Master's thesis

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

Erlend Mikkelsen Østvold

EBSP Indexer

An open-source alternative to commercial EBSD software

Master's thesis in Engineering and ICT
Supervisor: Jarle Hjelen
Co-supervisor: Bjørn Haugen
June 2023

NTNU
Norwegian University of
Science and Technology

Erlend Mikkelsen Østvold

EBSP Indexer

An open-source alternative to commercial EBSD software

Master's thesis in Engineering and ICT
Supervisor: Jarle Hjelen
Co-supervisor: Bjørn Haugen
June 2023

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

Acknowledgements

This thesis considers the development of EBSP Indexer, an application for processing, indexing and analyzing Electron Backscatter Patterns (EBSPs). EBSP Indexer is the continuation of the prototype which resulted from the pre-study [1], developed by Hallvard Tangvik Tellefsen Relling, Olav Leth-Olsen and Erlend Mikkelsen Østvold. The contributions from the master's thesis of Relling [2] have been crucial to the continuation which lead to EBSP Indexer, and includes a more in-depth perspective on the capabilities of Dictionary Indexing (DI). The master's thesis of Leth-Olsen [3] considers the use of DI to identify martensite in low-carbon steel, while also implementing additional functionality for calibrating pattern center that is present in the final version of EBSP Indexer. The source code of EBSP Indexer may be used to complement the reading of this thesis, which may be found either as an attachment or by the Digital Object Identifier: 10.5281/zenodo.7925262 provided by Zenodo [4]. Locations of highlighted code are given relative to the root directory of the source code.

This thesis has been supervised by Professor Jarle Hjelen from the Department of Materials Science and Engineering, and co-supervised by Associate Professor Bjørn Haugen from the Department of Mechanical and Industrial Engineering, at the Norwegian University of Science and Technology (NTNU). I would like to reach out a special thanks to our supervisor Prof. Hjelen for his guidance, feedback, and efforts in formulating the scope of the project. Additional thanks go to my co-developers and master's students Relling and Leth-Olsen, as well as my co-supervisor Assoc. Prof. Haugen. Technical help from Ph.D. Candidate Håkon Wiik Ånes, developer and maintainer of kikuchipy [5], the Python library which provides the foundation for EBSP Indexer, has been greatly appreciated. Thanks to our associates from the Electron Microscopy lab at NTNU, senior engineers Sergey Khromov and Yingda Yu, and also the students who enrolled in TMT4166 Module 3, who helped test the application. The contributions of everyone have led to what EBSP Indexer is today.

Lastly, I am thankful for the opportunity I was given to write my master's thesis in cooperation with the Department of Materials Science and Engineering at NTNU. This thesis builds upon the very core of my chosen field of study, creating bridges between the disciplines of science and software.

Abstract

Electron backscatter diffraction (EBSD) has become a well-established technique for investigating microstructures within crystalline materials, and so the interest in pushing the limitations of EBSD has grown. Dictionary Indexing (DI) has proven to be a valuable alternative when Hough Indexing (HI) fails to yield accurate crystallographic information. This is due to HI relying on image feature extraction, whereas DI takes advantage of dynamically simulated EBSD patterns (EBSPs) that reproduce accurate intensity distributions. Many commercial EBSD software features indexing routines based on DI, but no free alternative in the form of a traditional desktop application exists. There do however exist open-source solutions which offer EBSD tools for processing, indexing and analyzing EBSPs, usually accessible through an Application Programming Interface (API). Two such solutions are the Python libraries kikuchipy [5] and PyEBSDIndex [6].

EBSP Indexer was developed as an open-source application, which uses a Graphical User Interface (GUI) to increase accessibility to EBSD tools offered by kikuchipy and PyEBSDIndex. The application was developed using principles of interaction design and aims to improve usability for students and material scientists who are new to EBSD. A usability test was conducted to indicate the performance, compatibility and overall user experience of the application, which received an average score of 8.4 out of 10 from participants. Observations made during the test and data from a questionnaire were used to further improve features, which resulted in version 0.1.0 being released online. The application was packaged as an installer for Windows and an app for macOS, which were made available to download through Zenodo [4] and SourceForge [7]. Additional developer resources were made in an attempt to establish an open-source community, which provides the possibility for future development and maintenance of the source code.

This thesis continues the work of the pre-study [1] by contributing the following:

- Addition of new software features, and overhauling of existing ones
- Carrying out a usability test of a pre-release version of the software
- Distributing and publishing the first official release of EBSP Indexer
- Developer resources in the form of guidelines, tutorials and documentation
- Submission of an abstract and a research poster to The 20th International Microscopy Congress (IMC20)

Patch v0.1.1 of EBSP Indexer is already in development, and aims to fix minor issues which were postponed or discovered after release. Future work consists of improving stability on Mac, overhauling the existing code design, implementing unit tests, and the possibility of Hybrid Indexing, which combines DI and HI to increase indexing speed while retaining high levels of accuracy.

Sammendrag

Elektron tilbakespredningsdiffraksjon (EBSD) er blitt en godt etablert metode for å undersøke krystallstrukturer og -orienteringer i krystallografiske materialer, og som resultat har interessen for å presse begrensingene av EBSD økt. *Dictionary Indexing* (DI) has vist seg å være et verdifult alternativ i tilfeller der *Hough Indexing* (HI) ikke har mulighet til å gi god nok presisjon. Dette er fordi HI er avhengig av ekstraksjon av Kikuchi bånd i bilder, mens DI benytter seg av dynamisk simulerte EBSD mønstre (EBSPs) som inneholder presise intensitetsfordelinger. Mange kommersielle programvarer tilbyr indiserings metoder basert på DI, men det er en mangel på slike løsninger som er gratis og i form av en tradisjonell skrivebords applikasjon. Derimot finnes det en del åpen-kilde kode løsninger som tilbyr EBSD verktøy for prosessering, indisering og analysering av EBSPs, som vanligvis er tilgjengelige gjennom et applikasjonsprogrammeringsgrensesnitt (API). To slike løsninger er Python bibliotekene kikuchipy [5] og PyEBSDIndex [6].

EBSP Indexer ble utviklet som en åpen-kilde applikasjon, som benytter seg av et grafisk brukergrensesnitt (GUI) for å forbedre tilgjengelighet av EBSD verktøy fra kikuchipy og PyEBSDIndex. Utvikling av applikasjonen har tatt hensyn til prinsipper innenfor interaksjon design, og sikter på å forbedre brukervennlighet mot studenter og andre nybegynnere innen EBSD. En brukervennlighets test ble gjennomført for å gi en indikasjon på ytelse, kompatibilitet, og brukeropplevelse, som deltagerne ga en gjennomsnitts score på 8.4 av 10. Observasjoner under testen og data fra et egetlaget spørreskjema, ble brukt til å forbedre og videreutvikle funksjonalitet, som resulterte i at versjon 0.1.0 ble gitt ut på nett. Applikasjonen ble pakket sammen og distribuert som installasjonsfil for Windows og som app for macOS, der begge ble gjort tilgjengelige for nedlasting på Zenodo [4] og SourceForge [7]. I tillegg ble det laget og publisert en rekke utviklerresurser for å etablere et åpen-kilde samfunn, som skal kunne gi mulighet til videreutvikling og vedlikehold av kildekoden.

Denne oppgaven bygger på arbeidet som ble gjort i prosjektoppgaven [1], ved å bidra med følgende:

- Tilføyelse av ny programvare funksjonalitet, i tillegg til fornying av eksisterende implementasjoner.
- Utførelse av brukervennlighets test av en forhånds-utgivelse av programvaren
- Distribuere og publisering av den første offisielle utgivelsen av EBSP Indexer
- Utviklingsressurs i form av retningslinjer, opplæring og dokumentasjon
- Innlevering av sammendrag og forskningsplakat til Den 20. Internasjonale Mikroskopikongress (IMC20)

Oppdatering v0.1.1 av applikasjonen er allerede under utvikling, og vil inneholde små feilrettinger som ble utsatt eller oppdaget etter første utgivelse. Fremtidig arbeids består av å forbedre stabilitet på Mac, fornye kode design, implementasjon av enhetstesting, og muligheten for *Hybrid Indexing*, som kombinerer DI og HI for å øke indiserings-hastighet og samtidig beholde høye presisjon.

Table of Contents

List of Source Codes

List of Figures

List of Tables

Acronyms

API Application Programming Interface. 2, 3, 11, 20–22, 29, 30, 56, 57, 59

BCC Body-centered Cubic. 9, 21

CPU Central Processing Unit. 20, 22, 32, 46

DI Dictionary Indexing. 1, 2, 7, 9–11, 14, 15, 18, 20, 30, 35, 36, 38, 43, 48, 49, 54, 57, 60

DOI Digital Object Identifier. 30, 46

EBSD Electron Backscatter Diffraction. 1–3, 5, 8–10, 12, 13, 16, 20, 37, 41, 42, 53, 55–57, 59, 62

EBSP Electron Backscatter Pattern. 1–3, 5–11, 20, 21, 38, 42, 46

FCC Face-centered Cubic. 2, 9, 21

GPL General Public Licence. 20, 21, 24, 29, 46, 51, 62

GPU Graphics Processing Unit. 19, 22, 30, 32

GUI Graphical User Interface. 3, 4, 11–15, 18, 19, 22–25, 27, 28, 30, 32, 41, 43, 46, 53, 54, 56–62

HI Hough Indexing. 1, 2, 7–11, 17, 18, 20–22, 30–32, 35, 36, 38, 39, 43–45, 48, 49, 53, 54, 57, 60

IDE Integrated Development Environment. 27, 28

IMC20 The 20th International Microscopy Congress. 46, 52, 59, 62

IPF Inverse Pole Figure. 2, 6, 13, 21, 31, 33, 36, 42, 48, 57

IQ Image Quality. 7, 9

LTS Long-Term Support. 23, 24, 28

MSVC Microsoft Visual Studio C++ Redistributed 2015-2022. 24, 27

OIM DC TSL OIM Data Collection. 12, 13, 15, 20–22, 38, 39, 42, 53, 57

OS Operating System. 20, 28, 37, 46

PC Pattern Center. 6, 20–22, 43, 46

ROI Region of Interest. 6, 7, 21, 36, 39–41, 43, 44

SDSS Super Duplex Stainless Steel. 7, 14, 15, 38, 44

SEM Scanning Electron Microscope. 1–3, 5, 7, 38, 46

UI User Interface. 15, 20, 53, 58

UX User Experience. 11, 24, 38, 53, 57

1 Introduction

There is a lack of non-commercial, traditional desktop applications within the Electron Backscatter Diffraction (EBSD) community, which features indexing routines based on dictionaries of diffraction patterns, developed by S. Singh and M. De Graef [8].

This thesis is the continuation of the preliminary work of the pre-study [1], conducted during the Fall semester of 2022. Necessary background on the topic of EBSD is given in the subsections below, to better understand the current situation of software used in the industry, as well as the previous work that has led to this thesis. Details about the goals and scope are described, which aims to address the above problem.

1.1 Background and motivation

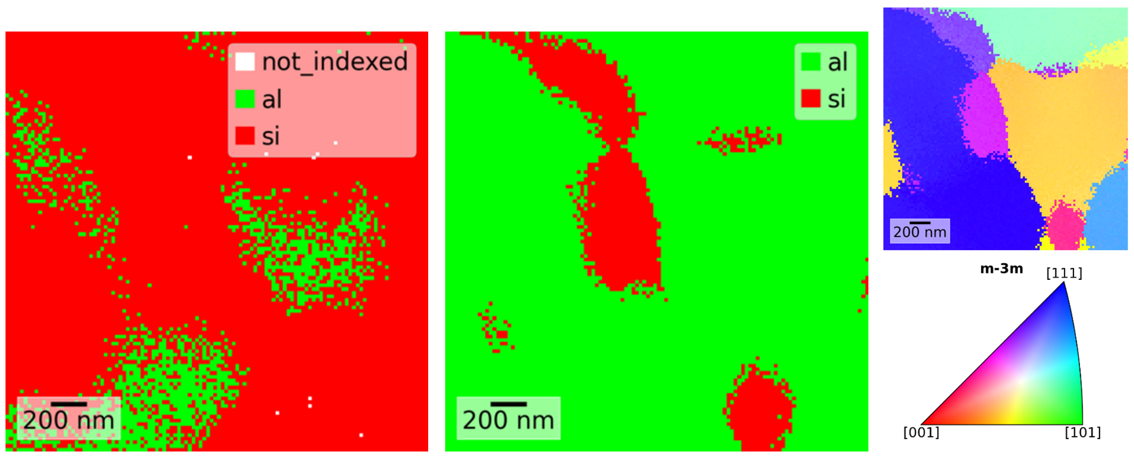
Information about a crystalline material's microstructure is vital to understand the behavior of the material. Many sciences like mechanical engineering, nanotechnology, material science, and geomechanics, are concerned with the strength, ductility, and fatigue resistance of the material, all of which can be derived from inspecting the microstructure. This knowledge can in return deliver methods for manufacturing metallic materials of increased strength-to-weight ratio and energy absorption capacity, which can be produced at a lower cost of raw materials and emissions [9].

To gain insight into the microstructure of a crystalline material, a common approach is to use EBSD, a specific signal mode deployed using a Scanning Electron Microscope (SEM). A sample of considerable size, taken from the crystalline material, is placed inside the vacuum-sealed chamber of the SEM, before being exposed to a high-energy beam of electrons emitted from a probe above the sample [1]. The electrons interact with the sample and are reflected towards a thin, transparent phosphor screen, generating an Electron Backscatter Pattern (EBSP) that is captured by a light-sensitive camera, from which crystallographic information can be derived. The high-energy beam is moved across the surface of the sample, generating large datasets which might consist of multiple hundreds of thousands EBSPs. The resolution of an EBSP is typically 100x100 pixels, where each pixel is 8-bit, allowing datasets to be streamed at speeds of several hundred patterns per second. As a result, the size of a dataset can become several gigabytes. To make sense of information like grain sizes, orientations, phases, and more, the dataset of EBSPs must be indexed either during or after the scan, the latter usually being on a separate computer. The indexing procedure is commonly done using one of two approaches, either by Hough Indexing (HI), developed from the work of Lassen et al. [10], or by pattern matching, which usually consists of dictionary indexing and refinement of orientation [11].

HI [10] consists of performing the Hough Transform [12] on EBSPs to identify the high-intensity lines, also known as Kikuchi bands. The identified bands are evaluated in comparison to theoretical values based on the material structures present in the sample, to determine the orientation of the grain that the EBSP originated from [1]. HI has been considered the golden standard in indexing methods since its first introduction in 1992, due to providing reliable results at an affordable number of required computations. The approach does however lack precision when indexing data from samples that consist of materials with similar crystal structure, due to the resulting EBSPs being almost indistinguishable when it comes to determining the material of origin. Identification of the bands can be troublesome when EBSPs are subjected to noise, a common consequence of lowering the accelerating voltage of the beam to increase spatial resolution [13, 1]. Another problem arises when the EBSP is a superposition of two differently oriented grains, common to occur at grain boundaries of fine-grained samples [11].

Pattern matching techniques provide an alternative in cases where HI struggles to produce correct indexing results. A general framework for dictionary-based indexing (DI) was developed by Singh and Graef [8] in 2016, where indexing is done by comparing the experimental patterns with a dictionary of simulated patterns, which contains all scattering vectors [14]. In practice, the intensity distributions are reproduced correctly in simulated patterns, resembling that of an experimental pattern, eliminating the need to rely on theoretical positional values of the bands, derived from kinematical modeling. Standard mathematical image correlation can then be utilized to determine a match [11].

To showcase the accuracy of DI on a fine-grained sample, phase maps from an Al-10Si dataset are shown in Figure 1a and Figure 1b using both indexing techniques. To indicate how accurate these phase maps are, an Inverse Pole Figure (IPF) map with corresponding color key is shown in Figure 1c, which shows the orientation of each indexed EBSP. The sample from the Al-10Si alloy is a good example for showcasing the strengths of DI, due to the Face-centered Cubic (FCC) structure of aluminum resembling that of silicon's diamond structure, which follows similar FCC Bravais lattice [15, 1].



(a) Hough Indexing (b) Dictionary Indexing (c) IPF map

Figure 1: Comparison between phase maps from Hough Indexing (a), and Dictionary Indexing (b) on a dataset from an Al-10Si alloy. Each pixel in a map corresponds to an EBSP. IPF map with corresponding color key (c) shows orientations that match the mapping of phases done with DI.

Source: Adapted from the pre-study [1]

Not only does HI assign silicon to the majority of the crystal map, but the areas containing aluminum are also dispersed with undefined borders compared to the dictionary-indexed phase map. Most importantly, the phase map from DI matches the assignment of individual grains, identified by the orientations displayed in the IPF map [1]. Areas of silicon correspond to orientations colored pink and purple. Note that the IPF map was created from the dictionary-indexed crystal map, however HI assigns similar orientations.

There exist commercial companies dedicated to providing instruments and software solutions for SEM and EBSD. Some popular software choices are delivered by EDAX [16], BRUKER [17] and Oxford Instruments Nanoanalysis [18], all of which provide software that is locked behind expensive licensing [1]. There do exist free alternatives, but few of these offer pattern-matching techniques and the ones that do lack the traditional desktop application format that commercial companies offer. These are usually developed by open-source communities and are instead distributed as source code, which is accessed through an Application Programming Interface (API). APIs are less restricted in terms of functionality, offering flexibility to solve any problem regardless of shape and size. Two projects that are regularly updated and maintained by their communities are EMsoft [14] and kikuchipy [5]. Open-source EBSD toolboxes for MATLAB also exist, MTEX [19] and AstroEBSD [20], but do require MATLAB licenses to be accessed. An overview of EBSD software that includes pattern matching routines for indexing is shown in Table 1.

Table 1: Overview of maintained EBSD software which takes advantage of pattern matching.

Source: Adapted from the pre-study [1]

Software Title	Company/Developers	Open-source	Platform
TSL OIM Data Collection	EDAX, AMETEK.inc [16]	No	Windows
ESPRIT Dynamics	BRUKER [17]	No	Windows 7+
AZtecCrystal	Oxford Instruments [18]	No	Windows 10 64-bit
EMsoft	M. De Graef et al. [14]	Yes	Fortran
AstroEBSD	T. B. Britton et al [20]	Yes	MATLAB
MTEX	F. Bachmann et al. [19]	Yes	MATLAB
kikuchipy	H. W. Ånes et al. [5]	Yes	Python

The Python library kikuchipy is being developed by Ph.D. Candidate Håkon Wiik Ånes and in close relation with the Electron Microscopy Lab at the Norwegian University of Science and Technology (NTNU), where former master students have contributed to its development. To lower the user threshold to entry-level for an API like kikuchipy, a type of interface and interaction style must be chosen. A Graphical User Interface (GUI) fits the usability and user experience goals of simplifying interaction and is why most commercial software adopts this approach. The chosen method for distributing the software should also reinforce this premise of simplicity. Although installing kikuchipy is made easy thanks to package installers like the package installer for Python (Pip), it still requires a Python distribution to be already installed in the user's system environment. Traditional desktop applications are often installed using a single installer file, making it less of a nuisance for users who are not familiar with navigating development environments.

1.2 Pre-study and scope of this thesis

Just like the pre-study, the scope of this thesis is contained within the 2nd step of the EBSD process [Figure 2], concerning the assignment of orientations to EBSPs in a dataset, done by the use of software [1]. The step takes place after the dataset is acquired from the SEM, and before the analytical interpretation formed from post-processing the results. Development of an EBSD software that can bridge the gap between these two steps is at the center of attention.

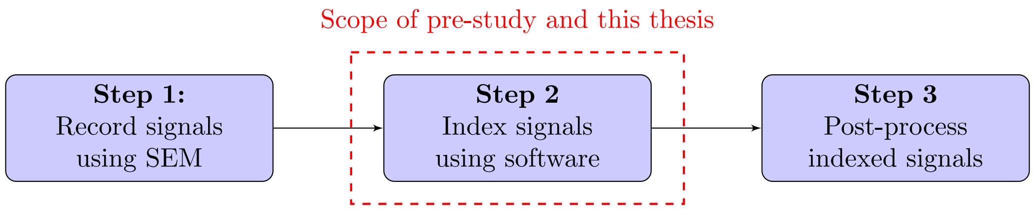

Scope of pre-study and this thesis
Step 1: Record signals using SEM
Step 2 Index signals using software
Step 3 Post-process indexed signals

Figure 2: Simplified flow chart showing steps that the EBSD process is composed of. The scope of this thesis regards step 2 of the process.

Considering that this thesis continues the work of the pre-study, the overall objective still applies; *To make the rich functionality of kikuchipy more accessible to users who are not familiar with the library, or coding in general, by developing a Graphical User Interface (GUI).* The result of the preliminary work was a GUI-based software, thought to be in a pre-release and test-ready state, which provides the foundation for a worthy solution [1]. However, further inspection shows quality that resembles that of a prototype, due to new issues being discovered. An in-depth description of these issues is given in Section 3.1. Although the objective implies a target group who are not concerned with the details and complexity of programming, the GUI also aims to provide an unrestricted experience for more advanced users. This was solved by implementing an interactive

Python interpreter as a terminal inside the GUI [1]. The software was packaged to provide an executable on Windows, which could run without a Python installation. A solution for creating an app bundle on macOS was known but did not see enough time to be tested during the pre-study. All the commercial solutions in Table 1 seem to lack support for macOS, and that is why it would be of great interest to develop a solution that supports this.

With the necessary functionalities in place, the focus shifts from creating features to refining them. The software has also been allowed to be deployed and tested among students who have enrolled in the course *TMT4166 - Experimental Materials Chemistry and Electrochemistry*, and signed up for Module 3 *Scanning electron microscopy, EBSD* [21]. This can provide valuable feedback and data, and serve as an indication of the software's performance and stability. The result should be the first official public release of the software, alongside the necessities needed to build an open-source community to maintain the software. From this, the following objectives can be formulated:

- Resolve the prototype's issues by implementing new features and overhauling existing ones. The result should be distributed as an installer on Windows and as an app bundle on macOS, and should provide a stable experience for end users.
- Conduct a usability test that involves users of the target group to receive feedback and data, which can be used to ensure the completion of the previous point.
- Publish the software, document code, create tutorials, and define guidelines to create the necessary foundation for an open-source community.

2 Theory

In this section, theoretical background on EBSD is given, with a focus on indexing and refinement techniques that are featured in the software. Concepts and principles within interaction design are explained, to provide the necessary background for describing the development process of the software. Details on what a good usability test and questionnaire entail are also presented.

2.1 Electron backscatter diffraction

In order to bridge the gap between an engineering discipline and software, the necessary theoretical background must be facilitated. Understanding the science behind the functionality that users will benefit from in their daily tasks, is necessary for the development of an interactive product that understands the user's needs. This section is based on the literature study conducted and presented in the pre-study, with some added extensions.

EBSD has become a widely accepted technique deployed by SEM for characterizing crystallographic microstructures of metals, due to the rapid expansion of available SEMs [22]. Instead of relying on light to form images, a SEM takes advantage of the magnification capabilities of using electrons to generate signals that can form images. In the case of EBSD, the image shows an Electron Backscatter Pattern (EBSP) made from Kikuchi lines, named after Seishi Kikuchi, who together with Shoji Nishikawa discovered the first case of EBSP in 1928 [23]. From these patterns, crystal properties like orientation, phase, shape and size of grains, texture, and strain can be obtained, all of which enable the study of microstructure phenomenons, such as re-crystallization, grain growth and phase transformations. Knowledge of how metals act on a submicron scale can determine the overall performance, an advantage that contributes to a superior design within many engineering disciplines. For example, in the manufacturing of turbines, the material of the blades should be properly aligned to favour strength in the strain direction.

Before a sample is studied, the appropriate preparation of the sample must be made. The types of preparations vary between materials, but all with the purpose of making the sample perfectly flat without contaminating or damaging the surface [22]. The quality of the EBSPs are highly influenced by the surface of the sample, and the capabilities of EBSD may therefore limit the investigation of some materials. The prepared sample is placed and angled about 70° inside a vacuum chamber, where it is exposed to a high energy beam of electrons, typically a 20 kV probe current, emitted by the SEM [Figure 3] [1].

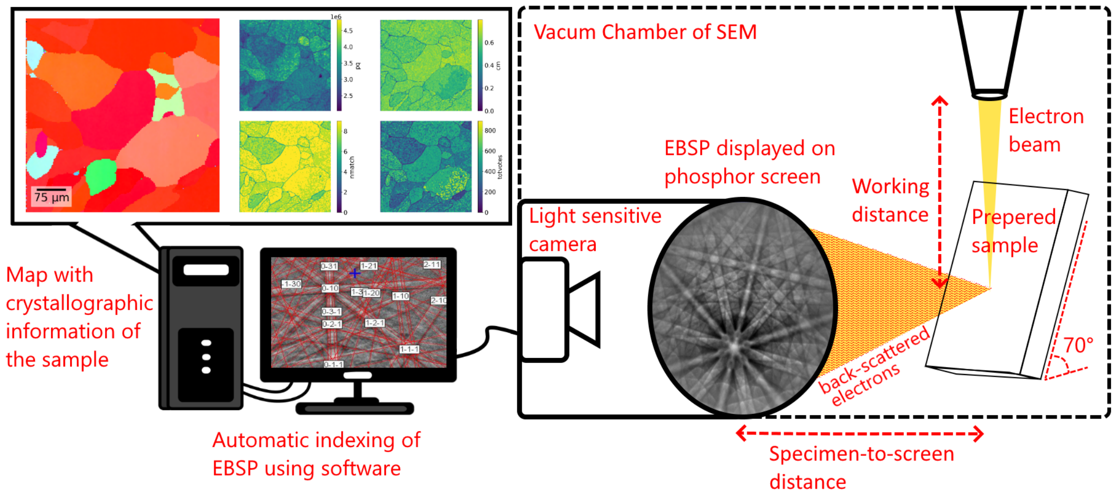

75 µm
Vacum Chamber of SEM
Electron beam
EBSP displayed on phosphor screen
Working distance
Prepered sample
Light sensitive camera
Map with crystallographic information of the sample
back-scattered electrons
70°
Automatic indexing of EBSP using software
Specimen-to-screen distance

Figure 3: Illustration of how an experimental setup for EBSD analysis might look.

The electrons interact with the crystal of the material, described by the geometry and arrangements of atoms, and about 5% of electrons are diffracted towards the pattern acquisition device, which consists of a transparent phosphor screen and a light-sensitive camera [1]. The specimen-to-screen distance is usually around 20 mm and affects the location of the projected center on the phosphor screen, also known as the Pattern Center (PC). Photons are emitted when the high energy electrons hit the phosphor screen, which is captured by the light-sensitive camera to create images of the EBSPs. The high energy beam is moved over the Region of Interest (ROI) of the stationary sample, which results in the diffraction geometry and PC shifting at each point. The impact of this shift depends on the size of the ROI and might require dynamic calibration of PCs from spot to spot [24].

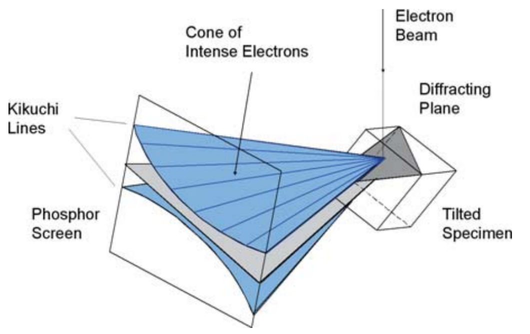
The diffraction of electrons that form Kikuchi lines on the phosphor screen act in accordance with Bragg's law:

$$2 * d_{hkl} * \sin\theta_{hkl} = n * \lambda \tag{1}$$

θ_{hkl} is the Bragg angle between the diffracting plane hkl and the electron beam, which is equal to the distance from the center of the Kikuchi line to its perimeter. The wavelength λ is inversely proportional to the accelerating voltage used, which is why such a high voltage is required to produce distinct Kikuchi lines that can be identified [1]. The order of reflection is n, usually equal to 1, and takes the additional distance the electron must travel into consideration [Figure 4b]. d_{hkl} is the interplanar spacing between parallel planes with Miller index hkl and can be derived from the lattice parameters of the crystal structure. For a cubic crystal structure with lattice parameter a, interplanar spacing between $\{hkl\}$-planes can be calculated by:

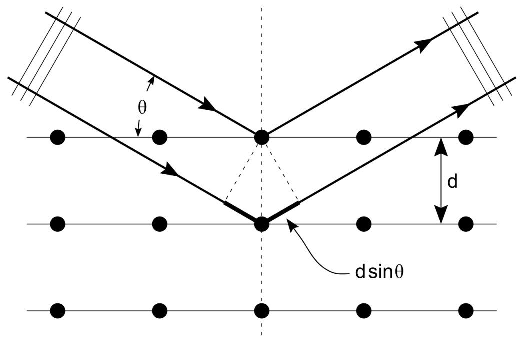
$$d_{hkl} = \frac{a}{\sqrt{h^2 + k^2 + l^2}} \tag{2}$$

The center of the Kikuchi line can be interpreted as the intersection between the lattice plane of $\{hkl\}$ and the phosphor screen, meaning intersections between lines represent zone axes [Figure 4a] [1]. Crystal orientations are derived from θ_{hkl}, acquired by assigning different families of planes to the considered Kikuchi lines, which is done by Equation 1 and Equation 2 for cubic crystal structures. Orientations may be visualized in an Inverse Pole Figure (IPF) map, a two-dimensional graphical representation that shows the location of a sampled point's direction relative to a chosen crystal reference system [25]. Directions in the reference system can then be conveniently visualized as values of red, green and blue in a color key [Figure 1c].



(a) Back-scattered electrons forming a cone of intense electrons that appears as Kikuchi Lines on the phosphor screen.

Source: Schwarzer et al. [24]



(b) Electrons interacting with the atoms of the sample surface according to Bragg's law (Equation 1).

Source: Wikipedia, The Free Encyclopedia [26]

Figure 4: Illustrations of how electrons interact with atoms in the prepared sample (b) and are diffracted as a cone of high-intensity electrons, appearing as Kikuchi lines on the phosphor screen (a).

The quality of experimental EBSPs may vary depending on the material, preparation technique used, the EBSP detector, and the specifications of the SEM, e.g. accelerating voltage, step size and resolution. It is therefore common for EBSD software to include options for image processing. Some options that are available in kikuchipy are the removal of static and dynamic backgrounds, and averaging of patterns based on their neighboring patterns [5]. Anomalies, visible damage to the phosphor screen, and background noise, all of which are constant and present in all patterns acquired, may be removed by subtracting the static background using a flat field image. The flat field image contains an even background by combining noise generated from differently oriented grains in the ROI [1]. Consequently, relative intensities are lost, and kikuchipy rescales intensities to use the complete available data range, usually an 8-bit grey scale. Relative intensities can however be reintroduced by dynamic background removal, which includes subtracting each pattern with a Gaussian blurred version of itself [5]. Some EBSPs may benefit from neighbor averaging, where a number of adjacent patterns, defined by the size of the kernel used, are combined with the processed EBSP. The result is less noisy patterns, due to noise from camera operating conditions being targeted. The current kikuchipy implementation of this does however not take into account grain boundaries, meaning EBSPs at these boundaries might become a combination of differently oriented grains. The resulting EBSP will then seem to have a completely different orientation. It is therefore advisable to skip this step of image processing when dealing with fine-grained materials, or when grain boundaries are studied.

Image Quality (IQ) may be quantified by measuring the sharpness of some Kikuchi lines, which can be done by performing a Fast Fourier Transform of the original pattern [24, 10]. An IQ map of the ROI will then show grain boundaries and surface damage due to producing lower quality signals [Figure 5].

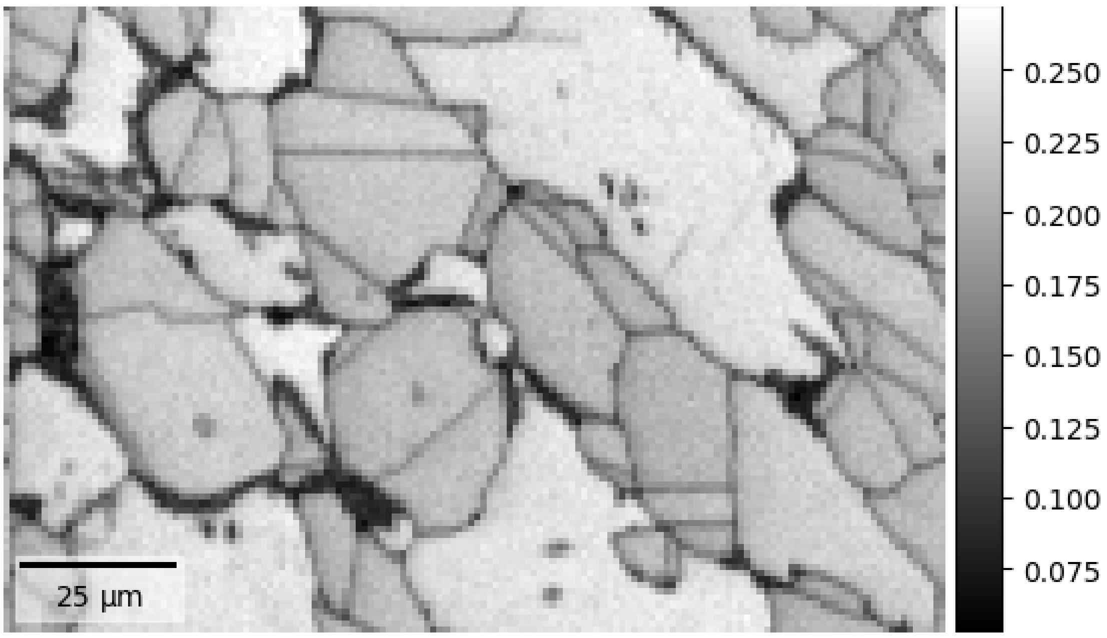


Figure 5: IQ map of an Super Duplex Stainless Steel (SDSS) dataset that contains sigma phase. Both static and dynamic background has been removed, to better showcase which crystals produce low-quality patterns. The dark spots are therefore most likely due to the presence of the sigma phase. IQ is represented by a measure of how sharp the image of the EBSP is, based on the procedure defined by Lassen et al. [10].

The procedure of indexing EBSPs was automated in 1992 with, among other things, the work *Image processing procedures for analysis of electron back scattering patterns* by Lassen et al. [10], and can today be performed in two ways; Hough Indexing (HI) and Dictionary Indexing (DI).

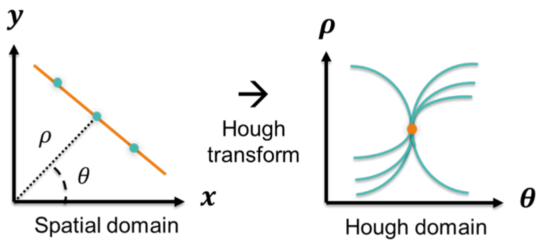
2.1.1 Hough indexing

HI was introduced as part of the fully automated EBSP analysis by Lassen et al. [10], and is still the most used method for indexing which relies on image analysis [11]. Extraction of image features, more specifically Kikuchi lines, is done using the Hough Transform. The initial Hough transformation was first introduced in 1962 by Hough [12], and later in 1972 as part of a major leap in computer vision, when it was used by Duda and Hart [27] to detect lines and simple shapes within digital images.

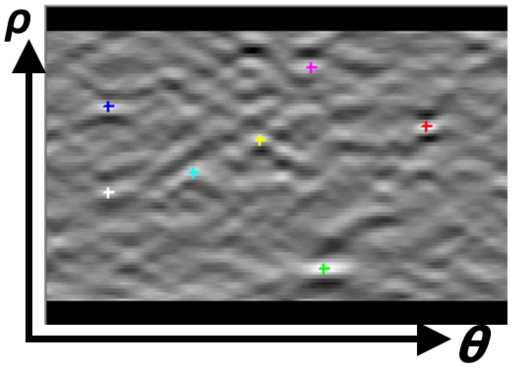
Depending on the software, HI may be implemented in slightly different ways. They do however all include applying a Hough transformation to the digital images of the EBSPs. The digital image can be regarded as a Cartesian grid which consists of many points (pixels), specified by the resolution used by the pattern acquisition device. Each point contains an intensity value and can be identified by its location (x, y) in the grid. The intensity value is an integer within a finite range which depends on the bit size of the data. Commonly a single 8-bit grey channel is used, which implies a range from 0 to 255. All possible lines through a point (x, y), parameterized by ρ and θ, can be described as:

$$\rho = x * \cos\theta + y * \sin\theta \tag{3}$$

ρ is the shortest perpendicular distance from the origin $(0, 0)$ of the grid to the line, and θ is the angle formed between the line and an axis of reference [Figure 6a]. A point (x, y) in the spatial domain can then be visualized as a single sinusoidal curve of constant intensity in Hough space, which may also be regarded as a Cartesian grid with axes for ρ and θ. It is common for EBSD software to allow the user to define the step size of this Cartesian grid, but usually, a step size of $1°$ is used. By transforming each point (x, y) to a curve (θ, ρ), and assigning the accumulation of intensities of overlapping curves to a single cell, high-intensity peaks are formed in Hough space [Figure 6a]. These are often called butterfly-peaks, due to the width of lines in the spatial domain, which translates to minor differences in distance ρ and a constant angle θ. The result is multiple neighboring high-intensity cells around the intersection of curves, which can easily be identified even though the originating line is fractured [Figure 6b].



(a) Hough transform of a line in the spatial domain to a peak in Hough space.



(b) Identification of peaks in Hough space.

Figure 6: Hough transformation of multiple points on a line to a single peak in Hough space, which can easily be identified. The EBSP of origin in image b is from the dataset described in Section 1.1.

Source: Adapted from the pre-study[1]

Optimization techniques to increase accuracy and performance may be used depending on the implementation. An ideal assumption is that each pixel contributes to a single line in the image instead of all possible lines, even though it might originate from an intersection between bands [24, 1]. To simulate this, the intensity of each pixel may be directly mapped onto existing peaks by a second Hough transformation, most likely improving accuracy. This can be considered as a discrete Radon transform (Radon 1917) [28], which is the generalized approach from which the

Hough transform originated [24]. The accumulated intensity R_i of a line i, or cell i in Hough space, can be calculated from:

$$R_i(\rho_i, \theta_i) = \sum_{j=0}^{J} \sum_{k=0}^{K} f(x_j, y_k) * \delta(\rho_i - x_j * \cos\theta_i - y_k * \sin\theta_i) \tag{4}$$

J and K are the numbers of pixels in each dimension of the image, $f(x_j, y_k)$ is the intensity of pixel (x_j, y_k), and Dirac delta function δ equals 1 if the pixel is part of line i and 0 otherwise [1].

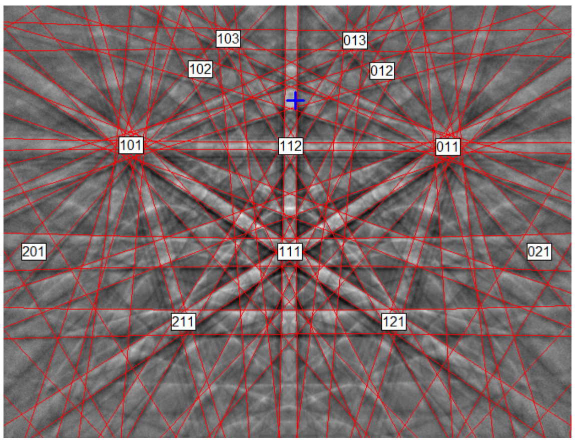


Figure 7: Indexed pattern where bands have been assigned families of planes.

Source: Oxford Instruments [29]

The problem has then been reduced from identifying lines to identifying single peaks of high intensity, which is much more applicable to be performed automatically by a computer. The next step is calculating the angles between experimental Kikuchi lines, and comparing them to theoretical interplanar angles, which are based on reflecting planes present in the crystal structure of the material [11]. This makes knowledge of the sample's structure a prerequisite, but for cubic crystal structures, it is often sufficient to specify a FCC or BCC structure. A lookup table of reference angles can be derived from theoretical geometry calculated from kinematic structure factor [11]. This also allows for the computation of relative intensities, but requires a more detailed specification of the crystal structure to be used in the indexing routine.

The experimental Kikuchi lines with the highest intensity and smallest width are compared to the lookup table and are ranked accordingly. Comparisons can either be done by *full band set indexing*, or by triplet/quadruplet indexing [11]. The former compares all identified experimental angles at the same time, while the latter puts Kikuchi lines in subsets of size 3 or 4 before each subset of bands are used to vote on a single solution. Triplet/quadruplet indexing takes advantage of possible geometry, which usually yields better results but requires more comparisons. Solutions are used to assign each Kikuchi line to a family of planes [Figure 7], from which orientation matrix can be calculated with respect to the position and angle of the pattern acquisition device [11].

2.1.2 Dictionary indexing

Increasing precision and accuracy of measured orientations have been of interest since the introduction of automated indexing. Some materials' crystallographic structures are almost indistinguishable from one another, making it impossible to determine an EBSP's origin of phase using HI, which depends on the extraction of image features. For example, ferrite (BCC structure) and austenite (FCC structure) can easily be distinguished by the use of HI, however constituent phases such as martensite have a body-centered tetragonal structure (BCT), where the central axis has a different length c than the other two axes with length a, measured as the ratio c/a. Typically $c/a = 1.04$ for martensite, and is more or less proportional with the content of carbon [30]. Although sharing a similar crystal structure with ferrite, martensite is significantly harder and less stable. There has therefore been an interest in the ability to distinguish between these two phases with the use of EBSD, e.g. EDAX explored in 2009 the approach of using IQ maps to identify martensite [30]. Later in 2021, the work of Baghdadchi et al. [31] identified martensite through a manual selection which considered grain sizes and orientations, using AZtecCrystal [18]. The master's thesis of Leth-Olsen [3], co-creator of the software developed during the pre-study, attempts to distinguish martensite from ferrite in low-carbon steel, using kikuchipy's implementation of Dictionary Indexing (DI) in combination with simulated EBSPs of martensite.

