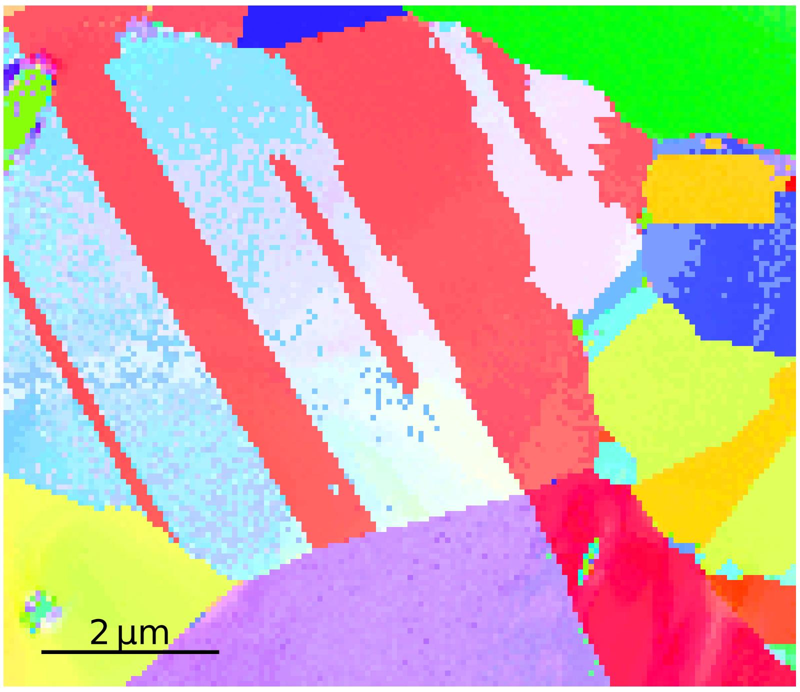
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

Viljar Johan Femoen

Mapping and navigating crystal orientations in the transmission electron microscope

Master's thesis in Nanotechnology
Supervisor: Antonius T. J. van Helvoort
June 2024



NTNU
Norwegian University of Science and Technology

Viljar Johan Femoen

Mapping and navigating crystal orientations in the transmission electron microscope

Master's thesis in Nanotechnology
Supervisor: Antonius T. J. van Helvoort
June 2024

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

ABSTRACT

The transmission electron microscope (TEM) is often the instrument of choice to study crystalline materials locally with nm resolution. As electron diffraction patterns are unique for each unique orientation of a given crystal, they can be used to determine the orientation of a crystalline sample. A user might want to view their sample from a specific crystallographic direction, e.g. for lattice imaging. However, this orientation might not be present in the sample as-is, and needs to be aligned with the electron beam using the goniometer of a tiltable sample holder.

Here, a software tool was developed to predict tilt angles which align a chosen point or region on a crystalline sample to a target zone axis, given an initial orientation and TEM geometry. The tool, `tiltlib`, is based on the open-source Python suite Pyxem. To accurately predict tilt angles, the position of the tilt axes needs to be determined. A robust method for this is proposed, based on a tilt series of scanning precession electron diffraction (SPED) datasets.

A SPED tilt series with four 5° steps of polycrystalline silver (Ag) was used to determine orientations of the sample, and subsequently the tilt axis position, in a JEOL JEM 2100F TEM, using a new method for tilt axis identification. Additionally, a polycrystalline lithium manganese nickel oxide ($LiMn_{1.5}Ni_{0.5}O_4$) sample was used to experimentally verify predicted tilt angles. The tilt axis position was accurate to 1° to 2° and the tilt angles to reach the target zone were within 1° to 5°. This allowed the operator to observe the Laue circle, which was used for final alignment. Deviations are likely caused by the 1° template matching precision, as well as the sample being slightly misoriented when re-inserted into the sample holder. To summarize, `tiltlib` is shown to be a functional and useful tool for zone axis alignment, and will save time compared with manual search.

The orientations were mapped with template matching using Pyxem. This is computationally expensive, and especially time-consuming for low-symmetry crystals. An alternative approach is proposed and tested based on a new algorithm for discarding templates before full correlation. By interpolating correlation scores from a rough template bank, a finer bank with promising templates is run again.

As crystal symmetry can affect the analysis, the developed algorithm was tested against simulated datasets from space groups $Fm\bar{3}m$, $P6_3/mmc$, and $P2/c$, as well as a SPED dataset of $Fm\bar{3}m$ Ag. The results indicate the new algorithm indeed reduces runtime, and returns the same or similar results as without any pre-selection. However, the currently implemented pre-selection algorithm in Pyxem, based on azimuthal integration and radial correlation, clearly outperforms the proposed algorithm, both in terms of runtime and similarity with the un-filtered results.

SAMMENDRAG

Transmisjonelektronmikroskop (TEM) er ofte det foretrukne instrumentet for å undersøke krystallinske materialer med nm oppløsning. Siden elektrondiffraksjonsmønstre er unike for alle orienteringer av en gitt krystall, kan de brukes til å bestemme orienteringen til en krystallinsk prøve. En mikroskopist vil kanskje undersøke prøven fra en gitt orientering, f.eks. for gitteravbildning. Imidlertid kan denne orienteringen være utilgjengelig prøven som den er, og prøven må justeres med elektronstrålen ved å bruke goniometeret til en vippbar prøveholder.

Her ble det utviklet et verktøy for å forutsi vippevinkler som justerer et valgt punkt eller område på en krystallinsk prøve til en målsoneakse, gitt en initiell orientering og TEM geometri. Verktøyet, `tiltlib`, er basert på åpen kildekode Pythonpakken Pyxem. For å forutsi nøyaktige vippevinkler må posisjonen til vippeaksene bestemmes. En robust metode for dette er foreslått, basert på en vipperekke av sveipe-presesjonselektrondiffraksjon (SPED)-datasett.

