3 % Al and 56.7 % Si.

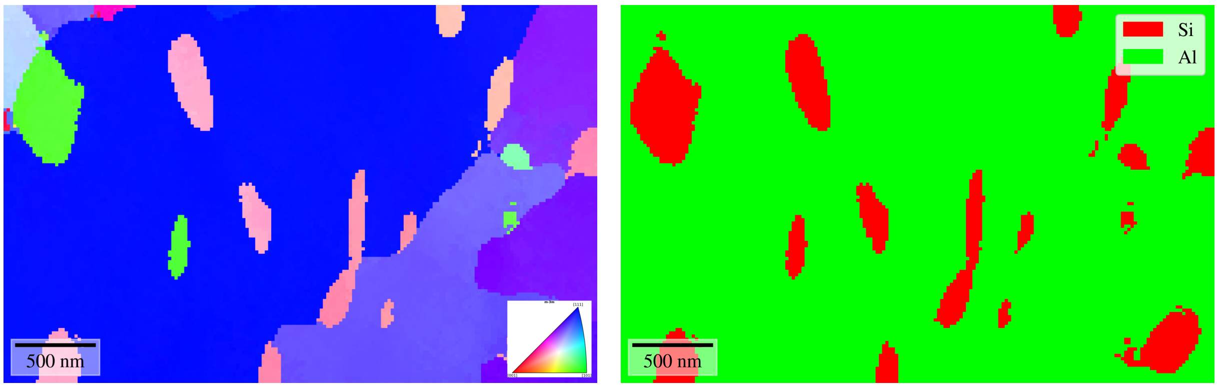


Figure 5.2: Indexing results from DI using **EBSP Indexer**. The unit cell orientations are presented in an inverse pole figure map (left) and the phase map is shown to the right. The phase fraction from DI was determined to be 89.4% Al and 10.6% Si.

The NCC-map from DI is given in Figure 5.3, where each pixel corresponds to the best matching value used to determine the orientation and phase. Darker regions have a lower match score. Two points indexed as aluminium and silicon, respectively, are marked and the corresponding experimental EBSP together with the simulated EBSPs for each phase is given in Figure 5.4. Histograms for the NCC match distribution for each phase are given in Figure 5.5, together with the mean score value for both aluminium and silicon, namely 0.402 and 0.301.

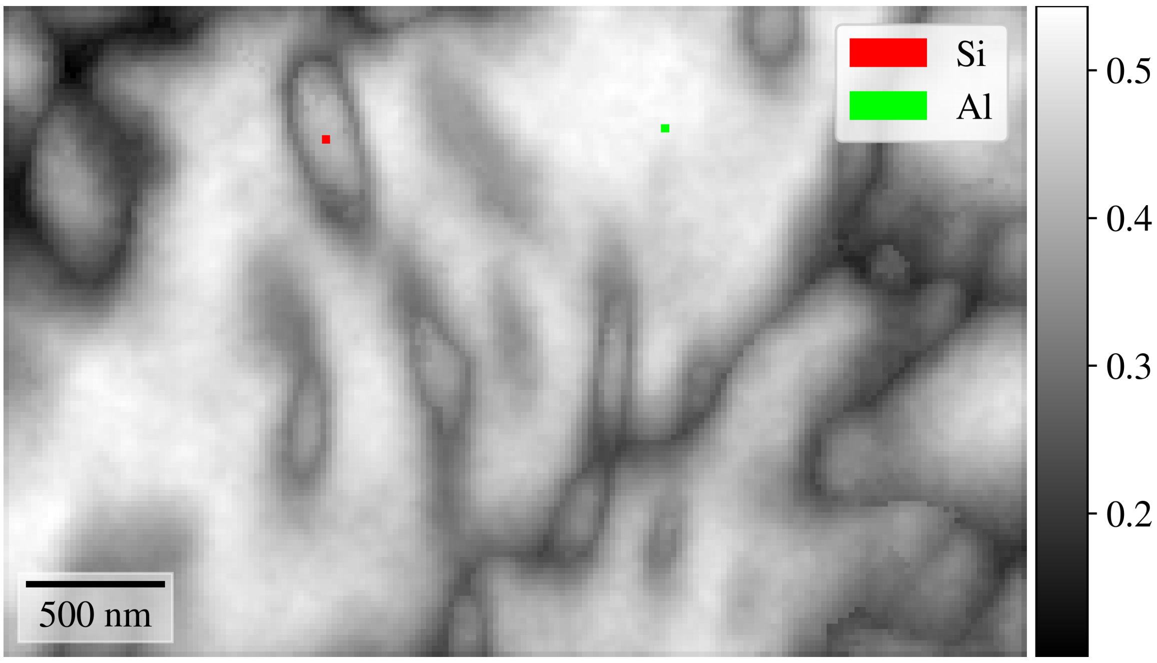


Figure 5.3: NCC-map for the merged indexing results. Two points indexed as aluminium and silicon are marked. The corresponding EBSP and the simulated diffraction patterns for each phase are given in Figure 5.4

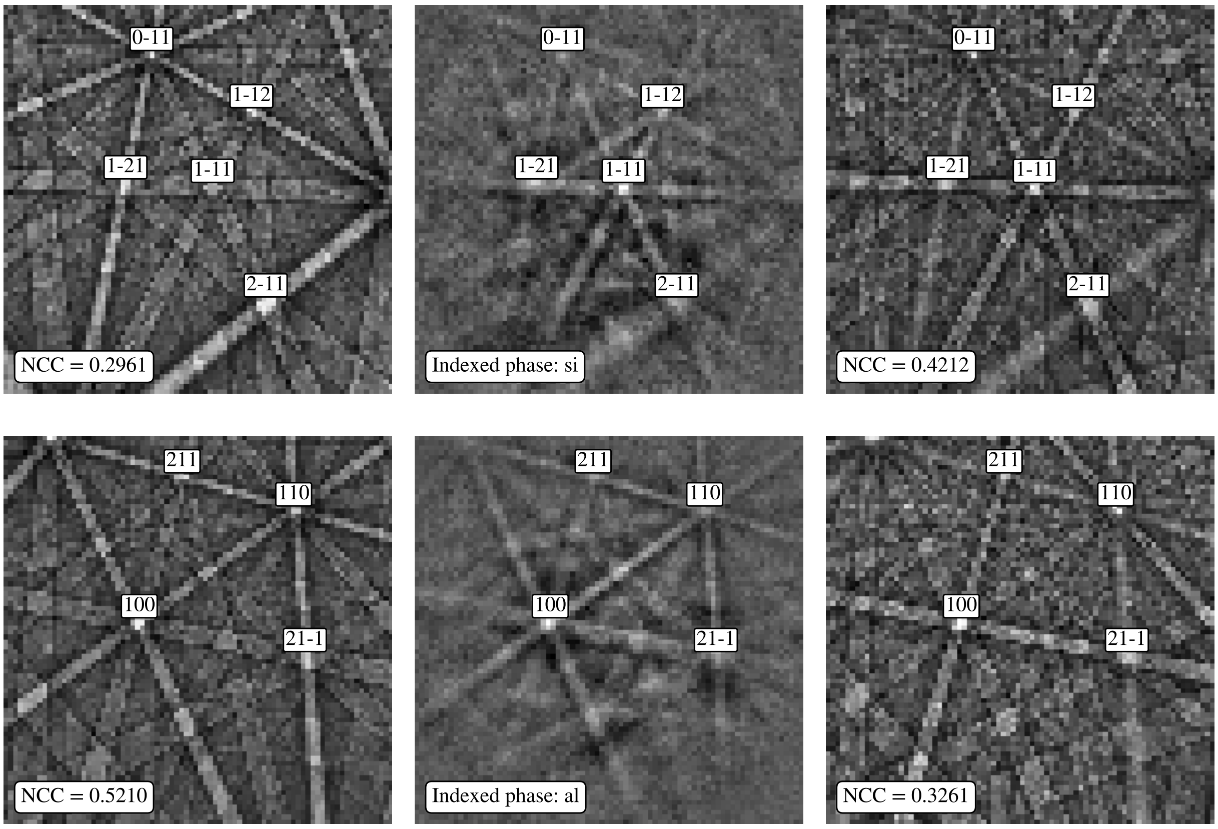


Figure 5.4: Comparison of two experimental patterns and the respective master pattern match for each phase. The simulated patterns for aluminium are given in the left column, the middle column holds the experimental EBSPs from two different locations, and the last column shows the simulated diffraction patterns for silicon. The upper EBSP is indexed as silicon and lower as aluminium.

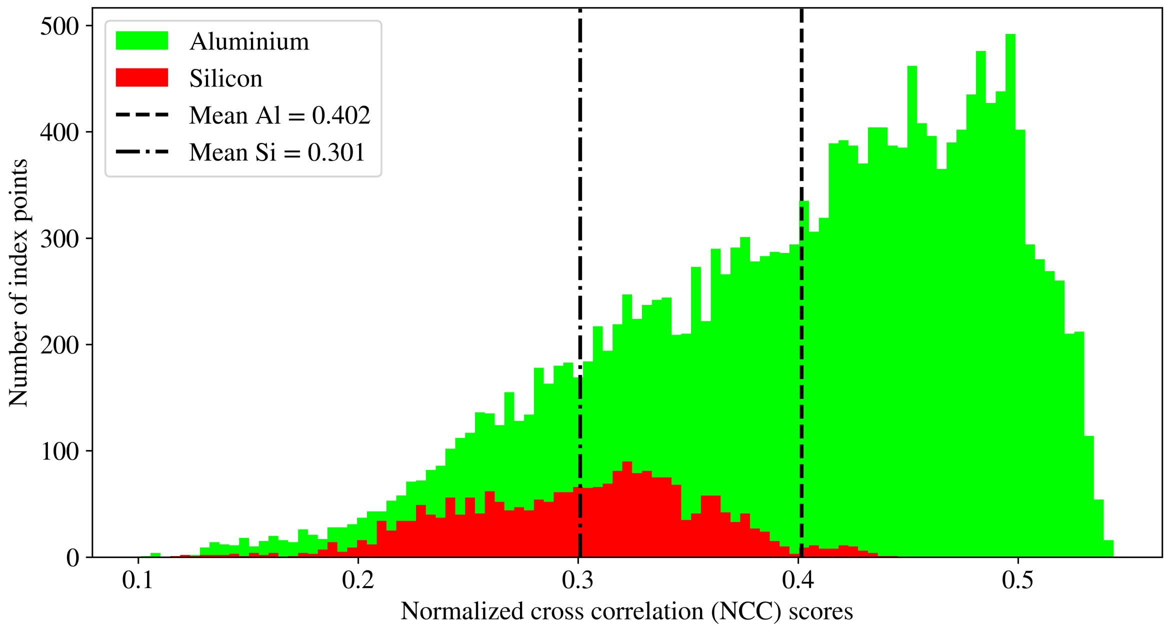


Figure 5.5: NCC histogram for aluminium and silicon. The mean NCC score is indicated for each phase.

5.2 Phase differentiation of several phases

To see how **EBSP Indexer** performs when indexing samples containing several phases, the DI routine in the program was run on an EBSD dataset from an SDSS sample with four phases. In addition to ferrite (δ) and austenite (γ), the χ and σ-phases had formed at grain boundaries.

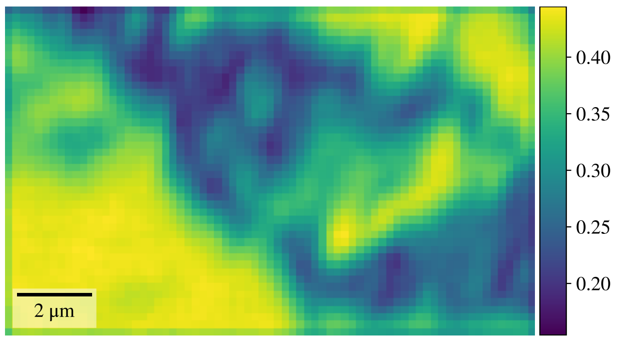


Figure 5.6: Image quality map from the processed EBSD dataset after removing the static and dynamic background and averaging with neighbouring patterns. The darker regions correspond to a lower pattern quality.

The dataset was acquired from a scan area of $14.2\mu m \times 8.8\mu m$ in a ZEISS ULTRA 55 FSEM with a NORDIF UF1100 EBSD detector. The acquisition pattern resolution was 120×120. Static and dynamic background subtraction and averaging with a Gaussian kernel with $\sigma = 1$ to improve the signal-to-noise ratio (SNR) was done with **EBSP Indexer**. An image quality (IQ) map of the processed dataset is provided in Figure 5.6. The indexing and orientation refinement parameters are summarized in Table 5.2 and the IPF map and phase map from DI are shown in Figure 5.7. The NCC map for the indexing result is found in Figure 5.8.

Table 5.2: Parameters for DI and orientation refinement.

PC (x, y, z)	(0.479, 0.788, 0.563)
Angular step size [°]	1.4°
Binning	None
Optimization algorithm	Nelder-Mead
Euler angle trust region	(1°, 1°, 1°)
$\mathrm{NCC}_{tolerance}$	1×10^{-4}

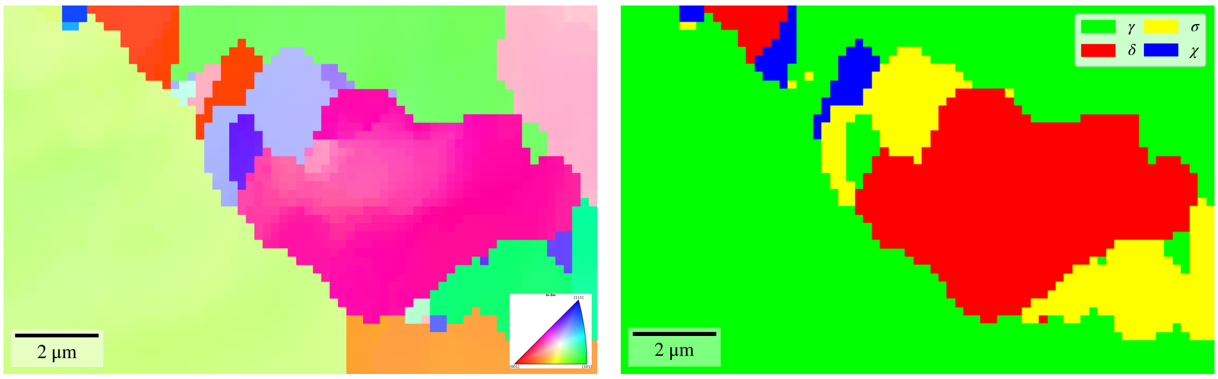


Figure 5.7: IPF map (left) and phase map for the SDSS with χ and σ precipitates. The phase fractions were determined to be: austenite (γ) 64.0%, ferrite (δ) 23.0%, σ-phase 10.8% and χ-phase 2.2 %.

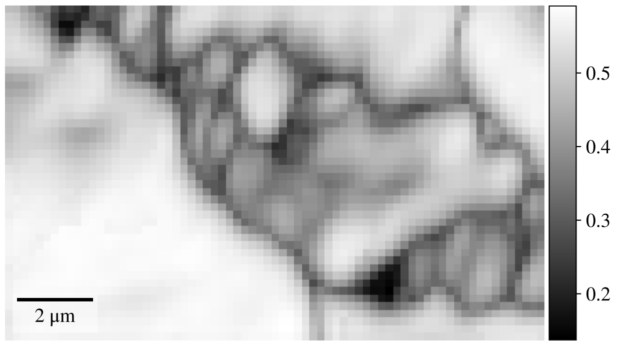


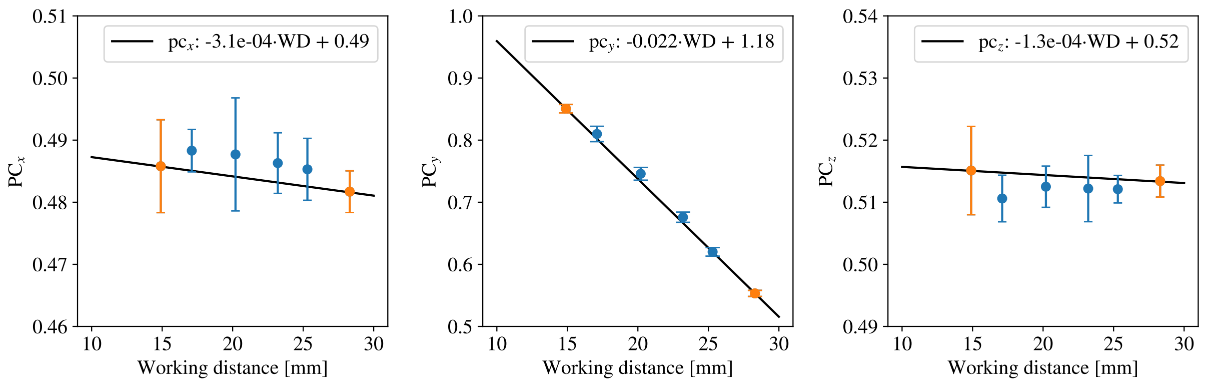
Figure 5.8: Normalized cross-correlation map for the SDSS sample.