The origin of DI began in 2013 with a new approach for the simulation of dynamic EBSP by Callahan and De Graef [32], based on a physics-based forward model presented by Winkelmann et al. [33] in 2007. The new approach led to the proposal of a new framework for EBSD indexing in 2015 by Chen et al. [34]. The framework takes each measured EBSP and identifies the most similar patterns in a dictionary, which consists of a discretization of the forward model mapped onto a dense grid of Euler angles [34]. Work continued in 2016 by Singh and Graef [8], which provided orientation sampling for indexing methods with *"... a near optimal covering of orientation space"* - Singh and Graef [8]. Generating the dictionary of possible oriented patterns was improved in 2019 by Foden et al. [35], where a new cross-correlation function was implemented, allowing for a smaller-sized dictionary and consequently an increase in patterns indexed per second. The work also introduced a follow-up step where the orientations of already indexed patterns are refined using the same library from which the dictionary was generated from [Figure 8] [35].

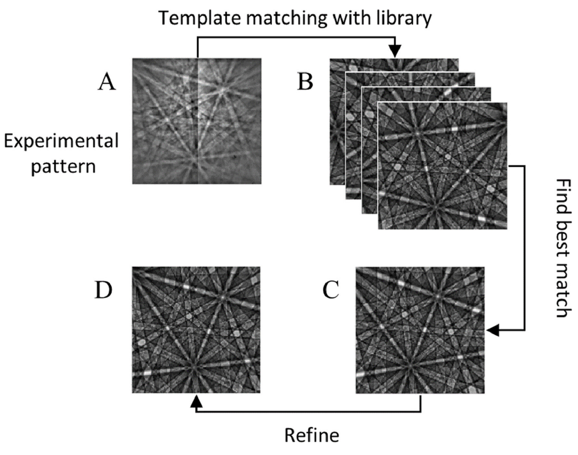


Figure 8: Steps of pattern matching, a term commonly used to describe indexing that consists of both DI and Refinement of orientations.

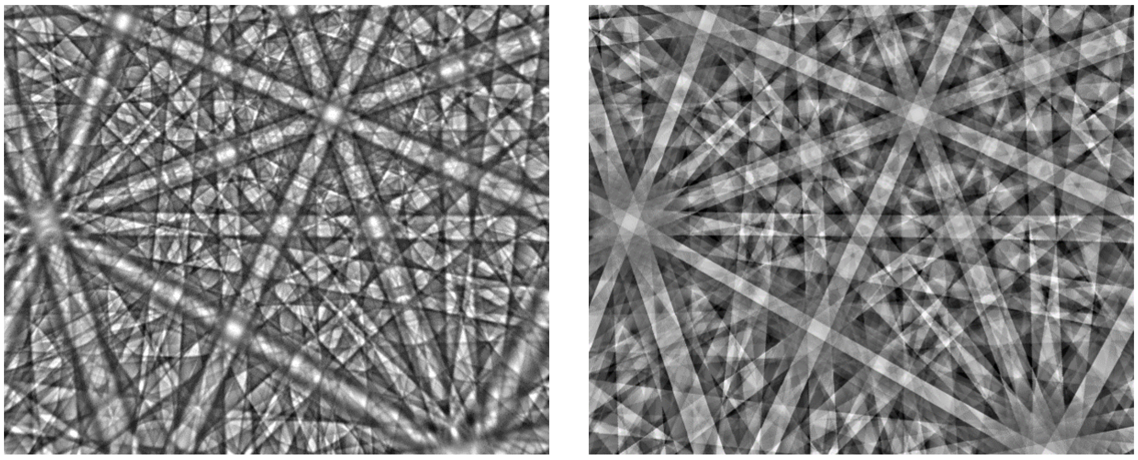
Source: Ram et al. [36]

To simulate EBSPs with correct intensity distributions, the energy of the backscattered electrons leaving the sample is modeled using a Monte Carlo simulation [32]. Both the tilt angle and crystal structure of the sample must be known. For crystalline samples, a second modulation of the backscattered electrons must be modeled for a given energy, which measures how many backscattered electrons that exit the crystalline sample in a specified direction [36]. The total count is computed for each possible symmetrically unique direction, where the number of directions is sampled to be finite [36]. The resulting intensities are projected onto the surface of a 2-sphere, which is called a master pattern. The master pattern then consists of the full range of possible orientations [11]. Master patterns can be simulated with EMsoft [14], and stored on a computer using the Hierarchical Data Format.

To generate a dictionary of simulated EBSPs from the master pattern, orientations must be sampled. The cubochoric method [37, 8] is popular due to incorporating crystallographic symmetry when sampling, and is also available as an option in kikuchipy. EBSP images are computed from the sampled orientations and are compared to each experimental EBSP to measure similarity. Some popular algorithms used to compare images are cross-correlation, normalized inner product, mutual information, and Manhattan distance [36]. The orientation of the sampled EBSP with the highest similarity score is then assigned to the experimental EBSP.

Indexed EBSPs with a misorientation within approximately $\pm 7°$ may be refined [Figure 8] by taking advantage of a shift in the vertical or horizontal directions of the sampled EBSP, due to the duality in the gnomonic projection [36]. This shift may be measured by a 2D Fast Fourier Transform cross-correlation, and with the correct pattern center, an approximation of the misorientation may be made [36]. The work of Lervik [38] implemented solutions into kikuchipy, based on the findings of Singh et al. [39], for both refining already indexed orientations, and also refining pattern center. Refinement is done by maximizing the similarity between simulated and experimental patterns, while either the orientation of the indexed EBSP or pattern center is held constant [39, 38]. A third solution which aims to refine both orientations and pattern center was also implemented, using global optimization of the six-dimensional space presented by Pang et al. [40].

The number of comparisons between experimental and sampled patterns makes DI significantly less computationally efficient than HI. The number of sampled patterns also increases depending on the complexity of the symmetry group of the sample. Work has however been done to increase indexing speeds for low-symmetry crystals. The addition of orientation refinement also allows for a more sparse dictionary, with larger sampling sizes, to be used to speed up DI without the final result losing accuracy [36]. Only the position and relative intensity of Kikuchi lines can be produced from the kinematical approach used in HI, whereas the dynamical simulated master patterns correctly reproduce intensity distributions within patterns [Figure 9] [11]. The intensity of each pixel in the patterns is considered when patterns are compared, instead of basing the solution of derived angles. This gives DI the potential for distinguishing between similar crystallographic structures, e.g. martensite and ferrite [3].



(a) Many-beam dynamical simulation (b) Standard kinematic theory

Figure 9: Comparison of an EBSP sampled from a master pattern generated using many-beam dynamical simulation (a), and theoretical geometry of an EBSP derived from the kinematic approach used in HI (b).

Source: Adapted from the simulations of Winkelmann [41]

2.2 Graphical user interface

A Graphical User Interface (GUI) is one of many design approaches that resides under the umbrella term *interaction design* [42]. Interaction design of a GUI differs from software engineering by the shift in the type of solution for the common issue at hand. While designers are concerned with what design is needed to be created in order to solve the issue, software engineers deal with the details of how to execute the design. The work of this thesis addresses both, which requires the roles of programmer and designer to be assumed. As a designer, one must know how users act and react to events presented to them through technology, so the User Experience (UX) may be adapted in an effective manner [42, p.39].

Paradigms for human-computer interaction began forming in the 1980s and were based on observations of how a single user interacts with a screen-based interface [42, p.97]. The core features were developed, which are still used today, and consisted of windows, icons, menus and pointer (WIMP) [42]. The term WIMP was later replaced by the much more generalized Graphical User Interface (GUI), which can also be applied to interfaces that do not necessarily take advantage of all features implied by WIMP. However, they both follow the model-view-controller software pattern, where only specific information of the internal software model is displayed to the user through a view [43]. The view partially consists of a controller component, where the user is presented with the available functionality. The alternative to GUI is a command-line user interface, where functionality is accessed through typing text, possible by the command-line interpreter which communicates with the software on behalf of the user. This usually requires some prior practice or knowledge of the underlying API, whereas the main advantage of GUI is that operations become intuitive and easier to master [43].

When designing a GUI, there exists a large number of principles, *"... derived from a mix of theory-based knowledge, experience, and common sense"* - Sharp et al. [42, p.57], which the designer may consider when prioritizing the User Experience (UX). The most common design principles, described by Sharp et al. [42], are visibility, feedback, constraints, consistency, and affordance.

Functions need to be **visible** to communicate that the user can interact with them, and highly influences the probability of whether the user will interact with it or not. When users interact with functionality, they expect **feedback** about the action which they just performed. Pointing users in the direction of the next step can also be a powerful feature of feedback. Both visibility and feedback can often be overlooked when designing automated functionality, where users are not properly informed of what is happening behind the scenes. Users might be confused if values are automatically changed when interacting with a completely different functionality. Although in most cases, users do not need to know every detail of the process and are instead interested in the results, however, it is still something to consider.

Users are in constant need of guidance, and **constraints** can be an effective way to restrict the user from performing actions that may result in mistakes, and should therefore not be allowed. Implementing constraints into a GUI is commonly done through shading functionality and options grey [42], but they may also completely disappear from the user's point of view. Note that while the former gives feedback to the user through the display, the latter does not, even though they both restrict the user's available actions. This knowledge can be used to reduce the number of graphical elements present on the display when feedback is redundant. Information may also be displayed in a certain way to constrain the perception of the information, e.g. putting data into a specific type of graph [42, p.60]. *First Principles of Interaction Design* is a popular set of principles by Tognazzini [44], and describes a duality within the topic of autonomy;

1. *Enable users to make their own decisions, even ones aesthetically poor or behaviorally less efficient*
2. *Exercise responsible control*
3. *Use status mechanisms to keep users aware and informed*

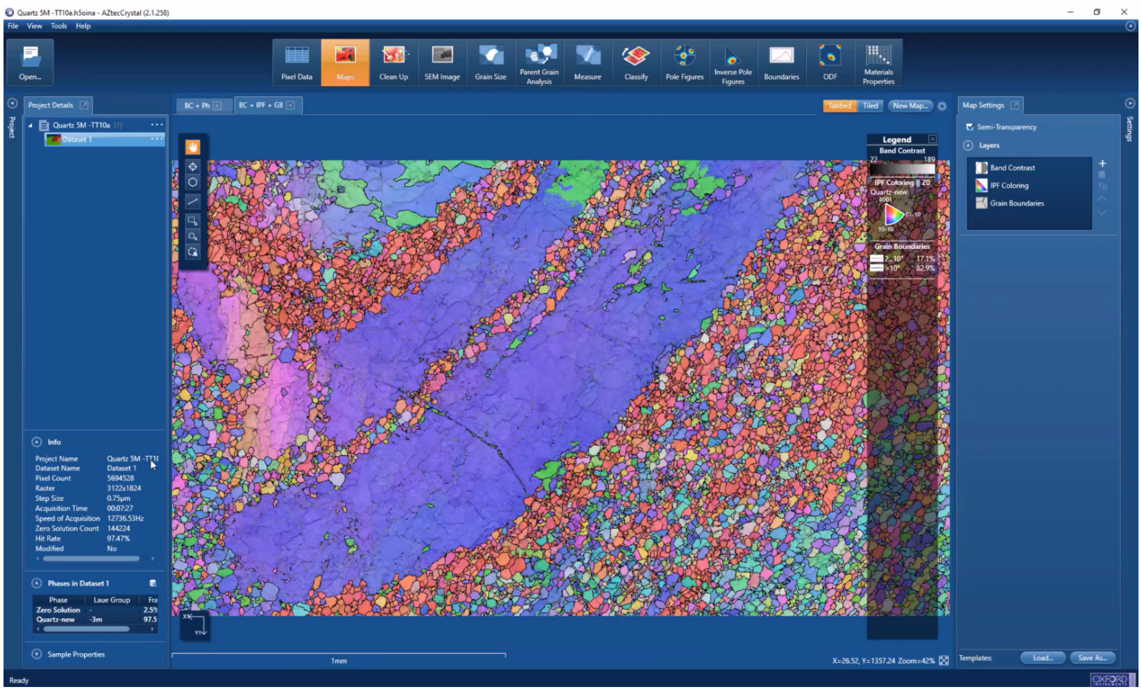
Whereas the 1st principle allows users the freedom of customizing and interacting with the design to their liking, to avoid users feeling frustrated and angry, the 2nd states that necessary control is needed to avoid making fatal mistakes [44]. An example of a disallowed interaction would be one that will crash the software. One should use the 3rd principle to exercise this control, e.g. by making options unavailable to the user depending on the state of other options.

To indicate that similar functionality is used for achieving similar tasks, **consistency** within the design is key [42]. This applies not only to how functionality looks but also to how users expect certain actions to be performed. An example of this is how on Windows one usually right-clicks on an object to open a context menu with additional options for that object. Users of Windows have come to associate this action with bringing up more options, and creates the expectation that other software running on the same system adapts the same premise for achieving the same result. Consistency influences the rate at which users learn to navigate the GUI, due to users being only required to learn a single mode of operation to operate on similar GUI-elements. **Affordance** is a principle that is similar to consistency, where the visual representation of the object makes it perceptually obvious how one should interact with it. *To afford means "to give a clue" (Norman, 1988)* - Sharp et al. [42, p.63]. In practice, this is done by giving GUI-elements distinct looks that correspond to their functionality, e.g. a button having a border that empathizes contrast against the environment it resides in, to communicate its ability to be clicked.

2.2.1 Existing GUIs for EBSD

To better understand how existing solutions for EBSD software implement GUI, both OIM DC [16] and AZtecCrystal [18] from Table 1 are examined. Additionally, a web-based prototype for EBSD is inspected, which was developed by Ole Natlandsmyr during his master's thesis in 2021 [45].

Oxford Instruments advertises AZtecCrystal as *"the most modern, comprehensive Electron Backscatter Diffraction (EBSD) data processing software package on the market today"* [18]. AZtecCrystal was launched in 2019, which shows in its modern design that does not utilize GUI-elements native to the system [Figure 10]. Navigating the different modules of AZtecCrystal is done through the top-centered menu bar, which consists of buttons with both icons and text [Figure 10]. The text is short but precise and captures the essence of the functionality of the module it leads to. Each module offers a different main component, but it is always placed at the center of the screen, where it takes up the majority of the screen space. Additional details and settings are placed respectably on the left and right sides of the main component. Because of this consistency throughout the software, some knowledge can be transferred between modules, contributing to a quicker learning experience. The different panels can also be undocked from the main application window, and be displayed on different monitors, utilizing more available screen space to improve visibility.



Info

Project Name	Quartz 5M -TT1(
Dataset Name	Dataset 1
Pixel Count	5694528
Raster	3122x1824
Step Size	0.75μm
Acquisition Time	00:07:27
Speed of Acquisition	12736.53Hz
Zero Solution Count	144224
Hit Rate	97.47%
Modified	No

Phases in Dataset 1

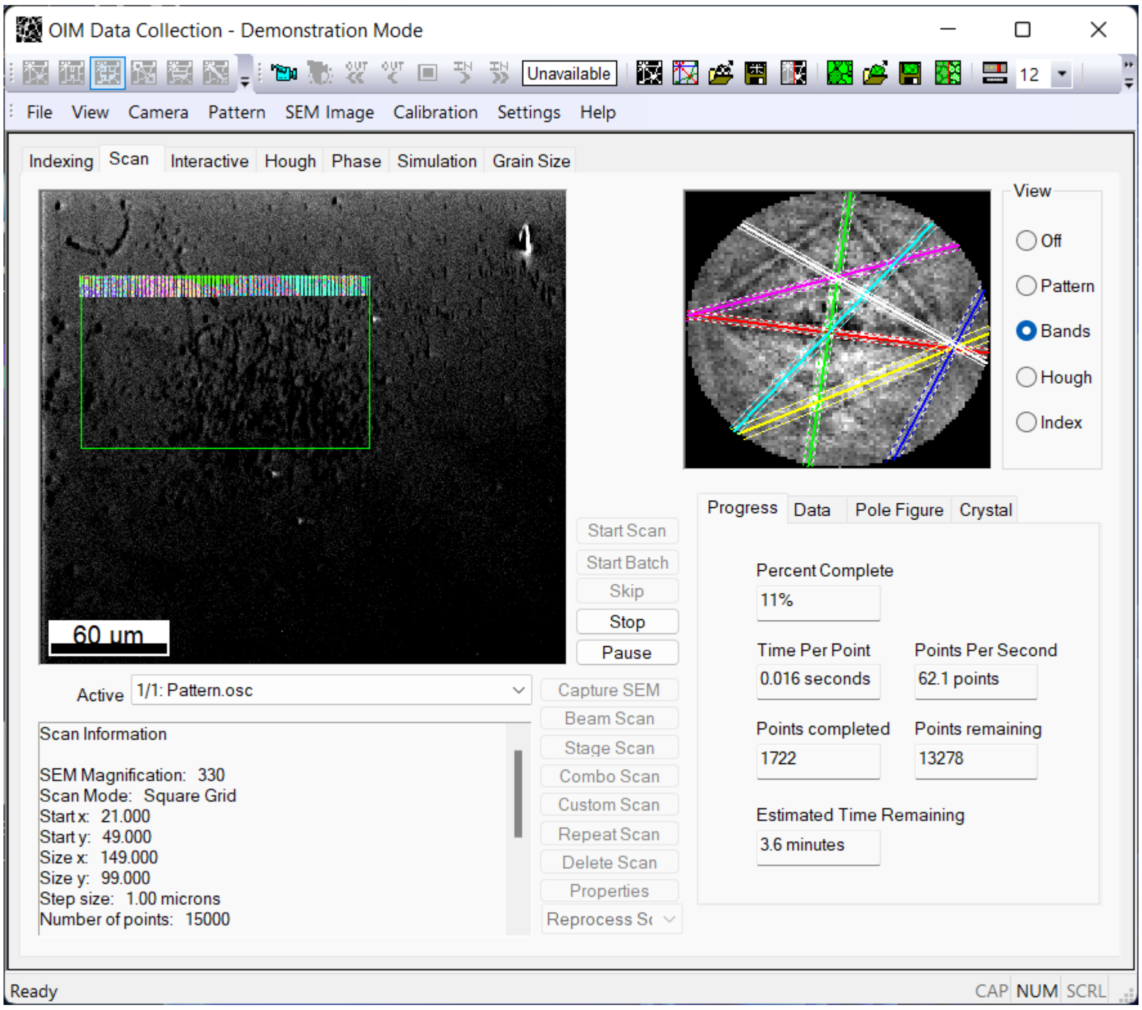
Phase	Laue Group	Fra
Zero Solution	-	2.5%
Quartz-new	-3m	97.5

Sample Properties

Figure 10: GUI of AZtecCrystal which shows the inspection of an IPF map.

Source: Demo uploaded to Oxford Instruments' official channel on YouTube [46].

The first OIM EBSD system on Windows was introduced in 1997, and EDAX has since released many iterations of the software and other analytical tools related to EBSD [47]. OIM DC, version 7.3.1 from April of 2017, adapts GUI-elements native to Windows [Figure 11]. One may argue that this makes the software look less modern compared to AZtecCrystal. This style is however common throughout many different types of software, due to the reason that it achieves a simple and intuitive design. On the other side, one does get the impression that the design of AZtecCrystal is carried out with a more professional look. The GUI displays a lot of different controller elements and information at the same time, which may be seen as cluttered by some users. Information and actions are grouped together where it makes sense, e.g. all buttons are stacked together, and progress information resides within its own bordered sub-page. Although the large number of buttons provides a lot of functionality, it does make navigating to the desired button a bit more tedious, especially for beginners. There are a lot of actions available to the user through the 3 rows of menu bars at the top, which are always displayed unless toggled off. The software offers undocking of some menu bars, but not panels at all, making this feature less efficient here than in AZtecCrystal. Although a great tool with many functionalities for EBSD, the interaction design of OIM DC seems to leave something to be desired.


OIM Data Collection - Demonstration Mode
Unavailable
File View Camera Pattern SEM Image Calibration Settings Help
Indexing Scan Interactive Hough Phase Simulation Grain Size
60 um
View
Off
Pattern
Bands
Hough
Index
Start Scan
Start Batch
Skip
Stop
Pause
Capture SEM
Beam Scan
Stage Scan
Combo Scan
Custom Scan
Repeat Scan
Delete Scan
Properties
Reprocess Sc
Progress Data Pole Figure Crystal
Ready
CAP NUM SCRL

View: [✓ Bands]

Active 1/1: Pattern.osc

Scan Information

SEM Magnification: 330
Scan Mode: Square Grid
Start x: 21.000
Start y: 49.000
Size x: 149.000
Size y: 99.000
Step size: 1.00 microns
Number of points: 15000

Percent Complete: 11%

Time Per Point: 0.016 seconds

Points Per Second: 62.1 points

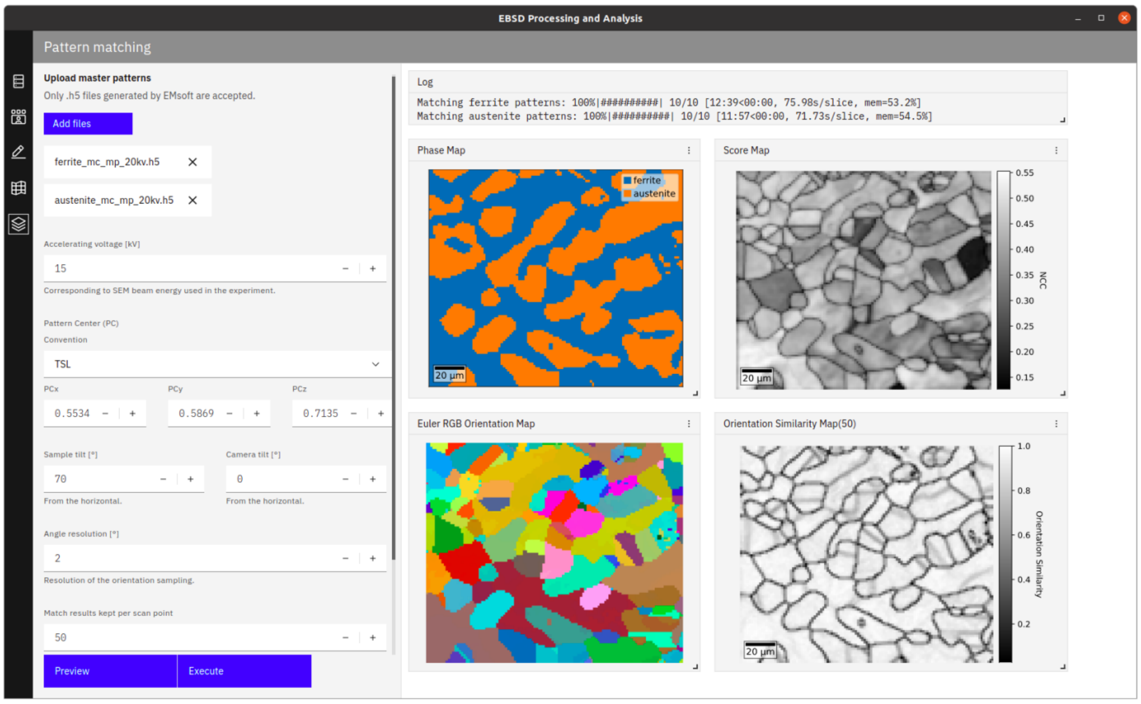
Points completed: 1722

Points remaining: 13278

Estimated Time Remaining: 3.6 minutes

Figure 11: GUI of TSL OIM Data Collection when indexing a dataset. The image is from the advanced demo mode of OIM DC v.7.3.1, where a SDSS with a sigma phase is indexed.

The work of Natlandsmyr [45] can in many ways be regarded as the predecessor to the software developed in this thesis. The prototype also aims to allow access to kikuchipy's DI implementation through a GUI, but the chosen approach is web-based instead of a local application on the computer. This solution introduces the need for a web server, which must either be hosted locally on the computer or externally. The design of the prototype has a complexity that resembles entry-level, which it achieves by its simple design [Figure 12]. The menu bar on the left shows different steps in the indexing process and provides an order to the intended workflow. Consistency can be found throughout the prototype, where the controller component is always presented on the left, where users insert inputs, while information and previews are shown on the right. Due to the workflow only targeting DI, the result resembles more a single tool rather than a toolbox of kikuchipy's functionality.



Upload master patterns
Only .h5 files generated by EMsoft are accepted.

ferrite_mc_mp_20kv.h5
austenite_mc_mp_20kv.h5

Accelerating voltage [kV]: 15
Corresponding to SEM beam energy used in the experiment.

Pattern Center (PC)
Convention: TSL
PCx: 0.5534
PCy: 0.5869
PCz: 0.7135

Sample tilt [°]: 70
From the horizontal.

Camera tilt [°]: 0
From the horizontal.

Angle resolution [°]: 2
Resolution of the orientation sampling.

Match results kept per scan point: 50

```
Log
Matching ferrite patterns: 100%|##########| 10/10 [12:39<00:00, 75.98s/slice, mem=53.2%]
Matching austenite patterns: 100%|##########| 10/10 [11:57<00:00, 71.73s/slice, mem=54.5%]
```

Figure 12: GUI of the prototype created by Natlandsmyr, where DI results of a SDSS are presented.

Source: Natlandsmyr [45]

By considering each GUI's strengths and weaknesses, developing a correctly balanced interaction design might be achieved. Takeaway ideas are the simple design of Natlandsmyr's prototype, the functionality and grouping of information used in OIM DC, and the interface of AZtecCrystal which provides good visibility.

2.3 Usability testing

To ensure an interaction design that satisfies the needs of the users, a usability test may be deployed. This method of progress already arose in the early 1980s, when computer scientists designed multiple different User Interfaces (UIs) to analyze and predict the performance of users carrying out tasks [42]. The studies were based on cognitive theories in order to determine the best approach for presenting users with operations, which takes into account humans' memory limitations [42, p.100]. The work of Weiser [48] in 1991 caused a significant shift in how these studies were carried out. Weiser predicts that computers and displays will become embedded in everyday items, and such require interaction designs that do not conflict with or interrupt the task being performed by the user [48]. This has in many ways already been realized, as computers and displays are now found in everyday household appliances, e.g. smart fridges, sewing machines, and light switches. It is especially important that users do not feel frustrated interacting with these technologies, considering that they are essential to everyday tasks. As a result, usability tests shifted from the study of only the human cognitive mind, to also considering the social context, emotions and environment in which the design is being used [42, p.97].

Figure 13: GUI displayed on a portable device, developed by Xerox Palo Alto Research Center, which may be connected to a variety of workstations.

Source: Weiser [48]

The measuring of usability of the design refers to how effective interaction is to learn, use, and be enjoyed by the user [42, p.50]. This may be done by directly asking the user questions, or gathering data related to effectiveness, efficiency, utility, safety, learnability and memorability [42, p.50]. Effectiveness takes into consideration all other aspects and gives an overall indication of whether the user can fulfill the purpose of the software. Efficiency is about reducing the number of steps needed to complete tasks, while the utility is concerned with providing enough functionality to do so. All of this should be carried out safely so that the user is not put in an undesirable situation. Such situations may cause fears perceived by the user, who want to avoid the consequences of making mistakes [42, p.51]. Furthermore, how easily functionality can be learned and memorized is especially important when the design's target group consists of users who have less experience with similar designs, e.g. students who are new to EBSD software.

Anticipating the user's needs requires the designer to have a deep understanding of the task domain and the user [44]. This includes anticipating what information and tools the user needs for each step of the process, which can be confirmed by sufficient usability testing. Usability tests should also be used for discovering what information should be communicated to the user, and should ideally be used regularly throughout the project to iterate design until the user performs at a satisfactory level [44].

First, the goal and reasons for gathering data must be established, so that the procedures and techniques used can be tailored to ensure the desired data is gathered [42, p.324]. Secondly, the participants must be chosen so that the targeted population represents the user group of the design. This is called sampling, and saturation sampling is when the whole population includes all types of members in the user group [42, p.324]. The appropriate relationship between the participants and the data gatherers must also be defined, which can be done by properly informing the participants of the cause of the usability test. Depending on what personal information is gathered, and the rules applied by the country, an informed consent which protects both participants and data gatherers might be needed. A type of triangulation, *investigation of a phenomenon from (at least) two different perspectives* - Sharp et al. [42, p.327], should be used to validate the results to some extent. Some common types are triangulation of data, meaning data is collected by varying sources, time, location and people, and methodological triangulation, where different techniques for gathering data are used [42, p.237]. To make sure that the planned usability test is viable, it is common to perform a pilot study before the main study [42, p.328].

Conducting a usability test can be done through the use of many different formats. Observations, review forms, and interviews are appropriate for testing software design. The most flexible technique for recording data is taking notes of observations done in a controlled environment [42, p.330]. Participating users are placed in the appropriate environment, e.g. a laboratory where research software is used, and given a set of tasks to perform. The tasks are supposed to highlight key functionality in the design and are the main source of data in the usability test. The formulation of the tasks should give just enough information so the user knows what the end goal of the task is. It is up to the user to figure out where to proceed to reach the end goal, with the only guidance being the hints provided by the design.

Another strategy, which better highlights the participant's thoughts, is an interview. These may be conducted in a structured or unstructured manner and may lead to improved feedback, by allowing participants to go in-depth on a topic. Unstructured interviews do however produce data that is not consistent across participants, due to different paths being explored. An alternative to interviews is a questionnaire, which has the advantage of being easily distributable to participants online [42, p.345]. These can contain both closed or open questions, but might lack the depth of data which is obtained by a dialog in an interview. Additional efforts must also be put into precisely worded questions, but these have the advantage of being more comparable between participants, due to answering the same structured questionnaire [42, p.346]. A viable option is to use these in conjunction with interviews, as long as there is an incentive to complete the questionnaire.

3 Development and methods

This section describes the continuation of *EBSD-GUI dev*, the version of the software which resulted from the work of the pre-study [1]. In addition to describing the iterations that the software went through during this thesis, the section also includes specifications for the software, how the software is distributed and published, and the establishment of an open-source community.

3.1 State of the software

Although complete with many features, the state of EBSD-GUI dev was somewhat rough and unpolished. This was of no surprise, as minor issues and bugs were intentionally given lower priorities, in order to implement all features within the deadline of the pre-study. These issues were instead left to be figured out as part of this thesis. Additional issues were also highlighted by the project's supervisor, Prof. Hjelen, who had used EBSD-GUI dev on multiple Windows computers for about a month.

Known issues of EBSD-GUI dev can be summarized by:

- Running the packaged executable on some Windows system can result in an `ImportError` when importing the `QtCore` module [Figure 15].
- Threading of specific tasks on macOS can result in leaked semaphore objects, instantly crashing the software.
- HI saves a copy of the dataset, taking up unnecessary storage space on the system [1].
- HI performed on lazy loaded dataset results in the following error:
  `Cannot determine Numba type of <class dask.array.core.Array>`
- Using relative paths to locate additional resources like icons, logos, and static text files, might fail when running the executable on some systems.
- Starting multiple tasks at the same time will result in the terminal being flooded with messages, making it very difficult to determine the state of the tasks.
- There are no written tests for existing code, meaning there is no automatic quality assurance for new additions and changes [1].
- Spelling errors and incorrect naming of some features.

Failing to execute the software on some Windows systems was most likely a compatibility issue with Qt6, which had been overlooked. Therefore a more thorough examination of Qt-supported platforms was needed. EBSD-GUI dev on macOS was very unstable, as the software would often crash due to leaked semaphore objects, which are objects shared between threads that need to be synchronized. Observations implied that this would happen when datasets were loaded, or maps were saved as images, inside threads. It was already known that the Matplotlib library was not thread-safe unless under certain conditions [49]. *You may be able to work on separate figures from separate threads. However, you must in that case use a non-interactive backend (typically Agg), because most GUI backends require being run from the main thread as well* - Hunter [49]. Precautions had already been made from this statement but only appeared to be working on Windows.

Although confirmed to work to some extent in a developer environment on macOS, a portable app had yet to be exported and tested. The software's implementation of HI was based on a pre-existing Jupyter Notebook used at the Electron Microscopy Lab of NTNU, which also included some of the above issues. This could be improved by basing the code of the HI Notebook provided by kikuchipy instead [5]. For example, the implementation of HI in EBSD-GUI dev required phases to be loaded from master patterns, even though the only required information is of the phase's microstructure. The user should be able to specify this themselves.

The only indication of a task being performed by the software was the print statements from the method being performed, which appeared in the GUI's terminal. End users should not be required to read through a large and unsorted log of messages to see what is happening during each task. Therefore it was necessary to extend this feature to improve the way information is communicated to the user. Another possible improvement was writing tests for existing code. The lack of tests meant a higher probability of functions breaking unnoticed, resulting in a less effective development process with new issues suddenly appearing. This also emphasized the need to conduct a usability test.

A development roadmap was created, and would include fixing the above issues, while also implementing the following new features:

- Support for packaging the software into an app structure that can run on macOS.
- Support refinement of orientations in crystal maps produced by HI.
- Support more phase structures when performing DI and refinement of orientations.
- Improve refinement speed by using the optimization algorithm Nealder-Mead [50] from the Python library NLopt [51], instead of the implementation from SciPy [52].
- Option for users to specify crystal microstructures, instead of being derived from master patterns, when doing HI [1].
- An improved detector-navigator for inspecting patterns, which can replace the interactive plot functions from the Python library HyperSpy [53].
- The ability to resize GUI elements inside the main application window, to better fit their content and adapt to different screen sizes and scaling configurations.
- Window with an overview of the tasks which are being performed, and includes the ability to cancel tasks.
- Create a user manual that gives a quick overview of the software [1].

Incorporating a proper usability test to take place midway during development, meant there was a low probability of all features being ready for testing. It was therefore important to distribute the goals of the roadmap across deployment-ready iterations of the software. This also meant fixing issues and overhauling existing features would take priority, to ensure a stable experience during the test. Most importantly, distributions of these updated versions would need to be made for both Windows and macOS. The planned iterations of the software are shown in Figure 14. Work would start with overhauling the existing features of EBSD-GUI dev. *EBSD-GUI Pre-release* was to become the version deployed in the usability test, and would include the new features that were ready for testing. User feedback would be taken into account to continue the development of a public v0.1.0 release of the software, later to be renamed *EBSP Indexer*. Later fixes for this version would be released as minor patches. If enough time was available, additional new features and other enhancements would be implemented as part of a v0.2.0 release.

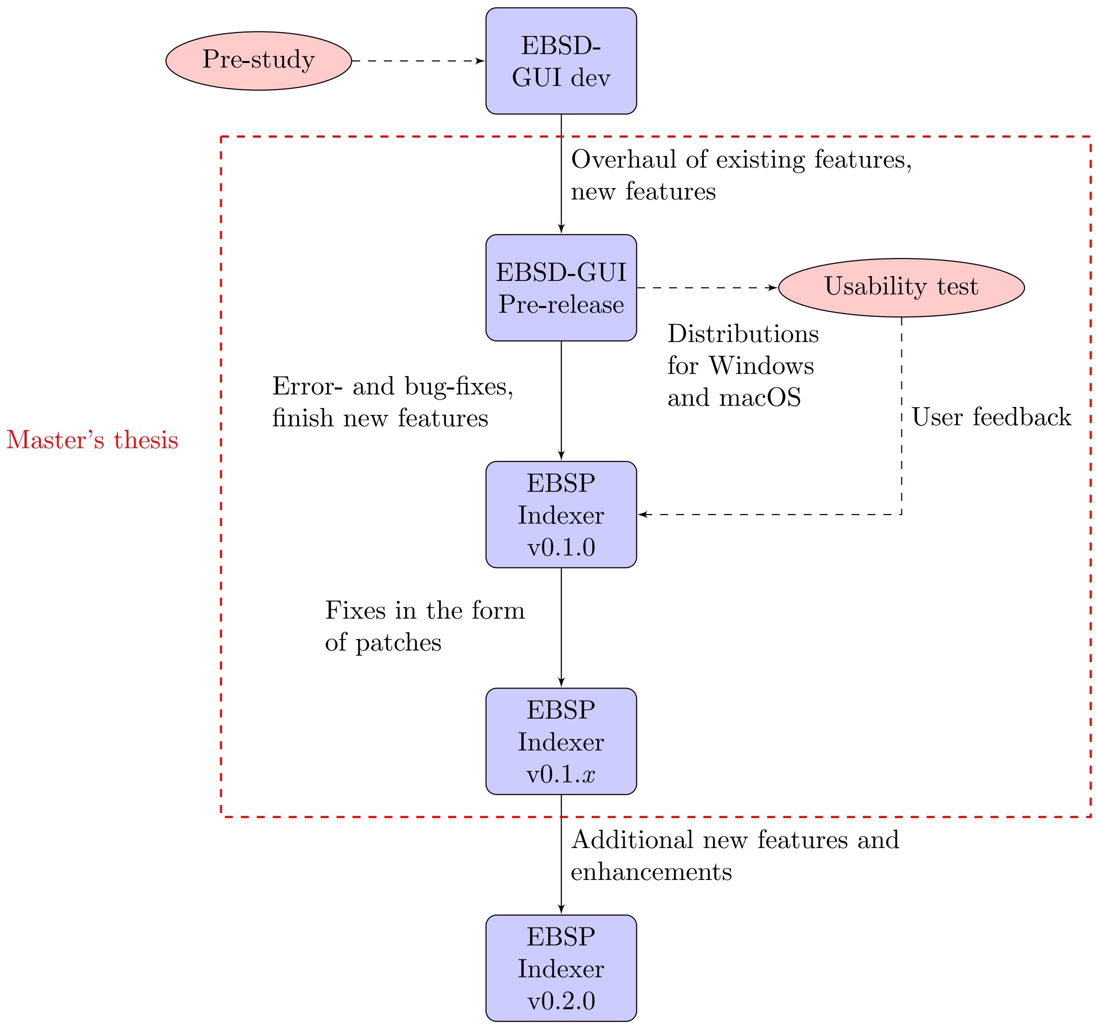

Pre-study
EBSD-GUI dev
Overhaul of existing features, new features
Master's thesis
EBSD-GUI Pre-release
Usability test
Distributions for Windows and macOS
Error- and bug-fixes, finish new features
User feedback
EBSP Indexer v0.1.0
Fixes in the form of patches
EBSP Indexer v0.1.x
Additional new features and enhancements
EBSP Indexer v0.2.0

Figure 14: Flow chart showing the planned development process of the software, which includes stable iterations of the software as blue boxes. Activities within the area marked with red dashes are discussed in this thesis. Distributions are made for the usability test, and user feedback is received. The x in EBSP Indexer v0.1.x indicates the number of patches needed. Version 0.2.0 is not within the scope of this thesis, but is described as future work in Section 7.

3.2 External packages and libraries

Seeing as some of the issues in EBSD-GUI dev might have been caused by incompatibilities and lack of support in some dependencies of the software, a re-examination of packages and libraries was in order. On Windows, the software consists of about 150 third-party Python libraries. Most of these are dependencies of kikuchipy, some of which are optional. These include PyEBSDIndex, orix, HyperSpy, diffpy.structure, diffsims, and many more. In addition, PyOpenCL is an optional requirement for PyEBSDIndex, which enhances indexing speed by utilizing the system's Graphics Processing Unit (GPU). The GUI is created using the PySide6 bindings for Qt, and the project is exported by PyInstaller. It is important to be aware of the licensing terms implied by these libraries, as they will determine the final licensing of the software when published.

Reasoning behind the specifications of the software was described in the pre-study [1], and can be summarized as follow:

- Windows and macOS, due to the popularity of these platforms among students.
- Python, because of its large number of available third-party libraries, and easy-to-learn syn-

tax, which lowers the barrier for other developers to contribute.

- kikuchipy, as it offers many features, among pattern matching capabilities, and is continually being supported with new releases.
- Qt, because of the ability to easily create traditional desktop applications, which offers an OS-native look.
- PyInstaller, due to being the most effective and reliable method for exporting applications from Python scripts, and has a large open-source community.

3.2.1 kikuchipy

The core idea of this project builds upon the functionality of kikuchipy, a Python library for processing, simulating, and analyzing EBSPs [5]. By being open-source, kikuchipy makes a serious contender to commercially available software like OIM DC [16] and ESPRIT DynamicS [17]. Some key features of kikuchipy are the implementation of DI, PC-optimization, and refinement of orientations in a crystal map. Analysis of multi-dimensional data is possible through HyperSpy, which kikuchipy extends upon [5]. Performance-heavy computations are done with the Numba library, which adds a Just-In-Time (JIT) compiler to Python, accessible through the addition of function decorators, allowing native machine code speed [54]. The implementation of Numba in kikuchipy uses automatic parallelization of functions, meaning multiple cores in the Central Processing Unit (CPU) are utilized to run computations in parallel. Numba also offers the possibility to target GPU architecture available through the Nvidia CUDA API [54], but this has yet to be utilized in kikuchipy. To handle the processing of data larger than the system's available random-access memory, the Dask library is used in combination with Numba decorated functions. Dask then handles both the pipelining and the parallelism [55], and the pre-study of Lervik [56] implemented larger-than-memory dictionary simulation into kikuchipy. The licensing of kikuchipy is determined by HyperSpy's more strict licensing, the GNU General Public Licence (GPL) v3.0 [57].

Due to kikuchipy's many dependencies, importing the library's classes and functions can take several seconds. As an example, importing `class EBSD` from the `kikuchipy.signals.ebsd` sub-module might take 5-7 seconds on a 6-year-old Linux system [58], due to the large import list of the sub-module. To avoid this, both kikuchipy and HyperSpy use lazy imports, meaning the import statement itself takes less than a second, while the long loading times of underlying libraries are delayed until they are first needed during runtime [58]. Note that this only applies to cases where kikuchipy is imported without accessing sub-modules directly, e.g. `import kikuchipy as kp`.

Development during the pre-study utilized version 0.7.0 of kikuchipy, but in early February 2023 version 0.8.0 was released with many new additions, changes and fixes. The ability to interact with PyEBSDIndex through kikuchipy's Application Programming Interface (API) was introduced, giving access to more convenient methods for performing HI. It was therefore decided to upgrade the project's version of kikuchipy to take advantage of these changes when overhauling the existing HI implementation. During development, kikuchipy would also release a few patches that included bug fixes and minor changes.

3.2.2 HyperSpy

HyperSpy aims to make it easy to apply analytical procedures that operate on multidimensional datasets [53]. External packages are able to register their own signals and components to fit their particular data type. For example, lazy loading of EBSD data in kikuchipy is made possible by extending HyperSpy's `LazySignal2D` class.