En SPED-vipperekke med fire 5°-trinn av polykrystallinsk sølv (Ag) ble brukt for å bestemme orienteringen av prøven, og deretter vippeakseposisjonen i en JEOL JEM 2100F TEM, ved hjelp av en ny metode for identifikasjon av vippeakse. I tillegg ble litsium mangan nikkel oksid ($LiMn_{1.5}Ni_{0.5}O_4$)-prøve brukt for å verifisere predikerte vippevinkler, for tre valgte områder. Vippeaksens posisjon var nøyaktig til 1° to 2° og vippevinklene for å nå målsoneaksen var innenfor 1° to 5°. Dette tillot operatøren å observere Laue-sirkelen, som ble brukt til endelig justering. Avvikene er sannsynligvis forårsaket av 1° maltilpasnings-presisjonen, samt at prøven er litt feilorientert når den settes inn i prøveholderen igjen. For å oppsummere; `tiltlib` er vist til å være et funksjonelt og nyttig verktøy for soneaksejustering, og vil spare tid sammenlignet med manuelt søk.

Orienteringene ble kartlagt med maltilpasning ved bruk av Pyxem. Dette er beregningsmessig tungt, og spesielt tidkrevende for lavsymmetrikrystaller. En alternativ tilnærming er foreslått og testet basert på en ny algoritme for å forkaste maler før full korrelasjon. Ved å interpolere korrelasjonsverdier fra en grov malbank kjøres en finere bank med lovende maler på nytt.

Siden symmetri kan påvirke analysen, ble den utviklede algoritmen testet mot simulerte datasett fra romgruppene $Fm\bar{3}m$, $P6_3/mmc$ og $P2/c$, samt en SPED datasett av $Fm\bar{3}m$ sølv. Resultatene antyder at den nye algoritmen faktisk reduserer kjøretiden, og returnerer samme eller lignende resultater som uten forhåndsforkastning av maler. Imidlertid blir den nye foreslåtte algoritmen overgått av den allerede implementerte forhåndsvalgalgoritmen i Pyxem, basert på kun radiell korrelasjon, både når det gjelder kjøretid og likhet med de ikke-filtrerte resultatene.

PREFACE

This thesis is the culmination of two semesters of work, first for my Project thesis, and then expanded and improved for my Master's thesis. The work was performed under supervision of Professor Antonius (Ton) T.J. van Helvoort, for the TEM Group at the Department of Physics.

The culmination of this work is the Python package `tiltlib`, available on PyPI and `https://www.github.com/viljarjf/tiltlib`. It has garnered some attention on GitHub, earning a 'star' from microscopists from around the world: Germany, USA, Australia, and the UK.

The project was composed of individual independent work, without the use of any AI tools, but it would not have been possible alone. A big thank you to Ton, for our weekly discussions and your valuable ideas and feedback. Your impressively extensive and thorough feedback will be missed. I would like to thank Dr. Tina Bergh, for preparing the sample and answering all my questions, as well as her and Dr. Emil Christiansen for collecting the data for me. Kaja Eggen Aune deserves a big thank for lending me her data, and for testing my code (even before the bugs were fixed). Kaja's data was aquired by Emil, Dr. Ruben Bjørge, Ton, and Dr. Inger-Emma Nylund, from a sample prepared by Inger-Emma. Thank you to Dr. Håkon Wiik Ånes for our orientation-related discussions, and to Dr. Carter Francis the rest of the Pyxem team for the feedback on my code and suggestions. Prof. Daniel Ugarte deserves a thank for his input on my work, and for inspiring it.

Finally, thank you to Hanna and all my fellow students, for putting up with me incessantly pointing at all the interesting findings on my screen, and putting up with me always winning in Mario Kart.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

LIST OF ABBREVIATIONS

ACOM	Automated crystal orientation mapping
API	Application-programmer interface
BF	Bright-field
CBED	Convergent-beam electron diffraction
CoM	Center of mass
CTEM	Conventional TEM
DED	Direct electron detector
DoG	Difference of Gaussians
EBSD	Electron back scattered diffraction
FCC	Face-centered cubic
FIB	Focused ion beam
FOLZ	First-order Laue zone
HAADF	High-angle annular dark-field
HOLZ	Higher-order Laue zone
HRTEM	High resolution TEM
IPF	Inverse pole figure
LMNO/$\mathbf{LiMn_{1.5}Ni_{0.5}O_4}$	Lithium manganese nickel oxide
NBD	Nano-beam diffraction
NCC	(Zero)normalized cross-correlation
PED	Precession electron diffraction
PF	Pole figure
RED	Rotation electron diffraction

SAED	Selected area electron diffraction
SED	Scanning electron diffraction
SEM	Scanning electron microscope
SP	Stereographic projection
SPED	Scanning precession electron diffraction
STEM	Scanning transmission electron microscopy
TEM	Transmission electron microscope
TM	Template matching
VBF	Virtual bright-field
ZOLZ	Zeroth-order Laue zone

CHAPTER
ONE

INTRODUCTION

1.1 Background and Motivation

Understanding the micro-structure of crystalline materials is in many cases crucial for understanding their properties [1]. This includes e.g. grain size, orientation distribution (i.e. texture), strain, crystal phases and grain boundaries. The transmission electron microscope (TEM) is a powerful tool to study structure on the micro- and nano-scale, owing to its unprecedented spatial resolution [2]. Crystalline materials are especially suited for the TEM, as the diffraction mode allows the user to capture electron diffraction patterns arising from µm down to nm sized areas [2, p. 78 – 79]. Diffraction theory is a mature and quite well understood field [3, ch. 1-2, 4], facilitating the study of new and novel materials by means of a TEM.