5.3 Working distance calibration

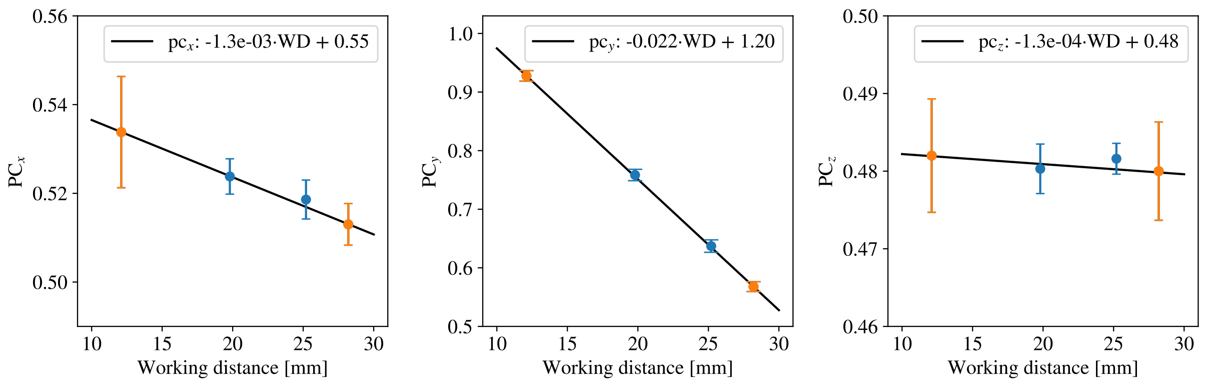
The calibration curves for two different microscope/EBSD detector setups are found in Figure 5.9. The pattern center for two working distances was determined from calibration patterns in **EBSP Indexer** for two different working distances. These values were then used for calibration using the **Working distance calibration**-tool. The input values for both microscope setups are given in Table 5.3. The resulting calibration curves with corresponding slope and interception point are given in Figure 5.9 for both microscope setups. The calibration curves for JEOL IT100 07050 setup shows that both the pc_x- and pc_z-coordinate stays relatively constant, whereas the $pc_y -$ coordinate changes with a slope of $-0.022\ \mathrm{mm}^{-1}$. For Zeiss SUPRA55 VP there is a slightly bigger change in both the pc_x and pc_z positions and the slope of the pc_y calibration curve is $-0.022\ \mathrm{mm}^{-1}$.

Table 5.3: PC coordinates for two working distances for two microscope/EBSD detector-setups used in PC calibration.

	WD [mm]	pc_x	pc_y	pc_z
JEOL IT100 07050				
Position 1	14.9	0.4858	0.8506	0.5151
Position 2	28.3	0.4817	0.5532	0.5134
ZEISS SUPRA55 VP				
Position 1	12.1	0.5338	0.9276	0.482
Position 2	28.2	0.513	0.5679	0.48



(a) JEOL IT100 07050



(b) ZEISS SUPRA55 VP

Figure 5.9: The calibration curves for two microscope/EBSD detector-setups. The orange points are the endpoints used for the calibration and the blue points correspond to the pattern center coordinate determined at intermediate positions. The error bars show the standard deviation of the mean value from the PC coordinated determined by using calibration patterns.

Chapter 6

Discussion

This chapter will discuss the results both from the development and application of the software. The first part will discuss the development and functionality of the software, evaluating the current implementation of the different modules, together with some ideas as to what could be done to further improve the software. The second part focuses on the results from different applications of the developed software.

6.1 Development and functionality

6.1.1 The dictionary indexing module

In the development of version 0.1.0 of **EBSP Indexer** the aim has been to provide the user with an indexing software that gives easy access to the dictionary indexing routine, meeting the basic needs when working with EBSD datasets and DI. This was to be done without flooding the user with options, and the number of parameters to be changed in the dialog window has been kept to a minimum. The current implementation does therefore not allow for much fine-tuning in the indexing routine by the user, and most settings are kept default, even though `kikuchipy` is inherently designed to offer a lot of room for flexibility through Python scripting. The current implementation provides easy access to DI, expanding the EBSD indexing toolbox for the average user.

With this said, as DI is a very computationally demanding indexing method [7], HI is still preferred when working with relatively straightforward samples, leaving DI as a tool for the characterization of more challenging and demanding materials. With more challenging cases one can argue that it should also be made more room for more flexibility in tuning the indexing parameters, to better match the full capability of `kikuchipy`, making the DI module more of an interactive tool for navigating the `kikuchipy` library, rather than a somewhat static shell using only some of the features available. This would require quite extensive development effort which there was not enough time for over the course of this project period.

Dictionary size preview

Since the size of the simulation dictionary is the main factor for how long the indexing routine will take, the preview of the size of the simulated dictionary was implemented to give the user an idea of how long the indexing task would take. It does not provide an exact time estimate, but it is easy for the user to see the effect of changing the angular step size. A bigger dictionary means a longer running indexing as every pattern is compared to every pattern in the simulation dictionary.

As of now the preview is only available for the space groups m-3m and 4/mmm by looking up the dictionary size for different angular resolutions hard-coded in the source code. It would certainly be beneficial to extend this functionality to all space groups by implementing a calculation that does not rely on hard-coded values. The preview should also be extended to provide an actual time estimate to the user for how long the indexing will run, as the preview today only provides information about the expected time relative to other angular step sizes. An actual time estimate is in general more useful, especially if the indexing routine is set to take several hours.

Orientation refinement

Refinement of orientation can today be selected directly from the DI dialog or as a separate step as described by Østvold in [42]. The refinement is one of the areas which would benefit from providing more flexibility in changing both the search region for orientations, which in the current implementation is limited to $\pm 1°$. The convergence limit could also be set as an optional value for tuning.

Handling of indexing results

In the current version of the DI implementation, all files from indexing are kept regardless, meaning that there can be quite a lot of files to navigate for the inexperienced user. The solution today is to store only the final crystal map in the main results directory and store all other files in the sub-directory `crystal_maps_data`, as shown in Figure 4.4. This could in future versions be improved by letting the user to a larger extent decide what files to keep, making the results directory less chaotic and easier to navigate for the user.

The user also have to decide prior to indexing what images to generate from the resulting crystal maps, and in the current version, there is no easy way of generating images post-indexing. This requires the user to load the indexed data in either other software like ATEX [43], or scripting using Python. In general, the software would certainly benefit from providing a more fully equipped imaging tool in order to generate high-quality figures suitable for scientific presentation.

6.1.2 Signal navigation

The signal navigation tool allows for easy investigation of EBSD datasets as well as the resulting crystal maps from either HI or DI. The implementation relies on two custom Python classes, `EBSDdataset` and `crystalMap` defined in `signal_loader.py`. The navigators used in the *Nav-*

igation view are only calculated when the user selects them in order to reduce memory usage, which proved itself to be especially important when dealing with large datasets, as some of the calculations take time and lead to the software freezing up during loading. This could be further improved by moving the larger calculations into parallel threads in the same manner as DI is run in a separate thread, this would improve the interaction with the software.

Geometrical simulations

The geometrical simulations are a convenient method to interactively verify the indexing results. The current implementation only shows the four strongest families of diffracting planes, which for most cubic phases is more than enough, but for eg. the tetragonal σ-phase this will for some orientations be insufficient to actually give a good verification of the orientation. The solution would be to implement a way of interactively changing the amount of diffracting planes used in the geometrical simulation, effectively overlaying more hkl-bands.

Region of interest selector

The current tool provides an easy method for cropping the EBSD dataset prior to indexing, this is especially useful to test the pattern center calibration and a set of indexing parameters on a smaller area prior to running indexing full datasets, as this often is a long-running task. The current selection tool only allows for cropping EBSD datasets prior to indexing but could be extended to also allow for cropping of the crystal maps generated from HI and DI as well, as this would be useful for cropping the data to focus on specific regions when presenting results.

6.1.3 Working distance calibration tool

The current implementation of the working distance pattern center calibration is functional and was implemented at the end of the development period as a proof-of-concept. The implementation still lacks some key features, like the possibility to change already existing calibrations. In the current version, all input data is lost, only keeping the interception point and slope for the lines for the x, y and z-coordinate. The functionality should be extended to save all data in a more convenient way for easier access for the user at a later point as well as allow for exporting and moving calibration data across devices.

The process of adding a new calibration requires quite a few steps and could be automated further. As of now, the working distance and pattern center coordinates for a single recording are stored separately, meaning that the user has to get the data from a minimum of four different text files to input the correct values. The first step to improve this would be to store the working distance and pattern center calibration value for a single recording in the same file, and secondly let the user automatically import these values by selecting the correct file. The option to manually input values should, of course, be kept as is, as users might use external software to calibrate the pattern center, but automatic reading of files is not just faster, but also reduces the room for mistakes and errors by the user.

6.1.4 User guide

The user guide provided in Appendix F is an extensive guide to performing EBSD analysis using **EBSP Indexer**. It is detailed, but it has been a goal to avoid letting it be too complicated. The guide has only been used by the student group during user testing and was revised before the final release of version 0.1.0. It will probably be necessary for additional revision of the content of the user guide as the software is distributed, and it would also have to be updated constantly alongside any further development.

6.2 Application of EBSP Indexer

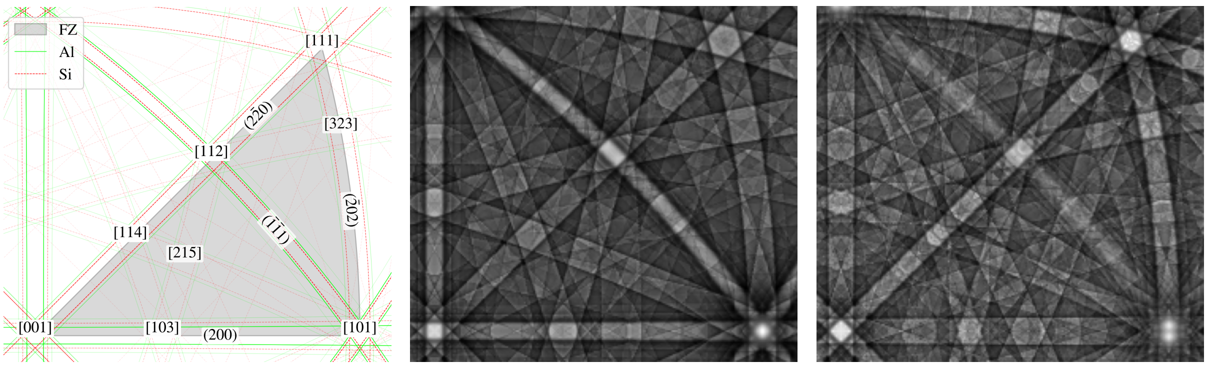
6.2.1 Al-10%Si

The results from indexing an Al-10%Si sample using both DI and HI demonstrate how DI can be used to reliably differentiate between phases belonging to very similar space groups, whereas conventional HI struggles to separate the two phases. The reason why conventional HI struggles can be explained by the diffracting planes for FCC (Fm-3m) and diamond structure (Fd-3m), and thus the position of the Kikuchi bands in the EBSPs, is almost exactly the same. By looking at the structure factor it can be shown that the only difference is that the diamond structure has an additional extinction rule for lattice planes where all *hkl*-indices are even and $h + k + l \neq 4N$. The strongest families of diffracting planes, sorted by $|F_{hkl}|$, for FCC and the diamond structure, are given in table 6.1. The plane families excluded by the extinction rules for the diamond structure are marked in ~~red~~. Since the difference between the two structures is just a few missing diffracting planes, the differentiation between aluminium and silicon cannot be based solely on the position of the Kikuchi bands.

Table 6.1: The strongest diffracting families of planes of the FCC and diamond structure. The planes are sorted by $|F_{hkl}|$ for aluminium. The forbidden planes according to the extinction rules for the diamond structure are marked in ~~red~~.

Crystal structure	Families of diffraction planes {hkl}								
FCC (Fm-3m)	{111}	{200}	{220}	{311}	{222}	{400}	{331}	{420}	{422}
Diamond (Fd-3m)	{111}	~~{200}~~	{220}	{311}	~~{222}~~	{400}	{331}	~~{420}~~	{422}

However, by investigating the master patterns for aluminium and silicon at 20 kV as well as a geometrical simulation of the Kikuchi bands for both phases given in Figure 6.1, it is clear that the intensity and width of the individual bands are different, allowing for differentiation between the two phases when this is taken into account. Comparing the master patterns the clearest difference between the two is the intensity of the bands making up the fundamental zone (FZ). The {200} and {111} bands are narrower and more intense for aluminium, and the {220} bands are narrower and more intense for silicon. This explains why DI is able to distinguish the two phases, which is further demonstrated in Figure 5.4 where the NCC-match of each master pattern is compared to two experimental patterns from the dataset. The NCC-score for the EBSP indexed as silicon is about 40% larger than the match with the aluminium master pattern, which can also be verified by a visual comparison of the patterns. For the EBSP indexed as aluminium, the NCC score is about 57% greater.



(a) (b) (c)

Figure 6.1: **(a)** Geometrical simulation of the Kikuchi bandwidths for aluminium and silicon. The fundamental zone is colored in grey, and the most intense zone axes [uvw] as well as the four most intense diffraction planes (hkl) for aluminium are labelled. **(b)** The master pattern of aluminium at 20 kV. **(c)** The master pattern of silicon at 20 kV.

The NCC-match distribution for each phase is given in Figure 5.5. The mean score value for aluminium is greater than for silicon, which is expected as the silicon particles are located at grain boundaries, where there is a greater overlap of EBSPs for different orientations, reducing the pattern quality and thus also the overall pattern match. The phase fraction determined by using DI is 89.4% Al and 10.6% Si. The scan area is too small to actually determine and confirm the phase composition of the sample, but gives, together with the NCC-histogram a good indication of the validity of the indexing results.

The combined use of conventional Hough indexing and EDS has been used in earlier publications to successfully differentiate between the two phases, as done in the analysis by C.-L. Chen and R. C. Thomson of an Al-12.5wt%Si-sample [44], the differentiation was possible as the grains were on the scale of several micrometres, well above the spatial resolution of EDS. In this demonstration, it is shown that DI is capable to differentiate and correctly identify Si particles on the size down to some hundred nanometers, which would not be possible with EDS analysis [6]. This approach also mitigates the need for an extra characterization step for phase differentiation.

6.2.2 Indexing of sample with four phases

The second application of the implemented dictionary indexer is the phase differentiation of an SDSS sample with σ- and χ-precipitates. This is primarily provided to show that the software scales well with the number of phases, as long as the user has access to the correct master pattern for the respective phases. The inverse pole figure map and phase map from indexing are given in Figure 5.7. The NCC-map in Figure 5.8 shows that the match overall is lower for the areas indexed as σ and χ, which is expected as pattern quality is lower in the corresponding areas shown in the image quality map in Figure 5.6. Other than a few singular pixels there is good agreement between the phase map and the different grains in the Inverse pole figure map.