There exists a package named HyperSpyUI that introduces a UI component of HyperSpy, similar to what this project tries to achieve by building upon kikuchipy. The UI even has a built-in interactive Python console, using the libraries IPython Kernel for Jupyter and Jupyter QtConsole, which *retains the raw power of HyperSpy* - Fauske [59]. An interactive Python console was also developed as part of EBSD-GUI dev but was built without any third-party console libraries. As a result, EBSD-GUI dev offers a less feature-rich console than HyperSpyUI, but one that is compatible with more libraries, e.g. PySide6, and can easily be maintained for future versions. The UI package is still being maintained, and version 1.2 was released in March 2023.

For this project, almost all interactions with HyperSpy are done through kikuchipy, meaning direct references to its API will not be common in the software's source code. Version 1.7.3 of HyperSpy is used because it is the minimal requirement for the current version of kikuchipy.

3.2.3 orix

Storing of orientations and phases in a crystal map is handled by the Python library orix, which implements objects and functions that represents 3D rotation vectors that accounts for crystal symmetry [60]. The data returned from the indexing procedures of kikuchipy are stored in an instance of the `CrystalMap` class from orix. The class includes plotting data by wrapping the `matplotlib.axes.Axes.imshow()` method from Matplotlib, which visualizes data as images. This makes it trivial to save phase- and IPF maps acquired through indexing. The library is lightweight compared to kikuchipy and HyperSpy, and does not implement lazy imports. Similarly to kikuchipy, orix follows the terms and conditions of GPL v3.0.

3.2.4 diffpy.structure

Both kikuchipy and orix depend on accurate descriptions of phases present in the dataset, which is handled by the diffpy.strucutre module [61]. Just like orix, it offers custom objects that store crystal structure data, such as atomic coordinates, displacement and occupancy. The module includes the classes `Structure`, `Atom` and `Lattice`, and also *includes definitions of all space groups in over 500 symmetry settings* - Juhás et al. [61]. The module is part of the larger DiffPy-CMI modeling framework [61], and inherits the framework's custom open-source license agreement. The licensee is granted a royalty-free nonexclusive license, and copies of the source code must include the copyright notice and the software license agreement itself. Modifying and redistributing the source code is also allowed, as long as users are notified of the modifications.

3.2.5 PyEBSDIndex

The PyEBSDIndex library includes objects and methods for performing HI on a dataset of EBSPs [6]. This is done by using an object of the class `EBSDIndexer`, where the user can specify the arrangements of phases present, PC coordinates, sample tilt towards the detector, and camera elevation. In addition, the Hough transform can be customized through a number of parameters, e.g. what fraction of the pattern should be considered, and also the number of detected bands used in voting. As of version 0.1.1 of PyEBSDIndex, the Hough-based indexing procedure can only differentiate between phases with FCC and BCC arrangements and does not take into account lattice parameters. Consequently, the indexing cannot differentiate between two phases of a similar arrangement. Future releases of PyEBSDIndex might address these limitations by utilizing `PhaseList` from orix, and `Structure` from diffpy.structure. OIM DC, and other commercial software, offers HI of multiple phases with different and/or similar arrangements, e.g. indexing of steel with austenite, ferrite and sigma phases. PyEBSDIndex needs to offer similar indexing capabilities in order to stay a serious contender.

The library was developed by employees of the US Naval Research Laboratory (NRL) and resides in the public domain [6]. Licensing is similar to the restrictions of diffpy.structure, and states the following: *This software can be redistributed and/or modified freely provided that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified* - Rowenhorst [6].

As mentioned above, kikuchipy v0.8 wraps the methods of PyEBSDIndex and makes it possible to use a sequence of PCs that is equal in size to the number of patterns. Each pattern is then assigned a PC, which compensates for variations that might be present in PC if the ROI is large. This provides more accurate results than using a single mean PC for all patterns. HI on a dataset that is lazy loaded through kikuchipy requires PyOpenCL, an optional dependency of PyEBSDIndex. This is because of a limitation in PyEBSDIndex, which can only handle Dask arrays when they are computed on the GPU with PyOpenCL. Dask arrays are used for the lazy storing of data. The CPU implementation uses Numba kernels, which assumes NumPy arrays instead [62].

3.2.6 PyOpenCL

PyOpenCL offers access to OpenCL, an open-source API for parallel computations, through Python code [63]. The API takes advantage of a system's available GPUs, which consists of hundreds of processing units, to do performance-demanding computations in parallel. This is also available for integrated graphics, a GPU built into the computer's CPU, which is commonly found on laptops. The code which defines the actual computations is written in C++, and is passed as Python strings to PyOpenCL methods. *PyOpenCL is open-source under the MIT license and free for commercial, academic, and private use* - Klöckner et al. [63].

PyOpenCL has been available as an optional dependency of PyEBSDIndex since release 0.1.0. It was however not taken advantage of in EBSD-GUI dev. This could greatly improve the performance of HI, and would achieve pattern indexing speeds similar to the ones found in commercial software like OIM DC. It was therefore made a priority to package the software to include PyOpenCL.

3.2.7 Qt and PySide

Development of the GUI of the software, is done with PySide6, a library which offers the official Python bindings for the Qt6 framework [64]. The Qt framework has become a large and open solution, titled The Qt Project. Although mainly overseen by The Qt Company, multiple leading companies, and organizations contribute to its development [65]. The core idea is to offer a toolbox for developers to create and customize graphical control elements that automatically translate on a cross-platform level. The framework is written in Qt-extended C++, which before compilation is parsed to regular C++ code by the Meta-Object Compiler (MOC), a preprocessor, which means extended features such as signals and slots can be compiled by any standard C++ compiler, e.g. Clang or the GNU Compiler Collection (GCC) [65].

Different elements can communicate with each other through the `Signal` and `Slot` mechanism, a central feature of Qt. This mechanism is often known as the observer pattern in object-oriented programming [1]. Objects, known as observers, subscribe to an interactive object, known as the subject. In Qt, this is done by the class of the subject having a `Signal` property, which emits whenever a condition is met. Subscribing is done through observers connecting global or class methods, marked with a `Slot` decorator in PySide6, to the `Signal` of the subject. Emitting the `Signal` will then notify all connected observers, and their respectable connected methods will, as of Qt v4.6, be invoked in the order in which they were connected [66]. As signals are class properties, multiple instances of the same class might share the same `Signal` instance. To solve differentiation between instances, signals can be associated with a datatype that they can emit, allowing a subject to send messages to the observers.

Qt also has a system that allows developers to utilize threads. Utilizing threads is important to avoid the GUI becoming unresponsive while running slow and demanding tasks [1]. Threads were implemented into EBSD-GUI dev at early stages, as indexing can take a considerably long time due to operations being performed on large amounts of data [1]. Instead of using the `QThread` class, a custom `Worker` class was created, which inherits from `QRunnable`. The `worker` object receives a method when initialized, and is passed to the global instance of `QThreadPool` by calling `QThreadPool.globalInstance().start(worker)`. The method is called if a thread is available, or queued for later execution if none are available. The advantage of using the `Worker` class instead of Qt's `QThread`, is the ability to add additional code that should always be executed when a thread is started or finished. The motivation for implementing this in EBSD-GUI dev was to redirect the standard output stream, so print statements from threads would appear inside the GUI, which is running in the main thread [Code 1] [1]. Signals and slots were used to convey the print statement from one thread to another, and would else lead to instabilities, eventually crashing without an error message. One should avoid code that leads to threads directly accessing GUI elements that live in the main thread, also known as the GUI Thread in Qt [67].

```
⋮
@QtCore.Slot()
def run(self):
    """
    Initialize the runner function with passed args, kwargs, and redirects the output.
    """
    with redirect_stdout(self.threadedStdout), redirect_stderr(self.errorRedirect):
        self.fn(*self.args, **self.kwargs)

⋮
```

Code 1: Redirecting output from a method that is called in a separate thread. To avoid crashes and instability issues, `redirect_stdout()` and `redirect_stderr()` must be called inside the run method with the `Slot` decorator. Taken from the EBSD GUI dev version of *utils/worker.py*.

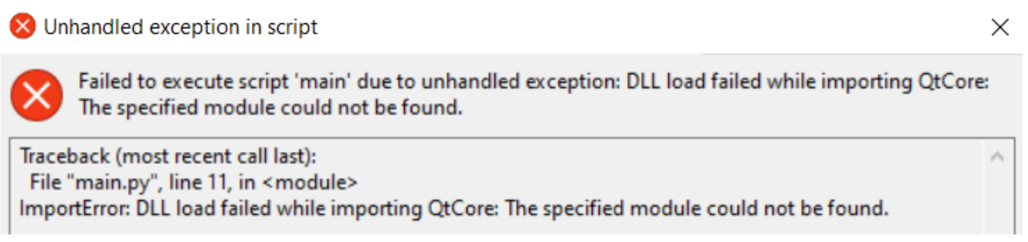
Depending on the Qt version, the lists of supported platforms are different [68]. Table 2 gives an overview of what platforms are supported by the Long-Term Support (LTS) versions of Qt5 and Qt6. In addition, Qt6 only supports x86_64 and arm64 architecture, while Qt5 supports both x86_64, arm64 and x86. This together with Table 2 implies that Qt6 cannot run on Windows 7 and Windows 8.1, nor a 32-bit based system in general.

Table 2: Overview of supported platforms for Qt 5.15 LTS and Qt 6.5 LTS.

Source: Adapted from Qt Documentation about Supported Platforms [68].

Platform	Qt5 LTS	Qt6 LTS	Compiler/Build Environment
Windows 11 21H2	Yes	Yes	MSVC 2019-2022, MinGW 11.2
Windows 10 21H2	Yes	Yes	MSVC 2015-2022, MinGW 8.1
Windows 10 (1809 or later)	No	Yes	MSVC 2019-2022, MinGW 11.2
Windows 7 / 8.1	Yes	No	MSVC 2017-2019, MinGW 8.1
macOS 11, 12, 13	Yes	Yes	Xcode 13 - 14
macOS 10.13	Yes	No	Xcode 11 - 13
Windows on ARM	No	Yes	MSVC 2019-2022

When it was decided during the pre-study to build the project's GUI with Qt6, the supported platform versions were not considered to such an extent. The main focus was compatibility between Windows and macOS, and as a result, the incompatibilities were discovered early in this thesis when EBSD-GUI dev was tested on multiple versions of Windows. The unhandled exception that occurred is shown in Figure 15.


Unhandled exception in script

Failed to execute script 'main' due to unhandled exception: DLL load failed while importing QtCore: The specified module could not be found.

```
Traceback (most recent call last):
  File "main.py", line 11, in <module>
ImportError: DLL load failed while importing QtCore: The specified module could not be found.
```

Figure 15: Unhandled exception in the script when running Qt6 on an unsupported platform or one that is missing runtime libraries. The exception is an `ImportError`.

It was also discovered that Microsoft Visual Studio C++ Redistributed 2015-2022 (MSVC) is required for running the software on Windows. This is a common prerequisite for applications that have been built using Microsoft C and C++ tools [69], and is necessary to compile and run Qt applications on Windows [Table 2]. MSVC can be downloaded from Microsoft's online documentation, which installs the required runtime libraries [69].

It was decided to continue the work using Qt6, and support for Windows 7 and 8.1 would not be prioritized. Instead, a way to package MSVC together with the software would be looked into. Support for Qt5 is currently only available to license holders, while free support for the LTS version of Qt6 is said to not expire until the 30th of Mars 2026.

3.2.8 PyQtDarkTheme and Linea Iconset

It was decided during development to upgrade the visual style of the software. The previous version of the software offered only the basic look of Qt, which consists of system-native elements, and lacked a unique and recognizable modern look.

Part of creating a visually appealing GUI, are the aesthetics and looks of the elements within it. However designing color pallets, borders, menus, buttons and so on can be a time-consuming process, and is usually saved for the very end of the development process. One may argue that aesthetics are outside the scope of this thesis, and one of the principles created by UX pioneer Bruce "Tog" Tognazzini states that *aesthetic design should be left to those schooled and skilled in its application: Graphic/visual designers* [44]. To maximize aesthetics while spending the least amount of time doing so, a pre-existing theme in the form of a Python library, named PyQtDarkTheme, was implemented. PyQtDarkTheme automatically applies a flat and modern theme to all visual elements of a Qt GUI, by simply importing and calling its setup function [70]. The library can apply both a dark or light theme `QPalette`, where the dark theme uses the Qt stylesheet from QDarkStyleSheet [71]. These two Python libraries use the MIT license.

In addition, icons from the Linea Iconset [72] were downloaded, and compiled to Python binary code through the Qt resource system. The icon set offers software-related icons, e.g. icons for trash [Figure 16], and various arrows, and is dedicated to the worldwide public domain through the Creative Commons Zero v1.0 Universal license [72].

Figure 16: Variations of a trash icon that may be used in different contexts.

Source: Linea Iconset by Ferrando and Sigidi [72]

3.2.9 PyInstaller

PyInstaller is a Python library that bundles a Python project, together with all its associative dependencies, into a single package that includes an executable specific to the operating system it was created on. The library is not a cross-compiler, meaning an executable for Windows must be made by running PyInstaller on Windows, and so on for other systems [73]. *PyInstaller is distributed under a dual-licensing scheme using both the GPL 2.0, with an exception that allows you to use it to build commercial products, and the Apache License, version 2.0, which only applies to a certain few files* - PyInstaller Manual [[73]].

PyInstaller relies on *hooks*, which in this case are files that point to other files in libraries needed to run the application. They are used during the analysis phase of the Python project [73]. PyInstaller does a suitable job of figuring out which files are needed by analyzing import statements, but for libraries which makes use of unusual import mechanisms, hooks are needed to reliably find all needed files. Hooks already exist for many of the major Python packages, such as NumPy and

PySide6, but not for less-known libraries like kikuchipy, HyperSpy and PyEBSDIndex, all of which require hooks to be properly packaged. For example, kikuchipy imports its classes and methods in a lazy way. Therefore simple hooks were created for these 3 libraries, all of which are defined similarly to Code 2.

```
from PyInstaller.utils.hooks import collect_all
datas, binaries, hiddenimports = collect_all('kikuchipy')
```

Code 2: PyInstaller hook for the kikuchipy. Hooks for HyperSpy and PyEBSDIndex were created similarly. Taken from *hooks/hook-kikuchipy.py*

3.3 Distribution of software

Distribution aims to deliver all necessary files of the application to the user's local system, in the most intuitive way possible. This includes assuring compatibility with supported operating systems, and files being unpacked to their intended location, without requiring the user to perform additional tasks. The application must also be available in the public domain for users to freely download, which requires distributions to be hosted by an online website, and also to apply the correct licensing terms. Specifications for recording changes, naming of versions, and honoring the contributors of the project, are important to preserve consistency throughout a project where developers come from different projects.

To ensure that the target group of students would be covered during the usability test, distributions for both macOS and Windows were made. Research conducted in 2019 by Vanson Bourne, a third-party research firm in the UK, showed that about 40% of students across 2 224 colleges use Mac, while the other 60% use Windows computers [74]. However, 71% of participants would prefer to use Mac [74], meaning the proportion of Mac users will most likely increase in the future.

The previously used approach from the pre-study was to use PyInstaller together with auto-py-to-exe [75]. The latter library implements a GUI where the user can set up parameters that are passed to PyInstaller. The initial argument for using auto-py-to-exe was the ability to save these parameters using the JavaScript Object Notation (JSON) format, which future developers could use to correctly package new versions of the application. However some arguments are system specific, and some consist of absolute paths that usually will require editing to be correct. To improve upon this process, it was decided to replace the use of auto-py-to-exe with a Python script that defines the correct arguments and paths, based on the system it is running on [Code 3]. The only required argument from the user is the target architecture, which can also be detected automatically by passing *'auto'* to the prompt.

```python
import os
import platform
import sys

import PyInstaller.__main__

sys.setrecursionlimit(sys.getrecursionlimit() * 5) # Fix needed for some systems
workdir = os.getcwd()

target_arch = input(
    "Specify the target architecture [auto, x86_64, arm64, universal2]: "
    ).lower().strip()

# Different seperators between macOS and Windows
if platform.system().lower() == "darwin":
    sep = ":"
else:
    sep = ";"

args_list = [
    '--noconfirm',
    '--onedir',
    '--icon', os.path.join(workdir, "resources/ebsd_gui.ico"),
    '--name', "EBSD-GUI",
    '--add-data', os.path.join(workdir, f"resources{sep}resources/"),
    '--add-data', os.path.join(workdir, f"*G.txt{sep}."),   # Add COPYING.txt
    '--additional-hooks-dir', os.path.join(workdir,"hooks"),
    '--hidden-import', "kikuchipy",
    '--hidden-import', "hyperspy",
    '--hidden-import', "pyebsdindex",
]
# Checks target architecture for valid values
if target_arch in ['x86_64', 'arm64', 'universal2']:
    args_list.append("--target-arch")
    args_list.append(target_arch)

# Skips creating application on macOS as it currently do not work
if platform.system().lower() == "darwin":
    args_list.append("--console")
else:
    args_list.append("--windowed")

args_list.append('main.py')
PyInstaller.__main__.run(args_list) # Runs the command with arguments
```

Code 3: New approach for exporting the Python scripts into an independent packaged application that is distributable. Taken from *export.py* during the transition from EBSD-GUI dev to EBSD-GUI Pre-release.

The distributable application exported from the Python scripts includes an executable file, denoted by the .exe extension if exported on Windows, and a runnable Unix file if exported on macOS. All files needed to run the application are included in the distribution folder. To easily install the distribution, different methods for Windows and macOS must be used.

3.3.1 Inno Setup (Windows)

Inno Setup makes it possible to compile a single EXE file, which can install the created distribution on any Windows version released since 2006 [76]. This installer can be heavily customized by writing code in the Pascal Script language, which can be done in Inno Setup's Integrated Development Environment (IDE). The final installer provides a Standard Windows wizard GUI, which many Windows users will already be familiar with. The wizard allows users to choose language, installation location, whether or not to associate specific file extensions with the software, and create shortcuts, among other things.

The customization provided allows for the running of other programs before, during or after the installation takes place. This provides an opportunity to automatically detect whether MSVC is already installed, and install the correct version of MSVC if needed. Code 4 shows a solution that checks whether the version satisfies at least v14.29, which was used during development, and is based on an answer found on Stack Overflow [77].

```
[Files]
; VC++ redistributable runtime. Extracted by VC2017RedistNeedsInstall(), if needed.
Source: "C:\EBSP-Indexer\VC_redist.x64.exe"; DestDir: {tmp}; Flags: dontcopy
...

[Code]
function VCRedistNeedsInstall: Boolean;
var
  Version: String;
begin
  if RegQueryStringValue(HKEY_LOCAL_MACHINE,
       'SOFTWARE\Microsoft\VisualStudio\14.0\VC\Runtimes\x64', 'Version',
       Version) then
  begin
    // Is the installed version at least 14.29 ?
    Log('VC Redist Version check : found ' + Version);
    Result := (CompareStr(Version, 'v14.29.30139')<0);
  end
  else
  begin
    Result := True;
  end;
  if (Result) then
  begin
    ExtractTemporaryFile('VC_redist.x64.exe');
  end;
end;

[Run]
Filename: "{tmp}\VC_redist.x64.exe"; \
  StatusMsg: Installing Microsoft Visual C++ Redistributable 2015-2022; \
  Parameters: "/quiet"; Check: VCRedistNeedsInstall; Flags: waituntilterminated
...
```

Code 4: Solution that automatically installs Microsoft Visual Studio C++ Redistributed 2015-2022 if needed. The script checks whether MSVC is installed, and only unpacks *VC_redist.x64.exe* if needed. Taken from *setup_script.iss*, and based on Stack Overflow answer [77].

EBSD-GUI dev is a 64-bit application, something Inno Setup has extensive support for [76]. The software should also be available on lab computers, and accessed by students. The installer should therefore be configured for non-administrative installation, as admin privileges are not required.

Lastly, Inno Setup provides uninstall and upgrade capabilities that are automatically included in the installer. The packaged application is given a Globally Unique Identifier (GUID), generated by the IDE of Inno Setup. The GUID is used by future installers, to recognize whether the same application is already installed. If the current version is lower than the installer's version, it will automatically replace the previous version with the newer one [78]. This is especially useful when providing LTS, which can be delivered as minor patches through the release of new installers.

3.3.2 Platypus (macOS)

In general, PyInstaller supports packaging Python scripts into files denoted by the .app extension. Although appearing to be a single file, this type of file contains a hierarchy of other files used to run the application. It can be seen as a wrapping around the distributed package, that adds instructions pointing to the file which should be executed to run the application. However, an issue was encountered when trying to run the distributed app made with PyInstaller, where the application would immediately close after verifying. Running the app bundle's Unix file in a terminal window would work, but the terminal window would stay open in the background which was not ideal.

Looking into the issue, a discussion thread for PyInstaller was found with users having similar issues, which proposed some solutions [79]. The issue is most likely caused by the GUI-library used, however only solutions for Python's tkinter library are provided. The most viable option was therefore a workaround, which is to use Platypus [80] to bundle the app distribution from PyInstaller into a second app [79]. Platypus is a software tool that can create macOS application bundles from command line scripts or programs in a number of programming languages [80]. It can be installed via Homebrew and was originally made to ease sharing of scripts without requiring the use of a command line interface. The app bundle made with Platypus can then include a shell script with a single command which runs the Unix file within the original app bundle [Code 5].

```bash
#!/bin/bash
"./EBSP Indexer.app/Contents/MacOS/EBSP Indexer"
```

Code 5: Unix command which points to the Unix file within the app bundle.

This is similar to running the Unix file in a terminal, however, Platypus allows defining the interface type as none, hiding the terminal window with the running shell script. Another option provided by Platypus is to enable the software to run in the background, keeping the application's GUI from immediately closing on startup.

3.3.3 Publication

An important part of publishing software is to anticipate the community of users and developers that might grow along it [81]. Such a community ensures that the project is maintained in the future, and avoids degradation of software quality, often referred to as bit rot or software decay [82]. This refers to the changes that occur in the digital environment where the software persists, which negatively impacts the software. Major updates to the computer's OS or hardware drivers, might cause unexpected errors and bugs in the software. Preparations have already been made to avoid this, by writing guides on how to set up a development installation and to maintain the source code. The existing GitHub repository, where the source code resides, functions as a public space where it is easy to ask and discuss software-related questions. Developers may also contribute with code themselves, which is inspected and accepted by the repository's maintainers. With the project being dependent on a community of developers, a set of rules should be carried out to ensure quality and consistency.

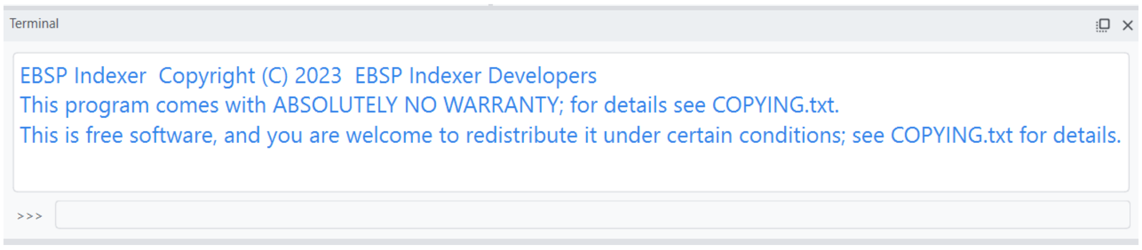
The contribution guidelines were specified as follows:

- All contributions to the project should be properly credited according to the all-contributors specification [83], adapted from the Contributor Covenant version 1.4 [84].
- Contributors associated with the project are part of the collective known as "EBSP Indexer Developers", which must be credited as the publisher of all associated software releases.
- All contributions must satisfy the terms of the GNU General Public Licence (GPL) v3.0, by which others can download, share or distribute the software within the licensing conditions.
- Code contributions should follow the Style Guide for Python Code [85] and use The Black Code Style [86].
- The version number of the software should be incremented according to Semantic Versioning 2.0.0 [87] when a new release is published.
- New releases should be created and compiled according to tutorials and should use PyInstaller together with Inno Setup (Windows) or Platypus (macOS).

The all-contributors implement specifications for including and crediting people who contribute to the project in any aspect, not just code [83]. This is done by implementing an all-contributors Bot into EBSP Indexer's GitHub repository, which features easy ways to add contributors to be displayed in a table inside the project's README file.

The Copyright laws enforced by the Norwegian Government, along with many other countries, imply that a work is automatically copyrighted the moment it is created [88]. This means that without including a software license that relaxes these terms, the software cannot be modified, built upon, or shared by others. Additionally, the software cannot be published without being made available wholly as open-source, because the software builds upon existing code, kikuchipy and HyperSpy, which applies GPL [57]. That is why this project must also be licensed under the terms of GPL, more specifically version 3.0. GPL is the most restrictive open-source licensing available, where releases of modifications must be documented and employ the same licensing terms. Limitations state that any type of warranty is not included, which limits the liability of the creators.

The software includes a copy of GPL version 3 inside the `COPYING.txt` file, which is required by the licensing terms. GNU also recommends that a short copyright notice is added to every file within the project and that the notice is displayed inside the software if possible. Therefore a notice was added to the top of `main.py`, and a message is displayed inside the terminal of the software [Figure 17].



```
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.
>>>
```

Figure 17: Copyright notice according to GPL v3.0 is displayed inside the application's Terminal.

The contribution guidelines describe the styling of code using the Style Guide for Python Code [85] and The Black Code Style [86]. Black has already been in use since the start of the project, however, the Style Guide for Python Code has only been used to some extent. Still, it is important to acknowledge these moving forward with development, to avoid unnecessary re-styling of code later. Semantic Versioning is another convention that is encouraged, where a version number consists of 3 numbers on the `MAJOR.MINOR.PATCH` format [87]. Depending on the impact of the change, different numbers are incremented. API changes which are incompatible with previous versions should increment the MAJOR version, and when functionality is added in a backward-compatible manner; the MINOR version [87]. Backward-compatible bug- and error-fixes are deployed as part of a PATCH version [87]. The version number can also include labels, which can specify the type of release, e.g. *Pre-release* or *Development build.* It is also common to start with version 0.1.0 as part of initial development and increment the `MINOR` version with each new release [89]. A 1.0.0 release is usually characterized by reliable software that is ready to be used in production, and built upon a stable API [89]. Releasing version 0.1.0 makes sense in this project, considering that the software depends on APIs from libraries, e.g. kikuchipy and PyEBSDIndex, which have yet to release a 1.0 version, indicating that they are still in development.

When publishing research software, it might be useful to register a Digital Object Identifier (DOI). The DOI can be used to reliably identify the published digital object, compared to Uniform Resource Locators (URLs) that might change over time. To make it easy for future authors and developers to cite the software, a DOI was created for the software by publishing it on Zenodo, a platform for sharing, curating, and publishing open-source data and software [90].

Another way to increase availability and visibility is to host the software on a website dedicated to users who want to download it. Ideally, this would be the same website dedicated to the project, however setting up a website is not part of this work's scope, and the required resources to do so have not been accounted for. There do however exist some external third-party websites, dedicated to hosting open-source software for users to download. SourceForge is such a website and is a popular platform for developers to publish their software. Although SourceForge is host to some reputable free software, like Apache OpenOffice and 7-Zip, one should be skeptical of websites that are advertised as ”free hosting”. The website has had a troubled history of controversial monetization models, where user's downloads would be bundled together with closed-source ad-supported software, often referred to as junkware or Potentially Unwanted Program (PUP) [91]. Although PUPs might not be malicious, they can result in vulnerabilities in a system's security. SourceForge was however acquired by a new company named Slashdot in 2016, which immediately abandoned this monetization model [91]. In 2020 the website was reported as safe to use again [92], and after inspecting the downloads it was decided to upload the software as a temporary solution.

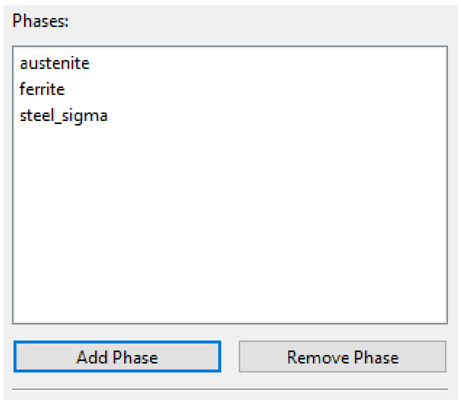
3.4 EBSD-GUI Pre-release

Development of EBSD-GUI Pre-release, the version to be deployed in the usability test, began by addressing issues regarding HI code, instability of the macOS version, the static nature of the GUI, and the lack of feedback regarding a task's state. Work was also started on an implementation for refining orientations in a crystal map that was indexed with either DI or HI.

3.4.1 Hough indexing overhaul

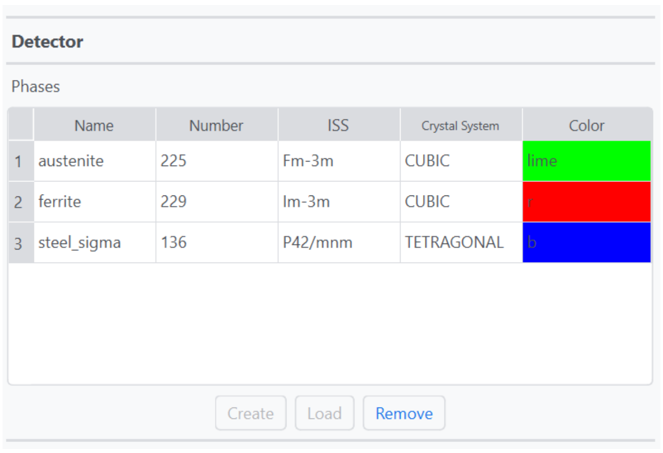
Following the possibility of accessing PyEBSDIndex's API through kikuchipy, introduced by the release of version 0.8.0, the function which performs HI in the software was completely rewritten. The new implementation avoids having to save an additional copy of the dataset and is much more concise. The addition of PyOpenCl greatly enhanced indexing speeds, from approximately 75 patterns/s to 215 patterns/s on a small dataset of 12 Mb with 60x60 patterns, using an old Intel(R) Core(TM) i3-6100U CPU with integrated HD Graphics 520. Performance boost is expected to increase with the size of the dataset, due to the constant overhead of sending data to GPU.

The original dialog window was kept, but its content and layout were improved. The parameters were divided into groups, indicating how parameters affect each part of the indexing process. For example, binning of signal shape is part of the signal group, and ρ-fraction and number of detected bands are part of the Hough Transform group. The original list of phases was replaced with a table instead, which now shows information about the space group and crystal system of the phase, and also what color will be assigned to the phase [Figure 18].


Phases:

austenite
ferrite
steel_sigma

(a) EBSD-GUI dev


Detector

Phases

	Name	Number	ISS	Crystal System	Color
1	austenite	225	Fm-3m	CUBIC	🟩 lime
2	ferrite	229	Im-3m	CUBIC	🟥 r
3	steel_sigma	136	P42/mnm	TETRAGONAL	🟦 b

(b) EBSD-GUI Pre-release

Figure 18: Comparison between the old phase list used in EBSD-GUI dev (a), and the new table implemented in "EBSD-GUI Pre-release" (b). The new table also restricts the user from creating or loading more phases than PyEBSDIndex is capable of, which is a maximum of 2. In EBSD-GUI Pre-release, it was possible to load 3 phases into HI by first doing DI. This would result in an error, later discovered by the usability test.

The ability to inspect the crystal system of the phase might be helpful to students, and to does who are less familiar with the details of a material's microstructure. Due to the HI implementation from PyEBSDIndex only supporting CUBIC crystal structures, it is particularly useful when troubleshooting incompatible phases.

The new table in Figure 18b adds an additional button labeled *Create*, which allows users to define the phase without loading it from an existing master pattern. The button opens a new dialog window, requiring the user to specify a name and a space group number for the phase. Information about the crystal system is derived from the space group number through the `Phase` class from orix. Specifying a custom structure is optional, as it is currently not being used by PyEBSDIndex during indexing. However, when PyEBSDIndex releases support for this, most if not all necessities for utilizing structures will already have been made. A custom structure should however be specified, in order to correctly display geometrical simulations on top of indexed patterns.

An option for editing the direction which specifies the color key used when creating an IPF map was added. This is useful when the user needs to analyze the orientations relative to a specific direction or wants to directly compare an IPF map with an existing one that uses a specific color-key direction.

3.4.2 Mac troubleshooting

In addition to the known issue of leaked semaphore objects, listed in Section 3.1, multiple issues related to bundling the software on macOS were discovered.

The first attempts of running a bundled version on macOS would result in an error, which occurred due to an object being of type `KeyboardModifier`, and a method, invoked when importing the Qt backend from Matplotlib, expecting the object to be of type `String` or `Integer`. The same issue was discovered on Matplotlib's official repository, pointing to a new "Enum system", introduced in version 6.4 of PySide6, being the culprit at fault [93, 94]. The new system required Matplotlib's Qt backend to adapt to the new system. As a quick solution, it was decided to downgrade PySide6 from v6.4 to v6.3.2, where the error did not exist.

When a bundle, which would open without error, was acquired for macOS, an issue related to multiprocessing was encountered. Multiple copies of the application kept opening in what seemed to be an endless loop. A user on Stack Overflow posted a similar issue when using PyInstaller and Qt, who had narrowed down the cause of the problem to happen when importing the sklearn package from scikit-learn, a dependency of kikuchipy used for multivariate analysis [95]. This library imports and uses tools from joblib to do parallel computing, which when frozen by PyInstaller causes an infinite loop on macOS. The user suggests excluding `sklearn.externals.joblib` from the bundle [95]. This can however break the functionality of scikit-learn and in turn kikuchipy. In a different post, a user suggests a fix by importing the multiprocessing library and calling `multiprocessing.freeze_support()` right before the main function of the application is called [96]. This fixed the issue of multiple copies of the GUI being created, however when inspecting the bundle using the resource tracker, one could see multiple processes still appearing in an endless loop. The final solution was found in a raised issue on PyInstaller's GitHub repository, which specified that freezing of multiprocessing must happen before import statements are called [97]. This fixed the issue, and the memory usage of the bundle finally stabilized.

In an effort to fix the leaked semaphore objects, the loading of datasets was put outside of threads. This does not affect performance if the dataset is loaded lazy, but will otherwise make the GUI stop responding for a moment, depending on how large the dataset is. Although it did not completely fix the leaked semaphores objects, it drastically decreased the frequency of their occurrences.

A specific problem was encountered on Mac computers with Apple silicon, a series of system-on-chips designed by Apple Inc. that is based on the ARM64 architecture. Using this architecture often requires custom support from the software to utilize the CPU and GPU properly, which is not always available seeing as the first Apple silicon was released only 3 years ago. As of January 2023, all new Mac models will be built using Apple silicon [98], making it important to support the ARM64 architecture to secure this future user group. The problem occurred during the triplet voting process of HI, where the CPU performs the computations instead of the GPU. The Python library RAY is responsible for distributed multi-processing, which only experimentally supports Apple's ARM64 architecture. The user guide of PyEBSDIndex highlights this, and suggests installing the base library using the Conda package manager instead of using the package installer for Python (pip) [6]. The default configured RAY package should be installed separately using pip [6]. Therefore the solution was to rebuild the virtual Python environment using Conda and use pip in cases where packages were not available to download.

3.4.3 Customizable GUI

Monitors and laptop screens come in a variety of sizes and resolutions, which affects the size and readability of GUI-elements being drawn on the display. Laptop screens are often smaller in size but retain a high-resolution display. In order to make elements appear larger without sacrificing the resolution of the display, and consequently lose image quality, both Windows and macOS use scaling. As a result, smaller-sized displays simply have less space to display GUI-elements, regardless of resolution. A solution would be to develop the software's GUI with this in mind, displaying fewer elements and tools at the same time. This would needlessly sacrifice visibility on larger displays and multi-monitor setups, so instead the solution became to introduce the ability to let users customize their GUI themselves.

A high level of customization was made possible by using Qt's `QDockWidget` class, which are *"secondary windows placed in the dock widget area around the central widget in a `QMainWindow`"* - Qt Documentation [99] [Figure 19]. The transition from using regular `QWidget` to `QDockWidget` was straightforward, due to the new class acting like a wrapper that contains the original widget. All widgets which previously resided in the central widget of the main window were moved into separate dock windows, to take advantage of the docking features.

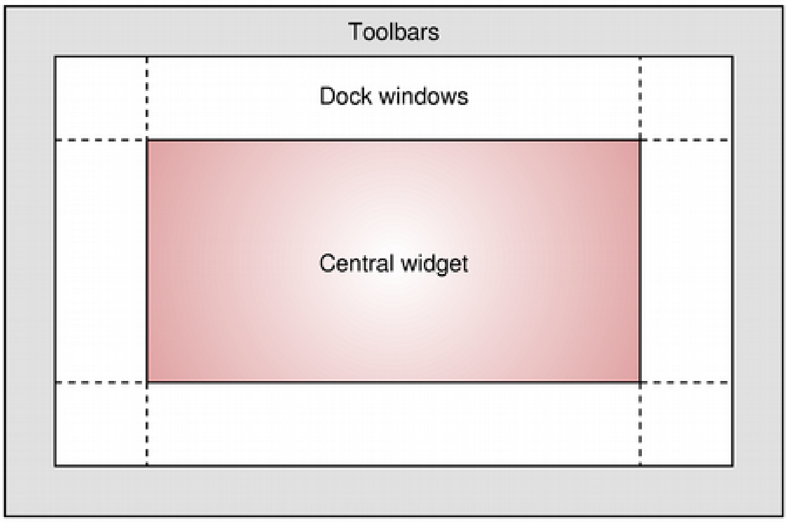


Figure 19: Allowed docking area inside the `QMainWindow` class from Qt.

Source: The official Qt Documentation [99]

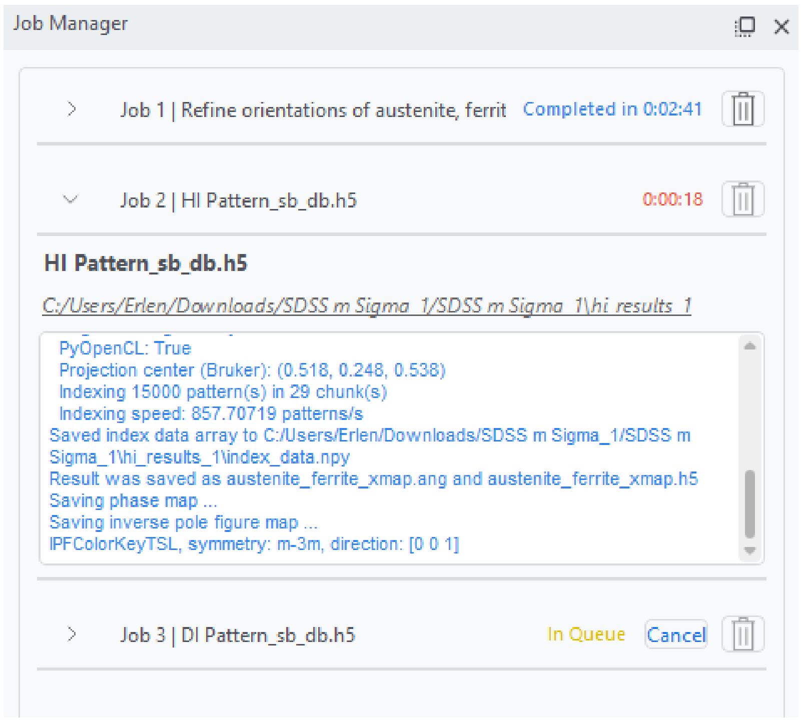
The dock windows can be resized, reordered, unlocked from the main window, maximized and hidden. The ability to unlock windows, allows widgets to be displayed in full-screen on separate connected monitors, e.g. for inspecting IPF maps in better detail. This is the same functionality that AZtecCrystal is capable of, described in Section 2.2.1.

Putting all the widgets into dock windows leaves the central widget empty, which then becomes a void in the middle of the main window when resizing the dock windows. A workaround was implemented which removes the central widget by calling `QMainWindow.setCentralWidget(None)`. This is not a recommended solution according to the official Qt Documentation, which states that *Creating a main window without a central widget is not supported. You must have a central widget even if it is just a placeholder* - official Qt Documentation [100]. On the other hand, the solution does seem to work fine, with multiple users online reporting so too [101].

3.4.4 Job Manager

To improve the feedback a user receives when they initiate long processes, such as signal-to-noise improvements or indexing, a new dock window, named Job Manager, was introduced to the main application window [Figure 20].

The Job Manager gives an overview of all currently running, completed, failed and queued jobs during the session. Jobs are ordered chronologically, providing a history of what tasks have been done and in what order, and the number of currently running jobs is shown on the status bar at the bottom of the main window. Each job shows the elapsed time, the title of the task being performed, the output path of the results, and a log that is updated in real-time with information about the tasks being performed. This is the same log that would previously appear in the terminal, and the same approach for redirecting the output stream described in Section 3.2.7 was used. However, it is now easier to identify what outputted messages belong to which task, and the complete log is also saved to the same folder as the results. If the task of a job fails, a short, red-colored message describing what went wrong is logged, and the full stack traceback of the error is displayed in the terminal instead.


Job Manager
Job 1 | Refine orientations of austenite, ferrit
Completed in 0:02:41
Job 2 | HI Pattern_sb_db.h5
0:00:18
HI Pattern_sb_db.h5
C:/Users/Erlen/Downloads/SDSS m Sigma 1/SDSS m Sigma 1\hi results 1

```
  PyOpenCL: True
  Projection center (Bruker): (0.518, 0.248, 0.538)
  Indexing 15000 pattern(s) in 29 chunk(s)
  Indexing speed: 857.70719 patterns/s
Saved index data array to C:/Users/Erlen/Downloads/SDSS m Sigma_1/SDSS m
Sigma_1\hi_results_1\index_data.npy
Result was saved as austenite_ferrite_xmap.ang and austenite_ferrite_xmap.h5
Saving phase map ...
Saving inverse pole figure map ...
IPFColorKeyTSL, symmetry: m-3m, direction: [0 0 1]
```

Figure 20: Job Manager containing jobs in different states. Job 1 is completed, while Job 2 is running, indicated by the timer which updates every second. Job 3 is queued and will not run until Job 2 is completed. Job 1 and Job 3 are both collapsed, while Job 2 shows additional information.

The idea and design of the Job Manager were inspired by the video editing software DaVinci Resolve by Blackmagic Design Pty Ltd. [102], commonly used by both Hollywood professionals and amateur filmmakers. Similar to indexing procedures, the rendering of a video may take between a few minutes to several days to complete. DaVinci Resolve implements the Render Queue, which gives an overview of the current and queued rendering jobs [Figure 21]. The Job Manager extends this concept with a log, which can be hidden with the Job's collapse button, to avoid flooding the list and consequently retain the principle of visibility.