Determining the crystal orientation from a diffraction pattern from a known structure is a straightforward task in principle, but often cumbersome and time-consuming [2, ch. 7, 5]. The general methodology is to recognize the characteristic Kikuchi band pattern [6–10], use the Laue circle to manually align to a recognizable low-index zone axis [11–15], or template matching (TM) [16–19]. Recent developments in TEM detector technologies[20], where reduced noise, increased dynamical range, and especially decreased capture and readout time, combined with scanning electron diffraction (SED) techniques such as 4D-STEM [21, 22], rotation electron diffraction (RED) [23] and SerialRED [24], has led to such large diffraction data quantities as to necessitate automation and streamlining the data processing.

Tools for automated crystal orientation mapping (ACOM) are available, e.g. ACT [25], NanoMEGAS' ASTAR [26], and the open-source Py4DSTEM[22] and Pyxem [27]. The latter three are based on TM, where simulated diffraction patterns are compared with diffraction patterns from the TEM [16, 28]. Pyxem being open-source is advantageous, as it can more easily exploit the rapid technological and architectural developments in computing, e.g. employing the GPU [29]. Community-driven further developments of open-source code is important to improve orientation analysis further, for example regarding accuracy and speed, beside dealing with ever increasing data sets.

When the orientation of a sample is determined at every point, the results can be further analyzed in programs such as Orix [30] or MTEX [31]. As the orientation mapping describes the sample-crystal relationship, only the microscope-gonio-sample relationship remains to fully describe the geometry. A complete description of the sample in the microscope reference frame would allow for navigation in crystal space by means of goniometer tilting. This concept exists in tools such as ALPHA-BETA [32], KSpaceNavigator [33], and τompas [34], but none of them integrate well with existing ACOM software, making them more reliant on low-index zones and more experienced operators. They all tackle the orientation determination problem themselves, by means of Kikuchi bands or Laue circles, which can be cumbersome for low-symmetry crystals. Furthermore, they all rely on single selected area electron diffraction (SAED) or nano-beam diffraction (NBD) patterns, which makes them unable to exploit SED and the diverse orientations present in polycrystalline or dispersed nanoparticle samples. Nanocartography [35] seems to support scanning transmission electron microscopy (STEM) data, which would mitigate this shortcoming, but is closed-source. Additionally, only some of the listed softwares include methods for accurately determining the positions of the tilt axes in the microscope, which is crucial for accurate alignment and a complete description of the geometry [36]. By implementing crystallographic navigation in the same framework as existing open-source ACOM software, here Pyxem, and packaging this with robust tilt axis identification methodology, the drawbacks of the existing software can be mitigated. An open-source environment additionally enables the use of existing libraries for automatic TEM control, such as JEOL's PyJEM [37] and PyED[38].

TM-based ACOM as a basis for zone axis alignment in an open-loop control system suffers from a few drawbacks, with angular orientation precision of 1.1° [39] (as opposed to 0.1° to 0.3° for Kikuchi-based orientation mapping [6, 39]), and unreliable stage control [40]. However, the prospect of a completely automatic pipeline for zone axis alignment makes TM an avenue worth investigating and improving. A beneficial approach to increase accuracy of TM-based orientation mapping, is to use precession electron diffraction (PED) [41, 42], whereby the precessed beam yields more kinematic-like diffraction patterns for easier simulations [41, 43]. Furthermore, one could perform more computationally expensive orientation mapping, e.g. using residual optimization on precessed diffraction pattern intensities [44]. This approach increases the angular orientation precision down to an impressive 0.03°, at the cost of many orders of magnitude longer computation time than conventional TM. Another approach would be to decrease the runtime, and rely on the precision of around 1° to be enough to observe the Laue circle. By employing the microscope operator for final alignment on the Laue circle, a closed-loop control system is established.

Even without using intensity in the orientation analysis, the default TM is too slow to be done on the fly. For example, TM on a 2 Gb scanning precession electron diffraction (SPED) data stack for a high symmetry phase like $m\bar{3}m$ and an angular resolution of 1 degree can take in the order of 30 minutes on a laptop computer. This hampers in-line implementation and use of a navigation tool based on TM, and limits the angular resolution of the mapped orientations, certainly for low symmetry phases which gives large template banks (e.g. for $2/m$ and 1° resolution, there are 14,593 entries in the template bank, compared to 1081 for $m\bar{3}m$). One way would be GPU based TM as suggested by Cautaerts et al. [29], which speeds up the calculation by means of parallelization rather than algorithmic considerations. Within Pyxem, described in the same paper, there is an algorithmically faster approach that pre-filter the bank based on matching azimuthally integrated templates and patterns. Alternative pre-selection approaches prior to full matching, with a small step size in the bank, could be considered. A speedup approach, without loosing angular resolution, would open new possibilities in material study and use of TM-based orientation analysis.

1.2 Aim of Study

This study aims to ease and improve crystallographic navigation and orientation analysis in the TEM, based on SPED, in an open-source environment. The aim will be addressed in two parts:

TEM navigation tool Creating and testing a navigation tool for aligning crystals to zone axes in a TEM.

TM algorithmic improvement A novel algorithm for template selection in TM, that reduces runtime without reducing accuracy.

Both of these will build upon the Pyxem software suite, to ensure transparency and an open platform, easily expanded and improved in the future.

As part of the first aim, a robust method for determining the position of the tilt axes in the TEM will be developed. It will be based on a tilt series of SPED data, where TM is used to produce orientation maps, and require minimal intervention. The navigation tool aims to take an orientation map as input, allowing the user to virtually align the different grains present in the scan as desired, rather than being restricted to single-crystal specimens or regions. Preliminary work on this aim started in August 2023, culminating in "Relating holder axes and template matching for grain orientation analysis"[45].

To reduce runtime of TM without loss of precision, a multi-step approach is suggested, implemented and tested, based on correlation score interpolation. The aim is to implement a three-step workflow where first, a template bank with low angular resolution (e.g. 3°) is used for TM. The correlation scores are then interpolated in orientation space, serving as an estimate of the correlation scores for orientations not part of the original template bank. Finally, the orientations with high estimated correlation scores are used for a second TM run, with each navigation position having a unique template bank based on the low angular resolution correlation scores. This algorithm aims to lower the runtime of TM, while retaining the output one would get from a template bank with high angular resolution (e.g. 0.2°).