6.2.3 Working distance calibration

As shown in figure 5.9, the calibration curves provide a reasonable guess for intermediate working distance positions. The blue dots correspond to the mean pattern center calculated using calibration patterns recorded at intermediate positions, and the mean value is well within the standard deviation for most positions, meaning that for many use cases, the initial value from the calibration curve may be used directly without any further calibration. As mentioned earlier, the use of a static mean pattern center is sufficient when working only with smaller area scans, and when working with more challenging indexing problems further pattern center optimization is necessary.

Chapter 7

Conclusions

With version 0.1.0 a fully functional and stable build of **EBSP Indexer** has been made available for use. The software has been tested on a wide range of datasets recorded on SEMs equipped with a NORDIF detector and has been tested on both Mac OS and Windows. The software allows for easy access to the Dictionary indexing (DI) routine, expanding the indexing toolbox for many users. The software also provides tools for cropping EBSD datasets as well as interactive investigation of the data both prior to and after indexing using either Hough indexing (HI) or DI. The investigation tool provides geometrical simulations which are overlaid the experimental patterns for visual confirmation of the indexing results. The DI module supports indexing all space groups as long as the user has access to the correct master pattern. A calibration tool for determining the pattern center (PC) from the working distance (WD) for a given SEM/EBSD detector configuration has been implemented and tested in the software, showing how this approach can be used for getting a very reasonable initial value of the PC suitable for indexing.

DI has been shown to successfully differentiate between the similar crystallographic space groups FCC (Fm-3m) and diamond structure (Fd-3m), allowing for direct phase differentiation at a sub-micron level, without the need for additional characterization methods.

Chapter 8

Future Work

- The DI implementation could be further improved by adding more flexibility in adjusting parameters as advanced functionality.
- A more advanced imaging tool should be developed, creating a more flexible tool for generating images suitable for scientific presentation.
- The signal navigator and the region of interest selector should be merged into a single tool.
- The calibration values for the SEM/EBSD detector setups should be stored in a more accessible way in order to make it easier to share the calibration across devices as well as future fine-tuning.
- The current HI module only supports the space groups BCC and FCC, but it has been announced that PyEBSDIndex will release a version with support for all space groups soon, and the software should be updated accordingly when this is available.
- DI is a computationally demanding indexing method and results in long-running indexing tasks, a combined method for indexing using both HI and DI should be implemented. HI can quickly index areas with high-quality patterns, and DI is then applied for areas where HI fails to provide confident results, like around grain boundaries.
- The discussion (Chapter 6) has provided some additional suggestions for improvements, which should be considered in future updates of the software.

Bibliography

[1] E. M. Østvold, 'EBSP Indexer - An open-source alternative to commercial EBSD software', M.S. thesis, Norwegian University of Science and Technology, Trondheim, 2023.

[2] O. Leth-Olsen, 'Distinguishing Martensite from Ferrite in Mild Steels with EBSD using Dictionary Indexing', M.S. thesis, Norwegian University of Science and Technology, Trondheim, 2023.

[3] A. J. Schwartz, *Electron backscatter diffraction in materials science*. Springer, 2009. DOI: https://doi.org/10.1007/978-0-387-88136-2.

[4] Y. Waseda *et al.*, 'Diffraction from Polycrystalline Samples and Determination of Crystal Structure', in *X-Ray Diffraction Crystallography: Introduction, Examples and Solved Problems*, Y. Waseda *et al.*, Eds., Springer, 2011, pp. 107–167. DOI: 10.1007/978-3-642-16635-8_4.

[5] N. C. K. Lassen, *Automated determination of crystal orientations from electron backscattering patterns* (IMM-PHD-1994-3). Technical University of Denmark, Sep. 1994.

[6] M. M. Nowell and S. I. Wright, 'Phase differentiation via combined EBSD and XEDS', *Journal of Microscopy*, vol. 213, no. 3, pp. 296–305, 2004, ISSN: 1365-2818. DOI: 10.1111/j.0022-2720.2004.01299.x.

[7] Y. H. Chen *et al.*, 'A Dictionary Approach to Electron Backscatter Diffraction Indexing', *Microscopy and Microanalysis*, vol. 21, no. 3, pp. 739–752, 2015. DOI: 10.1017/S1431927615000756.

[8] P. G. Callahan and M. De Graef, 'Dynamical Electron Backscatter Diffraction Patterns. Part I: Pattern Simulations', *Microscopy and Microanalysis*, vol. 19, no. 5, pp. 1255–1265, Oct. 2013. DOI: 10.1017/S1431927613001840.

[9] S. I. Wright *et al.*, 'Introduction and comparison of new EBSD post-processing methodologies', *Ultramicroscopy*, vol. 159, pp. 81–94, 2015. DOI: https://doi.org/10.1016/j.ultramic.2015.08.001.

[10] F. Ram and M. De Graef, 'Phase differentiation by electron backscatter diffraction using the dictionary indexing approach', *Acta Materialia*, vol. 144, pp. 352–364, Feb. 2018. DOI: 10.1016/j.actamat.2017.10.069.

[11] M. D. Graef *et al.*, *EMsoft-org/EMsoft: EMsoft Release 5.0.0*, Oct. 2019. DOI: 10.5281/zenodo.3489720. [Online]. Available: https://zenodo.org/record/3489720.

[12] H. W. Ånes *et al.*, *Pyxem/kikuchipy: Kikuchipy 0.8.4*, Apr. 2023. DOI: 10.5281/zenodo.7808659. [Online]. Available: https://doi.org/10.5281/zenodo.7808659.

[13] Í. Carneiro and S. Simões, 'Recent Advances in EBSD Characterization of Metals', *Metals*, vol. 10, no. 8, p. 1097, Aug. 2020. DOI: 10.3390/met10081097.

[14] S. Kikuchi, 'Diffraction of Cathode Rays by Mica', *Proceedings of the Imperial Academy*, vol. 4, no. 6, pp. 271–274, 1928. DOI: 10.2183/pjab1912.4.271.

[15] S. Nishikawa and S. Kikuchi, 'The Diffraction of Cathode Rays by Calcite', *Proceedings of the Imperial Academy*, vol. 4, no. 8, pp. 475–477, 1928. DOI: 10.2183/pjab1912.4.475.

[16] M. N. Alam *et al.*, 'High-Angle Kikuchi Patterns', *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 221, no. 1145, pp. 224–242, 1954. [Online]. Available: http://www.jstor.org/stable/100898 (visited on 17/04/2023).

[17] A. Winkelmann *et al.*, 'Many-beam dynamical simulation of electron backscatter diffraction patterns', *Ultramicroscopy*, vol. 107, no. 4, pp. 414–421, 2007. DOI: https://doi.org/10.1016/j.ultramic.2006.10.006.

[18] N. Krieger Lassen *et al.*, 'Automatic Recognition of Deformed and Recrystallized Regions in Partly Recrystallized Samples Using Electron Back Scattering Patterns', *Materials Science Forum*, vol. 157-162, pp. 149–158, May 1994. DOI: 10.4028/www.scientific.net/MSF.157-162.149.

[19] F. Ram *et al.*, 'Error analysis of the crystal orientations obtained by the dictionary approach to EBSD indexing', *Ultramicroscopy*, vol. 181, pp. 17–26, 2017. DOI: https://doi.org/10.1016/j.ultramic.2017.04.016.

[20] E. L. Pang *et al.*, 'Global optimization for accurate determination of EBSD pattern centers', *Ultramicroscopy*, vol. 209, p. 112 876, 2020. DOI: https://doi.org/10.1016/j.ultramic.2019.112876.

[21] S. Singh *et al.*, 'Application of forward models to crystal orientation refinement', *Journal of Applied Crystallography*, vol. 50, no. 6, pp. 1664–1676, Dec. 2017. DOI: 10.1107/S1600576717014200.

[22] A. J. Wilkinson *et al.*, 'Direct Detection of Electron Backscatter Diffraction Patterns', *Phys. Rev. Lett.*, vol. 111, no. 6, p. 065 506, Aug. 2013. DOI: 10.1103/PhysRevLett.111.065506.

[23] *EDAX Launches New Clarity Direct Electron Detector for EBSD*, 2020. [Online]. Available: https://www.edax.com/pressreleases/news/2020/may/edax-launches-new-clarity-direct-electron-detector-for-ebsd (visited on 17/04/2023).

[24] J. Hjelen *et al.*, 'Electron diffraction in the SEM', *Micron and Microscopica Acta*, vol. 22, no. 1, pp. 137–138, 1991. DOI: https://doi.org/10.1016/0739-6260(91)90128-M.

[25] S. Singh and M. D. Graef, 'Orientation sampling for dictionary-based diffraction pattern indexing methods', *Modelling and Simulation in Materials Science and Engineering*, vol. 24, no. 8, p. 085 013, Nov. 2016. DOI: 10.1088/0965-0393/24/8/085013.

[26] D. Roşca *et al.*, 'A new method of constructing a grid in the space of 3D rotations and its applications to texture analysis', *Modelling and Simulation in Materials Science and Engineering*, vol. 22, no. 7, p. 075 013, Oct. 2014. DOI: 10.1088/0965-0393/22/7/075013.

[27] H. W. Ånes *et al.*, *Pyxem/orix: Orix 0.10.2*, Oct. 2022. DOI: 10.5281/zenodo.7245549. [Online]. Available: https://doi.org/10.5281/zenodo.7245549.

[28] R. E. G. R. C. Woods, *Digital Image Processing (4th Global Edition)*, 4th. New York City, NY: Pearson, Mar. 2017.

[29] K. Kunze *et al.*, 'Advances in automatic EBSP single orientation measurements', *Textures and Microstructures*, vol. 20, pp. 41–54, 1993.

[30] S. I. Wright and B. L. Adams, 'Automatic analysis of electron backscatter diffraction patterns', *Metallurgical Transactions A*, vol. 23, no. 3, pp. 759–767, Mar. 1992. DOI: 10.1007/BF02675553.

[31] H. W. Ånes *et al.*, 'Processing and indexing of electron backscatter patterns using open-source software', *IOP Conference Series: Materials Science and Engineering*, vol. 891, no. 1, p. 012 002, Jul. 2020. DOI: 10.1088/1757-899X/891/1/012002.

[32] M. D. Graef, 'A dictionary indexing approach for EBSD', *IOP Conference Series: Materials Science and Engineering*, vol. 891, no. 1, p. 012 009, Jul. 2020. DOI: 10.1088/1757-899X/891/1/012009.

[33] K. Z. Baba-Kishi, 'Electron backscatter Kikuchi diffraction in the scanning electron microscope for crystallographic analysis', *Journal of Materials Science*, vol. 37, no. 9, pp. 1715–1746, 2002, ISSN: 0022-2461. DOI: 10.1023/A:1014964916670.

[34] K. Z. Baba-Kishi, 'Measurement of crystal parameters on backscatter kikuchi diffraction patterns', *Scanning*, vol. 20, no. 2, pp. 117–127, 1998. DOI: 10.1002/sca.1998.4950200210.

[35] J. A. Small and J. R. Michael, 'Phase identification of individual crystalline particles by electron backscatter diffraction', *Journal of Microscopy*, vol. 201, no. 1, pp. 59–69, 2001. DOI: 10.1046/j.1365-2818.2001.00788.x.

[36] A. K. Dahle *et al.*, 'Eutectic nucleation and growth in hypoeutectic Al-Si alloys at different strontium levels', *Metallurgical and Materials Transactions A*, vol. 32, no. 4, pp. 949–960, Apr. 2001. DOI: 10.1007/s11661-001-0352-y.

[37] D. Chen *et al.*, 'Effect of microscopic parameters on EBSD spatial resolution', *Ultramicroscopy*, vol. 111, no. 9, pp. 1488–1494, Aug. 2011. DOI: 10.1016/j.ultramic.2011.06.007.

[38] S. I. Wright *et al.*, 'Electron imaging with an EBSD detector', *Ultramicroscopy*, vol. 148, pp. 132–145, 2015. DOI: https://doi.org/10.1016/j.ultramic.2014.10.002.

[39] F. d. l. Peña *et al.*, *Hyperspy/hyperspy: Release v1.7.3*, Oct. 2022. DOI: 10.5281/zenodo.7263263. [Online]. Available: https://zenodo.org/record/7263263.

[40] D. Rowenhorst and H. W. Ånes, *Usnavalresearchlaboratory/PyEBSDIndex: PyEBSDIndex 0.1.2*, May 2023. [Online]. Available: https://pyebsdindex.readthedocs.io/en/stable/index.html.

[41] D. Johnstone *et al.*, *Pyxem/diffsims: Diffsims 0.5.2*, May 2023. DOI: 10.5281/zenodo.7962969. [Online]. Available: https://zenodo.org/record/7962969.

[42] E. M. Østvold *et al.*, *EBSP Indexer*, May 2023. DOI: 10.5281/zenodo.7925262. [Online]. Available: https://doi.org/10.5281/zenodo.7925262.

[43] B. Beausir and J.-J. Fundenberger, *Analysis Tools for Electron and X-ray diffraction: ATEX - software*, 2017. [Online]. Available: www.atex-software.eu.

[44] C.-L. Chen and R. C. Thomson, 'The combined use of EBSD and EDX analyses for the identification of complex intermetallic phases in multicomponent Al–Si piston alloys', *Journal of Alloys and Compounds*, vol. 490, no. 1, pp. 293–300, Feb. 2010. DOI: 10.1016/j.jallcom.2009.09.181.

Appendix A

dictionary_indexing.py

Index

Classes

Module **`dictionary_indexing`**

Classes

`class DiSetupDialog (parent=None, pattern_path: str = None)`

QDialog(self, parent: Optional[PySide6.QtWidgets.QWidget] = None, f: PySide6.QtCore.Qt.WindowFlags = Default(Qt.WindowFlags)) -> None

Dialog box for setting up dictionary indexing parameters.

Parameters

`parent` : `QWidget` , optional
Parent widget, by default None

`pattern_path` : `str` , optional
Path to the pattern file, by default None

Ancestors

PySide6.QtWidgets.QDialog, PySide6.QtWidgets.QWidget, PySide6.QtCore.QObject, PySide6.QtGui.QPaintDevice, Shiboken.Object

Class variables

`var staticMetaObject`

Methods

`def addPhase(self)`

Adds a phase to self.phaseList. The function is triggered by the add phase button. A phase is added from the master pattern .h5 file that is selected in the file browser.

The phase is added to the phase list and the phase table is updated.

`def checkConfirmState(self)`

Checks the state of the confirm button and enables/disables it depending on the state of the phase list.

```
def dictionary_indexing(self,
                        ebsd: kikuchipy.signals.ebsd.EBSD | kikuchipy.signals.ebsd.Laz
```

yEBSD)

Main function for dictionary indexing. The indexing parameters are stored in the options dictionary which are read from the ui-elements in the GUI.

Parameters

`ebsd` : `EBSD | LazyEBSD`
: The EBSD pattern containing experimental EBSPs used for refinement.

Returns

`merged` : `CrystalMap`
: class orix.crystal_maps.CrystalMap

Notes

The function will save different files depending on the number of phases in the sample and if refinement is performed or not.

For single phases without refinement, the function will save the following files to the results directory: - xmap: The crystal map from the phase For single phases with refinement, the function will save the following files to the results directory: - xmap_refined: The crystal map from the phase after refinement And the following files to the raw data directory: - xmap: The crystal map from the phase

For multi-phase samples without refinement, the function will save the following files to the results directory: - xmap: The merged crystal map from all phases And the following files to the raw data directory: - xmap_: The crystal map for each phase

For multi-phase samples with refinement, the function will save the following files to the results directory: - xmap_refined: The merged crystal map from all phases after refinement And the following files to the raw data directory: - xmap: The merged crystal map from all phases - xmap_: The crystal map for each phase - xmap_refined_: The crystal map for each phase after refinement

If the phase contains no inversion symmetry, both hemispheres are loaded for the master pattern.

`def estimateSimulationSize(self)`

Estimates the size of the simulation dictionary based on the number of phases and the angular step size by looking up the number of rotations for each phase in the SAMPLE_ROTATIONS dictionary and summing them up.

Notes

The estimation is only done if all phases belong to are either the CUBIC point group m-3m or the TETRAGONAL point-group 4/mmm. In any other case, the number of simulations is set to N/A.

```
def getOptions(self) -> dict
```

Collects all values from user input and stores them in a dictionary.