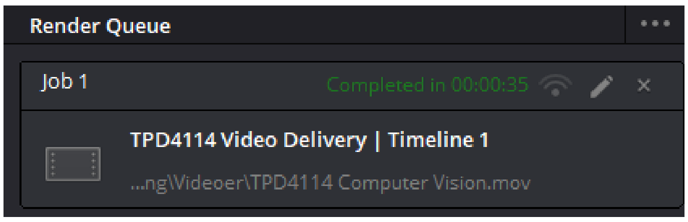

Render Queue
Job 1
Completed in 00:00:35
TPD4114 Video Delivery | Timeline 1
...ng\Videoer\TPD4114 Computer Vision.mov

Figure 21: Render Queue of DaVinci Resolve, which became an inspiration when designing the Job Manager.

In practice, each job displayed in the Job Manager is created using the new class `WorkerWidget`, which contains an instance of the previously developed `Worker` class. To display the state of the worker, the `Worker` class was extended with signals that can be emitted from a thread [Code 6]. Because signal properties are globally shared between instances of the class, the worker's unique identifier `self.id` is emitted to make sure the correct instance of `WorkerWidget` is updated.

```
⋮
@Slot()
def run(self) -> None:
    """
    Initialise the runner function with passed args,
    kwargs, and redirects standard output and error.

    Raises
    ------
    Exception
        If an error occurs in the `self.fn` method
    """
    with redirect_stdout(self.thdout), redirect_stderr(self.error_redirect):
        try:
            self.isStarted.emit(self.id)
            self.fn(*self.args, **self.kwargs)
            self.isFinished.emit(self.id)
        except Exception as e:
            self.thdout.errorwrite(f"{e} (See terminal for traceback)")
            self.failed = True
            raise e
        finally:
            if self.failed:
                self.isError.emit(self.id)
⋮
```

Code 6: The addition of signals to extend the functionality of Code 1. Signals are emitted from the thread to update whenever the task is started, finished or failed. `self.failed` is defined inside the `__init__()` function of the worker class. Taken from the EBSD GUI Pre-release version of *utils/worker.py*.

The new implementation causes an unwanted side effect when running jobs concurrently, because of how `redirect_stdout()` and `redirect_stderr()` act within threads. Even though both methods are called inside the thread, the latter of the two threads being initiated will also redirect the former's output stream. This means that if Job 1 and Job 2 are running concurrently, the output of Job 1 will appear in the log of Job 2. No solution for this was found, so it was decided to instead limit the number of allowed active threads to 1. Note that this does not affect the number of threads used during the indexing procedures, so the loss of performance is negligible to none at all.

Solutions for interrupting a started job were looked into, but no good or safe option was viable for this particular use case. Once a thread is spawned in Qt, it is given a set of instructions and will not terminate until these are completed. One may have communication between threads by using signals, showcased in Code 6, but the ability to cancel a running method would still require implementing checkpoints in the code, where the method is allowed to terminate. This was considered to not be worth the work because the ability to cancel a job during indexing would require adding checkpoints in the underlying libraries.

3.4.5 Implementing refinement of orientations

Previously, refinement of orientations was only available as an option in the DI routine. It was decided to add this as a separate tool, similar to how the processing of patterns is separated from the indexing routine [1]. This makes it possible to first inspect the crystal maps before deciding whether or not to perform refinement. This also allows refinement of orientations in crystal maps indexed using HI, given that dynamical simulations are provided as master patterns.

A new menu and dialog window for refinement were added, which allows the user to choose the refinement method and add optional keyword arguments. The optimization algorithm Nealder-Mead [50] from the Python library NLopt [51] was added in addition to the methods provided by SciPy [52]. The dialog window has a similar layout to DI and HI, to retain consistency throughout the different functionalities.

While implementing the functionality, a bug in orix v0.11.0 was encountered, where refining orientations of the second phase in a single multi-phased crystal map would result in a refined crystal map without information about the phase. The issue was reported on the kikuchipy discussion forum, and the root of the problem was identified [103]. The issue was resolved and released as patch v.0.11.1 for orix. Crystal maps containing points that were assigned the `not_indexed` label, possible by using HI on low-quality patterns, became another special case to take into account. Refinement of points that are labeled `not_indexed` had not been considered a possibility, and therefore no fail-safe mechanism for this existed in kikuchipy v0.8.3 or prior, resulting in a raised error. The issue was reported to the kikuchipy discussion forum, where it was stated that the issue would be fixed in the next patch [104]. To create a workaround before the deadline of the usability test, a navigation mask was created for the not indexed points [Code 7]. This would however lead to missing points inside the crystal map, which would appear as black spots in the IPF map and phase map.

```
⋮
# Not needed as of Kikuchipy 0.8.4
# --------------------------------
if not xmap_phase.all_indexed:
    nav_mask_phase = ~(
        xmap_phase.phase_id == xmap.phases.id_from_name(mp_key)
    )
    nav_mask_phase = nav_mask_phase.reshape(xmap.shape)
else:
    nav_mask_phase = None

⋮
```

Code 7: Temporary fix using navigation mask before the release of kikuchipy v0.8.4. Taken from the EBSD-GUI Pre-release version of *scripts/refinement.py*

To take full advantage of refinement, each point in the crystal map should contain indexing scores for all phases present in the ROI. This means that multiple phases are considered when refining each orientation, which allows refinement to re-assign the phase of points in the crystal map. That is why during indexing of a multi-phased sample in the current DI implementation, refinement is performed on each single-phased crystal map before merging the final result, which only keeps the best scoring orientation in each point. Due to this, the implementation of refinement must be able to handle both single- and multi-phased crystal maps created both with HI and DI. A single-phased crystal map from HI is stored similarly to one indexed using DI, giving a total of 3 different scenarios to consider:

1. Single-phased crystal map from HI or DI
2. Multi-phased crystal map from DI
3. Multi-phased crystal map from HI

In addition, due to the new ability to define and use custom phases during HI, described in Section 3.4.1, these custom phases must match the structure of the phase defined by the dynamical simulation used during refinement.

Because of the complexity needed for the refinement implementation, and the waiting time for the release of kikuchipy v0.8.4, it was decided to not include this functionality as part of the usability test.

4 Usability test

The following subsections present how a questionnaire was created, and how the usability test was conducted. The complete questionnaire that was given to participants can be found in appendix A. Highlights from gathered data and observations are described, and the full report of all collected data from the questionnaire can be found in appendix B. The data was used to prepare the first official release of the application.

4.1 Questionnaire

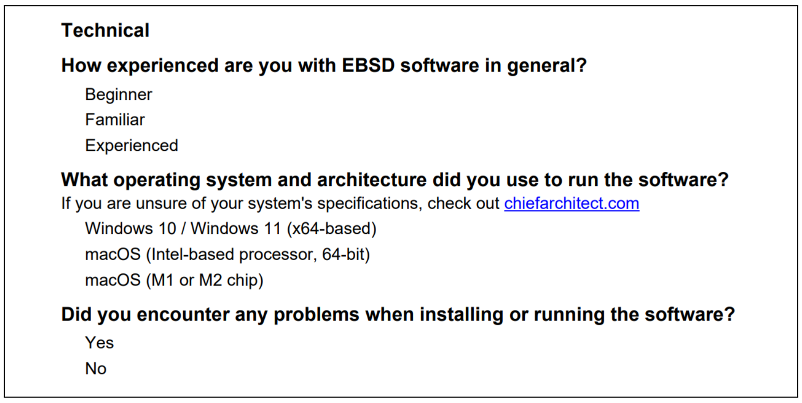
An online questionnaire was created to be used during the usability test, described in Section 2.3, in order for the gathered data to be of similar structure, and consequently be efficiently analyzed [42, p.346]. A questionnaire can be helpful to get an overview of issues that might have been overlooked during the testing,

It is important to establish why a questionnaire is desirable, in order to motivate the participant to fill out the form properly [42, p.327]. To make the participants feel safe about what data is gathered and for what purpose, it should be made obvious that personal information is protected. The following information and motivation should therefore be communicated to the participant at the beginning of the questionnaire:

This form is anonymous and all gathered data will be used for improving the software. Note that the data will be published in our masters' theses, which will be available to the public.

A questionnaire without any observer or interviewer being present should only be used if the motivation is high enough [42, p.346], so it would be smart to give the questionnaire during the lab session to ensure that filled-out forms are received.

In order to put responses into context, and to establish data from different perspectives, it is common for a questionnaire to gather basic demographic information [42, p.347]. Personal information may be classified as any type of information that can lead to the identification of the participant afterward. NTNU's procedures and tools for data gathering are described in an online guide, which aims to ensure that the proper information security level is applied [105]. The guide also specifies that certain tools cannot be used; *"NTNU does not have a data processor agreement with Google, and this means that researchers and students at NTNU can not use software and tools from Google when collecting personal data."* - NTNU Knowledge base [105]. For this particular questionnaire, it is useful to establish the EBSD skill level of the participant, and what OS was used to run the application. This can highlight compatibility issues that are specific to an OS. Although the population of participants is small, this should not be enough information to be classified as personal. To be on the safe side, the questionnaire was created using the Norwegian online tool named *"Nettskjema"*, recommended by NTNU, instead of an online Google form.


Technical

How experienced are you with EBSD software in general?

Beginner
Familiar
Experienced

What operating system and architecture did you use to run the software?
If you are unsure of your system's specifications, check out chiefarchitect.com

Windows 10 / Windows 11 (x64-based)
macOS (Intel-based processor, 64-bit)
macOS (M1 or M2 chip)

Did you encounter any problems when installing or running the software?

Yes
No

Figure 22: Questions used to establish the demographic of the participants. The full list of questions can be found in appendix A.

Questions should be formulated in a compact and precise manner, to best communicate to the participant what data is of interest. Depending on the response format, clear instructions on how to answer the question should be given [42, p.347]. *"Nettskjema"* also makes it possible to use formats that restrict the user from giving multiple answers, e.g. only allowing one checkbox to be checked at a time. These can be used to establish what functionality the participants have used. Rating scales can be used to gain insight into participants opinions, e.g. how easy to understand or usable a feature is [42, p.348]. This type of rating scale is called a Likert scale and can be designed by gathering a pool of statements about the features which should be investigated [42]. How participants experience the learning curve of the application, and whether the design and layout are complex, are some statements that can point design of UX in the correct direction. It was decided to use a discrete scale with 5 points, ranging from strongly disagree to strongly agree, to provide users with enough options to express themselves. Participants would also be asked to rate their complete experience, using a scale from 0 to 10, to give an indication of the overall experience using the application.

4.2 Lab sessions

The usability test was conducted as part of the teaching plan used in the third module of TMT4166 *Experimental Materials Chemistry and Electrochemistry*. The module makes up 1/3 of the course and teaches students to record EBSP using SEM, index patterns using HI and DI, and present the results using various maps [21]. A total of 11 students were put into groups, and given the task of using SEM to capture patterns from a Super Duplex Stainless Steel (SDSS), containing austenite and ferrite, and as it would later turn out, also sigma phase. The next step was to index the patterns using OIM DC available on the computers in the EM-lab, and EBSD-GUI Pre-release which was made temporarily available for the students to download through Uninett FileSender. The students were given an introduction to EBSD-GUI Pre-release in a plenary session where they had access to the application on their own computers and Macs [Figure 23]. The students could then mimic the given walk-through themselves, and get assistance with any unexpected issues that occurred. A small dataset with SDSS patterns, without sigma phase, was given as an example to practice on, before the students were to index their own captured patterns the next week during the lab sessions. A manual, created by Relling [2], was also sent to the students. The manual guides the users through the core functionality of the application, from loading experimental datasets to inspecting indexed results.

Figure 23: Plenary session with students in TMT4166. After a walkthrough was given, students would receive help with installing and using the application.

The lab week consisted of 4 days, with one group of students attending each day. Each session started with making sure the students knew their assigned tasks and answering questions the students might have. It was made clear to the students that their interaction with the application would be observed, and that guidance would mainly be given if they were completely stuck or encountered unexpected issues. This restriction became less strict as students completed their tasks, in order to create discussion about the application. Students would often be asked to think out loud to better understand their decision-making, giving insight and creating an important source for data gathering. To speed up dictionary indexing, the students were asked to index a smaller Region of Interest (ROI) in the lab, and index the complete dataset at home. This decreased the time spent indexing from 1.5 - 3 hours to 10 - 20 minutes depending on the size of the chosen ROI and laptop specifications. While waiting for EBSD-GUI Pre-release to finish indexing, students used OIM DC on the lab computers to experience a more commercial solution for doing HI. Once their tasks were completed, students answered the questionnaire described in Section 4.1. The questionnaire took 2-5 minutes for the students to complete.

4.3 Feedback and data

Data was gathered from the questionnaire and from the observations made during the preliminary session and lab hours are presented. A short summary is given in Figure 24, and an overview of all issues discovered is given in Table 3. Issues have been assigned a level of severity, depending on the type and consequence of the issue. The reason is to get a better overview of the general complexity and seriousness of the occurring issues, which indicates how the application performed during the test. The levels of severity are defined according to the action that initiated the cause of the issue, and are as follows:

- Critical: The action cannot be completed.
- High: The action can be completed through adjustments *outside* of the application.
- Standard: The action can be completed through adjustments *inside* of the application.
- Low: The action can be completed without requiring adjustments.

Out of the 11 participants, only 1 participant was not able to run the application successfully. An attempt to fix the problem was made but to no use. A more detailed description of the problem is given in the entry labeled MAC1 in Table 3. The participant was one out of the three who used macOS and ended up not participating in the questionnaire, which is why 10 participants answered the given form instead of 11.

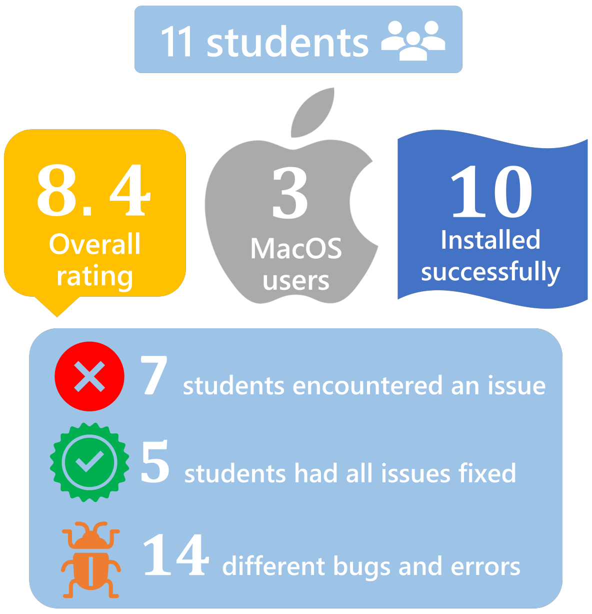

11 students
8.4
Overall rating
3
MacOS users
10
Installed successfully
7 students encountered an issue
5 students had all issues fixed
14 different bugs and errors

Figure 24: Summary of data gathered from usability test and questionnaire. Overall rating is the average of how participants rated their overall experience using the application. Out of the 11 participants, 1 was not able to install the application.

Table 3: Issues discovered during usability test of EBSD-GUI Pre-release. Each entry has a unique id depending on which system it was discovered on, a description of the issue, where in the application it appeared, and the type and severity of the issue. The table is sorted by severity.

Id	Description	Location	Type	Severity
MAC1	When loading the Python lib, the following error may occur: "Symbol not found: (_mkfifoat) Referenced: 'EBSD-GUI/Python', Expected: '.../libSystem.B.dylib'"	Startup	Crash	Critical
MAC2	Creating images in threads might result in 1 or 2 leaked semaphores on Intel-based Macs.	Pre-indexing maps, Dictionary indexing	Crash	Critical
WIN1	If Dictionary Indexing fails to locate a phase that is given by one of the master patterns, it fails when trying to merge.	Dictionary indexing	Error	Critical
WIN2	A directory containing a master pattern requires specific naming for Signal Navigation to be able to open the indexed crystal map.	Signal navigation	Error	High
MAC3	Punctuation in a file path results in the file not being properly read.	Main window	Error	High
WIN3	Native scaling set to 150% makes some dialog windows too large to fit within the display.	Dictionary indexing	Bug	High
WIN4	Calibration of Pattern Center cannot handle phases that have a non-cubic structure e.g. steel sigma phase.	Pattern refinement	Error	Standard
WIN5	A specific ROI would make indexing fail with the message: Unable to allocate 53.7 MiB for an array with shape (120, 222, 528) and data type float32.	Region of interest, Hough indexing, Dictionary indexing	Error	Standard
WIN6	Renaming of a processed dataset requires closing the application on some systems.	Pattern processing	Error	Standard
WIN7	PyOpenCL appears to not work with Intel HD graphics GPUs which are running old drivers from before 2019.	Hough indexing	Error	Low
WIN8	Updating the number of active jobs fail after running processing of patterns.	Pattern processing	Bug	Low
WIN9	Hough indexing can be started with 3 phases, even though the maximum allowed is two phases.	Hough indexing	Bug	Low
WIN10	System manager does not indicate that GPU is utilized on laptops with both Nvidia CUDA and Intel HD graphics, even though pattern speed is between 400-700 paterns/s.	Hough indexing	Bug	Low

Questionnaire

- All the students considered themselves beginners to EBSD software.
- Majority of students agreed that the features were easy to use and understand.
- Majority of students had no opinion on whether there should be more parameters to adjust.
- 1/3 of students found the design and layout of the application to be complex.
- 2/3 of students found the application slow and unresponsive at times.
- 2/3 of students knew the differences between the Image Viewer and Signal Navigation.
- 1/2 of students thought the different file formats were confusing.
- Majority of students agreed that they were able to easily complete their assigned tasks.
- 3/5 of students had a look at the user manual, which was rated on average 4.8 out of 5 in terms of usefulness.
- Students rated on average their overall satisfaction with the application an 8.4 out of 10.
- Only 1 student was *maybe* interested in contributing to the future development of the software.

Observations

- The Job Manager scales badly and results in multiple scrollbars, making it difficult for users to navigate and monitor their jobs. The initial location is awkward and small.
- In the dialog for ROI, some users were confused about the preview buttons, and thought maybe they would affect the processing.
- Processed datasets are available in the System Explorer while the processing itself is happening, which confused some users about the file being ready for use.
- Users had trouble finding and inspecting their results and knowing which files contained the indexed crystal map.
- Some users discovered that logs from Hough Indexing always display the Pattern Center using the BRUKER convention, regardless of the inputted convention. This is not considered a bug, but instead a miscommunication between the GUI and kikuchipy.
- When doing Pattern Center calibration, some users would adjust coordinates manually instead of automatically tuning them, resulting in a long and frustrating process for beginners. They were unsure of what misfit meant, and whether lower or higher values were better.
- The number of files generated from a 3-phased dataset using Dictionary Indexing is overwhelming for many users.
- One user affected by red-green color blindness struggled to tell when a job was finished, as he could not distinguish between text switching from red to green when completed.

Suggestions

- A confirmation prompt should be displayed when attempting to exit the application while a job is still running. One of the participants closed the application by accident and had to start their indexing from the beginning again.
- It would be helpful if the manual described how the user can move the docked widgets.

4.4 EBSP Indexer v0.1.0

This section describes the finalization of the software, based both on feedback received from the usability test, and the roadmap presented in Section 3.1. This version should not contain large changes to existing functionality that are outside the scope of the feedback. Such changes would potentially require another usability test, as functionality can unintentionally worsen.

4.4.1 Software portrayal

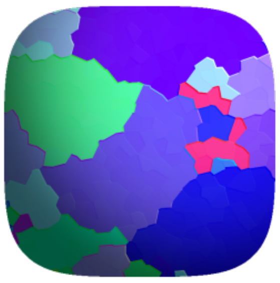


Figure 25: New icon for EBSP Indexer, of an IPF map originating from an aluminum dataset.

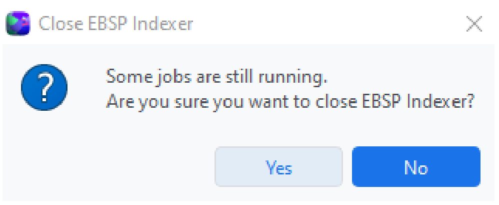
From the beginning of development, "EBSD-GUI" has been considered a temporary title for the software. To avoid potential confusion around a later name change, it was decided to do this before version 0.1.0 was released to the public. The term EBSD includes multiple processes, from using SEM, to indexing patterns, to generating and analyzing results [Figure 3]. It was necessary for the new title to better communicate the purpose of the software, and "EBSP Indexer" was proposed. This fits the core of the software, which is to improve and index Electron Backscatter Patterns (EBSPs), and is why the title was changed.

A new icon was designed to accompany the name change [Figure 25]. The icon should be simple, recognizable, and associated with the title of the software. Both kikuchipy and OIM DC use an illustration of an EBSP, so in order to be unique, a fully indexed crystal map represented as an IPF map was chosen. The color key direction used is $[1, 0, 1]$, creating a color pallet that is easy to recognize on most desktops. The IPF map was distorted using a crystallization image technique, to make it appear more simple and stylized.

4.4.2 Implementing usability feedback

Issue WIN3 from Table 3, observations related to the Job Manager, and the fact that 1/3 of students found the layout complex, indicate that the layout should be adjusted. The default location of the Job Manager was moved to the top right of the main application window, sharing the same height as the Image Viewer instead of the Terminal. The size of the log in each job was also reduced, so more jobs can be simultaneously displayed in the Job Manager, consequently removing the need for additional scrollbars. New dialog sizes were tested using a 150% scaling setup, to ensure that all buttons were accessible and visible, resolving WIN3.

In order to make the layout of the software more friendly to beginners of EBSD software, the main application window was made less cluttered on startup by hiding the empty Job Manager and Signal Navigation windows. These features will now automatically be shown when needed, e.g. when a job is started, or a dataset or crystal map is inspected. The Image Viewer will also now be focused when a new image is opened by the user, immediately displaying IPF maps and phase maps without requiring that the user navigates to the Image Viewer window.



Close EBSP Indexer

Some jobs are still running.
Are you sure you want to close EBSP Indexer?

Figure 26: Exit confirmation dialog when trying to exit the application while jobs are still running.

As suggested by one of the participants, an exit-confirmation dialog window was added, which appears when trying to close the application while jobs are still running [Figure 26]. This should make it so users avoid unintentionally closing the software in the middle of indexing, which could have resulted in hours of lost progress.

Both issues WIN4 and WIN9 [Table 3] are due to the lack of restrictions forced on the user, which allows users to encounter issues like these. Restrictions were added to PC calibration and HI, so non-cubic phases cannot be used, due to these not being supported by PyEBSDIndex. In addition, it is no longer possible to start HI with 3 or more phases. Forcing such restrictions upon users must be communicated properly, in order to avoid users feeling confused and deprived of their freedom [44]. Therefore phases that are unsupported were implemented to be shaded grey, indicating that they are disabled and non-selectable. A red message is also displayed in HI with information about what is wrong, e.g. the phase table displayed in Figure 18b would result in the message:

Structure P42/mnm in steel_sigma is currently not supported, only CUBIC Crystal Systems.

Issue WIN6 and the observation of a processed dataset being available in the System Explorer before being finished, highlighted an overlooked issue in the dialog for signal-to-noise improvement. Dataset files were loaded in a lazy manner, meaning the processing would happen after the resulting datasets had been saved to the system, and would be left open after the processing was done. This was changed so that only the preview of the dataset is loaded lazy, while the processed dataset is loaded completely beforehand.

Some participants noticed that the log from HI would always display the PC in the BRUKER convention, which lead to some participants being confused about inserting the wrong PC coordinates. The PC is displayed automatically when doing HI using kikuchipy's wrapper of PyEBSDIndex, which always internally uses the BRUKER convention. This would require the print statement to be changed inside the source code of kikuchipy, which was discussed on the forum of kikuchipy [106]. This resulted in "BRUKER" being added to the print statement in the release of patch v0.8.3 of kikuchipy, which should reduce confusion.

Accessibility which had not previously been considered was highlighted by the usability test. About 1 in 12 males are affected by color blindness, where difficulty differentiating between red and green is the most common [107]. One should avoid color combinations that are difficult to distinguish for users who are affected by any type of colorblindness. The previously green "completed" label displayed on jobs in the Job Manager, was replaced with one of blue color, and the use of blue and red in the GUI was made to resemble that of the color charts designed by data visualization specialist Ivan Kilin [108]. These colors are less saturated, which also makes colored messages easier to read. Finally, the ability to toggle between a light and dark theme design for the GUI was added to the settings menu, implemented by using PyQtDarkTheme. This is a very common feature shipped with most software, as the dark theme inflicts less eye strain when used in a poorly lit room.

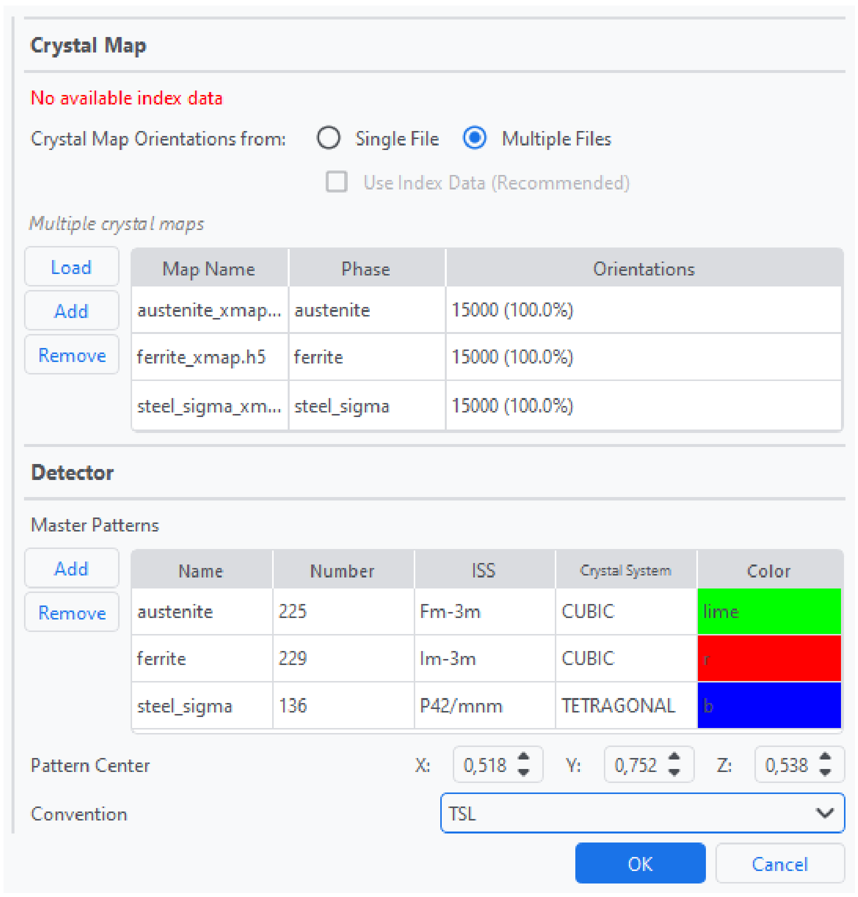
The work of Relling [2] solved issues WIN1 and WIN2, and also addressed the observations related to ROI and introduced improved sorting and naming of crystal maps generated from DI.

4.4.3 Finalizing refinement of orientations

As mentioned in Section 3.4.5, the development of refinement of orientations was not finished before the usability test. A generalized dialog window and code had to be developed, in order to support both single- and multi-phased crystal maps, indexed using HI or DI.

A multi-phased crystal map returned from kikuchipy's `EBSD.hough_indexing()` method, is structured so that each point only keeps information about the most likely solution for that point. It is however possible to keep all possible solutions for each point, by allowing the method to return the index data array, in addition to the crystal map, by setting `return_index_data = True`. The index data array is a NumPy array, which can be saved as a NumPy file for later use. Keeping the index data was added as an option to the HI dialog window, which also saves the path of the data in order to associate the data with the saved crystal map.

The DI implementation differs from HI in how results are handled. While HI saves a single crystal map, DI saves complete indexed crystal maps for each phase, in addition to the final merged crystal map. As a refinement of orientations should be done using the separately indexed crystal maps from each phase, the developed dialog window should support the ability to load multiple crystal maps from different files. Considering the above for HI, the ability for the user to choose between loading a crystal map from a single file or multiple from different files was implemented into the dialog window [Figure 27]. If a crystal map is loaded from a single file, and the path of the associated NumPy data file exists, the option for using index data is enabled and automatically checked.



Crystal Map

No available index data

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

☐ Use Index Data (Recommended)

Multiple crystal maps

Map Name	Phase	Orientations
austenite_xmap...	austenite	15000 (100.0%)
ferrite_xmap.h5	ferrite	15000 (100.0%)
steel_sigma_xm...	steel_sigma	15000 (100.0%)

Detector

Master Patterns

Name	Number	ISS	Crystal System	Color
austenite	225	Fm-3m	CUBIC	🟩 lime
ferrite	229	Im-3m	CUBIC	🟥 r
steel_sigma	136	P42/mnm	TETRAGONAL	🟦 b

Pattern Center X: 0,518 Y: 0,752 Z: 0,538

Convention TSL

Figure 27: Handling of phases from crystal maps and master patterns in refinement dialog. This approach shows the refinement of a SDSS with sigma phase, with orientations for each phase covering the whole ROI, shown by the number of orientations in each crystal map. Note that the figure only shows some of the available options in the refinement of orientations dialog window.

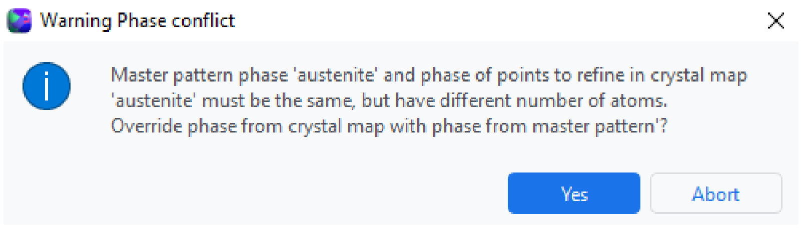
Crystal maps are extracted from each file added, or created from the index data if available. The name of the phase derived from a master pattern is then matched with the name of the phase in the crystal map so that the correct master pattern is used to refine the crystal map. This is done by iterating over each master pattern, crystal map, and the phases in the crystal map [Code 8]. By also using a navigation mask, we can ensure that only one phase is present at a time during the refinement process, which also makes it possible to refine multiple phases which originate from the same crystal map. The solution also allows for refining of the same phase from different crystal maps of the same ROI, which can have been indexed using different parameters, and the highest scoring solution would be kept after merging.

While developing the solution, kikuchipy released v.0.8.4, which takes `not_indexed` points into account when refining orientations. The navigation mask was therefore modified to include `not_indexed` points. However, it was discovered that merging crystal maps, which contain these points, would result in all points being labeled `not_indexed`. This was fixed later in the release of kikuchipy v.0.8.5. Unfortunately, this version of kikuchipy was not available before the release of EBSP Indexer v0.1.0, so a placeholder for the needed code was implemented and labeled *TODO* [Code 8]. Therefore the same problem described in Section 3.4.5 may occur when refining orientations in a crystal map created using HI. It was decided not to delay the feature due to this minor bug, and the fix would instead be shipped in a future v0.1.1 patch, once kikuchipy v0.8.5 was released.

```
⋮
ref_xmaps: list[CrystalMap] = []
ref_xmaps_navs: list[np.ndarray] = []
for mp_phase_name, mp in master_patterns.items():
    print(f"\nRefining with Master Pattern: {mp.phase.name}")
    for xmap in xmaps.values():
        for phase_id, phase in xmap.phases_in_data:
            if phase.name == mp_phase_name:
                # TODO Switch -2 with -1 when kikuchipy can merge not_indexed
                nav_mask_phase = ~np.logical_or(
                    xmap.phase_id == phase_id,
                    xmap.phase_id == -2,
                )
                nav_mask_phase = nav_mask_phase.reshape(xmap.shape)
                refined_xmap = s.refine_orientation(
                    xmap=xmap,
                    detector=det,
                    master_pattern=mp,
                    energy=energy,
                    navigation_mask=nav_mask_phase,
                    signal_mask=signal_mask,
                    trust_region=[1, 1, 1],
                    method=method,
                    method_kwargs=ref_kwargs,
                    compute=True,
                )
                ref_xmaps.append(refined_xmap)
                ref_xmaps_navs.append(nav_mask_phase)
⋮
```

Code 8: Solution for refining orientations in single or multiple crystal maps. The navigation mask should include phases with `id == -1` once kikuchipy v.0.8.5 is released. Note that this code does not contain the extraction of phases, nor the merging of refined crystal maps. Taken from *scripts/refinement.py*

To consider cases where the structure of the phase from the crystal map does not match the structure of the phase from the master pattern, a warning is displayed asking if the user would like to override conflicting structures [Figure 28]. The warning is displayed for each conflicting couple and requires confirmation before refinement can begin. Comparisons were done by using a hidden/private method of `kikuchipy.signals.util._crystal_map` named `_equal_phase`. Without overriding the structure of the conflicting phase, refinement would result in an error raised by kikuchipy, and that is why the second option of the warning is to abort refinement [Figure 28].


Warning Phase conflict

Master pattern phase 'austenite' and phase of points to refine in crystal map 'austenite' must be the same, but have different number of atoms.
Override phase from crystal map with phase from master pattern'?

Figure 28: Warning displayed when there is a conflict between phases. The message stating what is different is returned from the `_equal_phase` method.

5 Results

This section presents the software and other resources which were produced as a result of the work of Relling [2], Leth-Olsen [3], and this thesis. The contributions of this thesis are presented in more detail, where features are based on the released version of the software. A poster and an abstract description were also made, with the goal of informing potential users of the application. The plan is to display the poster at the venue of The 20th International Microscopy Congress (IMC20).

5.1 Software

The software developed, titled *EBSP Indexer*, provides users with a GUI that is capable of inspecting, processing, indexing and analyzing EBSPs acquired by a SEM. EBSP Indexer version 0.1.0 became the first official release of the software and was made available to download from the DOI: *10.5281/zenodo.7925262* provided by Zenodo [4], and SourceForge [7]. The available downloads include an installer for Windows and an app for macOS. The following OS versions and system architectures are supported:

- Windows 10/11 and macOS 13 Ventura
- x86_64 based CPU

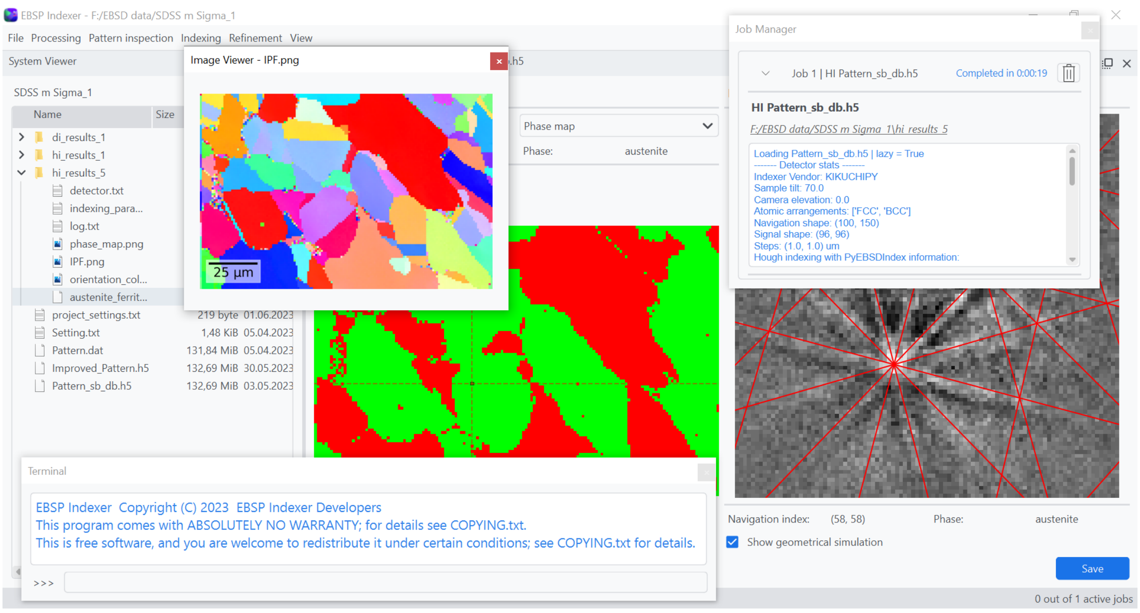
In addition, support for ARM64 based CPUs is labeled *experimental*, and currently not available to download as a packaged app. In the meantime, users of ARM64 may install the software from the source code. See Section 5.2 for more about this. The software is open-source and licensed under the terms and conditions of the GNU General Public Licence (GPL) v3.0.

Many core features of kikuchipy are made accessible through the use of GUI-elements, tied together by the application's main window [Figure 29]. The software and its feature are the results of the collective contribution of the EBSP Indexer Developers. Some features are however characterized by the involvement of certain contributors. Features offered by the GUI with respective authors are:

- Dictionary Indexing[1]
- Hough Indexing[2] [Figure 32, Figure 33]
- Refinement of Orientations[2] [Figure 34]
- Signal-to-noise Improvement[1,2] [Figure 31]
- Customizable Region of Interest[1]
- PC Calibration based on patterns[3]
- PC Calibration Curve based on Working Distance [3,1]
- Signal Navigation[1]
- Export Pre-indexing maps[1]
- Image Viewer[3]
- System Explorer[2,1] [Figure 30]
- Interactive Terminal[2] [Figure 35, Figure 36]
- Job Manager and Utilities for Threads[2] [Figure 35, Figure 36]
- Application Settings[3,1]

For a more thorough showcase of features, the manual by Relling [2] is recommended.

[1]Relling, [2]Østvold, [3]Leth-Olsen


EBSP Indexer - F:/EBSD data/SDSS m Sigma_1
File Processing Pattern inspection Indexing Refinement View
System Viewer
Image Viewer - IPF.png
25 µm
Phase map
Save
Show geometrical simulation

SDSS m Sigma_1

Name	Size	
di_results_1		
hi_results_1		
hi_results_5		
detector.txt		
indexing_para...		
log.txt		
phase_map.png		
IPF.png		
orientation_col...		
austenite_ferrit...		
project_settings.txt	219 byte	01.06.2023
Setting.txt	1,48 KiB	05.04.2023
Pattern.dat	131,84 MiB	05.04.2023
Improved_Pattern.h5	132,69 MiB	30.05.2023
Pattern_sb_db.h5	132,69 MiB	03.05.2023

Phase: austenite

Job Manager

Job 1 | HI Pattern_sb_db.h5 Completed in 0:00:19

HI Pattern_sb_db.h5

F:/EBSD data/SDSS m Sigma_1\hi_results_5

```
Loading Pattern_sb_db.h5 | lazy = True
------- Detector stats -------
Indexer Vendor: KIKUCHIPY
Sample tilt: 70.0
Camera elevation: 0.0
Atomic arrangements: ['FCC', 'BCC']
Navigation shape: (100, 150)
Signal shape: (96, 96)
Steps: (1.0, 1.0) um
Hough indexing with PyEBSDIndex information:
```

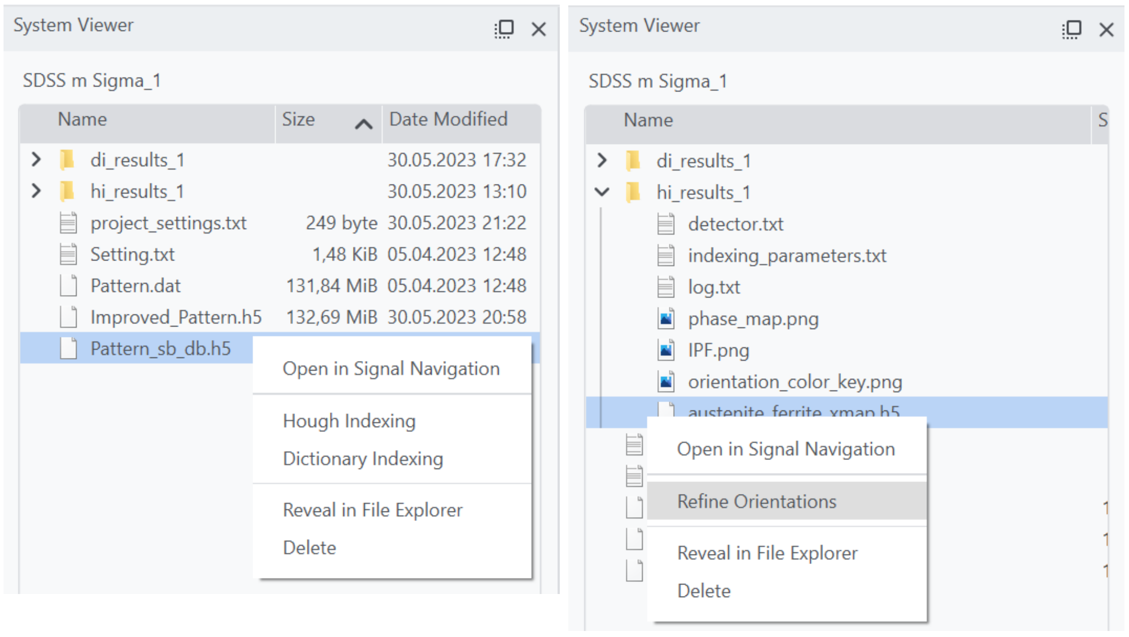
Navigation index: (58, 58) Phase: austenite

Terminal

```
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.
>>>
```

0 out of 1 active jobs

Figure 29: The application's main window has been made customizable by using the docking functionality provided by Qt. An up-scaled version of the image can be found in appendix C. The image demonstrates how dock windows may be unlocked from the main window. Any of these dock windows may be moved to other monitors, resized, hidden, or placed back into the main window. The added View menu in the toolbar at the top allows hidden windows to be shown or hidden again. The image shows Image Viewer (top-left corner), Job Manager (top-right corner), and Terminal (bottom-left corner) being unlocked from the remaining main window. The number of active jobs running, and allowed to run simultaneously, is shown in the bottom-right corner.