Note, that building and integrating the proposed developments on a dynamic community-driven open-source platform will require flexibility and adaptations. Contribution and fixes are required outside the specific applications (e.g. `tiltlib`). These will be listed separately in Appendix C.

1.3 Structure of the Thesis

The thesis is divided into seven chapters. In Chapter 2, relevant theory for crystallography, diffraction, the TEM, orientations, and orientation mapping is presented. Chapter 3 contains the methodology of the thesis, namely data collection and processing, as well as describing the methodology behind the code developed for this project. In Chapter 4, the results from orientation mapping, tilt axis identification, and zone axis alignment are presented, along with the results for the new TM algorithm. The results are subsequently discussed in Chapter 5. A conclusion is drawn, presented in Chapter 6, before suggestions for future work is presented in Chapter 7. The appendices give additional information: Appendix A contain listings of code developed for use in this project, apart from the navigation tool. Additional results are presented in Appendix B. Appendix C list the contributions to various open-source libraries made during this project. Finally, to demonstrate that the results of the work are presented to the community, an accepted abstract to the European Microscopy Congress is included Appendix D.

CHAPTER

TWO

THEORY

This chapter contains relevant theory for the study. First, crystallography is presented. This section is mainly based on chapters 1 to 6 in *The Basics of Crystallography and Diffraction* by Hammond [1]. The second section introduces diffraction, based on chapter 7 in Hammond, and chapter 2 in *Introduction to Solid State Physics* by Kittel [3]. Next, a brief overview of transmission electron microscopy is presented, based on chapters 2 and 12 from *Transmission Electron Microscopy and Diffractometry of Materials* by Fultz and Howe [2], chapter 3 in *Introduction to Conventional Transmission Electron Microscopy* by De Graef [46], and chapter 1 in *Texture Analysis in Materials Science: Mathematical Methods* by Bunge [47]. The fourth section outlines reference frames, and transferring between them using orientation relations. The final section gives an introduction to orientation mapping, drawing from Bunge, chapter 11 in *Physical Metallurgy* by Rollett and Barmak [48], as well as documentation from the Pyxem suite [27]. This chapter borrows heavily from the preliminary work on this project, "Relating holder axes and template matching for grain orientation analysis"[45].

2.1 Crystallography

A crystal is defined as a basis of one or more atoms, convolved with a lattice. Three-dimensional lattices are characterized by three translation vectors $\mathbf{a}_i$, $i \in \{1, 2, 3\}$, and are defined as the set of all points $\mathbf{r}$ such that $\mathbf{r}_{uvw} = u\mathbf{a}_1 + v\mathbf{a}_2 + w\mathbf{a}_3$ for integers u, v, w. The translation vectors are often given their own names: $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ for $\mathbf{a}_{1,2,3}$, respectively.

By applying a Fourier transform to an (infinite) crystal, we can apply the convolution theorem to replace the convolution of the basis and the lattice with a multiplication of the Fourier transformed basis and the reciprocal lattice. The reciprocal lattice is also a lattice, but its basis is different. One can transform a crystallographic basis $\mathbf{a}_i$ to a reciprocal basis $\mathbf{b}_j$ using

$$\mathbf{b}_j = \frac{2\pi}{V_c}\mathbf{a}_{j+1} \times \mathbf{a}_{j+2}, \tag{2.1}$$

wrapping around the indices at $j > 3$ such that e.g. $\mathbf{a}_4 = \mathbf{a}_1$. $V_c = \mathbf{a}_1 \cdot \mathbf{a}_2 \times \mathbf{a}_3$ is the unit cell volume.

2.1.1 Crystal Systems and Lattices

By choosing different translation vectors, one can create different crystal systems. These are characterized by the lengths a, b and c of the translation vectors, and the angles α, β and γ between them. Different combinations of lengths and angles are possible, e.g. $a = b = c$ and $\alpha = \beta = \gamma = 90°$ being the cubic crystal system. There are 7 different crystal systems in total, listed in Table 2.1.1.

Combining a crystal system with one of the four possible centerings yields a lattice. The centerings are: primitive (P), where there are only lattice sites at the vertices; body-centered (I), with an extra lattice site in the center of the unit cell; face-centered (F), with an extra lattice site in the center of each cell face; and base-centered (A, B, or C), with extra lattice sites in the center of only two opposite faces. An overview of all the centerings can be found in Figure 2.1.1(a). Some combinations of crystal systems and centerings have their own names, e.g. cubic F is often called face-centered cubic (FCC).

All combinations of centerings and crystal systems yield only 14 unique lattices, as many are duplicates. A complete list of the 14 fundamentally different lattices, called Bravais lattices, can be found in Table 2.1.1. An example of a duplicate entry would be cubic C, as it is equivalent to tetragonal P with a change of basis. As shown in Figure 2.1.1(b), a cubic C with lattice parameter $a = 1$ is a tetragonal P with $c = 1$, $a = \frac{\sqrt{2}}{2}$. Another example is cubic F, which has a rhombohedral primitive unit cell as shown in Figure 2.1.1(c). The choice of equivalent unit cells is arbitrary, and one can choose the representation that makes further work easier. For cubic F, it is most common to choose the larger cubic, non-primitive unit cell.

2.1.2 Point and Space Groups

Point groups are groups of the set of symmetry elements leaving the structure unchanged. The trivial point group, the monad, has no symmetry, and is denoted with 1 using Hermann-Mauguin notation. Using this notation, the point group with two-fold rotation, i.e. a diad, is denoted 2. A single mirror plane is the point group m. With two perpendicular mirror planes, one would arrive at the point group mm, one m for each axis and no symmetry elements for the final axis. This is not a complete description of the symmetry, since two perpendicular mirrors imposes a diad symmetry in the final perpendicular direction. As such, two mirror planes, and the resulting diad, forms mm2. If multiple symmetry elements share an axis, they are written as a fraction, e.g. 6/m for a hexad and a mirror plane. The final point group symmetry element used in Hermann-Mauguin notation for point groups is inversion axes, with a bar over a number. This entails a rotation followed by an inversion around a center. Only $\bar{3}$ is unique, the rest can be represented with mirrors, inversions, rotations, or a combination of these.