Returns

`options` : `dict`
dictionary storing all relevant parameters for indexing

```
def load_master_patterns(self) -> dict
```

Loads a master pattern for each phase in self.phaseList. The master patterns are stored in a dictionary with the phase name as key and the master pattern as value.

Returns

`mp_dict` : `dict`
dictionary storing the master pattern for each phase

```
def merge_crystal_maps(self, xmap_dict: dict, save_dir: str)
                       -> orix.crystal_map.crystal_map.CrystalMap
```

Takes in a dictionary with crystal maps and merges them in to one crystal map based on the matching score from dictionary indexing

Parameters

`xmap_dict` : `dict`
dictionary with crystal maps

Returns

`merged` : `CrystalMap`
class orix.crystal_maps.CrystalMap

```
def refine_orientations(self, xmaps: dict,
                        detector: kikuchipy.detectors.ebsd_detector.EBSDDetector,
                        ebsd_pattern: kikuchipy.signals.ebsd.EBSD | kikuchipy.signals.
                        ebsd.LazyEBSD)
```

Takes in a dictionary with crystal maps and performs orientation refinement on all crystal maps.

Saves individual crystal maps t file, both refined and unrefined crystal maps are stored.

Parameters

xmaps : `dict`
: Dictionary containing crystal maps from Dictionary Indexing.

detector : `EBSDDetector`
: An EBSD detector class storing the shape, pixel size, binning factor, detector tilt, sample tilt and projection center (PC).

ebsd_pattern : `EBSD | LazyEBSD`
: The EBSD pattern containing experimental EBSPs used for refinement.

Returns

xmaps_ref : `dict`
: Dictionary containing refined crystal maps.

```
def removePhase(self)
```

Removes a phase from self.phaseList. The function is triggered by the remove phase button. The phase is removed from the phase list and the phase table is updated.

```
def run_dictionary_indexing(self)
```

Starts the dictionary indexing in a separate thread. The function is triggered by the confirm button.

The results are stored in a folder in the working directory. The folder is named di_results_1, di_results_2, etc. depending on the number of results folders already present in the working directory.

The function calls the dictionary_indexing function in a separate thread. The following parameters are passed to the dictionary_indexing function: ebsd_pattern : EBSD | LazyEBSD The EBSD dataset to be indexed

```
def save_di_settings(self, xmap: orix.crystal_map.crystal_map.CrystalMap,
                     ebsd: kikuchipy.signals.ebsd.EBSD | kikuchipy.signals.ebsd.LazyEB
                     SD, original_sig_shape: tuple)
```

The dictionary indexing settings are saved in a text file in the results directory.

Parameters

xmap : `CrystalMap`
: Crystal map

ebsd : `EBSD | LazyEBSD`
: EBSD data

original_sig_shape : `tuple`
: Original signal shape

Returns

None

```
def save_inverse_pole_figure(self,
                             crystal_map: orix.crystal_map.crystal_map.CrystalMap)
```

Saves an inverse pole figure (IPF) map as a .png image for a given crystal map.

Parameters

`crystal_map` : `CrystalMap`
The crystal map for which the IPF map is generated.

Returns

None

```
def save_ncc_figure(self, crystal_map: orix.crystal_map.crystal_map.CrystalMap,
                    filename: str)
```

Saves a normalized cross-correlation (NCC) map as a .png image for a given crystal map.

The NCC map represents the correlation scores between the experimental pattern and the best fit simulated pattern. The map is saved as a grayscale image.

Parameters

`crystal_map` : `CrystalMap`
The crystal map for which the NCC map is generated.

`filename` : `str`
The filename specifying the type of NCC map. It indicates whether the map is refined, unrefined, or merged.

Returns

None

Notes

- For crystal maps with multiple rotations per point, the NCC scores are accessed from the first column of the scores array.
- For crystal maps with a single rotation per point, the NCC scores are accessed using the `get_map_data` method.

```
def save_orientation_similarity_map(self,
                                    crystal_map: orix.crystal_map.crystal_map.CrystalM
                                    ap, filedir: str, filename: str)
```

Saves an orientation similarity map (OSM) as a .png image for a given crystal map. The OSM values are accessed from the `orientation_similarity_map` method.

Parameters

crystal_map : `CrystalMap`
The crystal map for which the OSM is generated.

filedir : `str`
The directory where the OSM is saved.

filename : `str`
The filename specifying the type of OSM. It indicates whether the map is from a single-phase or multi-phase crystal map.

Returns

`None`

Notes

```
def save_project_settings(self)
```

The project settings are saved in a text file in the project directory.

```
def setNiterState(self)
```

Enables/disables the number of iterations spinbox and label depending on the state of the lazy checkbox.

```
def setupBinningShapes(self,
                       ebsd_signal: kikuchipy.signals.ebsd.EBSD | kikuchipy.signals.eb
                       sd.LazyEBSD)
```

Calculates the binning shapes that are possible for the EBSD dataset and adds them to the binning combo-box. The calculation is done using the modulo (%) operator on the signal shape.

Parameters

ebsd_signal : `EBSD | LazyEBSD`
The EBSD dataset to calculate the binning shapes for.

`def setupConnections(self)`

Setup connections for the dialog box buttons and widgets.

`def setupInitialSettings(self)`

Setup initial settings for the dialog box based on the current settings in project_settings.txt and advanced_settings.txt.

Notes

If the pattern center is not set in project_settings.txt, the pattern center will be calculated from the working distance if a calibration curve for the acquisition instrument is available. Otherwise, the pattern center is set to (0.5, 0.8, 0.5).

`def signal_mask(self, sig_shape: tuple)`

Adds a circular signal mask to the dictionary indexing parameters if the mask option is checked.

Parameters

`sig_shape` : `tuple`
: shape of the EBSD signal

`def updatePhaseTable(self)`

Updates the phase table ith the following information about the phase stored in self.phaseList.

Fill out relevant phase information for phases in self.phaseList.

Columns

NAME_COL = 0 NUMBER_COL = 1 ISS_COL = 2 CRYSTAL_COL = 3 COLOR_COL = 4

Appendix B

signal_loader.py

Index

Classes

Generated by *pdoc* 0.10.0.

Module **`signal_loader`**

Classes

`class EBSDDataset (dataset)`

A custom EBSDDataset class storing the EBSD, dataset type, navigation shape and navigators.

Parameters

`dataset` : `EBSD | LazyEBSD`
ebsd dataset

Notes

Available navigators for EBSD datasets are the Mean intensity map (default), Image quality map and Virtual BSE map.

Instance variables

`var image_quality`

Image quality (IQ) map

`var mean_intensity`

Mean intensity map

`var virtual_bse`

Virtual BSE (VBSE) image

Methods

`def compute_navigator(self, nav_num: int = 0)`

Computes a navigator to be used in the navigator view in the signal navigation widget.

Parameters

`nav_num` : `int`
The navigator id

Returns

`navigator` : `numpy.ndarray`

Returns the selected navigator.

Notes

Navigator ids:

0 = inverse pole figure, 1 = image quality, 2 = virtual bse image

```
class crystalMap (dataset: orix.crystal_map.crystal_map.CrystalMap, crystal_map_path: str,
              compute_all: bool = False)
```

A custom crystalMap class storing the CrystalMap, dataset type, a phase id array, navigation shape, corresponding EBSD dataset, navigators and Miller indices for diffraction planes used in geometrical simulation.

Parameters

`dataset` : `orix.crystal_map.CrystalMap`
: Crystal map containing the phases and unit cell rotations, generated from either Hough or Dictionary indexing.

`crystal_map_path` : `str`
: The crystal map file path

Notes

Depending on the crystal map different navigators are calculated.

Hough indexing:

- Single phase: Inverse pole figure
- Multi-phase: Inverse pole figure, Phase map

Dictionary indexing:

- Single phase: Inverse pole figure, Normalized cross-correlation map, Orientation similarity metric map
- Single phase with refined orientations: Inverse pole figure, Normalized cross-correlation map'
- Multi-phase: Inverse pole figure, Phase map, Normalized cross-correlation map, Orientation similarity metric map
- Multi-phase with refined orientations: Inverse pole figure, Phase map, Normalized cross-correlation map

Instance variables

`var inverse_pole_figure`

Inverse pole figure map. Used as default navigator for crystal maps.

`var normalized_cross_correlation_map`

Normalized cross-correlation for crystal maps from dictionary indexing.

`var orientation_simliarity_metric`

Orientation similarity metric for unrefined crystal maps from dictionary indexing.

`var phase_map`

Phase map for crystal maps with more than one phase.

Methods

`def compute_navigator(self, nav_num: int = 0) -> numpy.ndarray`

Computes a navigator to be used in the navigator view in the signal navigation widget.

Parameters

`nav_num` : `int`
The navigator id

Returns

`navigator` : `numpy.ndarray`
Returns the selected navigator.

Notes

Navigator ids:

0 = inverse pole figure, 1 = phase map, 2 = ncc_map, 3 = osm_map

`def detector(self, crystal_map_path) -> kikuchipy.detectors.ebsd_detector.EBSDDetector`

Loads the detector parameters stored together with the CrystalMap. The detector parameters are stored in the file detector.txt which is stored together with the crystal map.

Parameters

`crystal_map_path` : `str`
Crystal map file path.

Returns

`detector` : `kikuchipy.detectors.ebsd_detector.EBSDDetector`
An EBSD detector class storing its shape, pixel size, binning factor, detector tilt, sample tilt and projection center (PC).

```
def ebsd_signal(self, crystal_map_path: str) -> kikuchipy.signals.ebsd.LazyEBSD
```

Loads the corresponding EBSD dataset which the crystal map was generated from. The crystal_map_path is used to get the name of the EBSD dataset from indexing_parameters.txt stored together with the crystal map.

Parameters

`crystal_map_path` : `str`
: The crystal map file path

Returns

`ebsd_signal` : `LazyEBSD`
: Scan of Electron Backscatter Diffraction (EBSD) patterns

```
def hkl_simulation(self) -> list
```

Returns a list of Miller indices corresponding to the four strongest diffracting lattice planes for each phase in the CrystalMap.

Returns

`hkl_simulations` : `list`
: A nested list of Miller indices

Notes

Calculations of reciprocal lattice vectors and structure factors are done using `diffsim`.

Appendix C

signal_navigation_widget.py

Index

Classes

Generated by *pdoc* 0.10.0.

Module **`signal_navigation_widget`**

Classes

`class SignalNavigationWidget (mainWindow: PySide6.QtWidgets.QMainWindow)`

QWidget(self, parent: Optional[PySide6.QtWidgets.QWidget] = None, f: PySide6.QtCore.Qt.WindowFlags = Default(Qt.WindowFlags)) -> None

The signal navigation widget.

Ancestors

PySide6.QtWidgets.QWidget, PySide6.QtCore.QObject, PySide6.QtGui.QPaintDevice, Shiboken.Object

Class variables

`var staticMetaObject`

Methods

`def export_image(self, navigator_index: int, signal)`

Exports the current view and saves it as a .png image.

The user is presented with a dialog where directory and filename is set.

`def load_dataset(self, file_path: str)`

Takes in the path for the dataset.

The type of dataset is determined by a try/except-statement and the dataset is loaded by the correct signal loader.

EBSD is loaded using the kikuchipy loader, CrystalMap is loaded using the orix.io loader.

The dataset is passed as an argument to either EBSDDataset or crystalMap depending on the signal

The dataset object is passed to the plot_navigator function.

Parameters

`file_path` : `str`
file path to the dataset

```
def on_click_navigator(self, event,
                       dataset: scripts.signal_loader.EBSDDataset | scripts.signal_loa
                       der.crystalMap)
```

Trigger function when user clicks within the navigator view area.

```
def on_hover_navigator(self, event)
```

Trigger function when user hovers inside the navigator area.

```
def plot_navigator(self,
                   dataset: scripts.signal_loader.EBSDDataset | scripts.signal_loader.
                   crystalMap, nav_type: str, x: int = 0, y: int = 0)
```

Plots the selected navigator to the navigation view.

Parameters

dataset : `EBSDDataset | crystalMap`
  custom class EBSDDataset or crystalMap from signal_loader

nav_type : `str`
  EBSDDataset navigators: Mean intensity map, Image quality map, Virtual BSE map crystalMap navigators: Inverse pole figure, Phase map, Normalized cross-correlation map and Orientation similarity metric

x : `int`
  x index of dataset, by default set to 0

y : `int`
  y-index of dataset, by default set to 0

```
def plot_signal(self,
                dataset: scripts.signal_loader.EBSDDataset | scripts.signal_loader.cry
                stalMap, x_index: int, y_index: int)
```

Plots the EBSP for the current x_index, y_index to the signal view.

If the dataset is a crystalMap and self.add_geosim = True, a geometrical simulation is plotted.

Parameters

dataset : `EBSDDataset | crystalMap`
  custom class EBSDDataset or crystalMap from signal_loader

x_index : `int`
  x index of ebsd signal

`y_index` : `int`

y-index of ebsd signal

```
def setupConnection(self)
```

```
def showStatusTip(self, tip: str)
```

Appendix D

region_of_interest.py

Index

Classes

Generated by *pdoc* 0.10.0.

Module **`region_of_interest`**

Classes

`class RegionOfInteresDialog (parent, pattern_path=None)`

QDialog(self, parent: Optional[PySide6.QtWidgets.QWidget] = None, f: PySide6.QtCore.Qt.WindowFlags = Default(Qt.WindowFlags)) -> None

Region of interest dialog window. The user can select a rectangular region of interest in the navigator image. The selected region is saved as a new EBSD file.

Notes

The navigators are calculated from signal_loader.EBSDDataset class and the navigator to plot is read from the comboBoxNavigator.

Ancestors

PySide6.QtWidgets.QDialog, PySide6.QtWidgets.QWidget, PySide6.QtCore.QObject, PySide6.QtGui.QPaintDevice, Shiboken.Object

Class variables

`var staticMetaObject`

Methods

`def getSelection(self) -> dict`

Return dictionary with x0, x1, y0, y1 values

`def line_select_callback(self, eclick, erelease)`

Callback for line selection.

eclick and *erelease* are the press and release events.

`def on_click(self, event)`

Store click location when mouse is clicked

`def plot_navigator(self)`

Plot navigator image in a matplotlib widget. The navigator to plot is read from the comboBoxNavigator.

```
def run_roi_selection(self)
```

Save the selected ROI to a new file.

Note: The ROI is saved as a new file, the original file is not modified.

```
def setSavePath(self)
```

Set save path for processed pattern

```
def setupConnections(self)
```

Setup connections for all signals

```
def spinBoxBlockSignal(self, block: bool)
```

Block the signals of the spinboxes to prevent the callback function from being called when spinbox values are changed from the interactive rectangle selection.

```
def update_selection(self)
```

Update selection rectangle when spinbox values are changed

Appendix E

pc_from_wd.py

Index

Generated by *pdoc* 0.10.0.

Module **`pc_from_wd`**

Functions

`def pc_from_wd(microscope: str, working_distance: float, convention='TSL')`

Returns the pattern center as a numpy array for a given working distance and microscope. If there is no available calibration curve for the microscope, the default values (0.5, 0.8, 0.5) are returned.

Returns

`pc` : `numpy array`

Pattern center as a numpy array.

Notes

If the convention is set to "BRUKER", the y-coordinate is inverted.

Classes

`class wdCalibration (parent=None)`

QDialog(self, parent: Optional[PySide6.QtWidgets.QWidget] = None, f: PySide6.QtCore.Qt.WindowFlags = Default(Qt.WindowFlags)) -> None

Dialog box for adding a PC-WD calibration for a microscope/EBSD setup. The calibration is a linear interpolation between the two points and the slope and intercept for pc_x, pc_y and pc_z are stored in the advances_settings.txt with the microscope name as the key.

Ancestors

PySide6.QtWidgets.QDialog, PySide6.QtWidgets.QWidget, PySide6.QtCore.QObject, PySide6.QtGui.QPaintDevice, Shiboken.Object

Class variables

`var staticMetaObject`

Methods

`def addRow(self)`

Adds a new row to the table of calibration values.

`def checkValidInput(self)`

Checks if the input is valid.

The input is valid if the table contains only numbers and the microscope name is not empty.