SDSS m Sigma_1

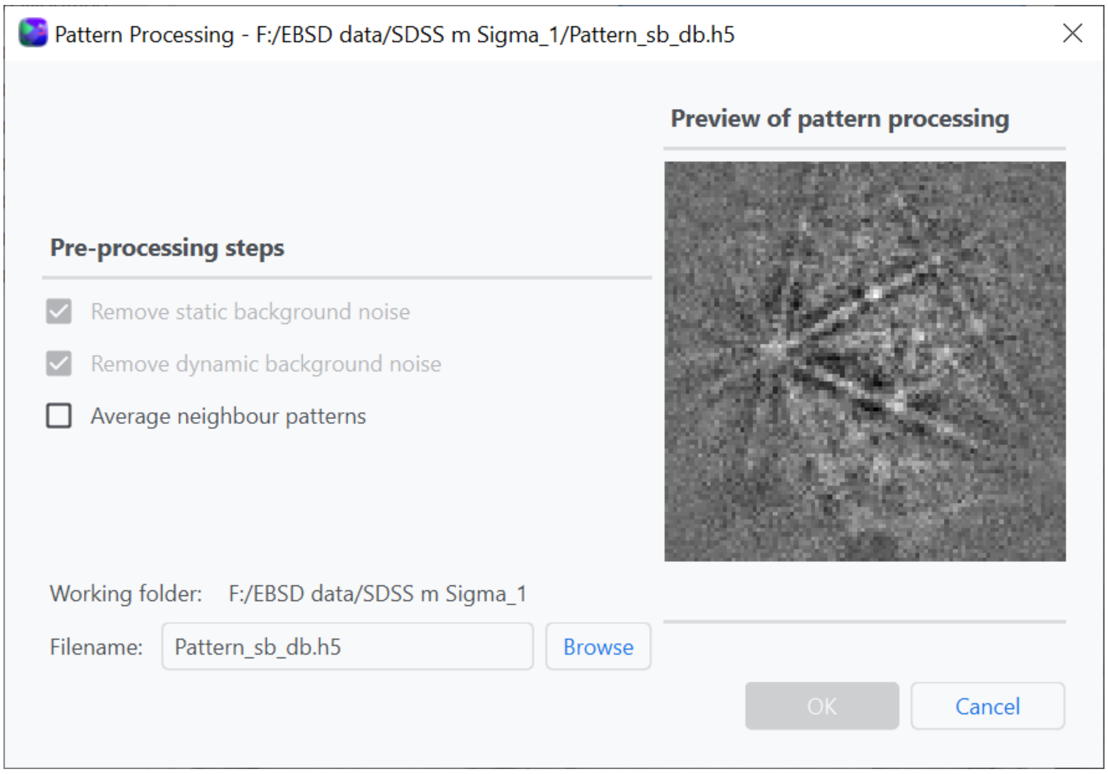
Name	Size	Date Modified
di_results_1		30.05.2023 17:32
hi_results_1		30.05.2023 13:10
project_settings.txt	249 byte	30.05.2023 21:22
Setting.txt	1,48 KiB	05.04.2023 12:48
Pattern.dat	131,84 MiB	05.04.2023 12:48
Improved_Pattern.h5	132,69 MiB	30.05.2023 20:58
Pattern_sb_db.h5		

SDSS m Sigma_1

Name
di_results_1
hi_results_1
detector.txt
indexing_parameters.txt
log.txt
phase_map.png
IPF.png
orientation_color_key.png
austenite_ferrite_xmap.h5

(a) Menu shown for an EBSD dataset (b) Menu shown for a crystal map

Figure 30: The `System Viewer` has been extended to support additional context menus that are shown when a file is right-clicked. The options of the menu depend on the type of file selected, e.g. a dataset has the option to do indexing (a), and a crystal map will show the option to refine orientations (b). All files have the option to be revealed in the file explorer of the system or be deleted. Deletion is done through an additional confirmation prompt to avoid users accidentally deleting files. The top option of the menu can be quickly accessed by double-clicking on the file.


Pattern Processing - F:/EBSD data/SDSS m Sigma_1/Pattern_sb_db.h5

Preview of pattern processing

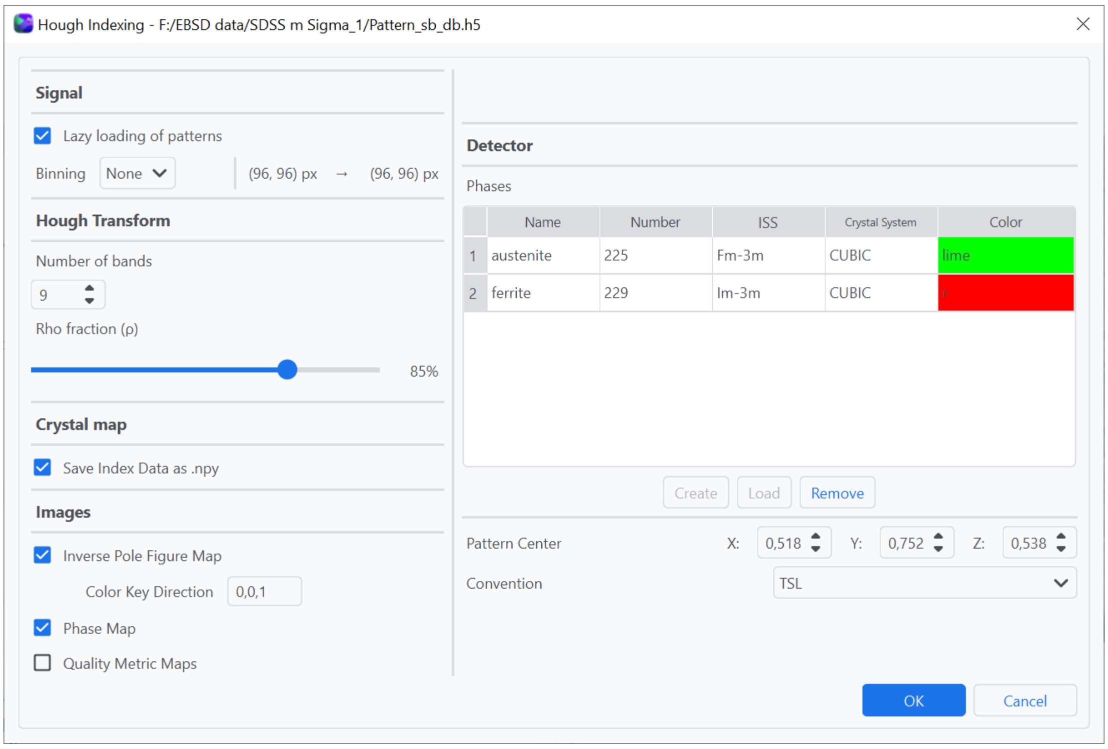
Pre-processing steps

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

Working folder: F:/EBSD data/SDSS m Sigma_1

Filename: Pattern_sb_db.h5

Figure 31: User restrictions were introduced to signal-to-noise improvement dialog, to avoid users applying the same processing to already improved patterns. If processing is applied without removal of static background, a warning is displayed due to DI relying on relative intensities (Section 2.1).


Hough Indexing - F:/EBSD data/SDSS m Sigma_1/Pattern_sb_db.h5

Signal

- [x] Lazy loading of patterns

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

Hough Transform

Number of bands: 9

Rho fraction (ρ): 85%

Crystal map

- [x] Save Index Data as .npy

Images

- [x] Inverse Pole Figure Map

Color Key Direction: 0,0,1

- [x] Phase Map
- [ ] Quality Metric Maps

Detector

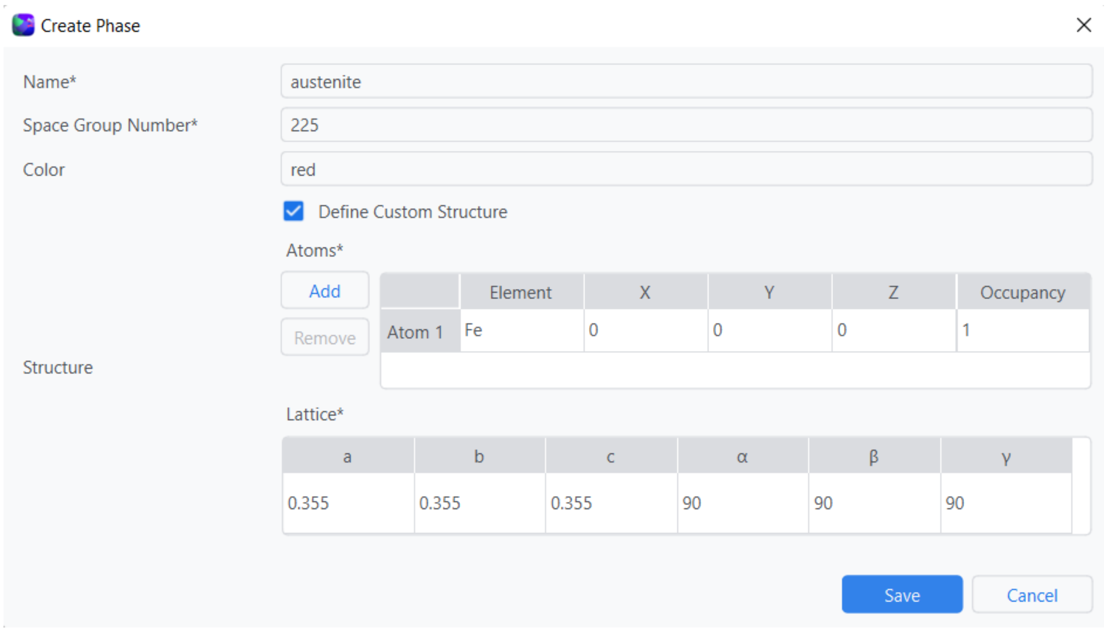
Phases

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

Pattern Center X: 0,518 Y: 0,752 Z: 0,538

Convention: TSL

Figure 32: Final version of the overhauled dialog window for Hough Indexing (HI). The binning option and phase table were improved to provide more information. The ability to save index data of the crystal map was added to improve later refinement of orientations of multi-phased samples. Specifying color key direction is available when IPF map is selected, and a number of restrictions and checks were implemented to ensure phases are compatible with HI.



Create Phase

Name*	austenite
Space Group Number*	225
Color	red

[✓] Define Custom Structure

Structure

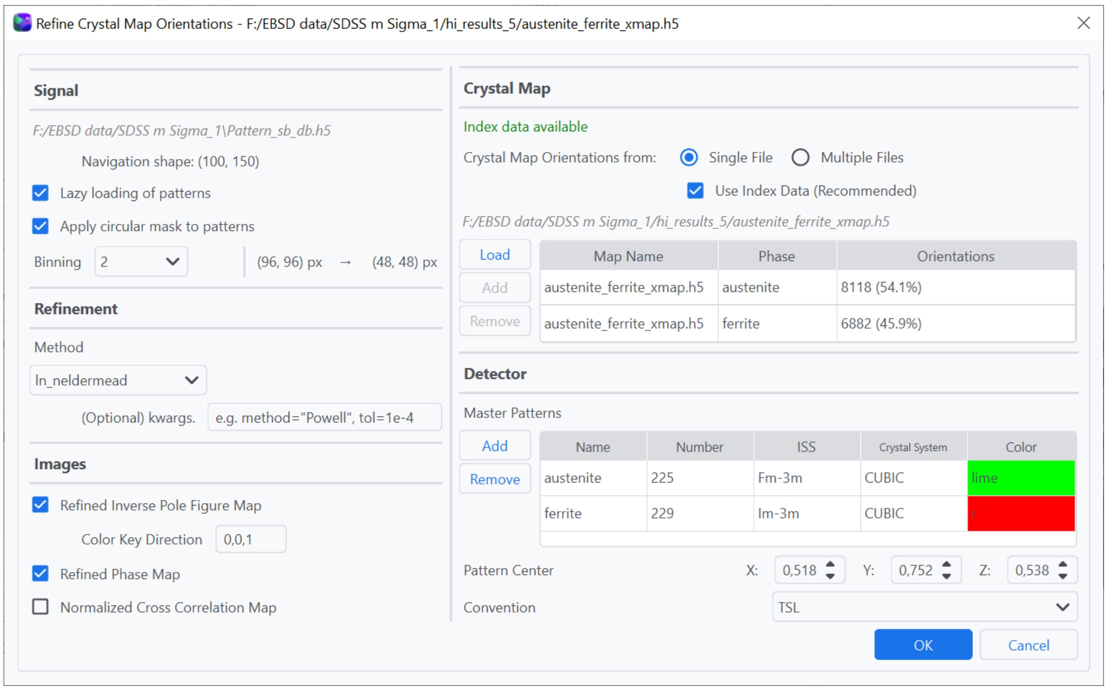
Atoms*

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

Lattice*

a	b	c	α	β	γ
0.355	0.355	0.355	90	90	90

Figure 33: New functionality for specifying a custom phase which can be used in HI. The image shows how austenite may be defined, which requires the space group number to be given. Specifying color is optional, and if none is given the default phase colors defined in the application settings are used. Defining a custom structure is optional, but is required for geometrical simulations to be properly displayed in Signal Navigation. Multiple checks are performed to guarantee that legal arguments are given.



Refine Crystal Map Orientations - F:/EBSD data/SDSS m Sigma_1/hi_results_5/austenite_ferrite_xmap.h5

Signal

F:/EBSD data/SDSS m Sigma_1\Pattern_sb_db.h5

Navigation shape: (100, 150)

[✓] Lazy loading of patterns

[✓] Apply circular mask to patterns

Binning 2 | (96, 96) px → (48, 48) 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)

F:/EBSD data/SDSS m Sigma_1/hi_results_5/austenite_ferrite_xmap.h5

Map Name	Phase	Orientations
austenite_ferrite_xmap.h5	austenite	8118 (54.1%)
austenite_ferrite_xmap.h5	ferrite	6882 (45.9%)

Detector

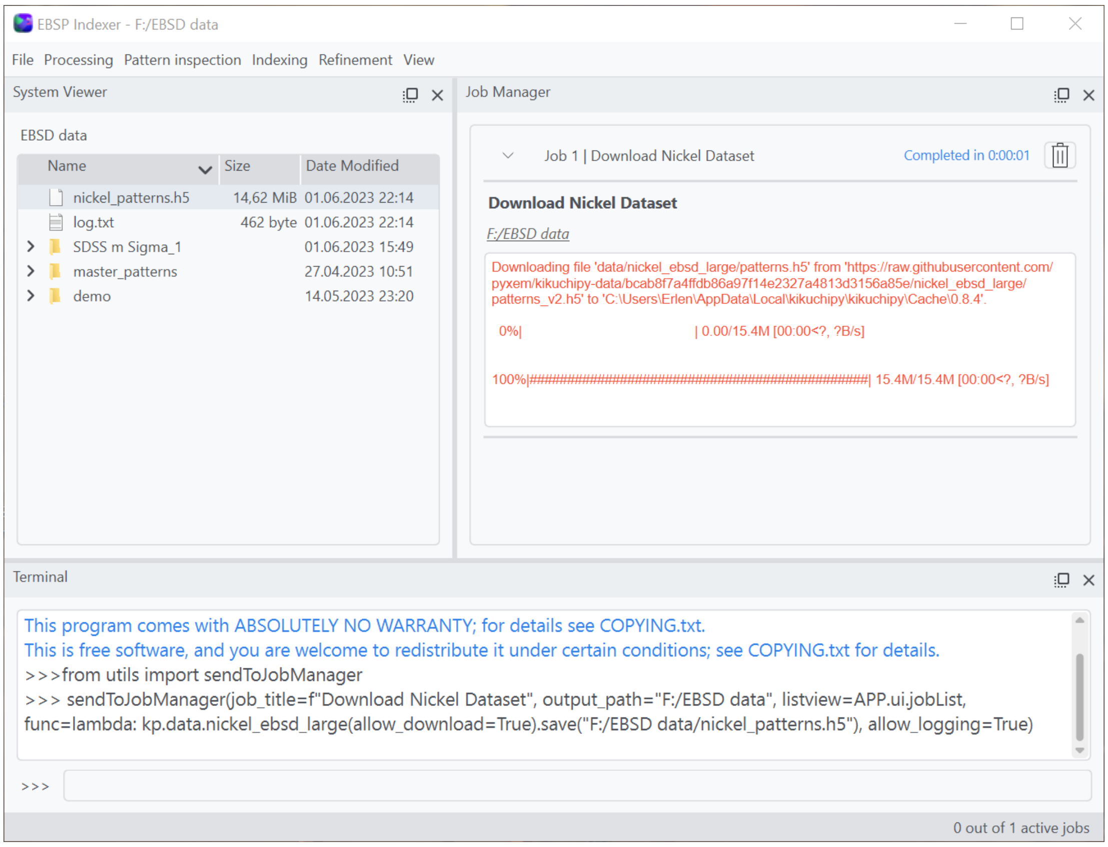
Master Patterns

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

Pattern Center X: 0,518 Y: 0,752 Z: 0,538

Convention TSL

Figure 34: Refinement of orientations dialog supports refinement of orientations from crystal maps indexed using both HI and DI. Refinement requires the original dataset, one or more crystal maps derived from the same dataset, and master patterns of the phases which are to be refined. Multiple refinement methods are available from the drop-down menu located in the Refinement group. If index data was saved during HI, it will automatically be enabled when choosing refinement for the crystal map which was derived from the index data.


EBSP Indexer - F:/EBSD data
File Processing Pattern inspection Indexing Refinement View
System Viewer
Job Manager
EBSD data

Name	Size	Date Modified
nickel_patterns.h5	14,62 MiB	01.06.2023 22:14
log.txt	462 byte	01.06.2023 22:14
SDSS m Sigma_1		01.06.2023 15:49
master_patterns		27.04.2023 10:51
demo		14.05.2023 23:20

Job 1 | Download Nickel Dataset   Completed in 0:00:01

Download Nickel Dataset

F:/EBSD data

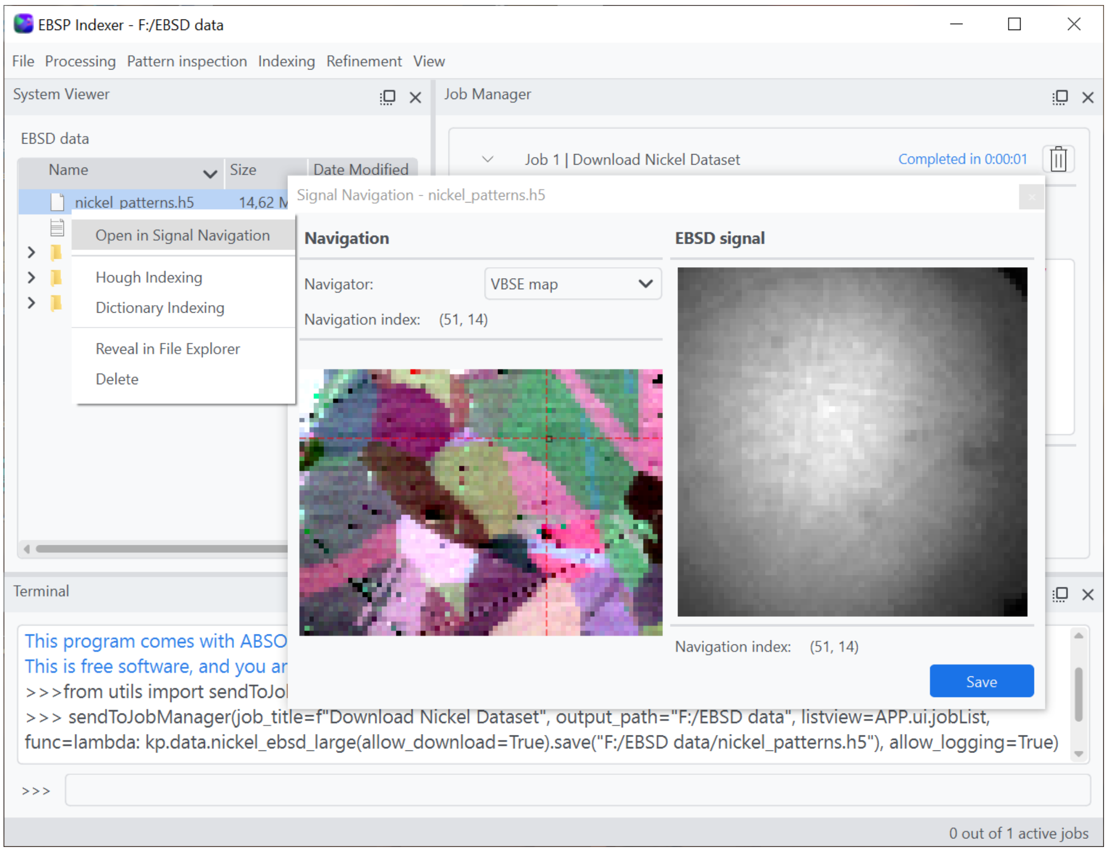
```
Downloading file 'data/nickel_ebsd_large/patterns.h5' from 'https://raw.githubusercontent.com/
pyxem/kikuchipy-data/bcab8f7a4ffdb86a97f14e2327a4813d3156a85e/nickel_ebsd_large/
patterns_v2.h5' to 'C:\Users\Erlen\AppData\Local\kikuchipy\kikuchipy\Cache\0.8.4'.

  0%|                                         | 0.00/15.4M [00:00<?, ?B/s]

100%|##################################################| 15.4M/15.4M [00:00<?, ?B/s]
```

Terminal

```
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.
>>>from utils import sendToJobManager
>>> sendToJobManager(job_title=f"Download Nickel Dataset", output_path="F:/EBSD data", listview=APP.ui.jobList,
func=lambda: kp.data.nickel_ebsd_large(allow_download=True).save("F:/EBSD data/nickel_patterns.h5"), allow_logging=True)
>>>
```

0 out of 1 active jobs

Figure 35: To showcase the flexibility of the Job Manager and the Terminal, a nickel dataset is downloaded to the current working directory. First, the `sendToJobManager` method is imported from the `utils` module of EBSP Indexer, using the terminal which acts as an interactive Python interpreter. There exists a method in kikuchipy that can download and save a small nickel dataset, and this method is passed as an argument to the `sendToJobManager` method. Calling `sendToJobManager` with appropriate arguments, shown in the terminal window, sends the kikuchipy method to a thread and adds a new job inside the Job Manager. The job then gives an overview of the running kikuchipy method. After the method is complete, the nickel dataset is saved as *nickel_patterns.h5*, which can be viewed in the System Viewer and inspected with Signal Navigation [Figure 36]. The log is also saved automatically if `allow_logging=True`.



```
This program comes with ABSO
This is free software, and you ar
>>>from utils import sendToJol
>>> sendToJobManager(job_title=f"Download Nickel Dataset", output_path="F:/EBSD data", listview=APP.ui.jobList,
func=lambda: kp.data.nickel_ebsd_large(allow_download=True).save("F:/EBSD data/nickel_patterns.h5"), allow_logging=True)
>>>
```

Figure 36: The downloaded nickel dataset from Figure 35 can be inspected using Signal Navigation to make sure the download was successful. Note that Signal Navigation was implemented by Relling [2].

5.2 Additional resources

To support an open-source community and future maintenance, a variety of user and developer resources were created and published. All of these resources, in addition to the source code, are available in the GitHub repository of the organization *EBSP Indexer* [109], and can be used free of charge within the GPL v3.0 terms and conditions. They were also made available on Zenodo [4] and SourceForge [7], in the same location where the installer and app reside. Users and developers are first greeted by the README file, found in appendix F, which gives an overview of the software's background, features, known issues of the latest release, minimum requirements, where the application is available to download, and contributing users. The repository was also made to contain a discussion forum, where users and developers can ask questions and give feedback, and the support page on SourceForge directs users to the same discussion forum.

Guides directed to developers are updated from the pre-study and include contribution guidelines, how to set up the development environment, how to update dependencies and distribute new versions of the application, and a suggested workflow on how to contribute to the project. These can be respectively found in appendices G, H, I and J.

Documentation of the source code that was contributed by this thesis includes scripts from the utility module *utils*, found in appendix K, and from the *scripts* module, found in appendix L. The documentation was made using pdoc3, an Auto-generate API documentation for Python projects [110].

Scripts for exporting the application and for creating the installer can be found in *export.py* and *setup_script.iss*, which are included in the root directory of the source code.

5.3 IMC20 abstract and research poster

An abstract description of EBSP Indexer was submitted to The 20th International Microscopy Congress (IMC20), a venue where participants from all over the world present cutting-edge research in microscopy [111]. The congress is held every four years IMC20, is hosted by the Korean Society of Microscopy (KSM), and is scheduled to take place 10 - 15 September 2023 at the Busan Exhibition and Convention Center in Korea. A research poster was also created to be displayed at the venue, which presents the motivation behind the software, and gives a brief overview of its features.

The submitted abstract and poster can be found respectively in Appendix D and E, and were written in cooperation with J. Hjelen and H. W. Ånes. At the venue, Y. Yu will be the presenting author.

6 Discussion

This section discusses aspects of the development process, the usability test, the final software, and the foundation that is to provide an open-source community. The intention is to highlight both positive and negative aspects to better understand what worked and what did not.

6.1 Development process

The development process has been characterized by multidisciplinary work within materials science, computer science, design of UX, and publishing of software. The development team has consisted of members with complementary competence within these disciplines, which has been of great value.

According to GitHub Insights, the complete project, which includes source code, resources and documentation, consists of 33 144 lines of code and text [Figure 37]. This is however somewhat misleading, as icons used in the GUI are converted to binary code and put into the Python script *resources_rc.py*, which consists of over 10 000 lines. This most likely explains the two massive peaks of contributed code found in Figure 37 during the transition from EBSD-GUI Pre-release to EBSP Indexer v.0.1.0. The first peak should instead be closer to the amount of code deleted in the same period, estimated to be around 2 000 lines. The second peak, which includes an addition and deletion of the same size, should most likely be neglected altogether, due to *resources_rc.py* being edited. Qt UI files are also technically counted twice, due to also being compiled to Python scripts. To get a more accurate overview of the work, which excludes auto-generated files, icons, images and licensing, a website named *Count LOC* [112] can help. The website states that the source code consists of 11 126 lines of code, where 5 015 lines define the Qt GUI, which seems to be a much more accurate estimation.

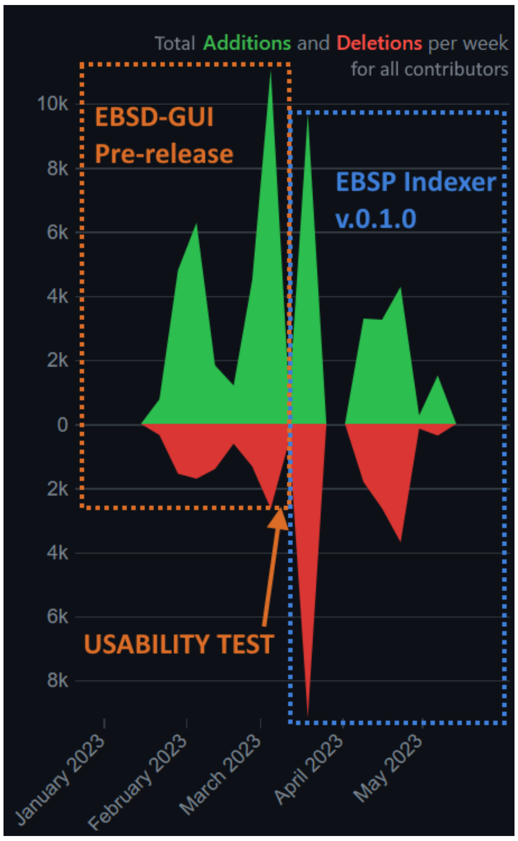


Figure 37: Contributions to the main branch of the GitHub Repository during the spring semester of 2023. The numbers in the graph on the right specify lines of code **and** text that was added and deleted from **all** contributors of the project. Note that the graph does not reflect the number of manually written lines, but instead, the amount of data edited.

Software design can be decomposed into three fundamental activities; 1. establish and understand requirements of the design, 2. create a design that aims to satisfy the requirements, and 3. evaluate the design [42, p.454]. Working in close cooperation with material scientists at the EM-lab at NTNU, through the use of weekly meetings, was valuable to the understanding of EBSD theory, and consequently, what was to be required by the interaction design. Requirements were elaborated on during the first weeks of the pre-study when the team was introduced to OIM DC, and kikuchipy through the pre-existing Jupyter notebooks used at the EM-lab. Multiple example datasets were indexed using these solutions, in order to experience the shortcomings of HI, and the inflexible and cumbersome editing of notebooks. These hands-on impressions were valuable once the designing process started, however, it would have been beneficial to study the documentation and examples provided by kikuchipy in more detail earlier on. This would have made it easier to take advantage of the structures within the classes of kikuchipy and create a design around them. This would probably not have a large impact on the design of the GUI, due to the focus of GUI usually being on input and output, but would improve code design. For example, the code implementing HI was rewritten to take advantage of the `Phase` and `PhaseList` classes from orix, and the metadata of the detector available in the `EBSD` class of kikuchipy.

The design process considered how functionality from kikuchipy could be made more easily accessible through interaction with GUI-elements, and how different elements would interact with each other. For example, a lot of functionality was built around the System Viewer [Figure 30], as functionality like inspecting, processing and indexing would require easy access to data. The design of the software is characterized by dividing functionalities into different menus and tools, which is heavily influenced by being a team project with three contributing members. This encouraged all members to work in parallel on different functionality that did not overlap, greatly improving the rate at which code was written. The performance of the code writing was however exposed to bottlenecks whenever the implementation of new functionality would require other features to already be in place. Attempts at implementing the fundamental functionality and utilities in the beginning were made, e.g. the System Viewer and the native File Explorer, for opening directories, and selecting and saving files. This was very successful, and these utilities were used throughout many features.

As development continued, the requirement for more utilities and fundamental features became apparent, e.g. threading of methods, and saving and loading of application settings. This would delay the production of functionality somewhat, as each member would eventually develop utilities that were specific to the functionality they were working on. An example of this is the phase lists shown in different dialog windows throughout the GUI, which is defined similarly in all scripts, resulting in boilerplate code. Instead, one should define a single generalized phase list that can be utilized throughout all dialog windows. The consequences of this became apparent when overhauling the phase list in HI, described in Section 3.4.1, which meant the phase list of DI and pattern center calibration had to also be overhauled to maintain consistency. Due to time constraints, this resulted in the phase lists related to pattern center calibration using the old design, while HI, DI and refinement of orientations use the new and improved one.

It is often difficult to predict what code needs to be generalized to be used throughout the software, and it will not be clear before consequences suddenly appear. There is a principle within programming called D.R.Y code, don't-repeat-yourself code, which aims to avoid this. This should definitely be considered in future development, and an effort to reduce and generalize existing code should be done. This is especially important for an open-source project to reduce the work needed for contributing, as developers can rely on existing utilities instead of having to develop their own. One should also keep in mind to extend this philosophy to include *don't-repeat-each-other* code, as the inconsistent phase list example showcases.

Another common concept within programming is the Test-Driven Development (TDD) approach, which requires developers to always write tests before new functionality is added. This is to ensure that both new and existing functionality works as intended, which is very useful when editing code in large open-source projects, e.g. kikuchipy. Unit tests are also useful in large projects, where tests are written for the smallest amount of code that can logically function on its own. Neither a TDD approach nor unit tests were implemented, as the project was regarded to be of a smaller scale considering that a lot of functionality was already provided by kikuchipy. The speed at which features were developed was probably increased in the beginning due to this decision, as features could quickly be tested manually by compiling and running the Python scripts. Manually testing features have however become much more cumbersome and less reliable as the project has grown. Editing and adding code now may affect existing functionality in unforeseen ways, which manual testing usually will not uncover. This is a common consequence of basing the final product on a prototype, which will simply postpone testing and maintenance problems [42, p. 553]. Implementing unit tests before any more features are added, will be beneficial to the project in the long run.

6.2 Usability test approach

As mentioned in the previous section, the last step of the design process is to evaluate what has been created. It is important to involve users who represent the target group, as these are the ones who will use the software on a regular basis [42, p.456]. It may also increase the users' feeling of ownership of the software, making it more likely that they will also support it in the future [42, p.458]. This becomes even more important in open-source software, due to it mainly relying on its users for maintenance and continued support. The usability test was chosen for this very reason, as it had the potential to also capture students' interests, who might contribute to the project in the future. However, only one participant indicated a minor interest in contributing to the project. Discussions with some of the participants highlighted that the scale of the project seemed intimidating, thinking it requires a lot of previous experience with software development, something that is less common among students of material science. This is true to some extent, however, contributions should be made from a multidisciplinary team, consisting of students who study both material- and computer science.

Out of the 12 available students, only one was not able to participate in the usability test. Targeting students first was of priority, as expensive EBSD software is likely less accessible to students compared to scientists, professors, and employees in larger organizations. The number of students who took part in the lab sessions at the same time was between 2 and 4, which was appropriate for having in-depth discussions. A larger number would most likely reduce the quality of the observations, but increase the reliability of the data gathered from the questionnaire. Both approaches are useful, but one can argue that the confirmation of having many participants becomes more important as the software reaches a production-ready state. Another principle highlighted by *First Principles of Interaction Design* [44] is to avoid only testing for learnability. This is to avoid data gathering which only focuses on the initial learning curve of the software, which ends up saying nothing about the long-term use or the end-state level of productivity [44].

As mentioned in the results, all participants regarded themselves as beginners, meaning neither experienced nor advanced users of EBSD software were tested. This might have slightly shifted the development focus to regard more user-friendly features during the transition from EBSD-GUI Pre-release to EBSP Indexer v.0.1.0, as described in Section 4.4.2. Future usability tests should be conducted to ensure that the design also satisfies the requirements and expectations set by other types of users in the target group. There is however the argument that one should not trust and support the current behavior of all users, as these may contain bad habits [42, p.463]. The creativity of designers is also affected, as the focus is put on satisfying requirements enforced by feedback. Instead, one should focus on how to improve the work of users, by implementing alternative workflows which have previously not been considered. However, one should also consider that *"users don't like to deviate from their learned habits if operating a new device with similar properties (Norman, 2013)"* - Sharp et al. [42, p.463]. The impression from conducting the usability test of this thesis, is that most of the gathered data are very valuable, indicated by the number of issues discovered [Table 3]. One should however still maintain a balance between following user-dictated feedback and one's own creative solutions.

Observing the users interacting with the design and asking them to elaborate on decisions, was useful when trying to understand their behavior. Investigating behaviors highlights the user's priorities, preferences, and implicit intentions [42, p.463]. Because of the plenary session, where participants were given a walkthrough of the software before the lab sessions, one may argue that the first impressions of the users were not captured by the usability test. This might be true, however, the software does require some knowledge of EBSD in order to be usable, making the software much more advanced than your usual phone app that has no prerequisites. Impressions say this was the correct decision, especially when considering that participants had been given tasks, which were to be featured in their final report in TMT4166, and which had to be completed within the time limit of 2 hours during their lab session.

The questionnaire was mainly used to confirm observations made during the usability test. The data was easy to analyze, due to the structure of the generated report from *Nettskjema*. A few of the statements that used the Likert scale resulted in participants selecting *no opinion*, giving the impression that the statement is neither false nor true. For example, most participants had no opinion on whether there should be more parameters to adjust. This is not too surprising, considering that all participants were students with no previous experience in EBSD, making it difficult to evaluate this statement. It could also mean that the participant thinks that the particular statement is correctly balanced. The length of the questionnaire can be regarded as short, considering that almost all users completed it within 2-4 minutes. This was difficult to predict, but in retrospect, a pilot test before the usability test could have provided an estimate of the length of the questionnaire. The opportunity to add more questions for data gathering would maybe have presented itself then.

Even though all the students considered themselves beginners to EBSD software, their general computer skills seemed to vary. It appeared as though this also greatly influenced the students' efficiency when navigating through menus. Uncertain students were less likely to try buttons and were generally ”clicking” less, worrying it might negatively impact their current work. Only the most exploratory of students right-clicked files in the system explorer to index and inspect data. In defense of the uncertain students, the system explorer did not indicate the ability to either left- or right-click the files shown. Hovering the mouse pointer over a file did not change its appearance from normal marking to selective marking, a feature overlooked during development. This was however added in the later version after testing. The detail is minor but important to make a GUI more intuitive, by giving users correct feedback to their actions. These assumptions are only based on the observations made during testing and would have benefited from empirical data gathering through the questionnaire. Considering the short length of the questionnaire, this could have been achieved by adding a question where participants would rate their computer competence level.

6.3 Software evaluation

The described scope of this thesis included the development of software that utilizes a GUI to ease interaction between users and kikuchipy, and to provide this in a packaged application that is easy to install for both Windows and macOS. It is safe to say that a GUI was successfully developed, and supports interaction with a lot of functionality from kikuchipy. To which extent the interaction has been made easier is difficult to determine, as expectations may vary from user to user. The usability test conducted indicates a user-friendly experience for users who are new to EBSD, with an overall rating of 8.4 out of 10. Meanwhile, advanced users familiar with the features of kikuchipy will most likely prefer the API-approach instead of a GUI, simply due to being more flexible.

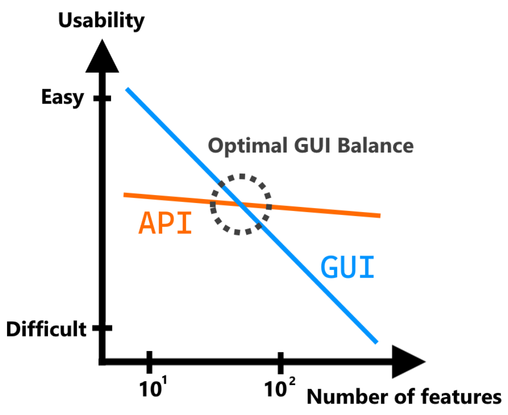


Figure 38: Usability as a function of the number of features differs depending on whether the software design is in the form of a GUI or an API.

There is an interesting balance between a GUI and an API in terms of usability and flexibility. As a GUI provides more options and features, the capabilities of the software increase, providing solutions for problems regardless of shape and size. However as the number of features approaches that of an API, the usability of the software will, without precautions, be reduced [Figure 38]. This is because it becomes more difficult to retain interaction design principles such as visibility and consistency. The amount of available visibility is finite within a display, and each feature will compete for this finite space. With a large number of different features, consistency must sometimes be given up in order for the design to make sense [42, p.64]. An API is not restricted in the same way, due to features being available through text-based code or commands, which is why usability is not affected by the number of features to the same extent as a GUI.

The developed application aims to provide a user experience within the optimal GUI balance between usability and the number of features [Figure 38]. With the software still in its early stages, v0.1.0, the GUI mainly supports basic functionalities that are to be expected from an EBSD-indexing software. Commercial software like OIM DC and AZtecCrystal operate on a much larger scale, with the support of large companies, and consequently offers many more features, options and UX improvements. Both of these applications seem to be close to an optimal GUI balance, with OIM DC arguably losing some usability due to the large number of features available, described in Section 2.2.1. Although EBSP Indexer is far from such a state, the DI capabilities offered by kikuchipy make EBSP Indexer a viable alternative to some of these commercial options, especially when considering the price of these solutions. The GUI also implements a number of widgets and tools, which aims to make it easier to work with DI and HI, e.g. the System Viewer, Signal Navigation, and different dialog windows.

An example of where the GUI does not offer the same flexibility as the API of kikuchipy, is when generating maps from indexed results. Depending on the aspect ratio and resolution of the resulting crystal maps, generating images of IPF maps and phase maps requires some adaptation to make labels and titles the correct size, and the user often has a specific preference for how the resulting images should look. It is difficult to provide the user with enough options in a GUI to mirror the capabilities of a Python library like Matplotlib, and especially without the user feeling overwhelmed. The solution should be to make the generation of images more adaptive using Matplotlib, which takes into account the size and aspect ratio of the image, and also have default values that users may configure in the application settings.

6.3.1 Design principles

Evaluating the software in terms of design principles can give an indication of how usable the product is. Functionalities are accessed through menus or windows, which can be moved, hidden or resized, all of which are actions that improve visibility. One way to make features that are burrowed in sub-menus more visible is to add a second toolbar with icons, e.g. from the Linea Iconset described in Section 3.2.8, where these features can be accessed from. This should however not be prioritized yet, seeing as the number of features offered by the GUI is limited.

The implemented Job Manager greatly improves feedback to users, e.g. by displaying inputted indexing parameters in the log. There are also minor details that provide user actions with feedback, e.g. the dialog window for setting up parameters in HI closing after parameters are submitted by clicking *Ok*, followed by the Job Manager automatically appearing if hidden. Another example of feedback is the use of pop-up dialog windows, which are displayed to make sure the user understands their actions, e.g. the exit confirmation dialog [Figure 26], and the warning displayed about Phase conflict [Figure 28]. Many constraints and error handling have been implemented to protect the user from stumbling into unwanted application states. Most of these are related to user input and actions, e.g. through the adaptive menus in the System Viewer which makes sure indexing is done using a dataset that has a format compatible with kikuchipy [Figure 30]. On multiple occasions, feedback about restrictions is given directly to the user, e.g. about PyEBSDIndex only supporting cubic crystal structures, or when index data is not available when refining orientations. These detailed messages are however not consistent throughout all features. This should be solved by implementing a more robust message utility, that can be used across all functions. Utilities are a great way to retain consistency throughout the application, and also to make editing code in a single place affecting all implementations of the utility in the application. This also means that changes made must be compatible with the different implementations of the edited utility.

Looking back on the implementation of refinement of orientations, one could have saved separate crystal maps instead of the NumPy data array during HI. This would remove the need for the additional checkbox *"Use Index Data"* [Figure 34], as refinement of orientations may be done the same way as DI, for the sake of consistency. On the other hand, since the availability of the index data is automatically detected, refinement of orientations requires fewer modifications when refining multi-phased crystal maps from HI, compared to DI. This highlights how the design process consists of iterating through alternative approaches, which are discovered by interacting with the design.

The look of the GUI provides details to indicate affordance, such as background colors changing to indicate that an item is selectable, and the mouse cursor changing appearance when an item is clickable. The use of PyQtDarkTheme also gives each GUI-element a distinct look to indicate its function and improve affordance.