Further combinations of rotations and mirror planes forms an infinite amount of point groups, as can be easily recognized from the infinite set of n-fold rotations. For crystallography, however, only a finite set of point groups exists, as only the symmetry elements of 1, $\bar{1}$, 2, 3, $\bar{3}$, 4, $\bar{4}$, 6, $\bar{6}$ and m are compatible with the translational symmetry of a lattice. This set of point groups is called the crystallographic point groups, and contains 32 three-dimensional point groups. Further use of the term "point group" will refer to these 32 crystallographic point groups,

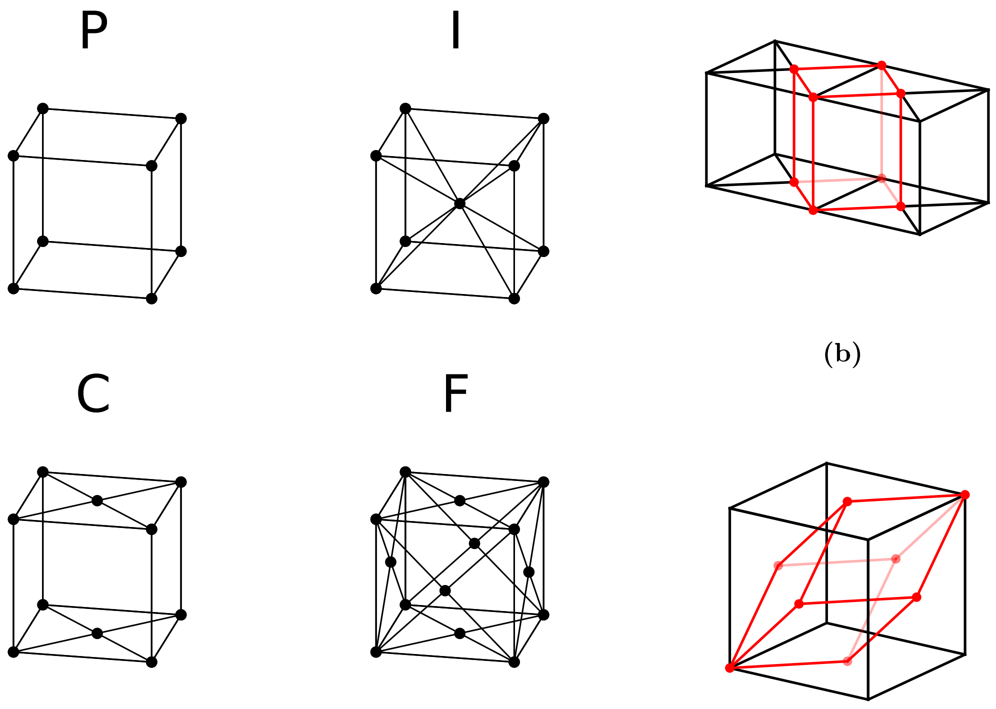


Figure 2.1.1: Lattice centerings and equivalence. (a) Different centerings displayed on a cubic crystal system. (b) Two unit cells of cubic C, with the contained tetragonal P unit cell. (c) The rhombohedral primitive unit cell of cubic F.

unless specified otherwise. These are listed in Table 2.1.1.

The three symbols in Hermann-Mauguin notation do not always correspond to the three crystal axes. For the cubic crystals, for example, this would not be useful, as the three directions are equivalent by the lattice symmetry. Therefore, for cubic space groups, the three symbols describe the symmetries along $\langle 1\,0\,0\rangle$, $\langle 1\,1\,1\rangle$ and $\langle 1\,1\,0\rangle$, respectively. This notation is described in Section 2.1.3. Furthermore, the notation has both a short and long version for each point group. Often, they are the same, but for e.g. m$\bar{3}$m, the full name is $\frac{4}{m}\bar{3}\frac{2}{m}$. As such, m$\bar{3}$m has a four-fold rotational symmetry and mirror symmetry for the $\langle 1\,0\,0\rangle$-direction, three-fold rotoinversion around $\langle 1\,1\,1\rangle$, and both a diad symmetry and mirror symmetry along $\langle 1\,1\,0\rangle$.

One can divide the point groups into the 11 centrosymmetric Laue groups, and the remaining 21 non-centrosymmetric point groups. Certain physical properties can only be exhibited by non-centrosymmetric crystals, e.g. piezoelectricity is only possible in the non-centrosymmetric point groups (except for 432). For kinematic electron diffraction simulations, on the other hand, all point groups are reduced to their corresponding Laue group, as scattering intensity imposes an inversion center.

Combining a point group with a compatible Bravais lattice yields a space group. As with centerings combined with a crystal system, not all combinations are unique or possible. In total, there are 230 space groups. These are denoted with a centering, followed by a point group. The simplest one is $P1$, a primitive tetragonal lattice

Table 2.1.1: A comprehensive list of 3D crystallographic point groups, and their corresponding crystal systems, adapted from [1, page 91].