`def get_microscope_name(self)`

Gets the microscope name from a text file. The file should contain the keys 'Manufacturer' and 'Model' and the values should be separated by a tab.

Notes

The microscope name is used as the key in the advanced_settings.txt file.

`def removeRow(self)`

Removes the selected row from the table of calibration values.

`def run_calibration(self)`

Runs the calibration and saves the calibration values to the advanced settings file. The calibration values are saved as a tuple of tuples with microscope name as the identifier.

`def setupConnections(self)`

Sets up the connections between the ui elements and the functions.

Appendix F

User guide - EBSP Indexer v0.10

EBSP INDEXER

User guide for version 0.1.0

Developers:
Erlend Mikkelsen Østvold
Hallvard Tangvik Tellefsen Relling
Olav Leth-Olsen

Contents

Acronyms

BCC Body-centered cubic. 8, 10, 13

DI Dictionary indexing. 2, 13, 15–17

EBSD Electron Backscatter Diffraction. 2, 4–6, 11, 13, 17

EBSP Electron Backscatter pattern. 4, 5, 10, 13, 15, 17

FCC Face-centered cubic. 8, 10, 13

HI Hough indexing. 2, 13–15, 17

IPF inverse pole figure. 17

MP master pattern. 8, 10, 14–16

PC Pattern center. 8–12, 14, 16

SDSS Super duplex stainless steel. 8

SEM Scanning Electron Microscopy. 2

SNR signal-to-noise ratio. 10

WD working distance. 8, 9, 11

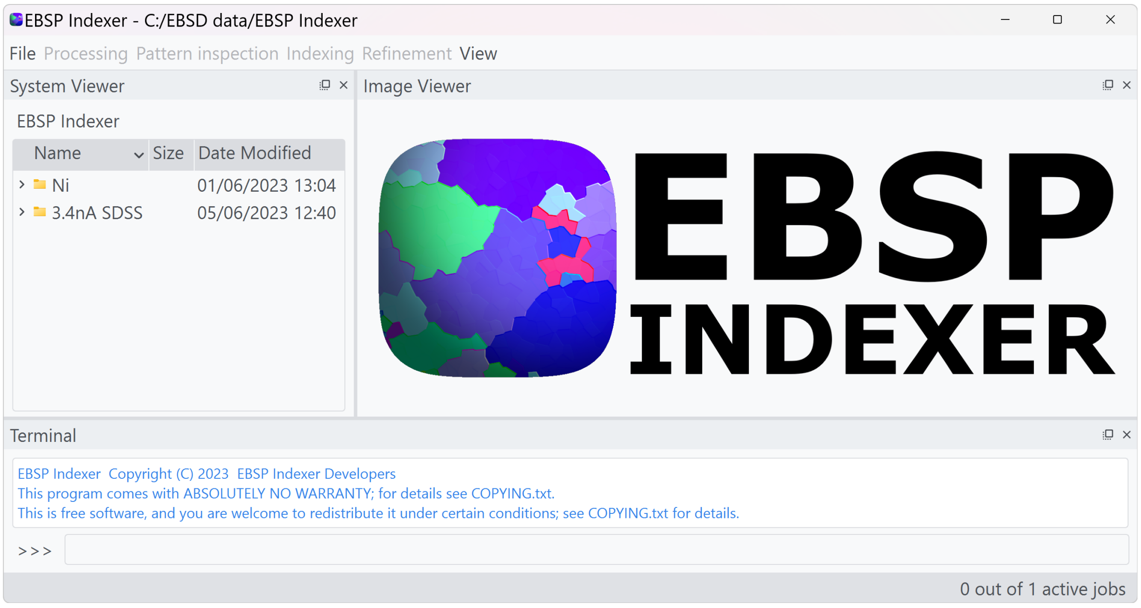
EBSP Indexer

EBSP Indexer[1][1] is an open-source program for studying EBSD datasets recorded in a SEM. The software is developed by Erlend Mikkelsen Østvold [2], Hallvard Tangvik Tellefsen Relling [3] and Olav Leth-Olsen [4] as part of their master thesis projects supervised by Prof. Jarle Hjelen. The software is built around the python library `kikuchipy` and [5] allows the user to investigate EBSD datasets, perform necessary pattern processing and setup calibration, and supports both Hough indexing (HI) and Dictionary indexing (DI). This guide is made for EBSD datasets acquired with SEMs equipped with a NORDIF EBSD detector, and the software has not been tested on datasets from other vendors to date.

[1]The full source code is available on Github

Loading and inspecting an EBSD dataset

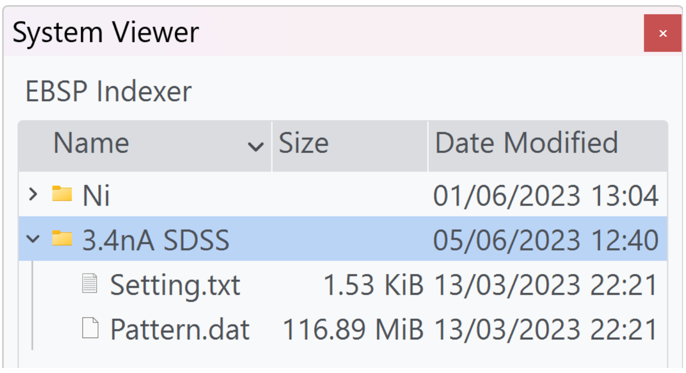
When opening the application, the main window is displayed as shown below. It consists of a *System viewer* for keeping track of files, an *Image viewer* for opening images and a *Terminal*, a Python interpreter with full access to `kikuchpy`.


EBSP Indexer - C:/EBSD data/EBSP Indexer
File Processing Pattern inspection Indexing Refinement View
System Viewer
Image Viewer
EBSP Indexer
Terminal
0 out of 1 active jobs

Name	Size	Date Modified
Ni		01/06/2023 13:04
3.4nA SDSS		05/06/2023 12:40

```
EBSP Indexer  Copyright (C) 2023  EBSP Indexer Developers
This program comes with ABSOLUTELY NO WARRANTY; for details see COPYING.txt.
This is free software, and you are welcome to redistribute it under certain conditions; see COPYING.txt for details.
>>>
```

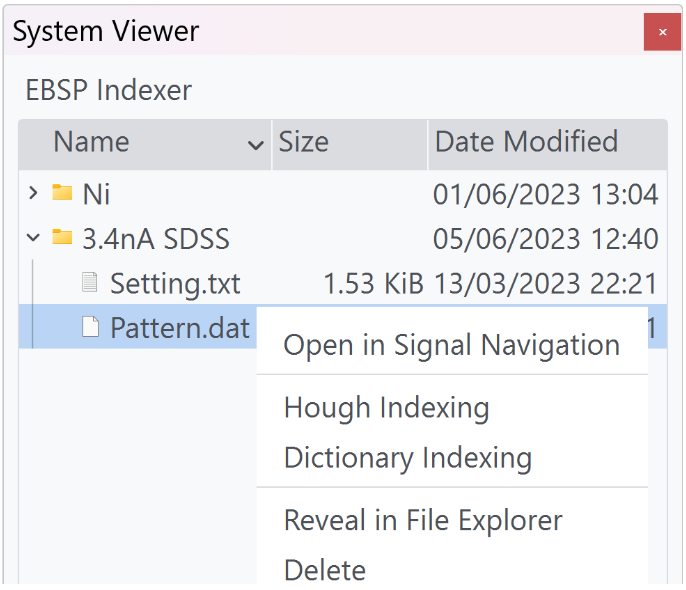
To open an EBSD recording go to **File** ⟶ **Open Workfolder** and select the folder where the dataset is stored. The folder will appear in the *System Viewer* to the left and look something like this:



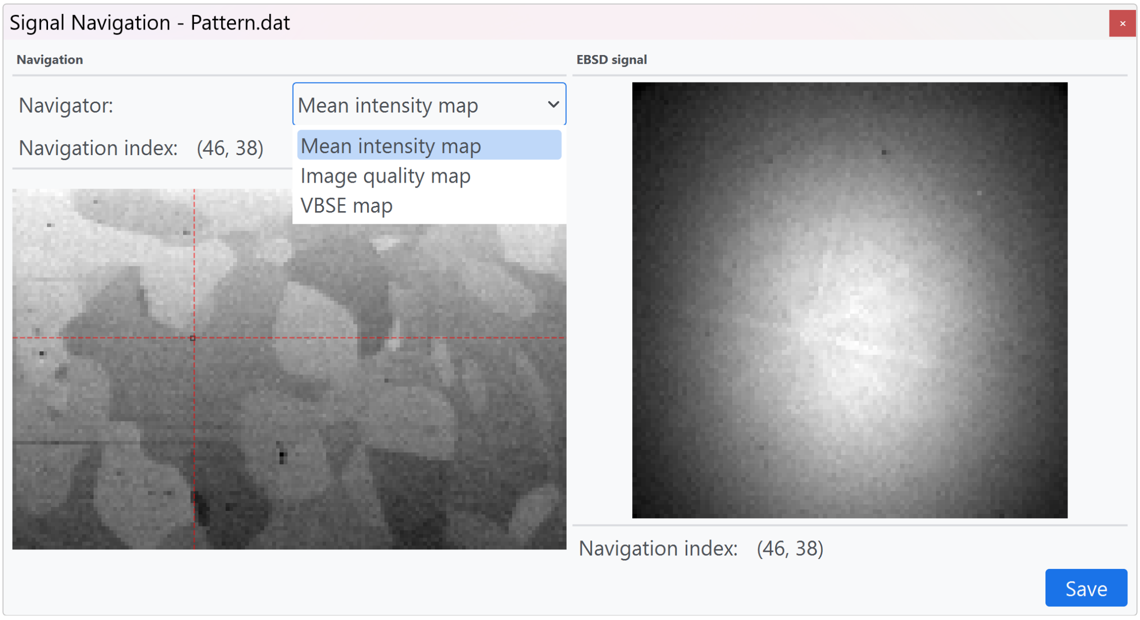
Name	Size	Date Modified
Ni		01/06/2023 13:04
3.4nA SDSS		05/06/2023 12:40
Setting.txt	1.53 KiB	13/03/2023 22:21
Pattern.dat	116.89 MiB	13/03/2023 22:21

1.1 Signal navigation - EBSD Dataset

The EBSD patterns are stored in the file called Pattern.dat, which is the NORDIF binary file format for storing EBSD patterns. To investigate the raw patterns, right-click on the file containing the patterns and press **Open in Signal Navigation**.



The **Signal Navigation** is split up into two sections, the *Navigation view* and the *EBSD signal view*. To navigate the dataset, simply click somewhere within the *Navigation view* and the corresponding EBSP will show up to the right.



For an EBSD recording there are three different images that can be used for navigating the dataset, the navigation image is changed the drop-down menu. By default, the navigator is a *Mean intensity map*.

1.2 Feature maps

It is possible to generate some different images from the EBSD dataset prior to indexing. See Table 1.1 for a description of the possible images. The images are generated by navigating to **Pattern inspection** ⟶ **Pre-indexing maps** and selecting one of the four options. The images are stored in a new folder with the name on the form {Pattern filename}_images and can be inspected by double-clicking on the file in the *System viewer*.

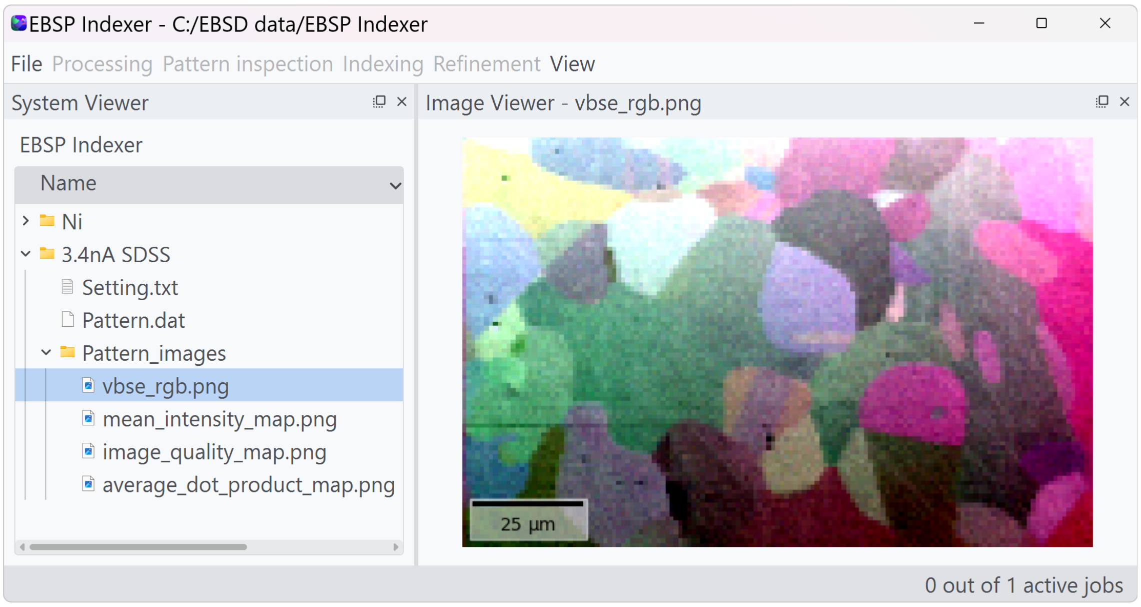


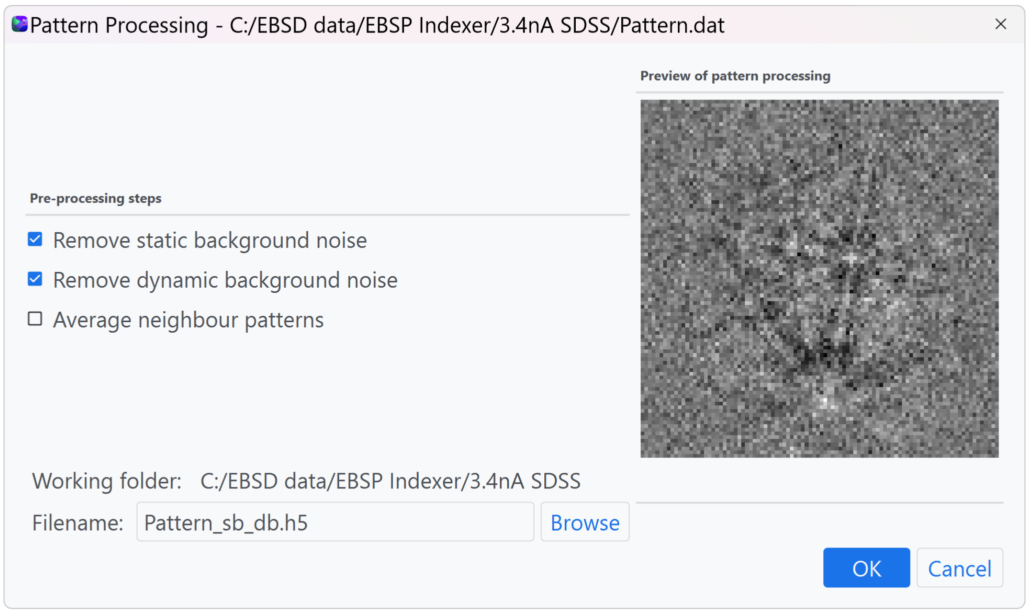
Table 1.1: Feature maps than can be generated in **EBSP Indexer**

Average dot product	Relative difference between neighbouring EBSPs
Image quality	The pattern quality of every EBSP
Mean intensity map	The mean intensity of every EBSP
Virtual backscatter electron	RGB Virtual backscatter electron (BSE) imaging

Pattern processing

2.1 Signal-to-noise improvement

A raw EBSD dataset typically has quite weak Kikuchi bands and unevenly illuminated patterns so it is necessary to perform some image processing in order to increase the band contrast and pattern quality. The application provides three processing steps, **Static background removal (sb)**, **Dynamic background removal (db)** and **Averaging with neighbouring patterns (anp)**. Static background removal is the process of subtracting the empty detector from every pattern, dynamic background removal applies a high-pass filter in Fourier space to remove "low frequencies", meaning large artefacts, from the individual patterns. Averaging is averaging the signal with neighbouring patterns to reduce noise. **Averaging should be used with care as it reduces the spatial resolution.** Pattern processing can be done by selecting a supported EBSD dataset and navigating to **Processing** $\longrightarrow$ **Signal-to-noise improvement**. The processed dataset is stored as a new file in the same directory.