Another aspect to consider is the accessibility of the features offered by the GUI. Today it is common for features to be designed or extended to support users with disabilities, as bad accessibility may end up excluding these users. Some designs take excessive control over the display, and as a result make irresponsible human-computer-interface errors, such as restricting text sizes and fonts to ones that even users with ordinary eyesight cannot read [44]. Many accessibility options are already configurable inside Windows and macOS, such as larger text sizes, automatic texting for hearing-impaired users, the ability to talk instead of using the keyboard, and many more. Qt takes advantage of many of these pre-configured accessibility options, and consequently, EBSP Indexer does so too, e.g. text sizes and scaling of GUI-elements that follow the user's pre-configured settings. Additional support was added to the System Viewer to enable the ability to use the keyboard for navigation, providing accessibility to users who might lack precise and accurate motor functions in their hands. A keyboard can also be used to navigate through most of the GUI-elements, such as dialog windows and toolbar menus, however not all features are covered. For example, operating Signal Navigation only supports the use of a mouse, which is due to this keyboard implementation requiring more effort in terms of programming complexity. Other accessibility issues were addressed in Section 4.4.2, and include the choice of colors that are beneficial to users affected by color vision deficiency and the option for using a darker GUI theme. Although mainly directed toward web applications, The W3C Web Accessibility Initiative [113] offers standards and support material to make it easier for developers to implement accessibility, which is worth looking into if the need for other accessibility options arises.

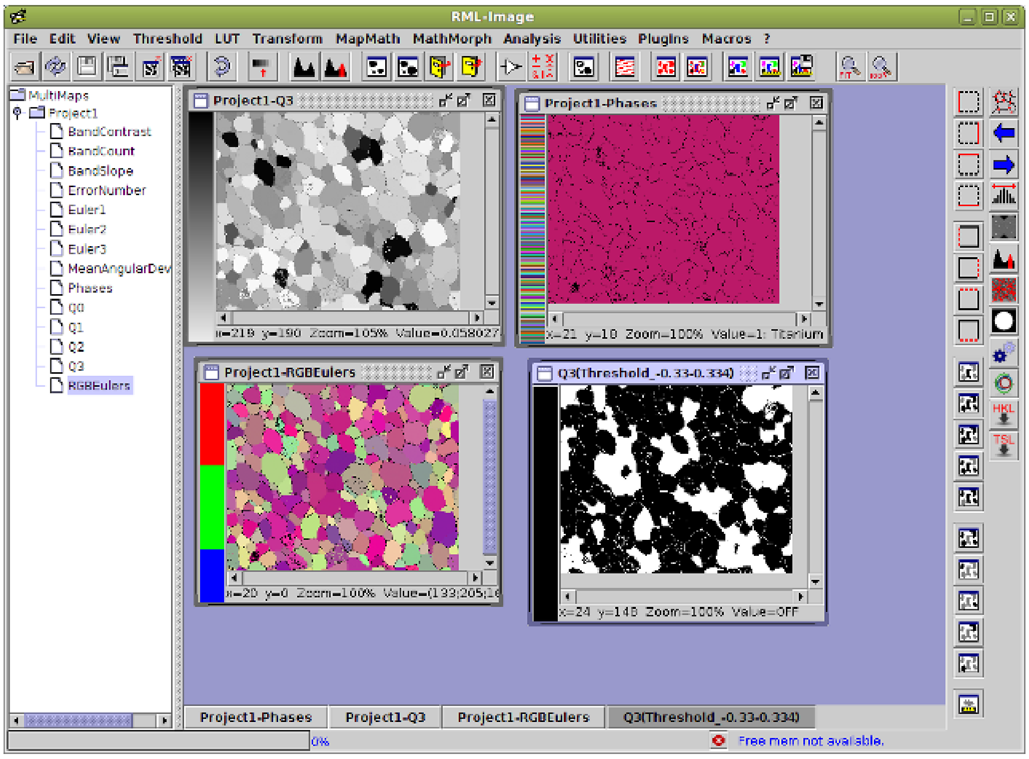
6.4 Source code and resources

The source code is available to download from 3 separate hosts, GitHub, Zenodo and SourceForge, making it accessible to anyone who would like to contribute to the development. The structure of the source code is straightforward, with Python scripts residing in folders with appropriate names that indicate their purpose, e.g. *utils* for utilities, *ui* for GUI-elements, and *resources* for icons and images. The structure would probably benefit from using more sub-folders, to improve the grouping of similar features. For example, the *scripts* folder contains almost all scripts that are not related to UI or utilities, and should be grouped into indexing, processing, pattern center, etc. Re-organization of the structure should be done once development has calmed down, to avoid incompatibilities with any features currently in development.

The quality of the source code is in many places characterized by having to meet deadlines, meaning the code was prioritized to work instead of being "beautiful". The approach has in many ways followed the philosophy of Joseph Leslie Armstrong, co-designer of the Erlang programming language, who states *"Make it work, then make it beautiful, then if you really, really have to, make it fast."* [1]. Just like how interactive design is an iterative process, so is the programming of software. For example, the code for System Viewer was originally located in *main.py*, but was moved to its own class to reduce the size of *main.py*. This improves the experience for new developers, who need to find and edit specific functionality within a feature, all of which is contained in its own dedicated Python script. This did however require the addition of Qt signals, in order for actions within the System Viewer to update other GUI-elements. This was done for Signal Navigation and Terminal as well, but the list within the Job Manager dock window is still accessed through *main.py*. The list should also be put into its own class and script, even though the code is small compared to other functionalities, as it maintains consistency for developers. One may argue that consistency within code design is equally as important as consistency in visual design, especially when developing an open-source project that should offer easy-to-edit code.

Many functions and classes throughout the source code have been given *docstrings*, string literals that are written within code with the purpose of providing documentation. These can then be used by the pdoc3 library to generate web-based documentation that is easy to navigate locally. This is meant to be a temporary solution, as the documentation is not available online, and was made to provide at least some documentation with the release of the software. Developers mainly benefit from basic knowledge of PySide6 and kikuchipy, both of which are already well documented. The combination of Python scripts and Qt-based GUI-elements are structured in a consistent way throughout the software, and uphold best practices [114]. Documentation is usually very important for the survivability of an open-source project and is even more important when the delivered software is an API. Although documentation is not strictly necessary in a GUI from a user's perspective, it is useful for a developer who joins the project mid-development. Something to also consider is the interactive Terminal with access to the running Python environment, which requires documentation in order to give users access to all the application's capabilities, showcased in Figure 35. In the future, one should look into the *Read the Docs* project, which both builds and provides hosting of open-source documentation for free.

While researching existing EBSD software, a similar project to EBSP Indexer was discovered. The project was part of the master's thesis of Philippe T. Pinard at McGill (Montréal, Québec, Canada), who developed an open-source software titled EBSD-Image that aimed to *analyze EBSD diffraction patterns and mappings* [115]. The software was written using the Java programming language, and it featured both a graphical- and a command line interface [Figure 39]. This was packaged into a Windows installer, and the software's website even states that a macOS version was under testing [115]. Unfortunately, the software was discontinued and was last updated in 2011. EBSD-Image appears to no longer be available to download.



x=219 y=190 Zoom=105% Value=0.058027
x=21 y=10 Zoom=100% Value=1: Titanium
x=20 y=0 Zoom=100% Value=(133;205;1
x=24 y=140 Zoom=100% Value=OFF

Figure 39: The GUI of EBSD-Image. Although the interface looks outdated in terms of today's expectations, it does have a similar layout to both EBSP Indexer and other EBSD software.

Source: Pinard [115]

Even though EBSD-Image provided a website, documentation, cross-platform support and many other features to establish an open-source community, the software was still discontinued. It is difficult to determine how well the software performed, and what the reason for its discontinuation was. Although the continuation of EBSP Indexer is not guaranteed either, it is safe to say that many resources have been created with the purpose of making the software succeed. Developer guides have been created with the purpose of maintaining the software and giving future developers an entry point for the project. A poster was also created to engage attendees at the IMC20. Whether these measures will be enough for the project to survive, remains to be seen.

7 Future work

There has already been developed patch v0.1.1, which is planned to release soon. The patch includes the upgrading of kikuchipy to v0.8.5, which fixes the issue related to the refinement of orientations resulting in only `not_indexed` points, described in Section 4.4.3. The release of the patch was delayed, and not included in this thesis, due to the distribution process of creating a Windows installer and a macOS app being time-consuming. In case any more issues arise, fixes for these may also be included in the patch and avoids users having to download new versions too often.

As stated in the results, the available downloads do not include an app for Macs with ARM64 architecture, even though one was made for the version which was distributed during the usability test. This is because of the instabilities experienced by Mac users during the usability test [Table 3], and it was decided to focus on ruling out issues on a single architecture at the time. Leaked semaphores persist in the macOS x86_64 app, although more rarely than before, and are highlighted by a known issue in the README file [Appendix F]. Due to time constraints and the availability of the Apple Silicon Mac in the group being limited in later stages of development, it was decided to label support for ARM64 as experimental. This version should be close to the stability of the x86_64 app. The development environment is however slightly trickier to set up to work as intended. All dependencies and their respective versions are stated in *requirements_macOS_arm64.txt*, located in the source code, however automatically installing these using Conda might fail, due to being a combination of Conda and Pip libraries.

Unfortunately, it was not enough time to develop all the planned features. The release of kikuchipy v0.8.0 highlighted the possibility of Hybrid Indexing, an approach developed by Ånes [116]. This is a combination of HI, DI and refinement, and aims to reduce indexing time by using results from HI when it is sufficient. Badly indexed map points from HI are re-indexed using DI [Figure 40], and refinement of orientations from both indexing techniques may be done before merging the results. Considering how much slower DI is than HI, it might be possible to save up to several hours of indexing time. Considering that GUI-elements already exist for HI, DI and refinement of orientations, implementing Hybrid Indexing into an EBSP Indexer version 0.2.0 should be straightforward. However, before the implementation of such a feature, one should first consider creating more GUI related utilities, to uphold the concept of D.R.Y code. This should make GUI-elements from HI and DI more reusable, which will benefit the implementation of Hybrid Indexing.

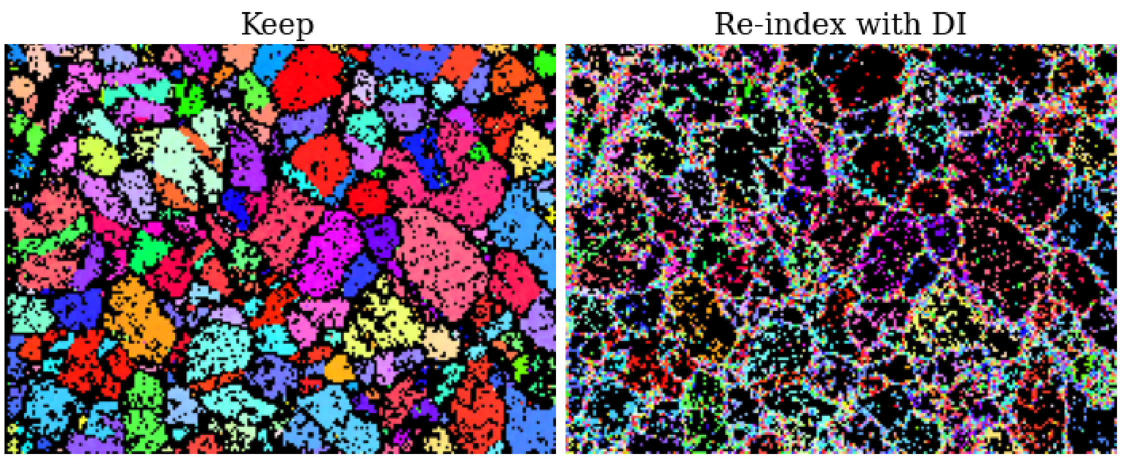


Figure 40: Re-indexing step in Hybrid Indexing, taken from the kikuchipy documentation. The maps showcase how many indexed points using HI that should be kept, and how many should be re-indexed using DI. The patterns are 60x60 pixels and are from a nickel dataset of 200x149 patterns, acquired by using a NORDIF UF-1100 detector. The fraction of points that need to be re-indexed is 43.18%.

Source: Ånes [116]

In addition to the improvements described in the discussion, it might be worth looking into some abnormalities which were discovered later in development. Optimizing the performance of the GUI should be prioritized, especially when considering that 2/3 of participants found the software slow and unresponsive at times, highlighted by the questionnaire. The GUI will stop responding while loading a dataset into Signal Navigation, and should therefore be loaded using a thread instead. The application also takes a long time to boot up the first time, about 1-3 minutes, due to many files being unpacked and loaded into memory. It might be worth taking advantage of lazy imports, to postpone loading of some libraries until they are needed by the application. The PyInstaller hooks described in Section 3.2.9 were written to include all files from kikuchipy, HyperSpy and PyEBSDIndex in the packaged distribution. These may be rewritten to include only required files, which can potentially reduce the size of the packaged application. On Windows, installing the application may trigger the firewall, displaying a warning stating that the application cannot be trusted. To avoid this, one may sign the installer using Microsoft's SignTool. GitHub Repositories offers a feature named *Dependabot*, which is used to detect security vulnerabilities in packages used by the software. The Dependabot in the EBSP Indexer repository displays 11 alerts related to security breaches, some of which are labeled as critical and of high priority. All of these are related to the packages used by the macOS versions, so it is recommended to look at these when investigating the leaked semaphore issue. Unfortunately, Dependabot is configured to only alert security breaches in the main repository branch, which is why these alerts were first discovered after the release of v0.1.0 when the main branch was updated.

Improvements discussed in Section 6, and future work described above, may be ordered in terms of priority. Some improvements will provide a better foundation for implementing new features and that is why these should have a higher priority.

1. Fix the known issues that are described in *README.md* [Appendix F]. The repository of EBSP Indexer will include an updated list of known issues.
2. Investigate compatibility issues on macOS using the security breaches found by the repository's Dependabot, to fix leaked semaphore objects. If nothing is found, it might be worth looking into an alternative solution for threading on macOS.
3. Distribute an app version for macOS that uses the ARM64 architecture.
4. Reduce boilerplate code by making more utility classes for reusable GUI-elements. These utilities should also be implemented in existing classes to provide consistency throughout the application and the source code.
5. Implement unit tests for existing classes and any new utilities added by point 4.
6. Implement Hybrid Indexing using unit tests and utilities from points 4 and 5. The feature should be part of the next minor update, v0.2.0 according to Semantic Versioning.
7. Create and host documentation using solutions provided by the Read the Docs project.
8. Improve accessibility by implementing keyboard support for more features, e.g. Signal Navigation.

8 Conclusion

Development of the EBSD-GUI dev, which resulted from the pre-study [1], was continued. The software used a Qt-based GUI to ease interaction with the EBSD processing, indexing and analyzing tools provided by kikuchipy and PyEBSDIndex. In this thesis, requirements for an EBSD software were established by understanding the basic theory of EBSD, principles of interaction design, and by investigating existing software solutions. These requirements, in addition to known issues of EBSD-GUI dev, were used to improve the software, which resulted in the test-ready version EBSD-GUI Pre-release. A usability test was conducted to obtain data about the performance of this version and to uncover issues that had been overlooked. The average participant rated the overall user experience as 8.4 out of 10. The gathered data and feedback were used to prepare the final result, which was the release-ready application EBSP Indexer v0.1.0. The software contributions of this thesis include:

- Overhaul of existing Hough Indexing implementation
- Addition of Refinement of orientations, implemented as a feature instead of an option that is only available in Dictionary Indexing
- Addition of Job Manager, and extension of utilities used for threading
- Addition of dock windows that may be moved, resized and hidden.
- Overhaul of existing System Viewer implementation
- Addition of user restrictions and feedback throughout multiple features, e.g. Signal-to-noise Improvement, Pattern Center calibration, Hough Indexing and Refinement of Orientations
- Fixes of issues discovered during the usability test, and improvements based on data from the questionnaire
- Improvements to the stability of the macOS version, including the possibility of packaging the software as an app

EBSP Indexer v0.1.0 was made distributable by using PyInstaller, Inno Setup and Platypus, and licensed using GPL v3. The distribution includes an installer for Windows and an app for macOS that supports the x86_64 architecture, and was published using Zenodo [4] and SourceForge [7]. The source code was also made available to download from these websites, in addition to the GitHub repository [109]. Multiple resources were created to support the establishment of an open-source community, which is necessary to secure future support for EBSP Indexer. Resources contributed by this thesis include:

- Publishing of EBSP Indexer v0.1.0
- Contribution guidelines
- Tutorials and guides for developers
- Documentation of the software contributions from this thesis
- Submission of an abstract to The 20th International Microscopy Congress (IMC20), and a research poster of EBSP Indexer to be presented at the venue.

Patch v0.1.1 of EBSP Indexer will be released soon, which includes fixes to Refinement or orientations, based on the release of kikuchipy v0.8.5. The stability of the macOS version was improved, but issues related to leaked semaphore objects still occur. This should be investigated further before making an attempt at packaging an app for the ARM64 architecture.

Bibliography

[1] Erlend Mikkelsen Østvold. 'Software development of a graphical user interface - Bridging the gap between electron backscatter diffraction and indexing software'. Unpublished pre-study, Norwegian University of Science and Technology, Department of Mechanical and Industrial Engineering. Dec. 2022.

[2] Hallvard Tangvik Tellefsen Relling. 'EBSP Indexer - using an open-source dictionary indexing software for phase differentiation'. MA thesis. Norwegian University of Science and Technology, June 2023.

[3] Olav Leth-Olsen. 'Distinguishing Martensite from Ferrite in Mild Steels with EBSD using Dictionary Indexing'. MA thesis. Norwegian University of Science and Technology, June 2023.

[4] Erlend Mikkelsen Østvold, Hallvard Tangvik Tellefsen Relling and Olav Leth-Olsen. *EBSP Indexer*. Version 0.1.0. May 2023. DOI: 10.5281/zenodo.7925262. URL: https://doi.org/10.5281/zenodo.7925262.

[5] Håkon Wiik Ånes et al. *pyxem/kikuchipy: kikuchipy 0.8.4*. Apr. 2023. DOI: 10.5281/zenodo.3597646. URL: https://doi.org/10.5281/zenodo.3597646.

[6] Dave Rowenhorst and Håkon Wiik Ånes. *PyEBSDIndex*. 2022. URL: https://pyebsdindex.readthedocs.io/en/latest/index.html.

[7] Erlend Mikkelsen Østvold, Hallvard Tangvik Tellefsen Relling and Olav Leth-Olsen. *EBSP Indexer*. Version 0.1.0. May 2023. URL: https://sourceforge.net/projects/ebsp-indexer/.

[8] S Singh and M De Graef. 'Orientation sampling for dictionary-based diffraction pattern indexing methods'. In: *Modelling and Simulation in Materials Science and Engineering* 24.8 (Nov. 2016), p. 085013. DOI: 10.1088/0965-0393/24/8/085013. URL: https://dx.doi.org/10.1088/0965-0393/24/8/085013.

[9] Håkan Hallberg and Mathias Wallin. *Microstructure Mechanics in Crystalline Materials*. 2023. URL: https://www.solid.lth.se/research/microstructure-mechanics-in-crystalline-materials/ (visited on 15/05/2023).

[10] Niels Christian Krieger Lassen, Dorte Juul Jensen and Knut Conradsen. 'Image processing procedures for analysis of electron back scattering patterns'. In: *Scanning microscopy* 6 (1992), pp. 115–121.

[11] Oxford Instruments. *Techniques for Electron Backscatter Diffraction (EBSD) Indexing*. 2023. URL: https://www.ebsd.com/ebsd-techniques/techniques-for-indexing (visited on 15/05/2023).

[12] Paul VC Hough. *Method and means for recognizing complex patterns*. US Patent 3,069,654. Dec. 1962.

[13] Saransh Singh et al. 'High resolution low kV EBSD of heavily deformed and nanocrystalline Aluminium by dictionary-based indexing'. In: *Scientific Reports* 8 (July 2018). DOI: 10.1038/s41598-018-29315-8.

[14] Marc De Graef et al. *EMsoft-org/EMsoft: EMsoft Release 5.0.0*. Version v5.0.0. Oct. 2019. DOI: 10.5281/zenodo.3489720. URL: https://doi.org/10.5281/zenodo.3489720.

[15] Damavandi and Esmaeil. *Why EBSD cannot distinguish the aluminum and silicon in Al-Si alloy?* July 2020. URL: https://www.researchgate.net/post/Why-EBSD-cannot-distinguish-the-aluminum-and-silicon-in-Al-Si-alloy.

[16] AMETEK Materials Analysis Division. *OIM Analysis*. 2022. URL: https://www.edax.com/products/ebsd/oim-analysis.

[17] Bruker Corporation. *Software: ESPRIT Family*. 2023. URL: https://www.bruker.com/en/products-and-solutions/elemental-analyzers/eds-wds-ebsd-SEM-Micro-XRF/software-esprit-family.html.

[18] Oxford Instruments Nanoanalysis. *AZtecCrystal*. 2023. URL: https://nano.oxinst.com/azteccrystal.

[19] F. Bachmann, Ralf Hielscher and Helmut Schaeben. 'Texture Analysis with MTEX – Free and Open Source Software Toolbox'. In: *Texture and Anisotropy of Polycrystals III*. Vol. 160. Solid State Phenomena. Trans Tech Publications Ltd, Mar. 2010, pp. 63–68. DOI: 10.4028/www.scientific.net/SSP.160.63.

[20] Thomas Benjamin Britton et al. '*AstroEBSD*: exploring new space in pattern indexing with methods launched from an astronomical approach'. In: *Journal of Applied Crystallography* 51.6 (Dec. 2018), pp. 1525–1534. DOI: 10.1107/S1600576718010373. URL: https://doi.org/10.1107/S1600576718010373.

[21] Norwegian University of Science and Technology. *TMT4166 - Course Content*. 2023. URL: https://www.ntnu.edu/studies/courses/TMT4166#tab=omEmnet (visited on 04/04/2023).

[22] Íris Carneiro and Sónia Simões. 'Recent Advances in EBSD Characterization of Metals'. In: *Metals* (2020).

[23] Shoji Nishikawa and Seishi Kikuchi. 'Diffraction of Cathode Rays by Calcite.' In: *Nature* 122.3080 (Nov. 1928), p. 726. DOI: 10.1038/122726a0.

[24] Robert Schwarzer et al. 'Present State of Electron Backscatter Diffraction and Prospective Developments'. In: Mar. 2010, pp. 1–20. ISBN: 978-0-387-88135-5. DOI: 10.1007/978-0-387-88136-2_1.

[25] Gregory S. Rohrer. '2.08 - Microstructural Characterization of Hard Ceramics'. In: *Comprehensive Hard Materials*. Ed. by Vinod K. Sarin. Oxford: Elsevier, 2014, pp. 265–284. ISBN: 978-0-08-096528-4. DOI: https://doi.org/10.1016/B978-0-08-096527-7.00027-1. URL: https://www.sciencedirect.com/science/article/pii/B9780080965277000271.

[26] Hydrargyrum. *Bragg diffraction 2.svg*. [Online; accessed 22-May-2023]. Dec. 2011. URL: https://en.wikipedia.org/wiki/Bragg%5C%27s_law#/media/File:Bragg_diffraction_2.svg.

[27] Richard O Duda and Peter E Hart. 'Use of the Hough transformation to detect lines and curves in pictures'. In: *Communications of the ACM* 15.1 (1972), pp. 11–15.

[28] Johann Radon. 'Uber die Bestimmung von Funktionen durch ihre Integralwerte langs gewissen Mannigfaltigkeiten'. In: *Berichte Schsische Akademie der Wissenschaften (Leipzig)* 69 (1917), pp. 262–277.

[29] Oxford Instruments. *Basics of Automated Indexing*. 2023. URL: https://www.ebsd.com/ebsd-explained/basics-of-automated-indexing (visited on 24/05/2023).

[30] Matt Nowell, Stuart Wright and J Carpenter. 'Differentiating Ferrite and Martensite in Steel Microstructures Using Electron Backscatter Diffraction'. In: *Materials Science and Technology Conference and Exhibition 2009, MS and T'09* 2 (Jan. 2009).

[31] Amir Baghdadchi, Vahid A. Hosseini and Leif Karlsson. 'Identification and quantification of martensite in ferritic-austenitic stainless steels and welds'. In: *Journal of Materials Research and Technology* 15 (2021), pp. 3610–3621. ISSN: 2238-7854. DOI: https://doi.org/10.1016/j.jmrt.2021.09.153. URL: https://www.sciencedirect.com/science/article/pii/S2238785421010590.

[32] Patrick G. Callahan and Marc De Graef. 'Dynamical Electron Backscatter Diffraction Patterns. Part I: Pattern Simulations'. In: *Microscopy and Microanalysis* 19.5 (2013), pp. 1255–1265. DOI: 10.1017/S1431927613001840.

[33] Aimo Winkelmann et al. 'Many-beam dynamical simulation of electron backscatter diffraction patterns'. In: *Ultramicroscopy* 107.4 (2007), pp. 414–421. ISSN: 0304-3991. DOI: https://doi.org/10.1016/j.ultramic.2006.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0304399106001975.

[34] Yu H. Chen et al. 'A Dictionary Approach to Electron Backscatter Diffraction Indexing'. In: *Microscopy and Microanalysis* 21.3 (2015), pp. 739–752. ISSN: 14358115. DOI: 10.1017/S1431927615000756.

[35] A. Foden et al. 'Indexing electron backscatter diffraction patterns with a refined template matching approach'. In: *Ultramicroscopy* 207 (2019), p. 112845. ISSN: 0304-3991. DOI: https://doi.org/10.1016/j.ultramic.2019.112845. URL: https://www.sciencedirect.com/science/article/pii/S0304399118302626.

[36] Farangis Ram et al. ‘Error analysis of the crystal orientations obtained by the dictionary approach to EBSD indexing’. In: *Ultramicroscopy* 181 (2017), pp. 17–26. ISSN: 0304-3991. DOI: https://doi.org/10.1016/j.ultramic.2017.04.016. URL: https://www.sciencedirect.com/science/article/pii/S030439911630198X.

[37] D Roşca, A Morawiec and M De Graef. ‘A new method of constructing a grid in the space of 3D rotations and its applications to texture analysis’. In: *Modelling and Simulation in Materials Science and Engineering* 22.7 (2014), p. 075013.

[38] Lars Andreas Hastad Lervik. ‘Orientation and Projection Center Refinement for EBSD Indexing in Python’. MA thesis. Norwegian University of Science and Technology, June 2021. URL: https://hdl.handle.net/11250/2785332.

[39] Saransh Singh, Farangis Ram and Marc De Graef. ‘Application of forward models to crystal orientation refinement’. In: *Journal of Applied Crystallography* 50.6 (Dec. 2017), pp. 1664–1676. DOI: 10.1107/S1600576717014200. URL: https://doi.org/10.1107/S1600576717014200.

[40] Edward L. Pang, Peter M. Larsen and Christopher A. Schuh. ‘Global optimization for accurate determination of EBSD pattern centers’. In: *Ultramicroscopy* 209 (2020), p. 112876. ISSN: 0304-3991. DOI: https://doi.org/10.1016/j.ultramic.2019.112876. URL: https://www.sciencedirect.com/science/article/pii/S030439911930292X.

[41] Aimo Winkelmann. *Electron Backscatter Diffraction (EBSD) Pattern Formation.* 2022. URL: https://www.ebsd.com/ebsd-explained/pattern-formation (visited on 26/05/2023).

[42] Helen Sharp, Jennifer Preece and Yvonne Rogers. *INTERACTION DESIGN, beyond human-computer interaction, Fifth Edition.* John Wiley & Sons, Inc., 2019.

[43] HEAVY.AI. *Technical glossary - Graphical User Interface.* 2022. URL: https://www.heavy.ai/technical-glossary/graphical-user-interface (visited on 19/05/2023).

[44] Bruce ”Tog” Tognazzini. *First Principles of Interaction Design (Revised & Expanded.* 2014. URL: https://asktog.com/atc/principles-of-interaction-design/ (visited on 18/05/2023).

[45] Ole Natlandsmyr. ‘Graphical User Interface Implementation for an Indexing Workflow of Electron Backscatter Diffraction Patterns’. MA thesis. Norwegian University of Science and Technology, July 2021. URL: https://hdl.handle.net/11250/2785344.

[46] Oxford Instruments Nanoanalysis. *Effective EBSD data processing using AZtecCrystal.* June 2021. URL: https://www.youtube.com/watch?v=UwmMC3iBSYg&ab_channel=OxfordInstruments.

[47] ©2022 AMETEK Inc. *AMETEK EDAX - Abous Us - History.* 2022. URL: https://www.edax.com/contact-us/about-us/history (visited on 19/05/2023).

[48] Mark Weiser. ‘The Computer for the 21st Century’. In: *SIGMOBILE Mob. Comput. Commun. Rev.* 3.3 (July 1999), pp. 3–11. ISSN: 1559-1662. DOI: 10.1145/329124.329126. URL: https://doi.org/10.1145/329124.329126.

[49] J. D. Hunter. ‘Matplotlib: A 2D graphics environment’. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[50] J.A. Nelder and R. Mead. ‘A simplex method for function minimization’. In: *The computer journal* 7.4 (1965), pp. 308–313.

[51] Steven G. Johnson. *The NLopt nonlinear-optimization package.* 2023. URL: http://github.com/stevengj/nlopt.

[52] Pauli Virtanen et al. ‘SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python’. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[53] Francisco de la Peña et al. *hyperspy/hyperspy: Release v1.7.3.* Oct. 2022. DOI: 10.5281/zenodo.7263263. URL: https://doi.org/10.5281/zenodo.7263263.

[54] Siu Kwan Lam, Antoine Pitrou and Stanley Seibert. ‘Numba: A llvm-based python jit compiler’. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC.* 2015, pp. 1–6.

[55] Matthew Rocklin. *Composing Dask Array with Numba Stencils.* 2018. URL: https://blog.dask.org/2019/04/09/numba-stencil (visited on 28/04/2023).

[56] Lars Lervik. ‘Larger than memory dictionaries for EBSD pattern indexing in Python’. In: (Dec. 2020). Unpublished project thesis, Norwegian University of Science and Technology.

[57] GNU. *GNU General Public License.* 2007. URL: https://www.gnu.org/licenses/gpl-3.0.html (visited on 18/04/2023).

[58] Håkon Wiik Ånes and Erlend Mikkelsen Østvold. *kikuchipy discussions - kikuchipy and hyperspy with pyinstaller #581.* 2022. URL: https://github.com/pyxem/kikuchipy/discussions/581%5C#discussioncomment-4302896 (visited on 30/04/2023).

[59] Vidar Tonaas Fauske. *HyperSpyUI - A Graphical interface for HyperSpy.* 2023. URL: https://hyperspy.org/hyperspyUI/index.html (visited on 28/04/2023).

[60] Håkon Wiik Ånes et al. *pyxem/orix: orix 0.11.1.* Mar. 2023. DOI: 10.5281/zenodo.7732341. URL: https://doi.org/10.5281/zenodo.7732341.

[61] Pavol Juhás et al. 'Complex modeling: a strategy and software program for combining multiple information sources to solve ill posed structure and nanostructure inverse problems'. In: *Acta Crystallographica Section A* 71.6 (Nov. 2015), pp. 562–568. DOI: 10.1107/S2053273315014473. URL: https://doi.org/10.1107/S2053273315014473.

[62] Håkon Wiik Ånes and Erlend Mikkelsen Østvold. *kikuchipy discussions - Hough indexing and lazy loading of patterns: Cannot determine Numba type of ¡class 'dask.array.core.Array'¿ #596.* 2023. URL: https://github.com/pyxem/kikuchipy/discussions/596#discussioncomment-4756310 (visited on 01/05/2023).

[63] Andreas Klöckner et al. 'PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation'. In: *Parallel Computing* 38.3 (2012), pp. 157–174. ISSN: 0167-8191. DOI: 10.1016/j.parco.2011.09.001.

[64] © 2023 The Qt Company Ltd. *Qt for Python.* Documentation. 2023. URL: https://doc.qt.io/qtforpython-6/ (visited on 01/05/2023).

[65] © 2023 The Qt Company Ltd. *About Qt.* 2022. URL: https://wiki.qt.io/About_Qt (visited on 01/05/2023).

[66] Yodan Tauber sepp2k. *Order of slots called on QObject.* 2015. URL: https://stackoverflow.com/a/1246964 (visited on 01/05/2023).

[67] © 2023 The Qt Company Ltd. *Threading Basics.* 2023. URL: https://doc.qt.io/qt-6/thread-basics.html (visited on 02/05/2023).

[68] © 2023 The Qt Company Ltd. *Supported Platforms.* 2023. URL: https://doc.qt.io/qt-6/supported-platforms.html (visited on 02/05/2023).

[69] Mahmoud Saleh et al. *Microsoft Visual C++ Redistributable latest supported downloads.* 2023. URL: https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170 (visited on 02/05/2023).

[70] Yunosuke Ohsugi. *auto-py-to-exe 2.26.0.* 2022. URL: https://pypi.org/project/pyqtdarktheme/.

[71] Colin Duquesnoy. *auto-py-to-exe 2.26.0.* 2021. URL: https://qdarkstylesheet.readthedocs.io/en/latest/readme.html.

[72] Dario Ferrando and Benjamin Sigidi. *Linea Iconset.* 2019. URL: https://linea.io/.

[73] David Cortesi et al. *PyInstaller Manual.* 2023. URL: https://pyinstaller.org/en/stable/ (visited on 16/04/2023).

[74] Matt Hanson. *71% of students would prefer a Mac to a PC – if they could afford one.* 2019. URL: https://www.techradar.com/news/71-of-students-would-prefer-a-mac-to-a-pc-if-they-could-afford-one (visited on 06/05/2023).

[75] Brent Vollebregt. *auto-py-to-exe 2.26.0.* 2022. URL: https://pypi.org/project/auto-py-to-exe/.

[76] Martijn Laan Jordan Russell. *Inno Setup.* 2022. URL: https://jrsoftware.org/isdl.php.

[77] Martin Prikryl MSalters. *How to install Microsoft VC++ redistributables silently in Inno Setup? - Answer.* 2023. URL: https://stackoverflow.com/a/51614652 (visited on 08/05/2023).

[78] Deanna Earley. *Creating an installer that will perform an update if an older version is already installed - Answer.* 2013. URL: https://stackoverflow.com/a/15639086 (visited on 08/05/2023).

[79] pepperpo. *Bundled MacOS App opens and closes immediately.* 2019. URL: https://github.com/pyinstaller/pyinstaller/issues/3820/%5C#issuecomment-489502526 (visited on 08/05/2023).

[80] Sveinbjorn Thordarson. *Platypus.* 2023. URL: https://sveinbjorn.org/platypus (visited on 08/05/2023).

[81] University of Bristol - Advanced Computing Research Centre. *How to publish research software.* 2023. URL: http://www.bris.ac.uk/acrc/research-software-engineering/software-howtos/how-to-publish-software/ (visited on 08/05/2023).

[82] Wikipedia contributors. *Software rot — Wikipedia, The Free Encyclopedia.* 2023. URL: https://en.wikipedia.org/w/index.php?title=Software_rot&oldid=1146126798 (visited on 08/05/2023).

[83] Kent C. Dodds et al. *All Contributors - Specification.* 2022. URL: https://allcontributors.org/docs/en/specification (visited on 08/05/2023).

[84] Coraline Ada Ehmke. *A Code of Conduct for Open Source Communities.* 2014. URL: https://www.contributor-covenant.org/ (visited on 08/05/2023).

[85] Guido van Rossum, Barry Warsaw and Nick Coghlan. *PEP 8 – Style Guide for Python Code.* 2001. URL: https://peps.python.org/pep-0008/ (visited on 09/05/2023).

[86] Carol Willing et al. *Black 23.3.0 documentation.* 2018. URL: https://black.readthedocs.io/en/stable/authors.html (visited on 09/05/2023).

[87] Tom Preston-Werner. *Semantic Versioning 2.0.0.* 2013. URL: https://semver.org/ (visited on 08/05/2023).

[88] Digitaliseringsdirektoratet. *Start and run Business - Copyright.* Nov. 2022. URL: https://www.altinn.no/en/start-and-run-business/planning-starting/protection-of-rights/copyright/ (visited on 08/05/2023).

[89] Matthew Setter. *Best Practices When Versioning a Release.* Jan. 2018. URL: https://www.cloudbees.com/blog/best-practices-when-versioning-a-release (visited on 09/05/2023).

[90] European Organization For Nuclear Research and OpenAIRE. *Zenodo.* en. 2013. DOI: 10.25495/7GXK-RD71. URL: https://www.zenodo.org/.

[91] Wikipedia contributors. *SourceForge — Wikipedia, The Free Encyclopedia.* 2023. URL: https://en.wikipedia.org/w/index.php?title=SourceForge&oldid=1146634359 (visited on 12/05/2023).

[92] Chris Hoffman. *Yes, You Can Download Software From SourceForge Again.* Jan. 2020. URL: https://www.howtogeek.com/218764/warning-don%5C%E2%5C%80%5C%99t-download-software-from-sourceforge-if-you-can-help-it/ (visited on 12/05/2023).

[93] Cristián Maureira-Fredes. *Qt for Python Release: 6.4 is finally here!* Oct. 2022. URL: https://www.qt.io/blog/qt-for-python-release-6.4-is-finally-here (visited on 12/05/2023).

[94] Antoine Beyeler and srikanthravipati. *[Bug]: TypeError: int() argument must be a string, a bytes-like object or a number, not 'KeyboardModifier'.* Oct. 2022. URL: https://github.com/matplotlib/matplotlib/issues/24155/#issuecomment-1279055783 (visited on 12/05/2023).

[95] Mitch. *PyQt5/PyInstaller app opens multiple copies in endless loop on OSX.* Mar. 2019. URL: https://stackoverflow.com/questions/54942950/pyqt5-pyinstaller-app-opens-multiple-copies-in-endless-loop-on-osx (visited on 12/05/2023).

[96] Oliver. *pyInstaller loads script multiple times.* Sept. 2015. URL: https://stackoverflow.com/a/32677108 (visited on 12/05/2023).

[97] Benny T. *Pyinstaller exe keeps opening itself.* Feb. 2019. URL: https://github.com/pyinstaller/pyinstaller/issues/4067 (visited on 12/05/2023).

[98] Wikipedia contributors. *Mac transition to Apple silicon — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-May-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Mac_transition_to_Apple_silicon&oldid=1154335221.

[99] © 2023 The Qt Company Ltd. *QDockWidget Class.* Documentation. 2023. URL: https://doc.qt.io/qt-6/qdockwidget.html (visited on 27/05/2023).

[100] © 2023 The Qt Company Ltd. *QMainWindow Class.* Documentation. 2023. URL: https://doc.qt.io/qt-5/qmainwindow.html#qt-main-window-framework (visited on 27/05/2023).

[101] Ivo. *QMainWindow with only QDockWidgets and no central widget.* 2010. URL: https://stackoverflow.com/q/3531031 (visited on 27/05/2023).

[102] Blackmagic Design Pty. Ltd. 2023. *DaVinci Resolve 18 - Video editing software.* 2023. URL: https://www.blackmagicdesign.com/products/davinciresolve (visited on 28/05/2023).

[103] Håkon Wiik Ånes and Erlend Mikkelsen Østvold. *Phase losing name and properties after refinement from single crystal map #620.* Mar. 2023. URL: https://github.com/pyxem/kikuchipy/discussions/620 (visited on 28/05/2023).

[104] Håkon Wiik Ånes and Erlend Mikkelsen Østvold. *Keeping not_indexed dummy phase after orientation refinement #621.* Mar. 2023. URL: https://github.com/pyxem/kikuchipy/discussions/621 (visited on 28/05/2023).

[105] Norwegian University of Science and Technology. *Data collection.* 2023. URL: https://i.ntnu.no/wiki/-/wiki/Norsk/datainnsamling (visited on 21/05/2023).

[106] Håkon Wiik Ånes and Erlend Mikkelsen Østvold. *PC convention / Indexer vendor changes on detector.get$_i$ndexer*#626. Mar. 2023. URL: https://github.com/pyxem/kikuchipy/discussions/626 (visited on 29/05/2023).

[107] National Eye Institute. *Color blindness.* 2019. URL: https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/color-blindness (visited on 02/04/2023).

[108] Ivan Kilin. *The best charts for color blind viewers.* 2022. URL: https://en.wikipedia.org/wiki/Mac_transition_to_Apple_silicon (visited on 05/05/2023).

[109] Erlend Mikkelsen Østvold, Hallvard Tangvik Tellefsen Relling and Olav Leth-Olsen. *EBSP Indexer Repository.* June 2023. URL: https://github.com/EBSP-Indexer/EBSP-Indexer.

[110] kernc. *pdoc3 0.10.0.* 2021. URL: https://pypi.org/project/pdoc3/.

[111] Prof. Keesam Shin et al. *About IMC20.* 2023. URL: https://www.imc20.kr/html/about/welcome.php (visited on 15/04/2023).

[112] Jolav and Ben Boyter. *Count LOC.* 2023. URL: https://codetabs.com/count-loc/count-loc-online.html.

[113] World Wide Web Consortium. *W3C Web Accessibility Initiative (WAI).* 2023. URL: https://www.w3.org/WAI/ (visited on 06/06/2023).

[114] Alan D. Moore. *QtDesigner and PyQt5: The right and wrong way to use them together.* Youtube video. 2020. URL: https://www.youtube.com/watch?v=XXPNpdaK9WA (visited on 06/06/2023).

[115] Philippe T. Pinard. *EBSD-Image.* McGill, Montréal, Québec, Canada. 2011. URL: https://ebsd-image.org/.

[116] Håkon Wiik Ånes. *kikuchipy User guide - Tutorials - Hybrid Indexing.* 2023. URL: https://kikuchipy.org/en/stable/tutorials/hybrid_indexing.html#Hybrid-indexing (visited on 07/06/2023).

Appendices

A EBSD-GUI Pre-release - Questionnaire

Nettskjema

EBSD-GUI 0.1.0 Pre-release - Review

This form is anonymous and all gathered data will be used for the single purpose of improving the software. Note that the data will be published in our master theses, which will be available to the public. If you have NOT used EBSD-GUI, please do not answer this form.

Technical

How experienced are you with EBSD software in general?

- Beginner
- Familiar
- Experienced

What operating system and architecture did you use to run the software?

If you are unsure of your system's specifications, check out chiefarchitect.com

- Windows 10 / Windows 11 (x64-based)
- macOS (Intel-based processor, 64-bit)
- macOS (M1 or M2 chip)

Did you encounter any problems when installing or running the software?

- Yes
- No

Problems

If multiple problems where encountered, please describe as many of them as possible in the questions bellow.

Briefly describe where the problem appeared. (E.g. during install, start-up, during indexing, etc.)