Crystal system	Centering	Unit cell	Laue groups	Remaining point groups
Triclinic	P	$a \neq b \neq c$ $\alpha \neq \beta \neq \gamma = 90°$	$\bar{1}$	1
Monoclinic	PC	$a \neq b \neq c$ $\alpha = \gamma = 90°$, $\beta > 90°$	2/m	m, 2
Orthorhombic	PICF	$a \neq b \neq c$ $\alpha = \beta = \gamma = 90°$	mmm	mm2, 222
Tetragonal	PI	$a = b \neq c$ $\alpha = \beta = \gamma = 90°$	4/m, 4/mmm	4, $\bar{4}$, $\bar{4}$2m, 422, 4mm
Trigonal	PR	$a = b = c$ $\alpha = \beta = \gamma \neq 90°$	$\bar{3}$, $\bar{3}$m	3, 3m, 32
Hexagonal	P	$a = b \neq c$ $\alpha = \beta = 90°$, $\gamma = 120°$	6/m, 6/mmm	6, $\bar{6}$, 622, 6mm, $\bar{6}$m2
Cubic	PIF	$a = b = c$ $\alpha = \beta = \gamma = 90°$	m$\bar{3}$, m$\bar{3}$m	23, 432, $\bar{4}$4m

with no additional symmetry, and a more symmetric one is $Fm\bar{3}m$.

When extending a point group with infinite repetition, new symmetry elements arise. These are glide planes and screw axes, which both entail a previously described symmetry plus a fractional translation.

A glide plane is a mirror plane, followed by a translation parallel to the mirror plane. The translation must be fractional, as a complete unit cell translation in any direction is symmetric by the definition of a lattice. The basic glides along the crystal axes are denoted a, b, or c in Herman-Mauguin notation, for a half glide along each respective axis. An e is used if two axes are possible and equivalent, e.g. for tetragonal P, a and b are equivalent. A half glide along a face diagonal is denoted n, and a quarter glide along a body diagonal is denoted d.

Screw axes are a rotation around an axis, followed by a translation along that axis. These are denoted n_m, for an n-fold rotation. m describes the total lattice translation after n screw operations, e.g. 4_1 describes a 90° rotation followed by a $\frac{1}{4}$ translation, as four $\frac{1}{4}$ translation gives a total translation of 1. 6_3 is a 60° rotation followed by a $\frac{1}{2}$ translation, and so on.

2.1.3 Crystal Planes and Directions

To label directions and planes in a crystal, one can employ Miller indices. Labeling of a plane is performed by finding the intercept of the plane with each crystal axis, taking their inverses, and multiplying by their largest common factor. As an example, the plane in Figure 2.1.2 intercepts the $\mathbf{a}_1$-axis at a, the $\mathbf{a}_2$-axis at $\frac{2b}{3}$, and the $\mathbf{a}_3$-axis at c. Inverting these values and multiplying out, one arrives at the plane

$(4\,3\,2)$. Negatives are represented with a bar, e.g. $(1\,\overline{2}\,1)$, and planes parallel to an axis get coordinate 0 in that axis. All planes infinitely repeat along their normal vector, with a spacing d_{hkl} calculated as follows:

$$d_{hkl} = \frac{2\pi}{|\mathbf{g}_{hkl}|}, \tag{2.2}$$

with the reciprocal lattice vector $\mathbf{g}_{hkl} = h\mathbf{b}_1 + k\mathbf{b}_2 + l\mathbf{b}_3$, using the reciprocal basis defined in Equation 2.1. For cubic crystals, this simplifies down to $d_{hkl} = a/\sqrt{h^2 + k^2 + l^2}$.

Crystal directions are labeled with $[u\,v\,w]$. In general, $\mathbf{g}_{hkl}$ is the normal vector of $(h\,k\,l)$, but for cubic systems this simplifies to $[h\,k\,l]$ being normal to $(h\,k\,l)$. Families of equivalent crystal planes are placed in curly braces, e.g. for cubic crystals, the $(1\,0\,0)$, $(\overline{1}\,00)$, $(0\,1\,0)$, $(0\,\overline{1}\,0)$, $(0\,0\,1)$, and $(0\,0\,\overline{1})$ planes form the $\{1\,0\,0\}$ family. Similarly, families of equivalent crystal directions are labeled $\langle u\,v\,w\rangle$.

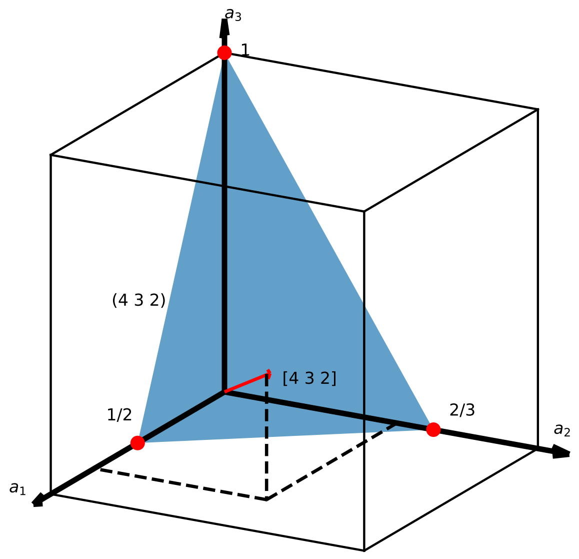


Figure 2.1.2: Example visualization of Miller indices. The blue plane intercepts the axes at $\frac{1}{2}a$, $\frac{2}{3}b$, and c. Inverting and multiplying to integers, we get the Miller indices $(4\,3\,2)$ for the plane. Its normal, $[4\,3\,2]$ for this cubic system, is shown in red.

Note that Miller indices are represented in the crystal basis. As the crystal basis is only orthonormal for cubic crystals, care must be taken when converting Miller indices to the Cartesian lab reference frame, and when calculating angles between crystal vectors and planes.

To represent the relation between Cartesian space and crystal space, a pole figure (PF) is often used. PFs are a projection of the unit sphere onto a plane, through a given pole. These allow the viewer to observe more of the direction space at once than a Cartesian projection shows. A PF can be constructed by placing a crystal in the center of a 3D sphere, and creating a pole for chosen crystallographic planes. A pole is placed where a ray from the center parallel to the plane normal intersects the sphere. This sphere is then projected onto the equatorial plane with the stereographic projection (SP), by casting a ray from all poles to a chosen pole. Note that, for poles on the same hemisphere as the pole chosen for projecting to, no intersection will be made with the plane, and as such these are not visible. This can of course be mitigated by showing a PF for both the north and south pole. A schematic representation of the construction of a SP PF can be seen in Figure 2.1.3.