Pattern Processing - C:/EBSD data/EBSP Indexer/3.4nA SDSS/Pattern.dat

Pre-processing steps

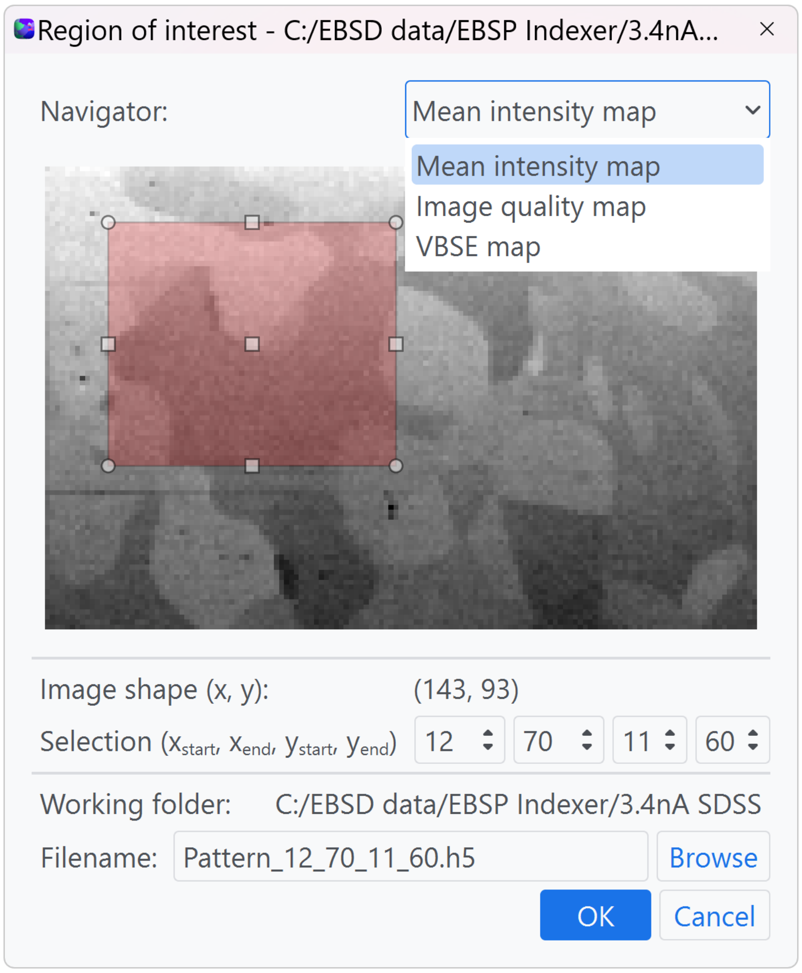
- [x] Remove static background noise
- [x] Remove dynamic background noise
- [ ] Average neighbour patterns

Working folder: C:/EBSD data/EBSP Indexer/3.4nA SDSS

Filename: Pattern_sb_db.h5

2.2 Region of interest selection

It is possible to make a copy of a region of interest in the *Region of interest* dialog window. The selection tool is found under **Processing** ⟶ **Region of interest**.


Region of interest - C:/EBSD data/EBSP Indexer/3.4nA...

Navigator: Mean intensity map

Image shape (x, y): (143, 93)

Selection (x_{start}, x_{end}, y_{start}, y_{end}) 12 70 11 60

Working folder: C:/EBSD data/EBSP Indexer/3.4nA SDSS

Filename: Pattern_12_70_11_60.h5

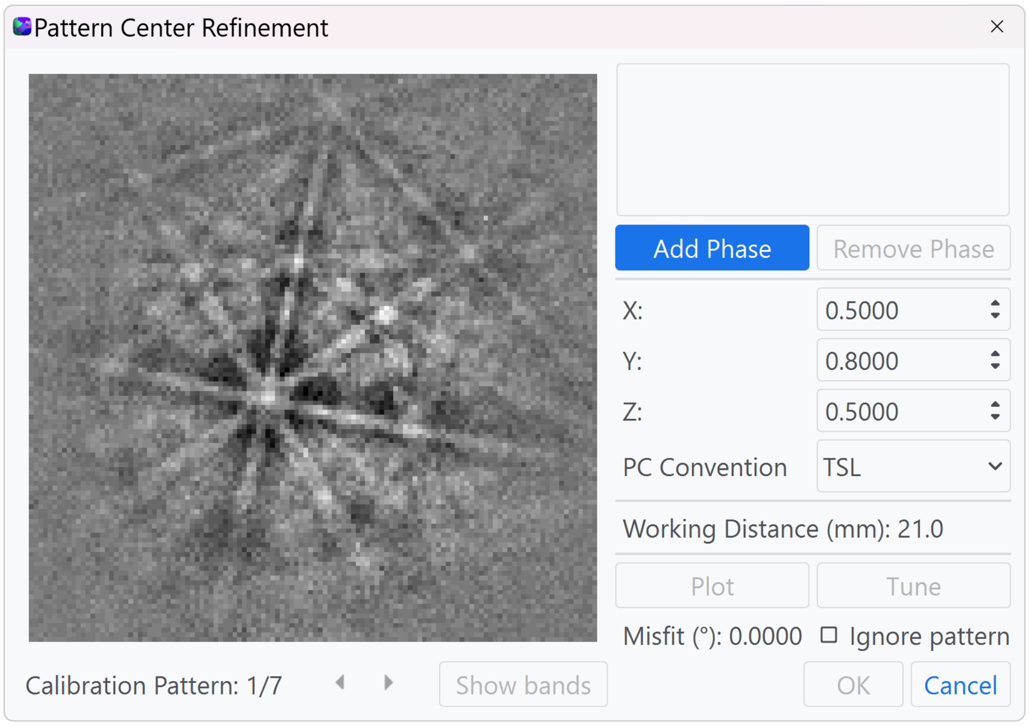
The dialog consists of a *Navigator* where a ROI selection can be made using the mouse. The *Navigator*-image can be changed in the drop-down menu. A selection can also be specified using the spin-boxes. The filename (*.h5) and working folder can be changed by pressing **Browse** and the new selection is saved by pressing **Ok**.

Pattern center calibration

The detector geometry has to be calibrated prior to indexing. The program offers three ways to calibrate the Pattern center (PC).

3.1 Using calibration patterns

Calibration using calibration patterns stored together with the dataset is found under **Processing** $\rightarrow$ **Pattern center optimization** $\rightarrow$ **Calibration patterns**. This opens a new dialog window.



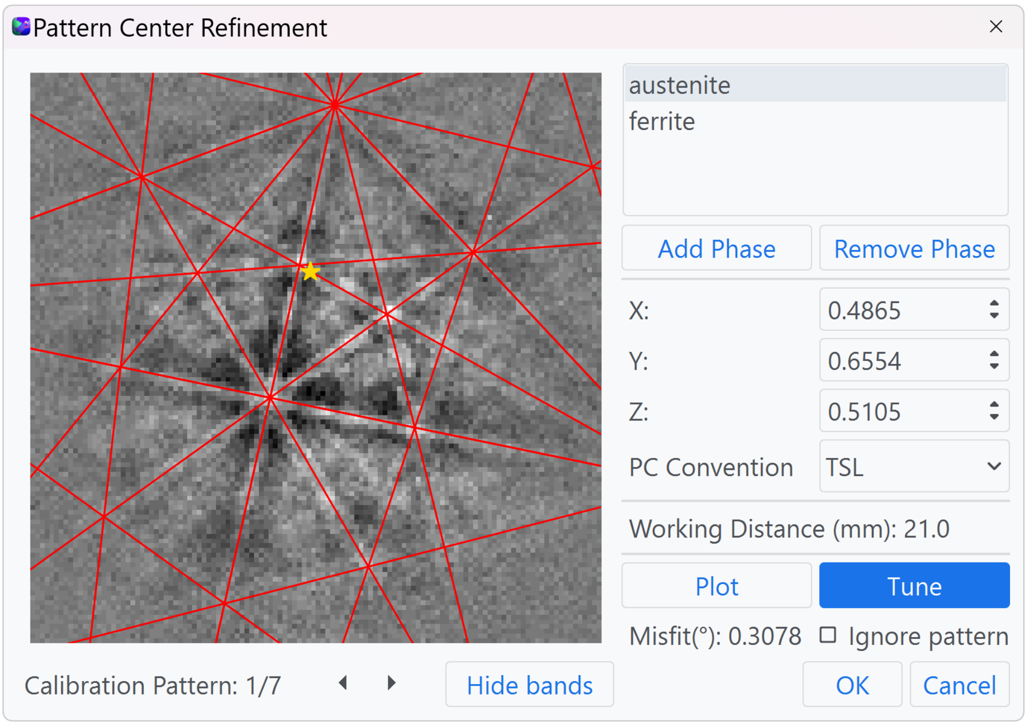
X: 0.5000
Y: 0.8000
Z: 0.5000
PC Convention TSL
Working Distance (mm): 21.0
Misfit (°): 0.0000
Calibration Pattern: 1/7

To start the calibration the expected phases in the sample have to be loaded. This is done by loading the master pattern (MP) for the corresponding phase by pressing **Add Phase**[*]. In the case of Super duplex stainless steel (SDSS) the two phases are *ferrite* and *austenite*.

The next step is to set the initial PC-coordinates. This can either be in the **TSL** convention where the origin is the lower left corner of the detector, or the **Bruker** convention with the origin in the upper left corner. The convention is specified in the drop-down menu **Convention**. X and Z will typically have a value around 0.5 whereas Y depends on the working distance

[*]Currently only supported for FCC and BCC phases

(WD) and will typically have a value between 0.5-0.8 (**TSL** convention). Press **Plot** to see how well the initial PC fits with the pattern. A god fit means that the red lines lay in the center of the bright bands in the pattern.

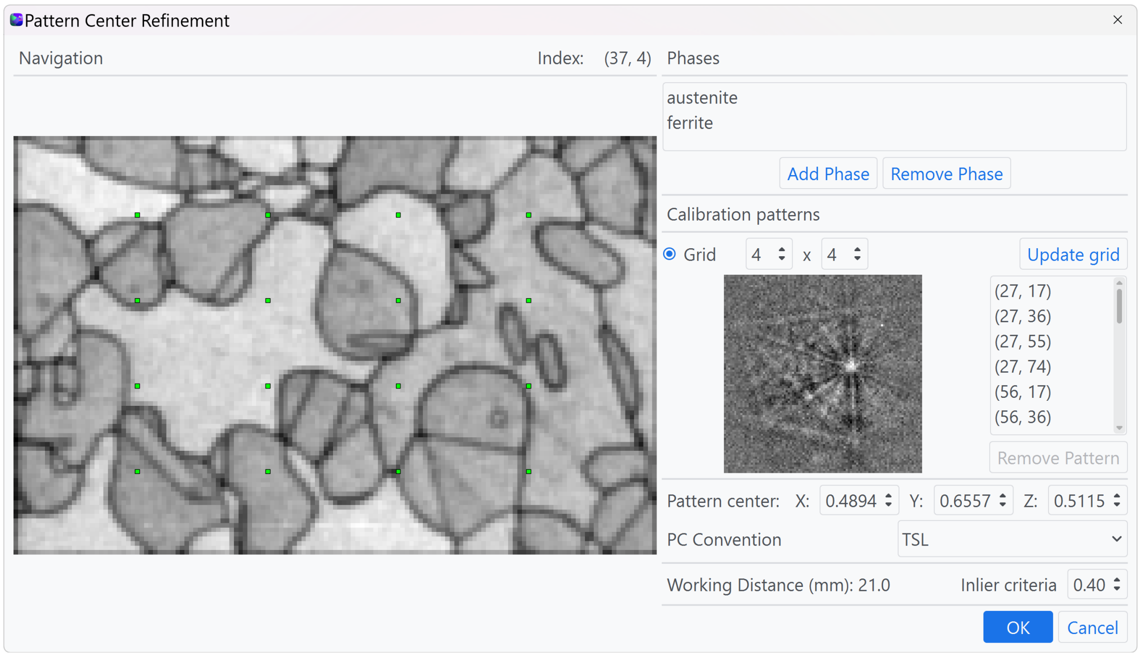


X: 0.4865
Y: 0.6554
Z: 0.5105
PC Convention TSL
Working Distance (mm): 21.0
Misfit(°): 0.3078 ☐ Ignore pattern
Calibration Pattern: 1/7

If the match is completely wrong, select another phase and plot again. Lastly, press **Tune** to perform a local optimization of the pattern center to get a better fit, a misfit below 0.6 is typically a good fit. This step can be repeated for every calibration pattern by using the arrows. If the optimization fails for a calibration pattern it can be excluded from the final calculation by checking **Ignore pattern**. Finally when pressing **Ok** the mean value is calculated and stored in the file project_setting.txt in the same directory as the dataset. If the option **Save individual pattern data to project settings file** is checked in the *Settings*-dialog, the individual PC-values for each calibration pattern are stored as well.

3.2 Custom selection of patterns

The second way to calibrate the PC is by performing an automated optimization on a selection of EBSPs from the dataset. This requires a dataset with a certain pattern quality and should be done on a dataset only after improving the signal-to-noise ratio. This method is found under **Processing** $\longrightarrow$ **Pattern center optimization** $\longrightarrow$ **Pattern selection** and opens a new dialog window.



austenite
ferrite

Grid 4 x 4

(27, 17)
(27, 36)
(27, 55)
(27, 74)
(56, 17)
(56, 36)

Pattern center: X: 0.4894 Y: 0.6557 Z: 0.5115

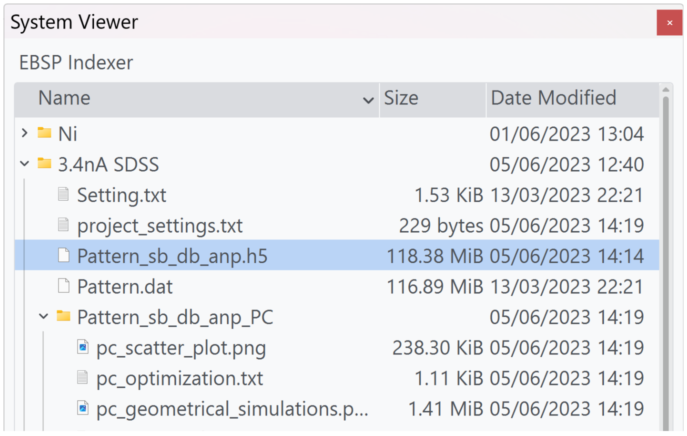
PC Convention TSL

Working Distance (mm): 21.0 Inlier criteria 0.40

The phases loaded from a MP by pressing **Add phase**[*]. The selection of patterns can either be done manually or in a grid by checking **Grid**. The size of the grid can be adjusted by using the spin-boxes and the grid is updated by pressing **Update grid**. The coordinates of the selected patterns are displayed in the list to the right.

Input the initial approximation for the PC-coordinated, this should be a relatively good approximation or else the optimization will most likely fail. The **Inlier criterea** sets the limit for what is accepted as a good match for the individual PC, where 1 corresponds to a perfect fit. The default value is 0.4 and can be adjusted using the spin-boxes. The PC stored to project_settings.txt is the mean value of the PC-values deemed as good matches. The optimization log and results are stored in a folder with the name on the form {pattern filename}_PC.