This element is only shown when the option 'Yes' is selected in the question 'Did you encounter any problems when installing or running the software? '

Briefly describe what the problem was. (E.g. the error message you got.)

This element is only shown when the option 'Yes' is selected in the question 'Did you encounter any problems when installing or running the software? '

Was the problem encountered solved?

This element is only shown when the option 'Yes' is selected in the question 'Did you encounter any problems when installing or running the software? '

If multiple problems, only select yes if all problems were solved.

- Yes
- No
- Don&#39;t Know

User Experience

Which of the following features in EBSD-GUI did you use?

Check multiple boxes.

- Calibration of pattern / projection center
- Signal-to-noise-improvement (Static or dynamic background removal, or averaging)

Nettskjema

- Region of interest (Cropping Pattern.dat)
- Hough indexing
- Dictionary indexing
- Creating pre-indexing maps (E.g. image quality, mean intensity, etc.)
- Signal navigation (Inspecting pattern.dat or crystal maps)
- Image Viewer (Viewing non-interactive images, e.g. *.png)
- Terminal command prompt. (Writing/ access python code inside the software)
- Settings (Changing settings of the software)

What is your opinion on the following statements?

Check one box in every row.

The software has a steep learning curve.

- Strongly disagree
- Somewhat disagree
- No opinion
- Somewhat agree
- Strongly agree

The features are easy to use and understand.

- Strongly disagree
- Somewhat disagree
- No opinion
- Somewhat agree
- Strongly agree

There should be more options / parameters to adjust.

- Strongly disagree
- Somewhat disagree
- No opinion
- Somewhat agree
- Strongly agree

The design and layout of the software is complex.

- Strongly disagree
- Somewhat disagree
- No opinion
- Somewhat agree
- Strongly agree

The different file formats are confusing.

- Strongly disagree
- Somewhat disagree
- No opinion

Somewhat agree
Strongly agree

The software is slow and unresponsive at times.

Strongly disagree
Somewhat disagree
No opinion
Somewhat agree
Strongly agree

The differences between the image viewer and signal navigation are clear.

Strongly disagree
Somewhat disagree
No opinion
Somewhat agree
Strongly agree

I was able to easily complete my assigned tasks.

Strongly disagree
Somewhat disagree
No opinion
Somewhat agree
Strongly agree

User manual

Did you have a look at the user manual at some point?

Yes
No
I did not have access to it

How useful did you find the manual?

This element is only shown when the option 'Yes' is selected in the question 'Did you have a look at the user manual at some point?'

Overview

Considering your complete experience with EBSD-GUI (from installation to indexing to results), how satisfied are you?

Do you have any final comments, ideas or feedback related to EBSD-GUI?

Would you be interested in contributing to future development of the software?

This is just to get an idea if anyone are interested.

Yes
Maybe
No

B EBSD-GUI Pre-release - Questionnaire results

Nettskjema

EBSD-GUI 0.1.0 Pre-release - Review

Oppdatert: 24. mars 2023 kl. 13:56

This form is anonymous and all gathered data will be used for the single purpose of improving the software. Note that the data will be published in our master theses, which will be available to the public.
If you have NOT used EBSD-GUI, please do not answer this form.

Technical

How experienced are you with EBSD software in general?

Antall svar: **10**

Svar	Antall	% av svar	
Experienced	0	0%	0%
Familiar	0	0%	0%
Beginner	10	100%	100%

What operating system and architecture did you use to run the software?

Antall svar: **10**

Svar	Antall	% av svar	
macOS (M1 or M2 chip)	1	10%	10%
macOS (Intel-based processor, 64-bit)	1	10%	10%
Windows 10 / Windows 11 (x64-based)	8	80%	80%

Did you encounter any problems when installing or running the software?

Antall svar: **10**

Svar	Antall	% av svar	
No	4	40%	40%
Yes	6	60%	60%

Problems

If multiple problems where encountered, please describe as many of them as possible in the questions bellow.

Briefly describe where the problem appeared. (E.g. during install, start-up, during indexing, etc.)

- kjøring av program for første gang
- Lazy loading did not work for hough indexing
- During Dictionary indexing
- Start-up
- When opening a produced h5 file
- During indexing

Briefly describe what the problem was. (E.g. the error message you got.)

- Mac skjønte ikke at det var punktum i en mappe
- GPU gammel-2018
- The OK button does not appear on the screen, unable to click on it
- File is damaged
- Internal conflict between phase name and pattern name
- one of the phases was not present and the program broke

Was the problem encountered solved?

Antall svar: **6**

Svar	Antall	% av svar	
Don't Know	0	0%	0%
No	1	16.7%	16.7%
Yes	5	83.3%	83.3%

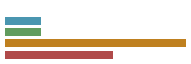
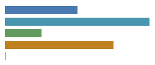
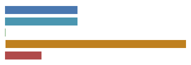
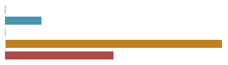
User Experience

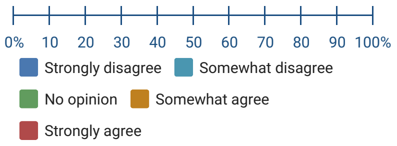
Which of the following features in EBSD-GUI did you use?

Antall svar: **10**

Svar	Antall	% av svar	
Settings (Changing settings of the software)	1	10%	10%
Terminal command prompt. (Writing/ access python code inside the software)	1	10%	10%
Image Viewer (Viewing non-interactive images, e.g. *.png)	9	90%	90%
Signal navigation (Inspecting pattern.dat or crystal maps)	7	70%	70%
Creating pre-indexing maps (E.g. image quality, mean intensity, etc.)	8	80%	80%
Dictionary indexing	10	100%	100%
Hough indexing	9	90%	90%
Region of interest (Cropping Pattern.dat)	9	90%	90%
Signal-to-noise-improvement (Static or dynamic background removal, or averaging)	10	100%	100%
Calibration of pattern / projection center	10	100%	100%

What is your opinion on the following statements?

Svar	Strongly disagree	Somewhat disagree	No opinion	Somewhat agree	Strongly agree	Diagram
The software has a steep learning curve.	2	2	3	3	0	
The features are easy to use and understand.	0	1	1	5	3	
There should be more options / parameters to adjust.	0	2	8	0	0	
The design and layout of the software is complex.	2	4	1	3	0	
The different file formats are confusing.	0	4	1	5	0	
The software is slow and unresponsive at times.	2	2	0	5	1	
The differences between the image viewer and signal navigation are clear.	1	2	1	2	4	
I was able to easily complete my assigned tasks.	0	1	0	6	3	



User manual

Did you have a look at the user manual at some point?

Antall svar: **10**

Svar	Antall	% av svar	
I did not have access to it	0	0%	0%
No	4	40%	40%
Yes	6	60%	60%

How useful did you find the manual?

Antall svar: **6** Snitt: **4.83** Median: **5**

Svar	Antall	% av svar	
5	5	83.3%	83.3%
4	1	16.7%	16.7%
3	0	0%	0%
2	0	0%	0%
1	0	0%	0%

Overview

Considering your complete experience with EBSD-GUI (from installation to indexing to results), how satisfied are you?

Antall svar: **10** Snitt: **8.40** Median: **7**

Svar	Antall	% av svar	
10	3	30%	30%
9	2	20%	20%
8	2	20%	20%
7	2	20%	20%
6	1	10%	10%
5	0	0%	0%
4	0	0%	0%
3	0	0%	0%
2	0	0%	0%
1	0	0%	0%
0	0	0%	0%

Do you have any final comments, ideas or feedback related to EBSD-GUI?

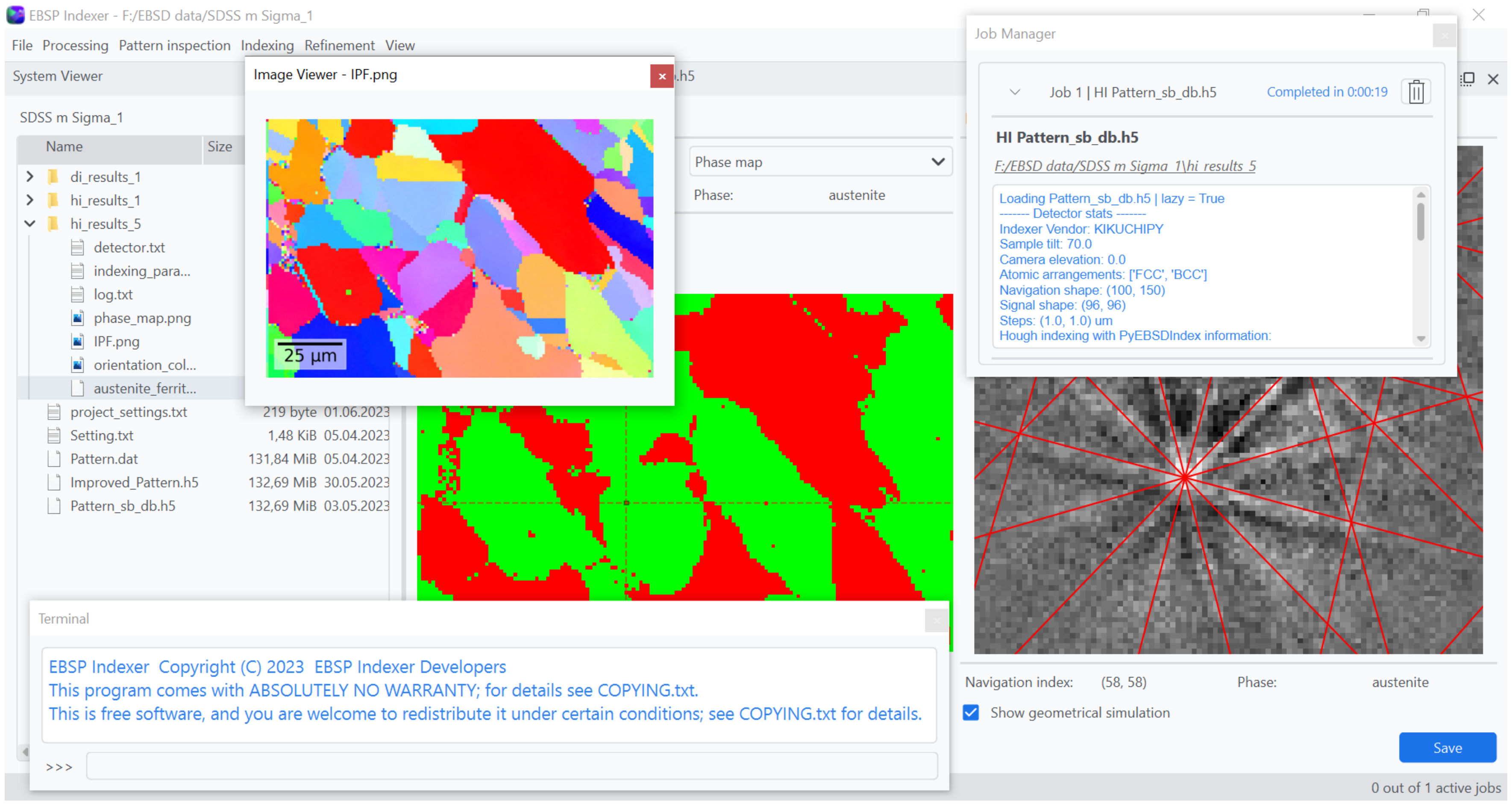
- Mulighet til å ha flere av samme type vinduer åpne, type se på forskjellen om man bruker 1. 6 step size og 4 step size, for så å legge resultatene ved siden av hverabdre for sammenligning
- The user guide was very helpful. Accessing of GPU in older computers might need refining as i could not use lazy indexing for hough.
- It was good.
- Good job guys!

Would you be interested in contributing to future development of the software?

Antall svar: **10**

Svar	Antall	% av svar	
No	9	90%	90%
Maybe	1	10%	10%
Yes	0	0%	0%

C Image of main application window


EBSP Indexer - F:/EBSD data/SDSS m Sigma_1
File Processing Pattern inspection Indexing Refinement View
System Viewer
Image Viewer - IPF.png
25 µm
Phase map
Phase: austenite
Job Manager
Save

SDSS m Sigma_1

Name	Size	
di_results_1		
hi_results_1		
hi_results_5		
detector.txt		
indexing_para...		
log.txt		
phase_map.png		
IPF.png		
orientation_col...		
austenite_ferrit...		
project_settings.txt	219 byte	01.06.2023
Setting.txt	1,48 KiB	05.04.2023
Pattern.dat	131,84 MiB	05.04.2023
Improved_Pattern.h5	132,69 MiB	30.05.2023
Pattern_sb_db.h5	132,69 MiB	03.05.2023

Job 1 | HI Pattern_sb_db.h5 Completed in 0:00:19

HI Pattern_sb_db.h5

F:/EBSD data/SDSS m Sigma_1\hi_results_5

```
Loading Pattern_sb_db.h5 | lazy = True
------- Detector stats -------
Indexer Vendor: KIKUCHIPY
Sample tilt: 70.0
Camera elevation: 0.0
Atomic arrangements: ['FCC', 'BCC']
Navigation shape: (100, 150)
Signal shape: (96, 96)
Steps: (1.0, 1.0) um
Hough indexing with PyEBSDIndex information:
```

Navigation index: (58, 58) Phase: austenite

[✓] Show geometrical simulation

0 out of 1 active jobs

Terminal

```
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.
>>>
```

D IMC20 abstract submission

EBSP INDEXER, an open-source software for Hough- and Dictionary indexing

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

Indexing of Electron Backscatter Patterns (EBSPs) has up to now mainly been performed by Hough indexing (HI) [1]. HI offers reliable indexing results, often at high speeds, but struggles to distinguish between phases of similar crystal structures and/or if patterns are very noisy. For example, HI usually fails to distinguish between aluminum and small silicon particles in an alloy since they both have an FCC structure. Dictionary indexing (DI) compares the experimentally acquired patterns with a dictionary of simulated patterns projected from a so-called master pattern of each phase, which are theoretically constructed patterns containing all scattering vectors [2]. Even though the approach of DI is slower than HI, it is powerful when indexing noisy patterns and/or separation of phases with similar crystal structures [3,4]. Datasets containing EBSPs with low (or moderate) noise and acquired from multiphase materials with different crystal structures should still apply HI, since HI is faster than DI.

DI and numerous other features are available through kikuchipy, a Python library for processing, simulating, and analyzing EBSPs [5]. EBSP Indexer was developed as an open-source software which aims to ease interaction between users and kikuchipy by implementing a graphical user interface that is easy to install. Users therefore do not require knowledge of kikuchipy, Python or programming.

EBSP Indexer allows for inspecting patterns in datasets generated by a scanning electron microscope, improve image quality of patterns, calibrate pattern/projection center, and crop additional regions of interest (ROI) before indexing the patterns using HI or DI. A number of parameters can be adjusted to make sure the indexing process fits the particular dataset. Additional refinement of orientations can also be done by iteratively optimizing the similarity between experimental and simulated patterns in a controlled manner. The result is a crystal map of the ROI which contains information about orientations, phases, metrics describing quality of the indexing, and more. Crystal maps can be saved as an .ang file or using the HDF5 file format for further analysis in other EBSD software.

The software is in early development (version 0.1.0) and will soon be available for Windows 10/11 and macOS for free. The project and source code are available at https://github.com/EBSP-Indexer.

References
1. N. C. Krieger Lassen, D. Juul Jensen, K. Conradsen. Image processing procedures for analysis of electron back scattering patterns. Scanning sicroscopy. 1992;6(1):115-21.
2. M. De Graef et al. EMsoft-org/EMsoft: EMsoft Release 5.0.0. Oct. 2019. doi: 10.5281/zenodo.3489720
3. O. Leth-Olsen et al. The challenge in differentiation between ferrite and martensite in low carbon steel by use of EBSD Dictionary Indexing, available for this conference
4. H. Relling et al. Differentiation between similar phases in an Al-10%Si alloy by EBSD dictionary indexing, available for this conference
5. H. W. Ånes et al. pyxem/kikuchipy: kikuchipy 0.8.4. 2023. doi: 10.5281/zenodo.3597646

E IMC20 EBSP Indexer poster

The poster is also included as an attached pdf for an improved viewing experience.

EBSP INDEXER, an Open-source Software for Hough- and Dictionary Indexing

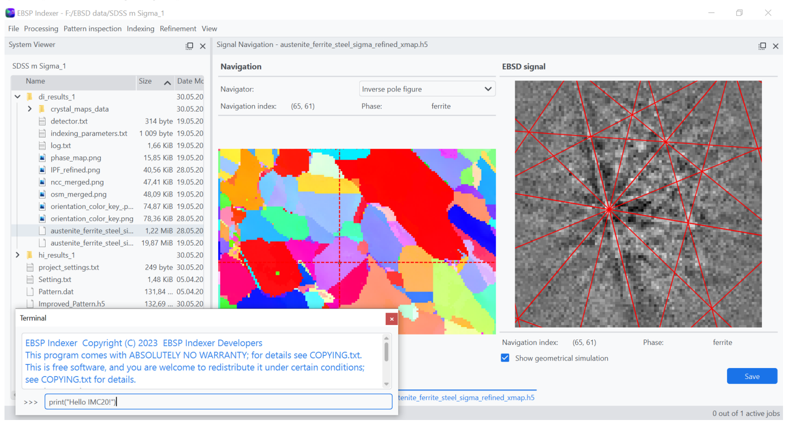
Erlend M. Østvold[a], Hallvard T. T. Relling[a], Olav Leth-Olsen[a], Håkon W. Ånes[a], Jarle Hjelen[a], Bjørn Haugen[a], Sergey Khromov[a] and Yingda Yu[a]
[a]Department of Materials Science and Engineering, Norwegian University of Science and Technology, 7491 Trondheim, Norway

Motivation

Hough Indexing (HI) [1] has been considered the standard indexing method for Electron backscatter diffraction (EBSD) mapping of microstructures in crystalline materials. The limitations of HI becomes apparent when patterns have a low signal-to-noise ratio, or when differentiating between phases of similar crystal structures. Dictionary Indexing (DI) [2] was developed to provide an alternative in cases where HI fails, which takes advantage of the full intensity information in a measured Electron Backscatter Pattern (EBSP) by matching experimental patterns with dynamically simulated ones. DI makes it possible to distinguish between phases of similar structures, e.g. silicon particles in an aluminum alloy [3], or martensite from ferrite in a low carbon steel [4].

Open-source Solution

EBSP Indexer was developed as an open-source solution that aims to ease interaction between users and the Python libraries kikuchipy [5] and PyEBSDIndex [6], which provides EBSD tools for DI and HI, by implementing a graphical user interface (GUI) that can be installed on either Windows or macOS.


EBSP Indexer - F:/EBSD data/SDSS m Sigma_1
File Processing Pattern inspection Indexing Refinement View
System Viewer
Signal Navigation - austenite_ferrite_steel_sigma_refined_xmap.h5
Navigation
EBSD signal
Navigator: Inverse pole figure
Navigation index: (65, 61)
Phase: ferrite
Show geometrical simulation
Save
0 out of 1 active jobs
Terminal

```
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.
>>> print("Hello IMC20!")
```

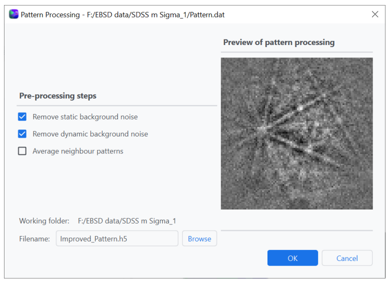
Figure 1: The main application window consists of widgets that can be resized, hidden, and displayed on multi-monitor setups. A number of dataset formats are supported, including Bruker, EDAX and NORDIF. EBSPs stored in datasets or crystal maps can be inspected using Signal Navigation. The running Python environment is accessible through the Terminal.

Features

Users require neither knowledge of Python nor programming, making it an entry-level application suitable for beginners and students.

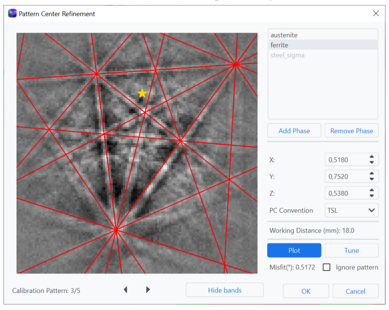
- Hough- and Dictionary Indexing
- Refinement of orientations
- Improvement of Signal-to-noise ratio in patterns
- Cropping of data, allowing a smaller region of interest
- Calibration of Pattern Center (PC) using either high quality calibration patterns, a custom selection of patterns, or a calibration curve based on different working distances
- Interactive inspection of experimental datasets and indexed crystal maps, using a variety of maps including Orientation, Phase, Image Quality, Normalized Cross-correlation and more

Dictionary Indexing of Austenite, Ferrite and Sigma phase


Pattern Processing - F:/EBSD data/SDSS m Sigma_1/Pattern.dat
Preview of pattern processing
OK
Cancel
Browse

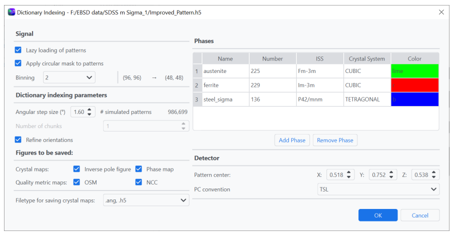
Pre-processing steps
[✓] Remove static background noise
[✓] Remove dynamic background noise
[ ] Average neighbour patterns
Working folder: F:/EBSD data/SDSS m Sigma_1
Filename: Improved_Pattern.h5

(a) Signal-to-noise Improvement



X: 0.5180
Y: 0.7520
Z: 0.5380
PC Convention: TSL
Working Distance (mm): 18.0
Misfit(°): 0.5172
Calibration Pattern: 3/5

(b) PC Calibration


Dictionary Indexing - F:/EBSD data/SDSS m Sigma_1/Improved_Pattern.h5
Add Phase
Remove Phase
OK
Cancel

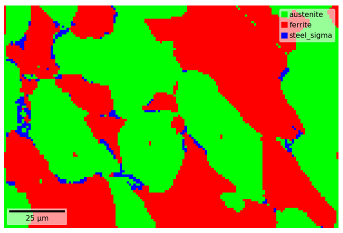
Signal
[✓] Lazy loading of patterns
[✓] Apply circular mask to patterns
Binning: 2 (96, 96) → (48, 48)
Dictionary indexing parameters
Angular step size (°): 1.60 # simulated patterns 986,699
Number of chunks: 1
[✓] Refine orientations
Figures to be saved:
Crystal maps: [✓] Inverse pole figure [✓] Phase 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
3	steel_sigma	136	P42/mnm	TETRAGONAL	🟦 b

Detector
Pattern center: X: 0.518 Y: 0.752 Z: 0.538
PC convention: TSL

(c) Dictionary Indexing

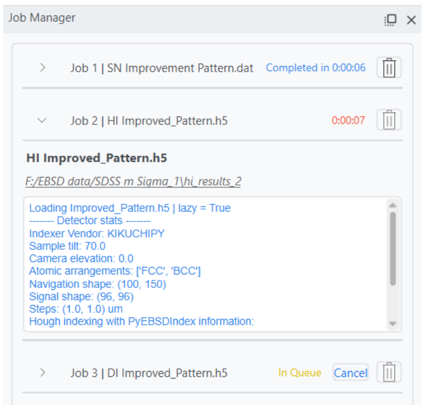


(d) Image Viewer

Figure 2: DI of EBSPs which consists of 96x96 pixels. Steps a and b are performed before DI, to obtain better indexing results.

Flexible Workflow

The interaction design resembles that of a toolbox, where users can customize parameters and mix functionalities. Users may for example use dynamical simulated master patterns to refine orientations which were indexed using HI.



```
Job 1 | SN Improvement Pattern.dat    Completed in 0:00:06
Job 2 | HI Improved_Pattern.h5        0:00:07
HI Improved_Pattern.h5
F:/EBSD data/SDSS m Sigma_1\hi_results_2
Loading Improved_Pattern.h5 | lazy = True
------- Detector stats -------
Indexer Vendor: KIKUCHIPY
Sample tilt: 70.0
Camera elevation: 0.0
Atomic arrangements: ['FCC', 'BCC']
Navigation shape: (100, 150)
Signal shape: (96, 96)
Steps: (1.0, 1.0) um
Hough indexing with PyEBSDIndex information:
Job 3 | DI Improved_Pattern.h5        In Queue
```

Figure 3: The Job Manager gives an overview of previously completed tasks, in addition to current and future tasks. The log within each job shows information from the current running task.

Available Resources

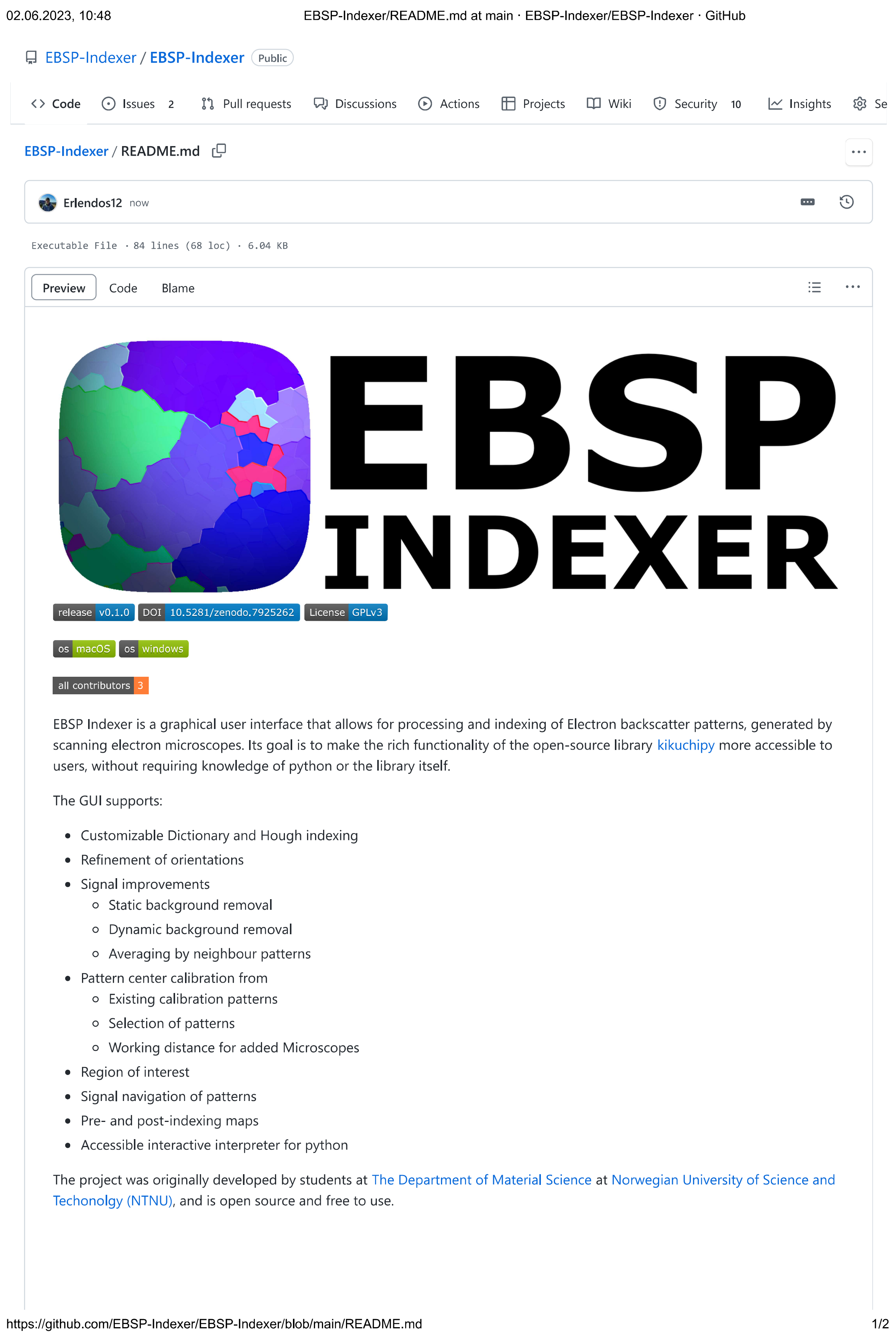
EBSP Indexer v.0.1.0 is available to download for Windows and macOS from our Zenodo DOI [7]. The source code, among other developer resources, are available in the EBSP Indexer repository on GitHub: *https://github.com/EBSP-Indexer/EBSP-Indexer*

[1] N. C. K. Lassen, D. J. Jensen, and K. Conradsen, "Image processing procedures for analysis of electron back scattering patterns," *Scanning microscopy*, vol. 6, pp. 115–121, 1992.
[2] Y. H. Chen, S. U. Park, D. Wei, *et al.*, "A Dictionary Approach to Electron Backscatter Diffraction Indexing," *Microscopy and Microanalysis*, vol. 21, no. 3, 739–752, 2015.
[3] H. T. T. Relling, O. Leth-Olsen, E. M. Østvold, *et al.*, "Differentiation between similar phases in an Al-10%Si alloy by EBSD Dictionary Indexing,", 2023.
[4] O. Leth-Olsen, H. T. T. Relling, E. M. Østvold, *et al.*, "The challenge in differentiation between ferrite and martensite in low carbon steel by use of EBSD Dictionary Indexing,", 2023.
[5] H. W. Ånes, L. Lervik, O. Natlandsmyr, *et al.*, *pyxem/kikuchipy: kikuchipy 0.8.6*, version v0.8.6, DOI 10.5281/zenodo.3597646, 2023.
[6] D. Rowenhorst and H. W. Ånes, *USNavalResearchLaboratory/PyEBSDIndex: PyEBSDIndex 0.1.1*, version v0.1.1, 2022.
[7] E. M. Østvold, H. T. T. Relling, and O. Leth-Olsen, *EBSP Indexer 0.1.0*, version 0.1.0, DOI 10.5281/zenodo.7925262, 2023.

NTNU – Trondheim
Norwegian University of Science and Technology

IMC20
The 20th International Microscopy Congress
10-15 Sep. 2023 | Busan, Korea

F EBSP Indexer README.md



02.06.2023, 10:48 EBSP-Indexer/README.md at main · EBSP-Indexer/EBSP-Indexer · GitHub

EBSP-Indexer / **EBSP-Indexer** Public

Code Issues 2 Pull requests Discussions Actions Projects Wiki Security 10 Insights Se

EBSP-Indexer / README.md

Erlendos12 now

```
Executable File · 84 lines (68 loc) · 6.04 KB
```

Preview Code Blame

EBSP INDEXER

release v0.1.0 DOI 10.5281/zenodo.7925262 License GPLv3

os macOS os windows

all contributors 3

EBSP Indexer is a graphical user interface that allows for processing and indexing of Electron backscatter patterns, generated by scanning electron microscopes. Its goal is to make the rich functionality of the open-source library kikuchipy more accessible to users, without requiring knowledge of python or the library itself.

The GUI supports:

- Customizable Dictionary and Hough indexing
- Refinement of orientations
- Signal improvements
  - Static background removal
  - Dynamic background removal
  - Averaging by neighbour patterns
- Pattern center calibration from
  - Existing calibration patterns
  - Selection of patterns
  - Working distance for added Microscopes
- Region of interest
- Signal navigation of patterns
- Pre- and post-indexing maps
- Accessible interactive interpreter for python

The project was originally developed by students at The Department of Material Science at Norwegian University of Science and Techonolgy (NTNU), and is open source and free to use.

https://github.com/EBSP-Indexer/EBSP-Indexer/blob/main/README.md 1/2

Known issues ❗

- macOS might experience leaked semaphores when creating images in threads, e.g. Inverse Pole Figure Map in Hough Indexing.

  As of now, it is recommended to run indexing **without** generating images on Mac.

- Refinement of orientations of a crystal map which includes not_indexed points, might produce results that cannot be opened in signal navigation.

- Updating the application's settings will set the current working directory to the specified default directory (if checked).

- Updating the application's settings will reset the current selected file, even though it appears to be selected.

- When saving a merged crystal map in the case where only one phase is identified, save will fail since a crystal map with the same name exists already.

- Creating jobs while the Job Manager is unlocked from the main window will result in an error, due to the parent widget of the job list changes.

Minimum requirements 🔧

- Windows 10/11 **or** macOS 13 (Ventura)
- x86_64-based CPU (arm64 chipset is experimental)

In addition, the windows version requires Microsoft Visual C++ Redistributable packages for Visual Studio 2015, 2017, 2019, and 2022. This is included in our installer, and is automatically installed if needed.

Downloads ⬇️

⚠️ Make sure to read the aboves about **Known issues** and **Minimum requirements** before downloading the software.

Installer for Windows and App for macOS are available to download from Zenodo or SourceForge.

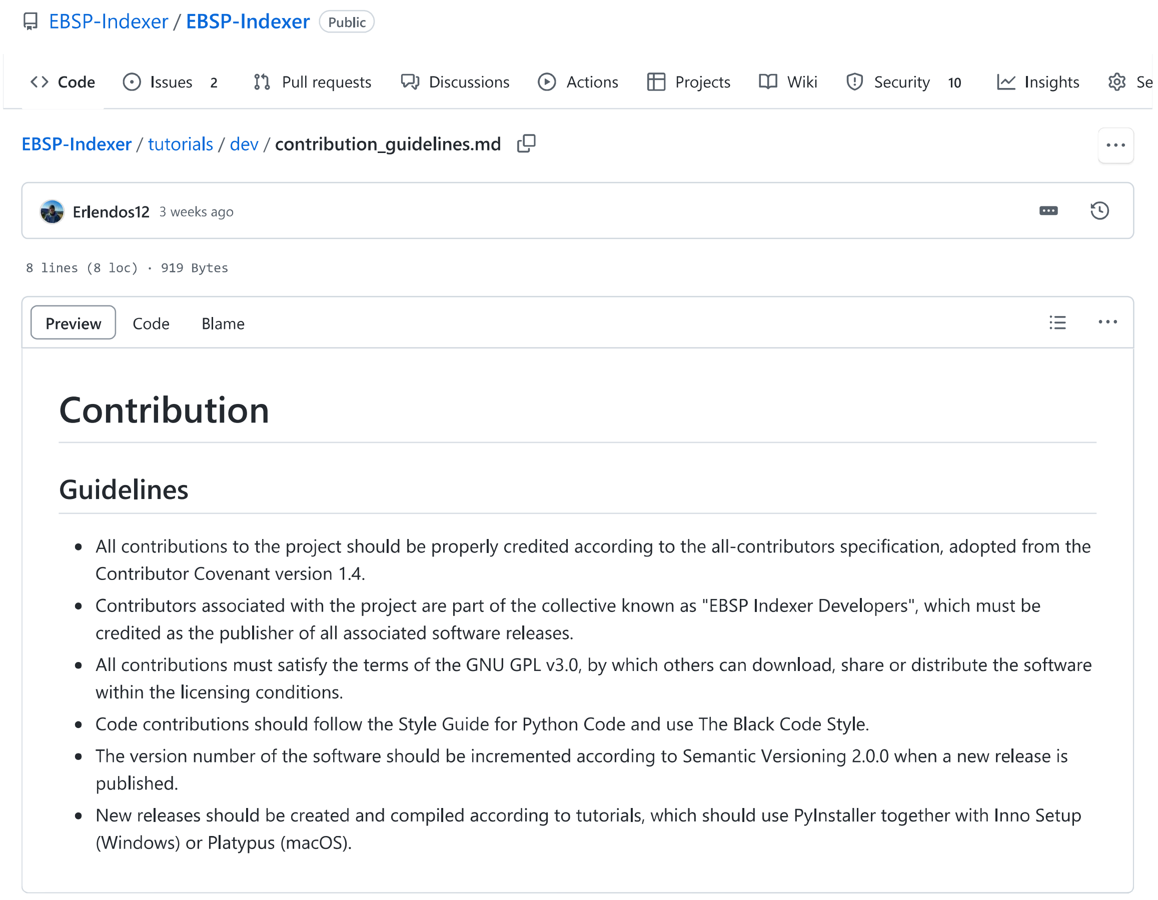
Contributors ✨

Thanks goes to these wonderful people (emoji key):

Erlend M. Østvold 💻 🐛 📖 🤔	olavlet 💻 🐛 📖 🤔	htrellin 💻 🐛 📖 🤔

This project follows the all-contributors specification. Contributions of any kind welcome!

G Developer Tutorials - Contribution guidelines


EBSP-Indexer / EBSP-Indexer Public
Code
Issues 2
Pull requests
Discussions
Actions
Projects
Wiki
Security 10
Insights
Se
EBSP-Indexer / tutorials / dev / contribution_guidelines.md
Erlendos12 3 weeks ago
8 lines (8 loc) · 919 Bytes
Preview
Code
Blame

Contribution

Guidelines

- All contributions to the project should be properly credited according to the all-contributors specification, adopted from the Contributor Covenant version 1.4.
- Contributors associated with the project are part of the collective known as "EBSP Indexer Developers", which must be credited as the publisher of all associated software releases.
- All contributions must satisfy the terms of the GNU GPL v3.0, by which others can download, share or distribute the software within the licensing conditions.
- Code contributions should follow the Style Guide for Python Code and use The Black Code Style.
- The version number of the software should be incremented according to Semantic Versioning 2.0.0 when a new release is published.
- New releases should be created and compiled according to tutorials, which should use PyInstaller together with Inno Setup (Windows) or Platypus (macOS).

H Developer Tutorials - Installation

EBSP-Indexer / **EBSP-Indexer** Public

Code | Issues 2 | Pull requests | Discussions | Actions | Projects | Wiki | Security 10 | Insights | Se

EBSP-Indexer / tutorials / dev / **development_installation.md**

Erlendos12 3 weeks ago

37 lines (33 loc) · 2.54 KB

Preview | Code | Blame

Setup of development installation

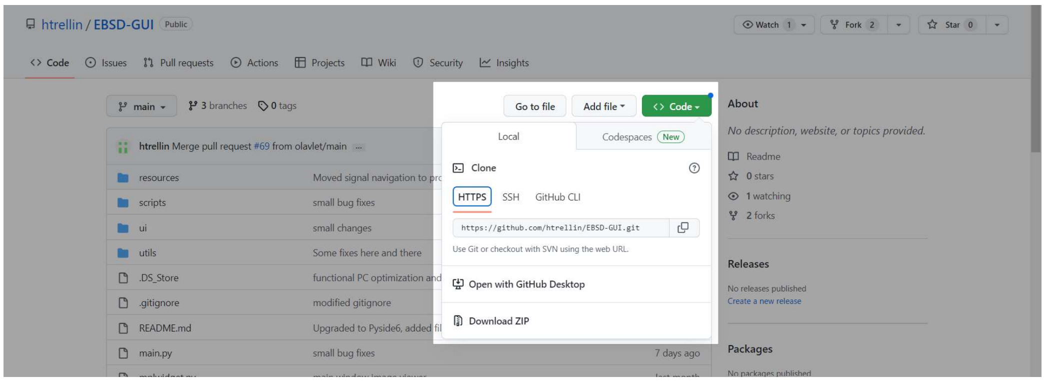
Devlopment Requirements:

- Python >= 3.10 (Tested on 3.10, but >=3.7 should work.)
- Microsoft Visual C++ Redistributable packages for Visual Studio 2015, 2017, 2019, and 2022
- Visual Studio Code (Optional, but recommended)
- Qt Designer (Optional for designing GUI elements)

In addition, it is recommended for users on MacOS with Apple Silicon to use a conda environment. See pyebsdindex's additional installation for macOS with Apple Silicon.

Setting up the repository

1. Clone or fork the repository to your local computer using git. It is recommended to put the repository just under your local disk, so that the root directory is C:/EBSP-Indexer. If you don't have much experience with git, we recommend Visual Studio Code's solution for git. Alternativly you could download the repo as a zip and upack it manually.



htrellin Merge pull request #69 from olavlet/main

resources	Moved signal navigation to pr	
scripts	small bug fixes	
ui	small changes	
utils	Some fixes here and there	
.DS_Store	functional PC optimization and	
.gitignore	modified gitignore	
README.md	Upgraded to Pyside6, added fil	
main.py	small bug fixes	7 days ago

https://github.com/htrellin/EBSD-GUI.git

Use Git or checkout with SVN using the web URL.

No description, website, or topics provided.

No releases published
Create a new release

No packages published

2. Open the repository in any terminal. The next step is to create a virtual evironment for the repository. This can be done by executing the following command in the root directory (same directory that includes requirements.txt):

   ```
   python3 -m venv env
   ```

   Then activte the enviroment in your current running terminal, e.g. for windows powershell by executing:

   ```
   & env/Scripts/Activate.ps1
   ```

   Make sure that the enviroment has been activated by executing:

   ```
   pip -V
   ```

The path returned should include ...\env\lib\site-packages\pip. NOTE: Make sure that the environment is ALWAYS activated when executing commands related to the project.