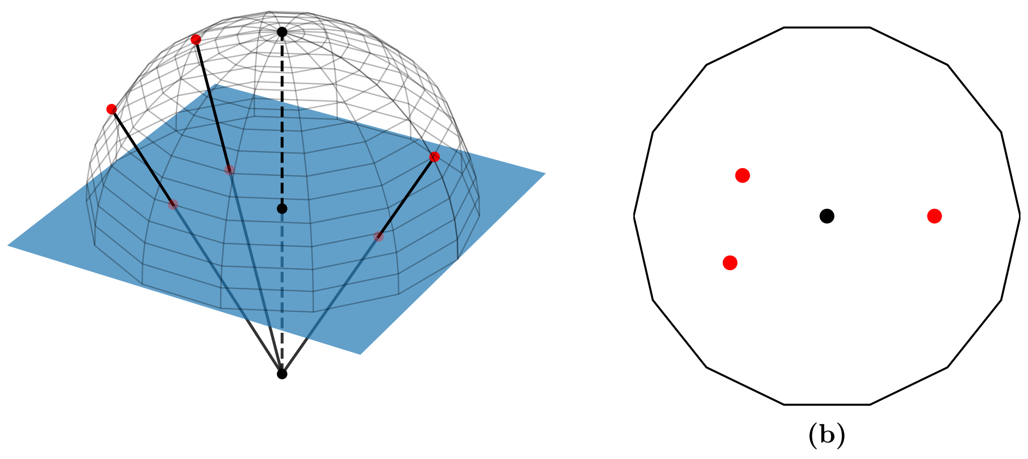

(a) (b)

Figure 2.1.3: Schematic representation of how a stereographic projection is constructed, and used to visualize a pole figure. Directions are cast out onto the unit sphere (the ray casting itself is not shown), represented by the red dots on the sphere mesh, before being projected onto the plane towards the point at $(0, 0, -1)$. (a) A 3D representation of the process. (b) The resulting pole figure.

The SP is specific to a chosen pole, often represented in the Cartesian lab frame of the sample. When a sample is viewed in lab-frame coordinates, it is usually not aligned with a crystal axis. One can therefore use inverse pole figures (IPFs), which are aligned to the crystal coordinates instead. As an example, in Figure 2.1.4(a) the IPF of a cubic crystal viewed from [0 0 1] can be seen.

When viewing IPFs of crystals, they often contain symmetrically equivalent poles. For a cubic crystal, for example, all $\langle 1\,1\,1 \rangle$, $\langle 1\,0\,1 \rangle$, and $\langle 0\,0\,1 \rangle$ are equivalent, making most of the IPF shown in Figure 2.1.4(a) redundant. One therefore often only show the reduced zone IPF, where only symmetrically unique poles may be present. A monoclinic crystal with the $\frac{2}{m}$ point group, for example, has a semicircle as its symmetry reduced IPF, whereas the m$\bar{3}$m point group has the spherical triangle between [0 0 1], [1 0 1] and [1 1 1]. These are shown in Figure 2.1.4(b) and Figure 2.1.4(c).

Additionally, these symmetry reduced IPFs are often shown as a color map, such that a certain color corresponds to a certain pole. This lets one show the orientations of spatially distributed crystals, by means of color-coding the orientation of

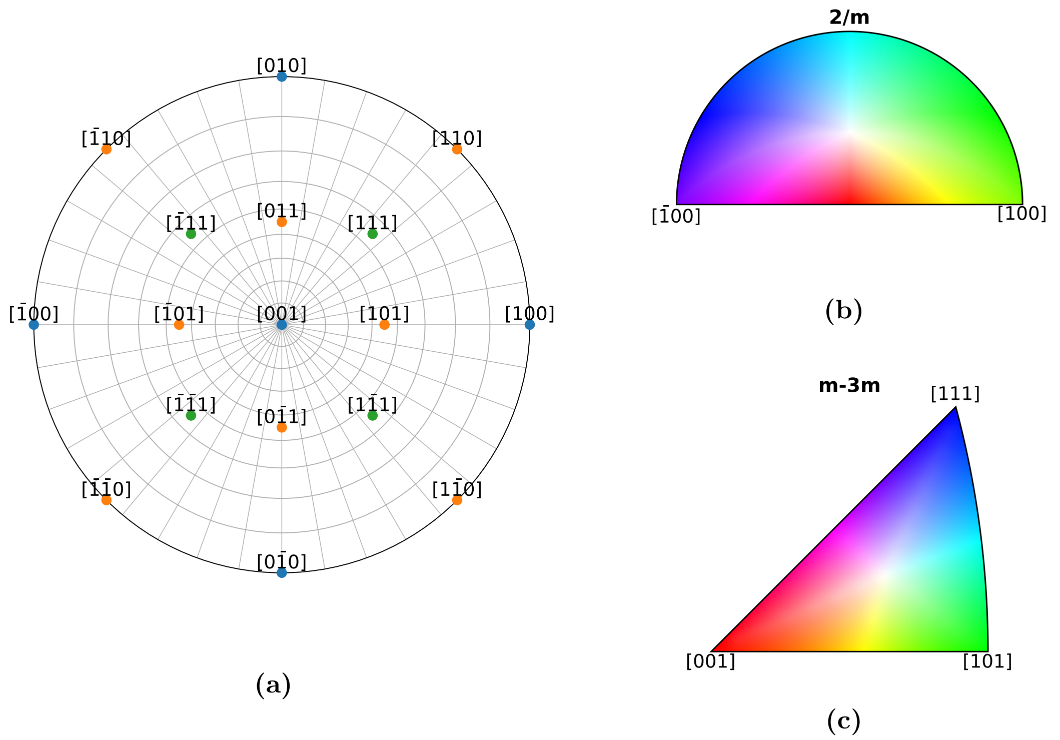
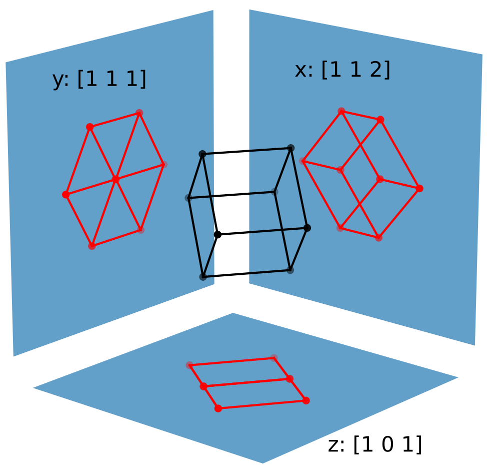