[*]Currently only supported for FCC and BCC phases

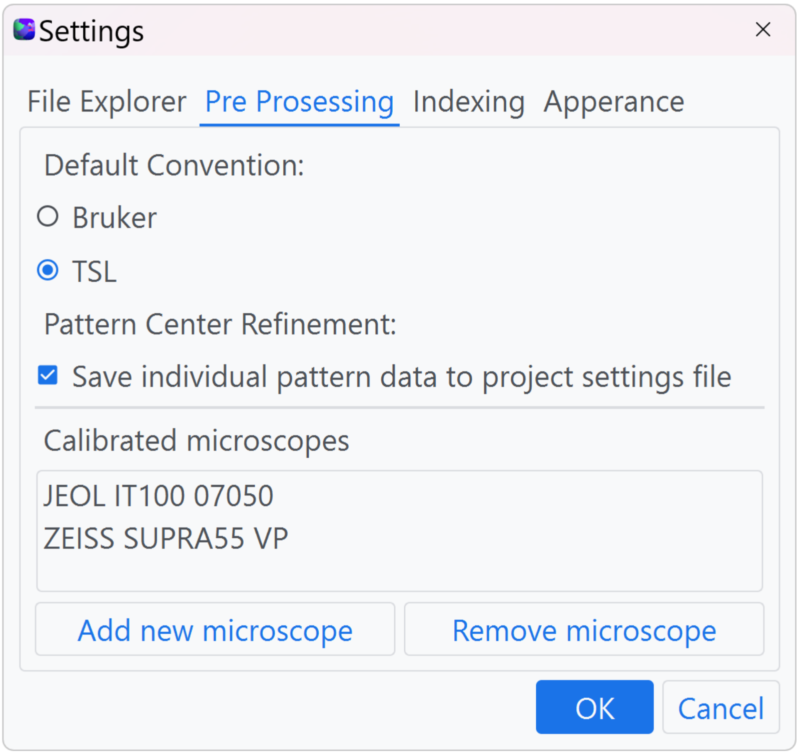

System Viewer

EBSP Indexer

Name	Size	Date Modified
› Ni		01/06/2023 13:04
⌄ 3.4nA SDSS		05/06/2023 12:40
Setting.txt	1.53 KiB	13/03/2023 22:21
project_settings.txt	229 bytes	05/06/2023 14:19
Pattern_sb_db_anp.h5	118.38 MiB	05/06/2023 14:14
Pattern.dat	116.89 MiB	13/03/2023 22:21
⌄ Pattern_sb_db_anp_PC		05/06/2023 14:19
pc_scatter_plot.png	238.30 KiB	05/06/2023 14:19
pc_optimization.txt	1.11 KiB	05/06/2023 14:19
pc_geometrical_simulations.p...	1.41 MiB	05/06/2023 14:19

3.3 Working distance calibration curve

The third way for PC calibration is to add a calibration curve for a specific microscope/EBSD detector-setup. This requires to have at least two EBSD recordings at two different working distances separated by a minimum of 10 — 15 mm. A new microscope is added by navigating to **File** → **Settings** → **Pre Processing**. A list of already calibrated microscopes is available in the dialog window, and a new one can be added by pressing **Add new microscope**.


Settings

File Explorer | Pre Prosessing | Indexing | Apperance

Default Convention:

- [ ] Bruker
- [✓] TSL

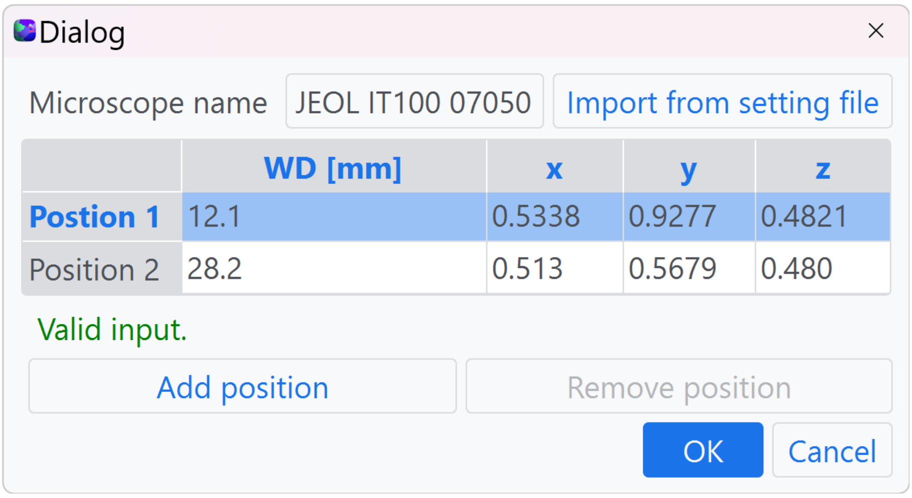
Pattern Center Refinement:

- [✓] Save individual pattern data to project settings file

Calibrated microscopes

JEOL IT100 07050
ZEISS SUPRA55 VP

In the calibration dialog window, the *Microscope name* can be imported from the Setting.txt-file stored together with the EBSD recording by pressing **Import from setting file**. It is possible to set the microscope name manually but is not recommended. To add a calibration curve, type in the PC values for a minimum of two different working distances in the table. More positions can be added by pressing **Add position**. Press **Ok** to store the calibration. To view the calibration line values, double-click on the microscope name in the *Settings* dialog window.



Microscope name JEOL IT100 07050

	WD [mm]	x	y	z
Postion 1	12.1	0.5338	0.9277	0.4821
Position 2	28.2	0.513	0.5679	0.480

Valid input.

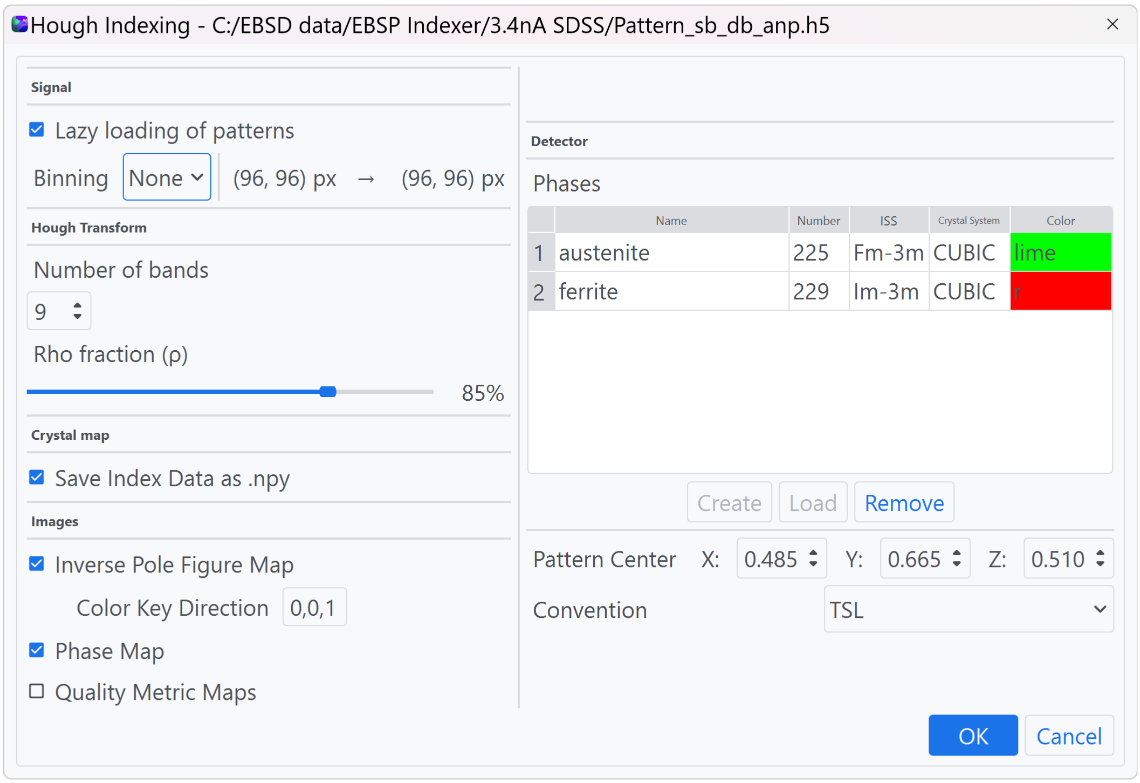
The PC can also be set manually if has been determined in other software.

Indexing

EBSP Indexer supports two routes for indexing EBSPs, Hough indexing[*] and Dictionary indexing. Both methods can be selected by right-clicking on the file storing the processed patterns (e.g. `Pattern_sb_db.h5`) and selecting one of the methods from the menu. The result from both indexing methods is a so-called *crystal map* storing information about the phase and unit cell rotations.

4.1 Hough indexing

The Hough indexing is done internally using the python library `PyEBSDIndex` [6]. The dialog window consists of parameters that can be changed and tuned for specific indexing needs.



Hough Indexing - C:/EBSD data/EBSP Indexer/3.4nA SDSS/Pattern_sb_db_anp.h5

Signal

- [✓] Lazy loading of patterns
- Binning: None — (96, 96) px → (96, 96) px

Hough Transform

- Number of bands: 9
- Rho fraction (ρ): 85%

Crystal map

- [✓] Save Index Data as .npy

Images

- [✓] Inverse Pole Figure Map
  - Color Key Direction: 0,0,1
- [✓] Phase Map
- [ ] Quality Metric Maps

Detector

Phases

	Name	Number	ISS	Crystal System	Color
1	austenite	225	Fm-3m	CUBIC	🟩 lime
2	ferrite	229	Im-3m	CUBIC	🟥 r

Create Load Remove

Pattern Center X: 0.485 Y: 0.665 Z: 0.510

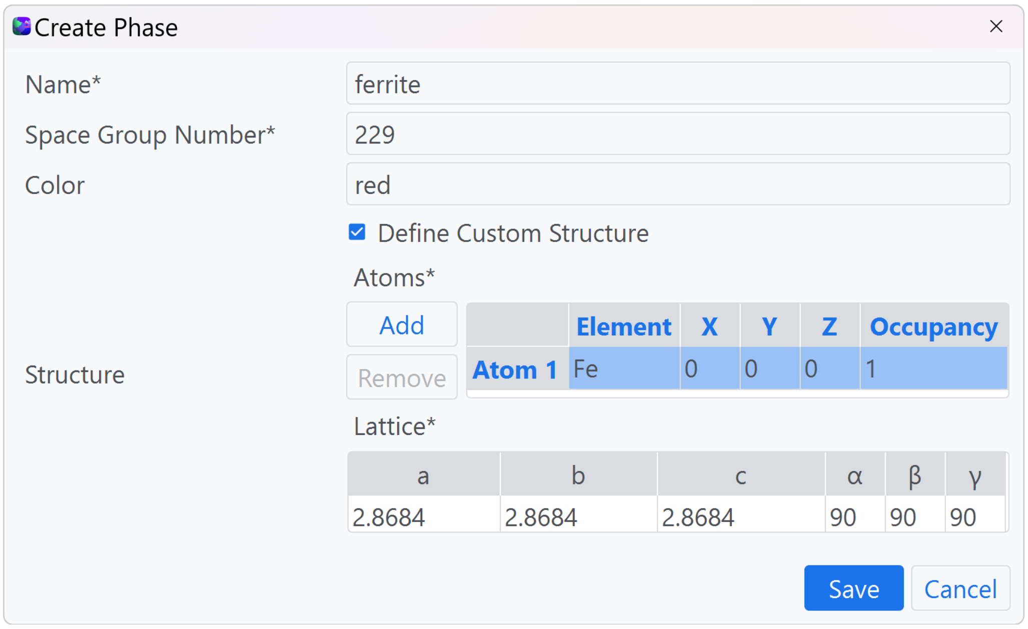
Convention: TSL

OK Cancel

Lazy loading of patterns means that the dataset is split up into smaller chunks, allowing for indexing of EBSD datasets larger than the available memory (RAM). This is enabled by default. *Binning* means to combine pixels on the pattern into larger ones, this is for most indexing

[*]Currently only supported for FCC and BCC phases

cases using HI not necessary. The ρ-fraction specifies how much of the pattern is to be used in indexing, leaving only the band information from the center of the detector. Lastly, it is possible to select what images to generate from the indexed patterns, for multi-phase samples a *Phase map* can also be generated. The *Phases*-table shows information about the phases the indexer will look for. A phase can either be loaded from a master pattern by pressing **Load** or added manually by pressing **Create**. This opens a window where the phase name, space group number, color and optionally the structure can be defined. If PC calibration has been done beforehand, the phases and the PC value are already set.



Name*	ferrite
Space Group Number*	229
Color	red
	✓ Define Custom Structure

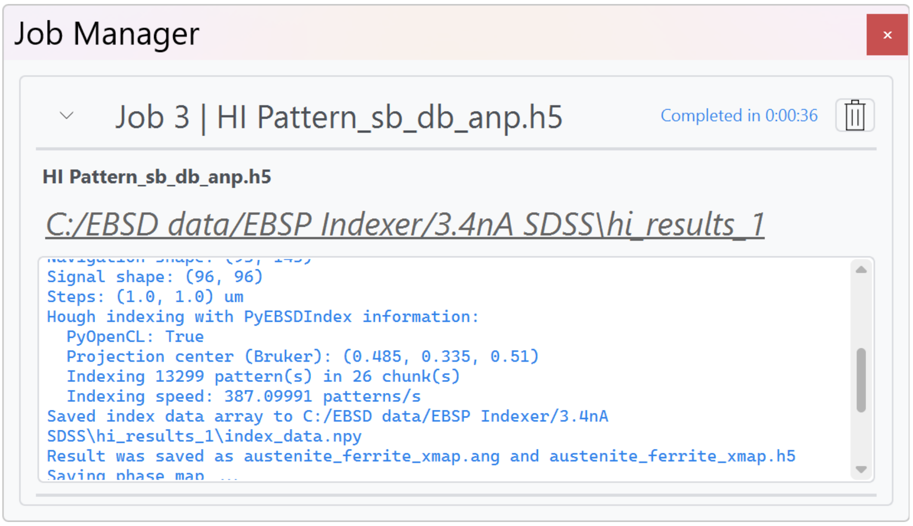
Atoms*

	Element	X	Y	Z	Occupancy
Atom 1	Fe	0	0	0	1

Lattice*

a	b	c	α	β	γ
2.8684	2.8684	2.8684	90	90	90

Press **OK** to start the indexing. The indexing progress is shown in the *Job manager* which will show up to the right in the main window.



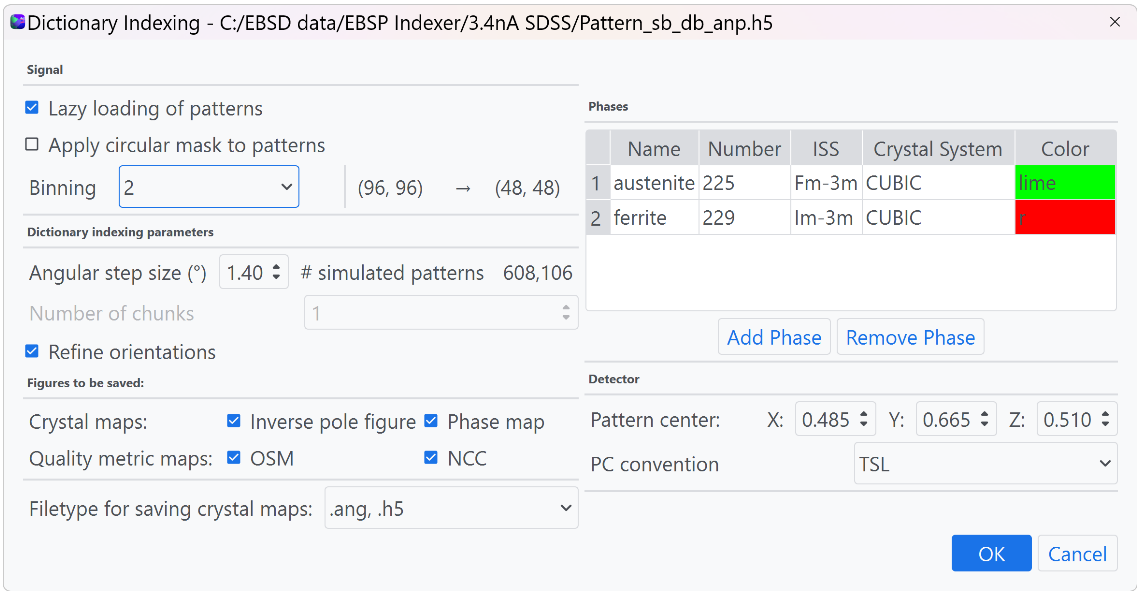
HI Pattern_sb_db_anp.h5

C:/EBSD data/EBSP Indexer/3.4nA SDSS\hi_results_1

```
Signal shape: (96, 96)
Steps: (1.0, 1.0) um
Hough indexing with PyEBSDIndex information:
  PyOpenCL: True
  Projection center (Bruker): (0.485, 0.335, 0.51)
  Indexing 13299 pattern(s) in 26 chunk(s)
  Indexing speed: 387.09991 patterns/s
Saved index data array to C:/EBSD data/EBSP Indexer/3.4nA
SDSS\hi_results_1\index_data.npy
Result was saved as austenite_ferrite_xmap.ang and austenite_ferrite_xmap.h5
Saving phase map ...
```

4.2 Dictionary indexing

Dictionary indexing is done using the indexer provided by `kikuchipy`. The dialog window is quite similar to HI but with some other parameters. DI supports all crystal systems as long as the correct master pattern is available.