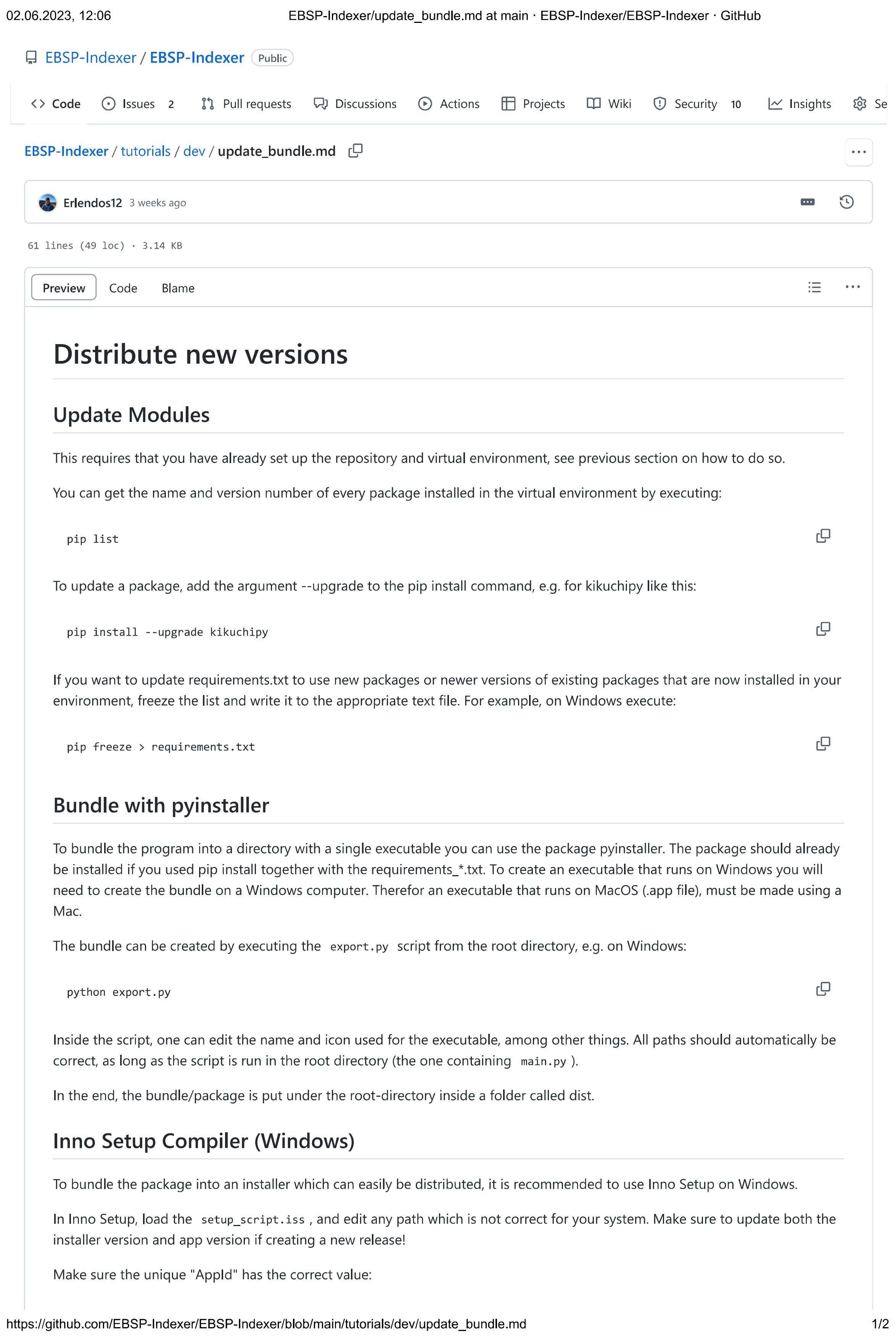
3. Once the environment is active, install the necessary packages from the appropriate requirements_*.txt for your system, e.g. for Windows:

```
pip install -r requirements_WINDOWS.txt
```

4. The program should now be able to run by executing the following in the root directory:

```
python main.py
```

I Developer Tutorials - Distribute new versions



02.06.2023, 12:06 EBSP-Indexer/update_bundle.md at main · EBSP-Indexer/EBSP-Indexer · GitHub

EBSP-Indexer / **EBSP-Indexer** Public

Code | Issues 2 | Pull requests | Discussions | Actions | Projects | Wiki | Security 10 | Insights | Se

EBSP-Indexer / tutorials / dev / update_bundle.md

Erlendos12 3 weeks ago

61 lines (49 loc) · 3.14 KB

Preview | Code | Blame

Distribute new versions

Update Modules

This requires that you have already set up the repository and virtual environment, see previous section on how to do so.

You can get the name and version number of every package installed in the virtual environment by executing:

```
pip list
```

To update a package, add the argument --upgrade to the pip install command, e.g. for kikuchipy like this:

```
pip install --upgrade kikuchipy
```

If you want to update requirements.txt to use new packages or newer versions of existing packages that are now installed in your environment, freeze the list and write it to the appropriate text file. For example, on Windows execute:

```
pip freeze > requirements.txt
```

Bundle with pyinstaller

To bundle the program into a directory with a single executable you can use the package pyinstaller. The package should already be installed if you used pip install together with the requirements_*.txt. To create an executable that runs on Windows you will need to create the bundle on a Windows computer. Therefor an executable that runs on MacOS (.app file), must be made using a Mac.

The bundle can be created by executing the `export.py` script from the root directory, e.g. on Windows:

```
python export.py
```

Inside the script, one can edit the name and icon used for the executable, among other things. All paths should automatically be correct, as long as the script is run in the root directory (the one containing `main.py`).

In the end, the bundle/package is put under the root-directory inside a folder called dist.

Inno Setup Compiler (Windows)

To bundle the package into an installer which can easily be distributed, it is recommended to use Inno Setup on Windows.

In Inno Setup, load the `setup_script.iss` , and edit any path which is not correct for your system. Make sure to update both the installer version and app version if creating a new release!

Make sure the unique "AppId" has the correct value:

https://github.com/EBSP-Indexer/EBSP-Indexer/blob/main/tutorials/dev/update_bundle.md 1/2

```
38A0E626-43E8-4E4A-8AE6-62FA6C9932D5
```

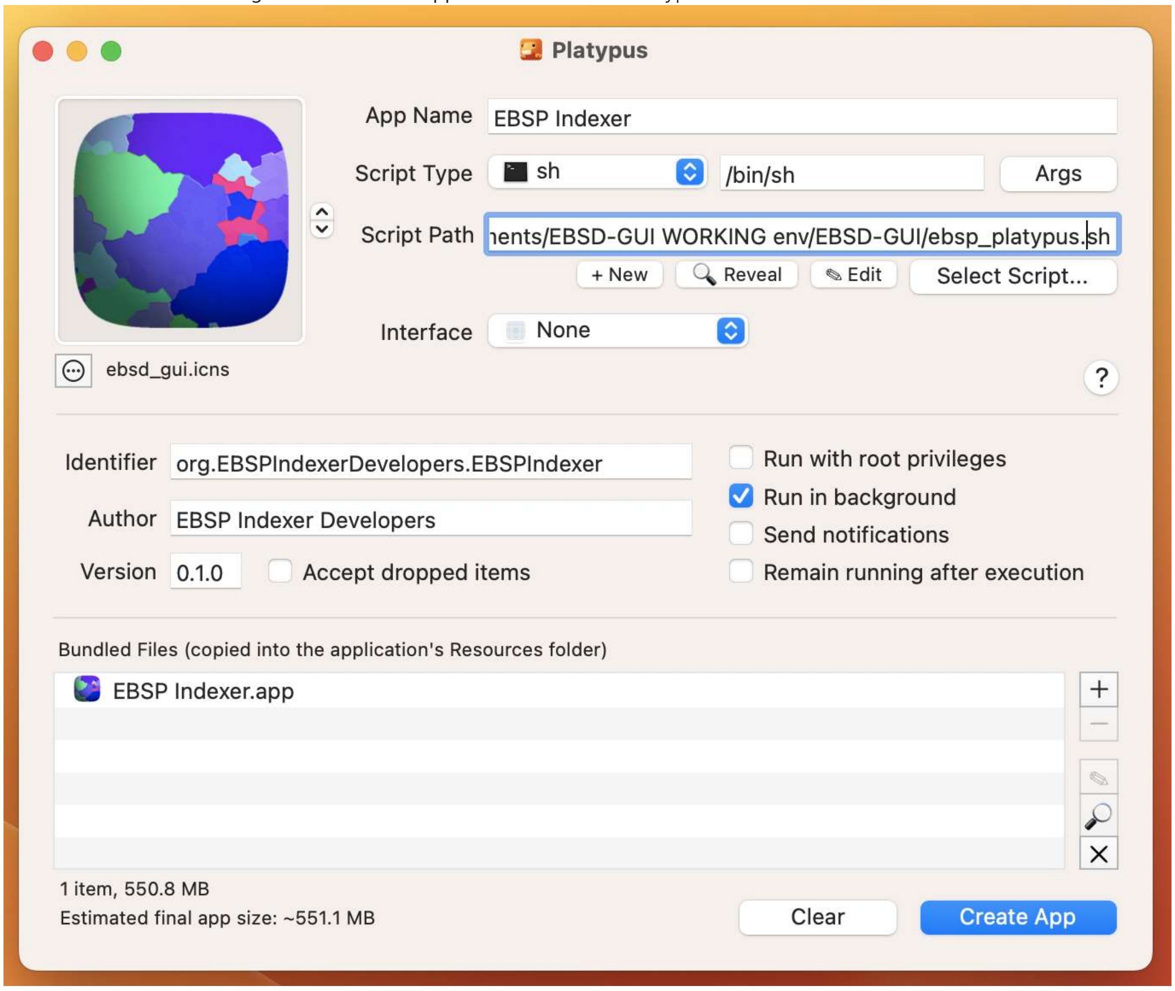
Use this so Windows recognizes the app and updates to the new version instead of installing a seperate new version.

Platypus (macOS)

There is a problem with PyInstaller bundling GUI applications on the .app format. The most viable option is a workaround, which is to use Platypus to bundle the app distribution from PyInstaller into a second app.

Platypus is a software tool which can create macOS application bundles from command line scripts or programs in a number of programming languages. It can be installed via Homebrew.

Make sure that the following is correct for the app bundle made with Platypus:


Platypus

App Name: EBSP Indexer
Script Type: sh — /bin/sh
Script Path: ents/EBSD-GUI WORKING env/EBSD-GUI/ebsp_platypus.sh
Interface: None
ebsd_gui.icns

Identifier: org.EBSPIndexerDevelopers.EBSPIndexer
Author: EBSP Indexer Developers
Version: 0.1.0
Accept dropped items: ☐
Run with root privileges: ☐
Run in background: ✓
Send notifications: ☐
Remain running after execution: ☐

Bundled Files (copied into the application's Resources folder)
EBSP Indexer.app

1 item, 550.8 MB
Estimated final app size: ~551.1 MB

- Script path pointing to `ebsp_platypus.sh` in the root directory.
- Script type is `sh` .
- Interface type is `None`
- The app bundled with PyInstaller must be added to the Bundlded Files list.
- Make sure that `"Run in background"` is checked
- The identifier used should be:

```
org.EBSPIndexerDevelopers.EBSPIndexer
```

- Author should be `EBSP Indexer Developers`
- Remember to update the version

J Developer Tutorials - Workflow

EBSP-Indexer / **EBSP-Indexer** Public

<> Code | Issues 2 | Pull requests | Discussions | Actions | Projects | Wiki | Security 10 | Insights | Se

EBSP-Indexer / tutorials / dev / workflow.md

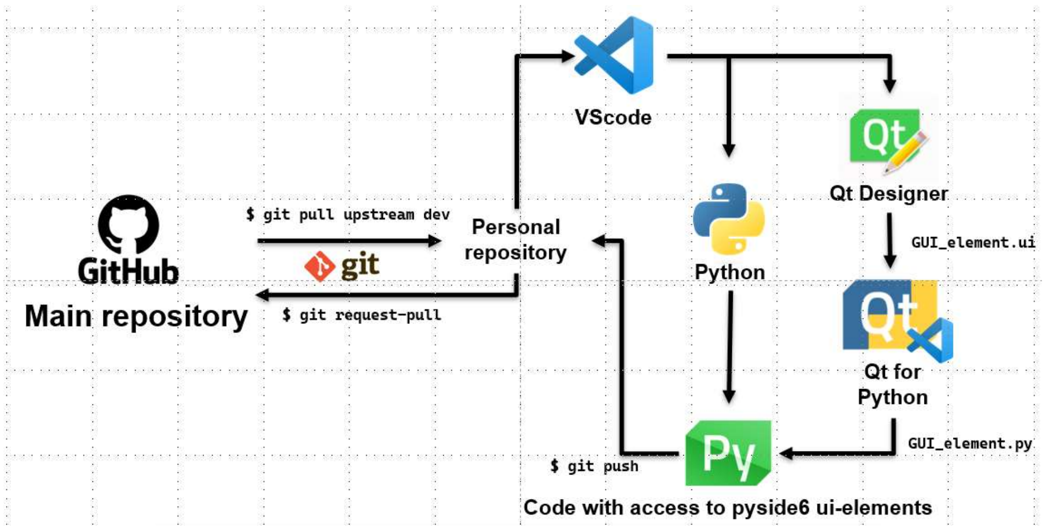
Erlendos12 3 weeks ago

49 lines (41 loc) · 2.52 KB

Preview | Code | Blame

Working with Qt Designer and python

The following describes a workflow for designing GUI elements and then add Python code to the elements.



Qt Designer

Design the gui component using Qt designer and save the file with an .ui extension. Note that when creating a new Qt form the template/ forms you are presented with decides the base class of that ui. This is important, as the future python class (containing python logic) should inherit this ui and in doing so its base class. This decides what Qt methods are available to the python class itself. There are 3 classes of importance:

- QMainWindow: A standalone window with no parent window/ widget. The main application uses this type of window.
- QDialog: A popup window that originates from another window, usually the main window.
- QWidget: Basic GUI element that can be part of a window or create custom elements

After the ui file is created, the file can be compiled to python code using:

```
pyuic4 input.ui -o output.py
```

There are VScode extensions that automatically compiles the .ui file to a .py file every time the .ui is uptated. It is highly recommended to install Qt for python extension.

Python logic

Create a python class in its own .py file that will contain the logic for the GUI element. The class should have the UI baseclass as its super. Import the Ui class that was generated in the .py file from the .ui file. This class will be the Ui class of our python logic class. After initializing the Ui class in our Python class to the variable self.ui, call self.ui.setupUi(self) to set up the designed GUI. Then another help method can be defined (e.g. setupConnections()) to connect self.ui and its elements to python functions/ logic.

Example template

```
from ui.ui_python_widget import Ui_PythonWidget()

class PythonWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.ui = Ui_PythonWidget()
        self.ui.setupUi(self)
        self.setupConnections()
        ...

    def setupConnections(self):
        *Connect GUI elements that exists in Ui_QtForm() to python functions here
        ...
```

Naming convention

- Functions/ variables containing PyQt logic and pure Qt elements uses camelCase
- Functions/ variables that contain python code and other libraries uses snake_case

K Documentation - Package utils

All packages (/)

Package **`utils`**

Sub-modules

`utils.filebrowser`

Utility for accessing a File Browser that is native to the OS

`utils.redirect`

Utility for mapping self.write to a function of choice

`utils.setting_file`

Utiltiy for saving and loading parameters as text files

`utils.threads`

A submodule containing differet utility methods and classes for threading

All packages (/) :: utils (/utils/)

Module **`utils.filebrowser`**

Utility for accessing a file explorer that is native to the OS

Classes

```
class FileBrowser (mode: Optional[int] = 0, dirpath: Optional[str] = 'C:/EBSP-Indexer',
                   filter_name: Optional[str] = 'All files (*.*)',
                   caption: Optional[str] = None)
```

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

Utility for accessing a file explorer that is native to the OS

Parameters

`mode` : `int`, optional
: The mode of the file explorer, which may be either OpenFile, OpenFiles, OpenDirectory, or SaveFile

`dirpath` : `str`, optional
: The directory that is first shown in the file explorer

`filter_name` : `str`, optional
: The filter that includes file extentions, which decides what files are shown, *or* allowed to be saved.

`caption` : `str`, optional
: The title of the File Browser window

Ancestors

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

Class variables

`var OpenDirectory`

`var OpenFile`

`var OpenFiles`

`var SaveFile`

`var staticMetaObject`

Methods

`def getFile(self) -> int`

Prompts the user with the file explorer window in the currently set mode

Returns

`int`
: 1 if one or more paths are set. Abort/ cancel returns 0

`def getPaths(self) -> list[str]`

Get the path which was last set by the `FileBrowser.getFile()` method

Returns

`List[str]`
: A list which contains all the paths that were set

`def setCaption(self, caption)`

Sets the title of the File Browser window

`def setDefaultDir(self, path: str)`

Sets the default directory if the path exsits

`def setFileFilter(self, text: str)`

Sets the filter of file extentions

`def setMode(self, browser_mode: int)`

Sets the mode, which may be either OpenFile, OpenFiles, OpenDirectory, or SaveFile

Module `utils.threads.thdout`

Utility for redirecting standard output from a thread through signals

Classes

`class ThreadedOutput`

QObject(self, parent: Optional[PySide6.QtCore.QObject] = None) -> None

QObject to be used in redirecting standard output from a thread through signals

Attributes

`outputLine` : `Signal[str]`
Emits the captured standard output as a string

`outputError` : `Signal[str]`
Emits the fully traced error as a string

Ancestors

PySide6.QtCore.QObject, Shiboken.Object

Class variables

`var staticMetaObject`

Methods

`def errorwrite(self, line: str) -> None`

Capture stderr and emit signal

`def flush(self)`

Dummy function to allow reciving of progressbars from kikuchipy

`def outputError(...)`

`def outputLine(...)`

```
def write(self, line: str) -> None
```

Capture stdout and emit signal

Module **`utils.threads.worker`**

Utiltiy for creating a Worker that performs a function of choice inside a thread

Functions

```
def sendToWorker(parent: PySide6.QtWidgets.QMainWindow | PySide6.QtCore.QObject,
                 func: Callable, *args, **kwargs)
```

Sends a function to a worker-thread with args and kwargs, and redirects standard output and error to parent.

Parameters

`parent` : `QMainWindow, QWidget`
: The parenting QWidget which is either a WorkerWidget or the main application window, contains methods for writing output

`func` : `Callable`
: The function callback to run on this worker thread, Supplied args and kwargs will be passed through to the runner

`args` : `tuple`
: Arguments to pass to the callback function

`kwargs` : `dict[str, Any]`
: Keywords to pass to the callback function

Classes

```
class Worker (parent: PySide6.QtWidgets.QMainWindow | PySide6.QtWidgets.QWidget,
              func: Callable, *args, **kwargs)
```

QRunnable(self) -> None

Parameters

`parent` : `QMainWindow, QWidget`
: The parenting QWidget which is either a WorkerWidget or the main application window, contains methods for writing output

func : `Callable`
: The function callback to run on this worker thread, Supplied args and kwargs will be passed through to the runner

args : `tuple`
: Arguments to pass to the callback function

kwargs : `dict[str, Any]`
: Keywords to pass to the callback function

Ancestors

PySide6.QtCore.QRunnable, PySide6.QtCore.QObject, Shiboken.Object

Class variables

`var staticMetaObject`

Methods

`def isError(...)`

`def isFinished(...)`

`def isStarted(...)`

`def run(self) -> None`

Initialise the runner function with passed args, kwargs, and redirects standard output and error.

Raises

`Exception`
: If an error occurs in the task of the thread

`def setupConnections(self)`

Module `utils.threads.worker_widget`

Utiltiy for creating the Jobs that are shown in the Job Manager of the main application window

Functions

```
def sendToJobManager(job_title: str, output_path: str,
                     listview: PySide6.QtWidgets.QListWidget, func: Callable,
                     allow_cleanup: bool = False, allow_logging: bool = False, *args,
                     **kwargs)
```

Method for threading a function func with arguments *args and keyword arguments* *kwargs, and adds a workerWidget to the listview.

Classes

```
class WorkerWidget (job_title: str, output_directory: str,
                    mainWindow: PySide6.QtWidgets.QMainWindow,
                    jobItem: PySide6.QtWidgets.QListWidgetItem, func: Callable,
                    allow_cleanup: bool = False, allow_logging: bool = False, *args,
                    **kwargs)
```

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

Parameters

`job_title` : `str`
The title of the job that is to be executed by the worker

`output_directory` : `str`
The location where results will appear

`mainWindow` : `AppWindow`
The main app winow which contains the job manager (should be replaced by the job manager list later)

`job_item` : `QListWidgetItem`

The item in the job manager list which contains the worker widget, is resized when the worker widget is resized.

func : `Callable`
: The function callback to run on this worker thread, Supplied args and kwargs will be passed through to the runner

allow_cleanup : `bool` , optional
: If the output_directory should be deleted if the task of the worker fails or is cancelled (default is False)

args : `tuple`
: Arguments to pass to the callback function

kwargs : `dict[str, Any]`
: Keywords to pass to the callback function

Ancestors

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

Class variables

`var counter`

`var staticMetaObject`

Methods

```
def adjustSize(self, show)
```

Resizes this Job Item that resides in the JobManager List, the resizing depends on whether to hide or show more information

```
def errorwrite(self, line: str) -> None
```

Capture stderr and display in outdisplay

```
def finalize(self, id: int, failed: Optional[bool] = False,
             cancelled: Optional[bool] = False, cleanup: Optional[bool] = False,
             logging: Optional[bool] = False)
```

Updates this Job to perform actions due to this Job's Worker being finished

Parameters

id : `int`
: The id of the Worker which was completed

`failed` : `bool` , optional
: Whether or not the Worker failed

`cancelled` : `bool` , optional
: Whether or not the Worker was cancelled

`cleanup` : `bool` , optional
: Whether or not to removes the output directory, cleanup can only remove empty directories for safety reasons

`logging` : `bool` , optional
: Whether or not to save the redirected output from the Worker as a log

`def removeMeSignal(...)`

`def sendCancelWorker(self)`

Attempts to cancel the Job which was sent to `QThreadPool.globalInstance()`

`def sendRemoveItem(self)`

Emits a signal for this JobWidget to be removed from the Job Manager

`def setupConnections(self)`

Connects class methods to UI signals

`def time_worker(self, id)`

Starts a timer that is displayed in this Job

`def updateActiveJobs(self)`

Updates the number of active jobs in the main application window

`def updateTimerDisplay(self)`

Updates the display that shows the Timer

`def write(self, line: str) -> None`

Capture stdout and display in outdisplay

`def writeoutput(self, line: str, fmt: PySide6.QtGui.QTextCharFormat = None) -> None`

Set text formatting and display line in outdisplay

L Documentation - Package scripts

All packages (/)

Package `scripts`

Sub-modules

`scripts.advanced_settings`

`scripts.color_picker`

`scripts.console`

Interactive console widget. Use to add an interactive python interpreter in a GUI application …

`scripts.create_phase`

Script for a dialog window where users can create a phase with custom structure

`scripts.dictionary_indexing`

`scripts.hough_indexing`

Script for a dialog window where users can specify parameters for Hough Indexing

`scripts.pattern_center`

`scripts.pattern_processing`

Script for a dialog window where users can specify parameters for Signal-to-Noise Improvements to be performed on a dataset

`scripts.pc_from_wd`

`scripts.pc_selection`

`scripts.pre_indexing_maps`

`scripts.refinement`

Script for a dialog window where users can specify parameters for refinement of orientations in an indexed crystal map

`scripts.region_of_interest`

`scripts.signal_loader`

`scripts.signal_navigation_widget`

`scripts.system_explorer`

Script which defines the System Explorer widget which resides in the main application window

Module **`scripts.console`**

Interactive console widget. Use to add an interactive python interpreter in a GUI application.

Original code created by deanhystad, source code available at https://python-forum.io/thread-25117.html (https://python-forum.io/thread-25117.html)

Classes

```
class Console (parent: PySide6.QtWidgets.QWidget, context: dict[str, typing.Any],
             history: int = 20, blockcount: int = 500)
```

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

A GUI version of code.InteractiveConsole.

Parameters

`parent` : `QWidget`
The widget which contains the console widget

`context` : `dict[str, Any]`
Specifies the dictionary in which code will be executed

`history` : `int` , optional
The max number of lines in the history buffer

`blockcount` : `int` , optional
The max number of lines in the output buffer

Ancestors

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

Class variables

`var staticMetaObject`

Methods

`def errorwrite(self, line: str) -> None`

Capture stderr and display in outdisplay

`def flush(self)`

Dummy function to allow reciving of progressbars from kikuchipy

`def push(self, line: str) -> None`

Execute entered command. Command may span multiple lines

`def resetbuffer(self) -> None`

Reset the input buffer.

`def setcontext(self, context: dict[str, typing.Any])`

Set context for interpreter

Parameters

context : `dict[str, Any]`
Specifies the dictionary in which code will be executed

`def setfont(self, font: PySide6.QtGui.QFont) -> None`

Set font for input and display widgets. Should be monospaced

`def setprompt(self, text: str)`

Sets the prompt of the output line

`def setscrollbarmax(self) -> None`

Sets the scrollbar of the output display to max

`def write(self, line: str) -> None`

Capture stdout and display in outdisplay

`def writeoutput(self, line: str, fmt: PySide6.QtGui.QTextCharFormat = None) -> None`

Set text formatting and display line in outdisplay `class LineEdit (parent, history: int = 100)`

QLineEdit(self, arg__1: str, parent: Optional[PySide6.QtWidgets.QWidget] = None) -> None
QLineEdit(self, parent: Optional[PySide6.QtWidgets.QWidget] = None) -> None

QLineEdit with a history buffer for recalling previous lines. Also accepts tab as input (4 spaces).

Parameters

`parent` : `QWidget`
The parent widget which contains the console widget

`history` : `int, Optional`
The max number of lines in the history buffer

Ancestors

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

Class variables

`var staticMetaObject`

Methods

`def clearhistory(self) -> None`

Clear history buffer

`def event(self, ev: PySide6.QtCore.QEvent) -> bool`

Intercept tab and arrow key presses. Insert 4 spaces when tab pressed instead of moving to next contorl. When arrow up or down are pressed select a line from the history buffer. Emit newline signal when return key is pressed.

`def newline(...)`

`def recall(self, index: int) -> None`

Select a line from the history list

`def record(self, line: str) -> None`

Add line to history buffer

`def returnkey(self) -> None`

Return key was pressed. Add line to history and emit the newline signal.

Module `scripts.create_phase`

Script for a dialog window where users can create a phase with custom structure

Classes

`class NewPhaseDialog (parent: PySide6.QtWidgets.QWidget, default_color: Optional[str] = None)`

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

Dialog window where users can create a phase with custom structure

Parameters

`parent` : `QWidget`
The parent widget

`default_color` : `str`, optional
The color which is assigned to the phase

Ancestors

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

Class variables

`var staticMetaObject`

Methods

`def addEmptyAtom(self)`

Adds an empty atom row to the table of atoms

```
def get_phase(self, name: str, space_group: int,
              structure: Optional[diffpy.structure.structure.Structure] = None,
              color: Optional[str] = '') -> orix.crystal_map.phase_list.Phase
```

Returns a phase which contains the input parameters from the UI

Parameters

`name` : `str`
: The name of the phase.

`space_group` : `int`
: The space group of the phase.

`structure` : `Structure`
: Structure object of the phase. If none is given, a default strucutre object is used in the returned phase.

`color` : `str`
: The color which is assigned to the phase.

Returns

`Phase`
: A phase containing parameters derived from the input parameters.

`def removeAtom(self)`

Removes an atom row from the table of atoms

`def setAtomButtons(self, structure_enabled: Optional[bool] = True)`

Enables/ disables the ability to remove atoms from the table

`def setupConnections(self)`

Connects class methods to UI signals

`def validatePhaseParameters(self) -> None`

Checks whether entries satisfies input requirements

If all input requirements are satisifed, the parameters are saved as a dicitonary in the class, named kwargs. If not; display message to the user about which entires are wrong.

All packages (/) :: scripts (/scripts/)

Module **`scripts.hough_indexing`**

Script for a dialog window where users can specify parameters for Hough Indexing

Functions

```
def log_hi_parameters(pattern_path: str, dir_out: str,
                      signal: kikuchipy.signals.ebsd.EBSD | kikuchipy.signals.ebsd.LazyEBSD =
                       None, xmap: orix.crystal_map.crystal_map.CrystalMap = None,
                      mp_paths: dict = None, pattern_center: numpy.ndarray = None,
                      convention: str = 'BRUKER', binning: int = 1, index_data_path=None)
```

Logs and saves additional parameters, which were used during indexing, to hi_results/indexing_parameters.txt

Assumes convention is BRUKER for pattern center if none is given

TODO: This method may be used to develop an utility for future logging, and is why it is currently not a class method.

Classes

```
class HiSetupDialog (parent: PySide6.QtWidgets.QMainWindow, pattern_path: str)
```

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

Dialog window where users can specify parameters for Hough Indexing

Parameters

`parent` : `QMainWindow`
The main application window

`pattern_path` : `str`
The path to the dataset that is to be indexed

Ancestors

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

Class variables

```
var staticMetaObject
```

Methods

```
def add_phase(self, name: str = '', space_group: int = None,
              structure: diffpy.structure.structure.Structure = None, color: str = '')
```

Loads a phase directly from the method's input parameters, which is used to load a phase specified by the dialog from `scripts.create_phase`

```
def create_phase(self)
```

Opens the dialog window from `scripts.create_phase` for specifying custom phases

```
def getBinningShapes(self, signal: kikuchipy.signals.ebsd.LazyEBSD)
                     -> dict[str, tuple[int, int]]
```

Calculates and returns a dictionary of available binning options depending on the pattern resolution in the `signal` parameter

Parameters

`signal` : `LazyEBSD`
The EBSD signal which the dictionary of binning sizes should be generated from

Returns

`dict[str, tuple[int, int]]`
Key is the binning integer as a string, and resulting binning resolution is specified as two intengers in a tuple

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

Returns the parameters specified by the UI in a dictionary structure

```
def hough_indexing(self,
                   s: kikuchipy.signals.ebsd.EBSD | kikuchipy.signals.ebsd.LazyEBSD)
```

Performs Hough Indexing of the loaded signal `s`, with parameters specified by the dialog window

Resulting crystal maps are saved to a folder named hi_results, which is placed in the same location as the dataset

Parameters

`s` : `EBSD, LazyEBSD`
    The loaded EBSD signal which contains patterns to be indexed

```
def load_master_pattern_phase(self, mp_path: Optional[str] = None)
```

Loads the phase of a mastter pattern file

Parameters

`mp_path` : `str` , optional
    The path to the master pattern which should be loaded, if none is given a file dialog appears where the user may choose the file

```
def load_parameters(self, signal: kikuchipy.signals.ebsd.LazyEBSD)
```

Reads parameters specified by the project of the dataset and the application settings

```
def remove_phase(self)
```

Removes selected rows of phases from tableWidgetPhase

```
def run_hough_indexing(self)
```

Loads the dataset, creates an output directory, and sends the `HiSetupDialog.hough_indexing()` method to the Job Manager

```
def save_ipf_map(self, xmap: orix.crystal_map.crystal_map.CrystalMap,
                 ckey_direction: Optional[Sequence] = [0, 0, 1],
                 ckey_overlay: Optional[bool] = False)
```

Save plot of inverse pole figure map and orientation colour key

Parameters

`xmap` : `CrystalMap`
    The crystal map which the orientations originates from

`ckey_direction` : `sequence`
    3D vector used to determine the orientation color key

`ckey_overlay` : `bool`
    Whether the colour orientation key is shown on top of the map or saved to seperate png, default is seperate

`def save_parameters(self)`

Writes parameters to the project of the dataset

`def save_phase_map(self, xmap)`

Save plot of phase map

Parameters

xmap : `CrystalMap`
The crystal map which the phases originates from

`def save_quality_metrics(self, xmap)`

Save plots of quality metrics

Parameters

xmap : `CrystalMap`
The crystal map which the quality metrics originates from

`def setAvailableButtons(self)`

Sets whether buttons are enabled or disabled depending on state

`def setupConnections(self)`

Connects class methods to UI signals

`def updatePhaseTable(self)`

Updates the display of the phase table

Constants

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

Module `scripts.pattern_processing`

Script for a dialog window where users can specify parameters for Signal-to-Noise Improvements to be performed on a dataset

Classes

`class PatternProcessingDialog (parent: PySide6.QtWidgets.QMainWindow, pattern_path: str)`

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

Dialog window where users can specify parameters for Signal-to-Noise Improvements to be performed on a dataset

Parameters

`parent` : `QMainWindow`
The main application window

`pattern_path` : `str`
The path to the dataset that is to be indexed

Ancestors

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

Class variables

`var staticMetaObject`

Methods

`def apply_processing(self, dataset: kikuchipy.signals.ebsd.EBSD)`

Applies processing to `dataset`

Parameters

dataset : EBSD
: The dataset that the processing is applied to

```
def average_neighbour(self, dataset: kikuchipy.signals.ebsd.EBSD,
                      show_progressbar: Optional[bool] = True)
```

Averages intensities within patterns by considering the pattern's neighbohurs.

Parameters

show_progressbar : bool

dataset : EBSD
The dataset that the processing is applied to

show_progressbar : bool
Whether progress bars should be displayed in the log

```
def close_dialog(self)
```

Method for closing the dialog window and delete the loaded preview

TODO: With the latest changes of EBSP Indexer v0.1.0, this method should not be necessary anymore, and `self.reject()` should be connected to the `self.ui.buttonBox.rejected` signal instead

```
def preview_processing(self)
```

Performs processing of the previewed pattern by considering what improvments that have been choosen

```
def remove_dynamic(self, dataset: kikuchipy.signals.ebsd.EBSD,
                   show_progressbar: Optional[bool] = True)
```

Removes the dynamic backgorund from the dataset

Parameters

dataset : EBSD
The dataset that the processing is applied to

show_progressbar : bool
Whether progress bars should be displayed in the log

```
def remove_static(self, dataset: kikuchipy.signals.ebsd.EBSD,
                  show_progressbar: Optional[bool] = True)
```

Removes the static backgorund from the dataset

Parameters

`dataset` : `EBSD`
The dataset that the processing is applied to

`show_progressbar` : `bool`
Whether progress bars should be displayed in the log

`def run_processing(self)`

Displays a warning if static background noise has not been removed, loads the dataset, and sends the `PatternProcessingDialog.apply_processing()` method to the Job Manager

`def setSavePath(self)`

Sets the path for where the processed dataset should be saved

`def setupConnections(self)`

Connects class methods to UI signals

`def setupInitialSettings(self)`

Sets the avaialblity of processing by checking what processing that has already been done on the dataset, which depends on the naming of the dataset

TODO: A more robust system to keep track of what processing has been done to a dataset

`def showImage(self, dataset: kikuchipy.signals.ebsd.LazyEBSD)`

Updates the preview of the improved pattern

Parameters

`dataset` : `LazyEBSD`
The dataset that the processing is applied to

All packages (/) :: scripts (/scripts/)

Module **`scripts.refinement`**

Script for a dialog window where users can specify parameters for refinement of orientations in an indexed crystal map

Classes

```
class RefineSetupDialog (parent: PySide6.QtWidgets.QMainWindow,
                         file_path: Optional[str] = '')
```

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

Dialog window where users can specify parameters for refinement of orientations in an indexed crystal map, using the dataset which the crystal map originated from and master pattern for the correspondig phases

Parameters

`parent` : `QMainWindow`
The main application window

`file_path` : `str`, optional
The path to a dataset which contains EBSD patterns, *or* a crystal map which contains indexed orientations

Ancestors

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

Class variables

`var staticMetaObject`

Methods

`def available_index_data(path: str) -> str`

Checks whether there exist index data associated with the crystal map that was indexed using Hough.

Parameters

`path` : `str`
: The path to the directory which contains the crystal map that may have index data attatched to it

Returns

`String`
: The path of the available index data, if no path is found retuns an empty string

`def clear_crystal_maps(self, force_clear: Optional[bool] = False)`

Clears the currently loaded crystal maps if there are multiple when switching to single mode, *or* if `force_clear = True`

Parameters

`force_clear` : `bool` , optional
: Whether or not the loaded crystal maps should be cleared regardless of UI state

`def getBinningShapes(self, signal: kikuchipy.signals.ebsd.LazyEBSD) -> dict`

Calculates and returns a dictionary of available binning options depending on the pattern resolution in the `signal` parameter

Parameters

`signal` : `LazyEBSD`
: The EBSD signal which the dictionary of binning sizes should be generated from

Returns

`dict[str, tuple[int, int]]`
: Key is the binning integer as a string, and resulting binning resolution is specified as two intengers in a tuple

`def getOptions(self) -> dict`

Returns the parameters specified by the UI in a dictionary structure

`def load_crystal_maps(self, xmap_path: Optional[str] = None, force_clear=False)`

Load one or multiple crystal maps, depending on the state of the UI

Parameters

`xmap_path` : `str` , optional
: The path to the crystal map which should be loaded, if no path is given a dialog window appears to let the user select a crystal map

`force_clear` : `bool` , optional
: Whether or not the loaded crystal maps should be cleared regardless of UI state

`def load_master_pattern(self, mp_path: Optional[str] = None)`

Load one or multiple master patterns

Parameters

`mp_path` : `str` , optional
: The path to the master pattern which should be loaded, if no path is given a dialog window appears to let the user select one or multiple master patterns

`def load_parameters(self, signal: kikuchipy.signals.ebsd.LazyEBSD)`

Reads parameters specified by the project of the dataset and the application settings

Parameters

`signal` : `LazyEBSD`
: A preview of the dataset as a lazy loaded EBSD Signal, which is used to derive the microscope and working distance that were used to obtain the patterns

`def promptOverridePhase(self, message) -> bool`

Prompts the user with a warning displaying what part of the strcture of a phase in a crystal map that contains conflicts with the master pattern used to refine orientations

Parameters

`message` : `str`
: The message that is displayed in the prompt

Returns

`Bool`
: Whether or not the phase strcutre should be overriden, depends on the user's input in the prompt

```
def refine_orientations(self,
                        s: kikuchipy.signals.ebsd.EBSD | kikuchipy.signals.ebsd.LazyEB
                        SD, xmaps: dict[str, orix.crystal_map.crystal_map.CrystalMap],
                        master_patterns: dict[str, kikuchipy.signals.ebsd_master_patte
                        rn.LazyEBSDMasterPattern], options: dict)
```

Performs refinemnet of orientations on all crystal maps in `xmaps` , by using the experimental signal `s` and the master patterns in `master_patterns`

Resulting crystal maps are saved to the specified output folder

Parameters

s : `EBSD, LazyEBSD`
: The loaded EBSD signal which contains patterns that `xmaps` originated from

xmaps : `dict[str, CrystalMap]`
: Ccontains the crystal maps which should be refined

master_patterns : `dict[str, LazyEBSDMasterPattern]`
: Contains master patterns that is used to refine orientations

options : `dict`
: Contains parameters specified by the UI

```
def remove_crystal_maps(self)
```

Removes the selected rows from the crystal map table

```
def remove_master_pattern(self)
```

Removes selected rows from the master pattern table

```
def run_refinement(self)
```

Loads the dataset, if index data is selected generate crystal maps from the data, checks whether phase structures are compatible and if not prompt the user with an override dialog,

When there are no phase conflicts between phases and master patterns, sends the `RefineSetupDialog.refine_orientations()` method to the Job Manager

```
def save_ipf_map(self, xmap: orix.crystal_map.crystal_map.CrystalMap,
                 ckey_direction: Optional[Sequence] = [0, 0, 1],
                 ckey_overlay: Optional[bool] = False)
```

Save plot of inverse pole figure map and orientation colour key

Parameters

`xmap` : `CrystalMap`
: The crystal map which the orientations originates from

`ckey_direction` : `sequence`
: 3D vector used to determine the orientation color key

`ckey_overlay` : `bool`
: Whether the colour orientation key is shown on top of the map or saved to seperate png, default is seperate

`def save_ncc_map(self, xmap: orix.crystal_map.crystal_map.CrystalMap)`

Saves the normalized cross-correlation map of `xmap` as an image

NCC scores are accesed differently depending on whether the xmap is the result of multiple merged crystal maps

Parameters

`xmap` : `CrystalMap`
: The crystal map which contains normalized cross-correlation scores

`def save_phase_map(self, xmap)`

Save plot of phase map

Parameters

`xmap` : `CrystalMap`
: The crystal map which the phases originates from

`def save_quality_metrics(self, xmap)`

Save plots of quality metrics

Parameters

`xmap` : `CrystalMap`
: The crystal map which the quality metrics originates from

`def setAvailableButtons(self)`

Sets whether buttons are enabled or disabled depending on state `def set_output_directory(self, output_path='')`

Sets the output directory where refined results are stored

`def setupConnections(self)`

Connects class methods to UI signals

`def updateCrystalMapTable(self)`

Updates the display of the crystal map table

Constants

FILE_NAME_COL = 0 PHASE_NAME_COL = 1 ORIENTATIONS_COL = 2

`def updateMasterPatternTable(self)`

Updates the display of the master pattern table

Constants

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

All packages (/) :: scripts (/scripts/)

Module **`scripts.system_explorer`**

Script which defines the System Explorer widget which resides in the main application window

Functions

`def openTxtFile(txt_path: str)`

Opens a file in the OS's default software for inspecting text

Parameters

`txt_path` : `str`
The path to the file that is to be opened

`def revealInExplorer(revealed_path: str)`

Reveals a path in the File Explorer that is native to the OS

Parameters

`revealed_path` : `str`
The path to the folder or file that is to be revealed

Classes

`class SystemExplorerWidget (parent: Optional[PySide6.QtWidgets.QWidget] = Ellipsis)`

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

Widget for a System Explorer that may show a hierarchy of files which are stored locally

Parameters

`parent` : `QWidget`
A QtWidget which is regareded as the parent of the System Explorer

Attributes

`pathChanged` : `Signal[str]`

Emits the new selected path whenever the path is changed

`requestSignalNavigation` : `Signal[str]`
Emits the path of the file that should be shown in the Signal Navigation widget

`requestImageViewer` : `Signal[str]`
Emits the path of the file that should be shown in the Image Viewer widget

Ancestors

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

Class variables

`var IMAGE_EXTENSIONS`

`var KP_EXTENSIONS`

`var SYSTEM_VIEW_FILTER`

`var staticMetaObject`

Methods

`def contextMenu(self)`

Displays a menu of options which depends on the type of class that the selected_path implies

`def deleteSelected(self, deletion_path)`

Deletes a file *or* a folder

Parameters

`deletion_path` : `str`
The path to the file *or* folder that is being deleted

`def displayDeleteWarning(self, deletion_path)`

Displays a confirmation prompt to the user about deleting a file *or* a folder

Parameters

`deletion_path` : `str`
The path to the file *or* folder that is to be deleted

```
def doubleClickEvent(self)
```

Emits one or more signals depending on the type of file that was double clicked

```
def onSystemModelChanged(self, new_selected: str, old_selected: str)
```

Emits the `self.pathChanged` signal with a new selected path

Parameters

`new_selected` : `str`
The new path selected

`old_selected` : `str`
The previously selected path which is replaced

```
def pathChanged(...)
```

```
def requestImageViewer(...)
```

```
def requestSignalNavigation(...)
```

```
def setSystemViewer(self, working_dir: str,
                    filters: Optional[Sequence[str]] = ('*.h5', '*.dat', '*.ang',
                    '*.jpg', '*.png', '*.txt'))
```

Sets the folder that is displayed in the System Viewer with a file filter applied

Parameters

`working_dir` : `str`
The path of the directory that is shown

`filters` : `Sequence[str]` , optional
A sequence of strings that specify file extentions to show, if none is specified use default values

```
def setupConnections(self)
```

Connects class methods to UI signals

NTNU

Norwegian University of
Science and Technology