Dictionary Indexing - C:/EBSD data/EBSP Indexer/3.4nA SDSS/Pattern_sb_db_anp.h5

Signal

☑ Lazy loading of patterns

☐ Apply circular mask to patterns

Binning: 2 (96, 96) → (48, 48)

Dictionary indexing parameters

Angular step size (°): 1.40 # simulated patterns: 608,106

Number of chunks: 1

☑ Refine orientations

Figures to be saved:

Crystal maps: ☑ Inverse pole figure ☑ Phase map

Quality metric maps: ☑ OSM ☑ NCC

Filetype for saving crystal maps: .ang, .h5

Phases

	Name	Number	ISS	Crystal System	Color
1	austenite	225	Fm-3m	CUBIC	🟩 lime
2	ferrite	229	Im-3m	CUBIC	🟥

Detector

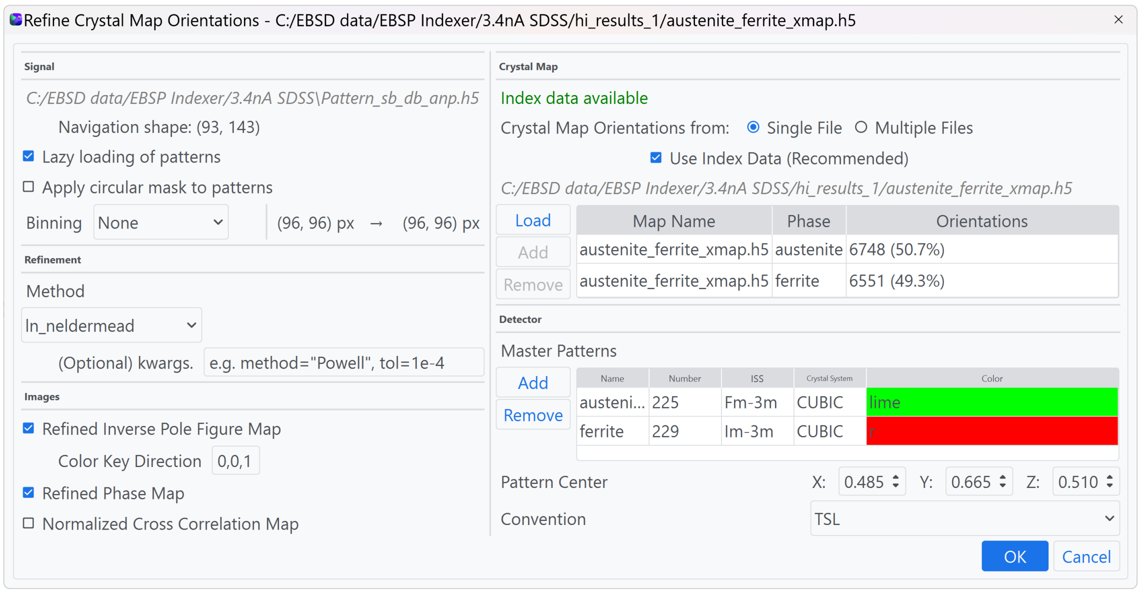
Pattern center: X: 0.485 Y: 0.665 Z: 0.510

PC convention: TSL

Apply circular mask to patterns will remove the pixels from the outer region of every EBSP, removing the influence of the weak diffraction signal from the edges of the detector. *Binning* will reduce the size of each EBSP and as DI is an image-comparison algorithm, reducing the number of pixels will increase the indexing speed, but extreme binning might reduce the accuracy. Typically the pattern resolution should not be lower than 30×30 for high-quality patterns. The *Angular step size* specifies the orientation sampling of the master pattern used for indexing. A higher number will give a coarser dictionary which again will reduce accuracy but increase indexing speed. A lower number will on the other hand increase the size of the dictionary meaning both a more computationally intensive and longer-running indexing. A value between $1.4° - 1.6°$ is usually sufficient for most indexing cases. If *Refine orientations* is checked, a local optimization of the orientations will be performed after the initial indexing, this is especially important when indexing multi-phase samples. The results can be stored in the `.h5` and/or `.ang` file format. `.h5` is needed for further study of the results within **EBSP Indexer**. .ang can be used in other software like ATEX or MTEX. The progress of the indexing is found in the *Job manager*.

4.3 Refinement

Orientation refinement can also be done as a separate step after indexing. This is done by right-clicking on the crystal map from either indexing method and choosing **Refine orientations**. This opens a new dialog window.



Refine Crystal Map Orientations - C:/EBSD data/EBSP Indexer/3.4nA SDSS/hi_results_1/austenite_ferrite_xmap.h5

Signal

C:/EBSD data/EBSP Indexer/3.4nA SDSS\Pattern_sb_db_anp.h5

Navigation shape: (93, 143)

☑ Lazy loading of patterns

☐ Apply circular mask to patterns

Binning: None — (96, 96) px → (96, 96) px

Refinement

Method: ln_neldermead

(Optional) kwargs. e.g. method="Powell", tol=1e-4

Images

☑ Refined Inverse Pole Figure Map

Color Key Direction: 0,0,1

☑ Refined Phase Map

☐ Normalized Cross Correlation Map

Crystal Map

Index data available

Crystal Map Orientations from: [✓ Single File] Multiple Files

☑ Use Index Data (Recommended)

C:/EBSD data/EBSP Indexer/3.4nA SDSS/hi_results_1/austenite_ferrite_xmap.h5

Map Name	Phase	Orientations
austenite_ferrite_xmap.h5	austenite	6748 (50.7%)
austenite_ferrite_xmap.h5	ferrite	6551 (49.3%)

Detector

Master Patterns

Name	Number	ISS	Crystal System	Color
austeni...	225	Fm-3m	CUBIC	lime
ferrite	229	Im-3m	CUBIC	🟥

Pattern Center: X: 0.485 Y: 0.665 Z: 0.510

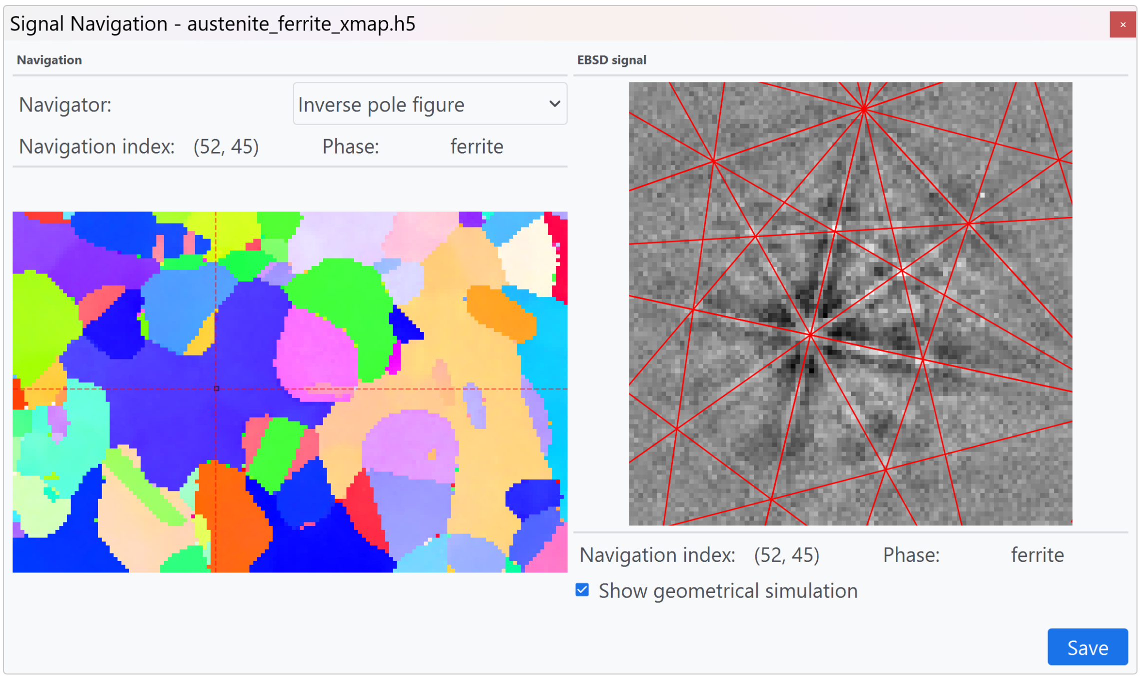
Convention: TSL

The refinement parameters are *Binning*, *Signal mask* and what refinement method should be used under *Method*. Under *Crystal Map* the indexing results to be refined are displayed. If the crystal map was generated from DI the check-box **Use Index Data** is disabled. When loading a multi-phase crystal map, the number of data points of each phase is displayed under *Orientations*. The detector geometry is defined under *Detector*. The MP for the phases to be refined have to be loaded and the PC-coordinates are set before pressing **Ok** to initiate the orientation refinement.

Inspecting indexing results

5.1 Signal navigation - Crystal map

The indexed results from both HI and DI can be inspected in *Signal Navigation*. To investigate the crystal map, right-click on the file and press Open in Signal Navigation. The layout is the same as for EBSD datasets.


Signal Navigation - austenite_ferrite_xmap.h5
Navigation
Navigator: Inverse pole figure
Navigation index: (52, 45) Phase: ferrite
EBSD signal
Navigation index: (52, 45) Phase: ferrite
Show geometrical simulation
Save

The default navigator for crystal maps is the inverse pole figure, a phase map is also available for multi-phase samples. The current phase of each point is displayed above the navigator, and the corresponding EBSP can be viewed by clicking. The indexing result can be verified visually by checking **Show geometrical simulation** which will overlay a set of simulated kikuchi lines. A good match means that the simulated lines lay in the center of the bright bands in the diffraction pattern. The current view can be exported as an image by pressing **Save**.

Bibliography

[1] E. M. Østvold, H. T. T. Relling and O. Leth-Olsen, *EBSP indexer*, version v0.1.0, May 2023. DOI: `10.5281/zenodo.7925262`. [Online]. Available: `https://doi.org/10.5281/zenodo.7925262`.

[2] E. M. Østvold, 'EBSP indexer - an open-source alternative to commercial EBSD software,' Master thesis, Norwegian University of Science and Technology, Trondheim, 2023.

[3] H. T. T. Relling, 'EBSP indexer - making dictionary indexing accessible,' Master thesis, Norwegian University of Science and Technology, Trondheim, 2023.

[4] O. Leth-Olsen, 'Distinguishing martensite from ferrite in mild steels with EBSD using dictionary indexing,' Master thesis, Norwegian University of Science and Technology, Trondheim, 2023.

[5] H. W. Ånes *et al.*, *Pyxem/kikuchipy: Kikuchipy 0.8.4*, version v0.8.4, Apr. 2023. DOI: `10.5281/zenodo.7808659`. [Online]. Available: `https://doi.org/10.5281/zenodo.7808659`.

[6] D. Rowenhorst and H. W. Ånes, *Usnavalresearchlaboratory/PyEBSDIndex: PyEBSDIndex 0.1.2*, version v0.1.2, May 2023. [Online]. Available: `https://pyebsdindex.readthedocs.io/en/stable/index.html`.

Appendix G

Abststract for IMC20

Differentiation between similar phases in an Al-10%Si alloy by EBSD dictionary indexing.

Relling, Hallvard[1], Østvold, Erlend[1], Leth-Olsen, Olav[1], Ånes, Håkon W.[1], Yu, Yingda[1], Khromov, Sergey[1], Kramer, Berit[1], Hjelen, Jarle[1]
[1]Norwegian University of Science and Technology, Norway

The phase distribution in an Al-10%Si alloy, with Si-particles at the sub-micron scale, has been investigated by EBSD. Since Al (FCC) and Si (diamond) have similar crystal structures, the conventional Hough indexing (HI) is struggling to differentiate between them due to their very similar EBSPs. Phase differentiation is also challenging using EDS mapping as the X-ray emission volume is larger than the particle size for light elements at high acceleration voltages (~20 kV) [1]. A newer indexing approach, dictionary indexing (DI), has been applied to separate the similar phases. DI is an image comparison method where the experimentally acquired patterns are compared with a dictionary of simulated EBSPs generated from a master pattern [2].

The specimen was prepared in a Buehler Vibromet 2. The EBSD dataset was acquired on a Zeiss Ultra 55 FESEM equipped with a NORDIF UF-1100 EBSD detector. The patterns with resolution 80x80 pixels were streamed to HDD at a rate of 300 fps. The acceleration voltage, tilt angle and step size were 20 kV, 70° and 20 nm, respectively. To reduce noise in the raw patterns a 2x on-line averaging was applied. The indexing was done using *EBSP Indexer* [3], a GUI based on the Python library kikuchipy [4]. Static and dynamic background correction was performed on the patterns before averaging with a 3x3 matrix. *Figure 1* shows the grains in an Inverse Pole Figure (IPF) map and the silicon particles (red) in the aluminium matrix (green) are shown in the phase map.

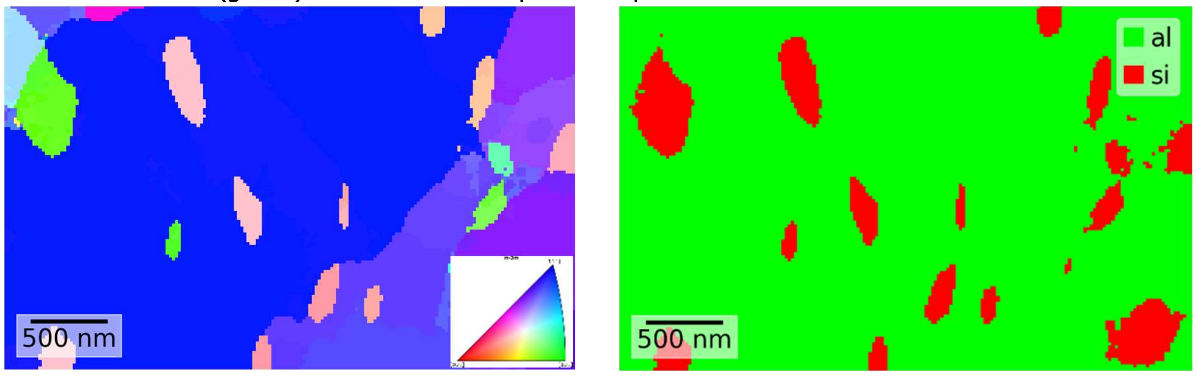


Figure 1: Inverse pole figure map (left) and phase map (right) of an Al-10%Si after indexing using DI.

Figure 1 demonstrates how DI can reliably distinguish between the two similar cubic phases, allowing for phase identification using EBSD without the need for additional characterisation methods.

References

1. M. M. Nowell and S. I. Wright (2004) Phase differentiation via combined EBSD and XEDS, *JOM* **213** doi: 10.1111/j.0022-2720.2004.01299.x
2. P. G. Callahan and M. De Graef (2013) Dynamical Electron Backscatter Diffraction Patterns. Part I: Pattern Simulations, *Microsc. Microanal.* **19** doi: 10.1017/S1431927613001840
3. E. Østvold et al. EBSP INDEXER, an open-source software for Hough- and Dictionary indexing, available for this conference
4. H. W. Ånes et al. (2023) pyxem/kikuchipy: kikuchipy, Zenodo, doi: 10.5281/zenodo.3597646