Figure 2.1.4: Utilizing the stereographic projection (SP) for plotting crystallographic directions. (a) Inverse pole figure (IPF) in a SP viewed from $[0\,0\,1]$, with $\langle 1\,0\,0 \rangle$, $\langle 1\,1\,0 \rangle$, and $\langle 1\,1\,1 \rangle$ labeled. (b), (c) Reduced zone IPFs for selected point groups, including a color map for the crystallographic directions.

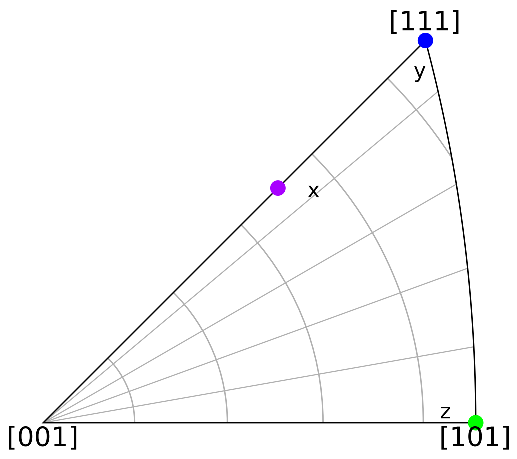
the crystal at a given point in space. Note that a single such color map is not sufficient to describe the orientation of a crystal, as viewing the orientation in a single direction (the chosen pole) gives no information of the orientation in any orthogonal directions. As such, when plotting spatially resolved crystal orientation maps, one normally show IPFs for both x, y and z-directions of the sample. This makes three colormaps, where for each position in space there are three colors, corresponding to the zone axis in each direction. A schematic of the interpretation of colormap-IPFs can be seen in Figure 2.1.5, where a single orientation is shown.

2.2 Diffraction

Diffraction, the study of interfering waves interacting with an object, is widely used for investigating crystalline materials. The length scales of the wave and the object must be comparable, and as such crystallographers mostly use X-rays and electrons. This chapter presents kinematic diffraction theory, based mostly on Kittel's *Introduction to Solid State Physics*[3]. The symbols follow Kittel, with the exception of the reciprocal lattice vector **g**, which Kittel capitalizes.



(a)



(b)

Figure 2.1.5: Schematic construction of the interpretation of colormap - IPFs for a cubic crystal, with Bunge Euler angles (0°, 45°, 35°). (a) The zone axis for each of the sample directions x, y and z. (b) The colors corresponding to the zone axes.

2.2.1 Bragg's Law

When a coherent wave of electrons is sent through a material, it will interact and scatter. Classically, this can be interpreted as elastic scattering with the Coloumb potential from the atoms in the material. If the material exhibits long-range order, e.g. crystalline materials, then one may observe well-defined maxima and minima in the scatter profile.

Bragg presented a simple picture of diffraction in a lattice: assuming a coherent incident wave with angle θ to a crystal plane $(h\,k\,l)$, it will reflect with the same angle θ. The wave would simultaneously reflect at the next parallel plane, traveling a distance $d_{hkl}\sin\theta$ further before reflecting again. To constructively interfere with itself, this extra path length to and from the next plane must equal an integer number of wave lengths. This is mathematically formulated in Bragg's law:

$$2d_{hkl}\sin\theta = n\lambda, \tag{2.3}$$

where λ is the wavelength and d_{hkl} is the lattice spacing for $(h\,k\,l)$ from Equation 2.2.

While Equation 2.3 correctly predicts scattering angles, and thereby allowing the measurement of lattice spacing, its simplistic description fails to predict diffraction peak intensities. For certain charge distributions and angles, the intensity is zero and the diffraction peak is absent, making Bragg's law impractical for diffraction predictions beyond plane spacings.

2.2.2 Diffraction Intensity

To account for scattering intensities, a systematic description of wave-lattice interaction is necessary. If a spatially distributed lattice property is responsible for scattering, we can describe this as

$$\rho(\mathbf{r}) = \rho(\mathbf{r} + \mathbf{r}_{uvw}) \tag{2.4}$$

due to the periodic nature of the lattice. ρ can for example be charge number density for electron and X-ray diffraction.

This can therefore be considered as ρ being limited to only one lattice site, and being convolved with the lattice. Applying a Fourier transform to simplify the lattice convolution, we get

$$\rho(\mathbf{r}) = \sum_{hkl} \rho_{\mathbf{g}} e^{i\mathbf{g}_{hkl}\cdot\mathbf{r}}, \tag{2.5}$$

where

$$\rho_{\mathbf{g}} = V_c \int_{\text{cell}} \rho(\mathbf{r}) e^{-i\mathbf{g}\cdot\mathbf{r}} dV.$$

This fulfills Equation 2.4, since $\exp(i\mathbf{g}_{hkl} \cdot \mathbf{r}_{uvw}) = 1$ for all integers h, k, l, u, v, w.

Assuming elastic collisions, the phase difference of waves scattered from two volume elements dV separated by $\mathbf{r}$ is $(\mathbf{k} - \mathbf{k}') \cdot \mathbf{r}$ for an incident wave $\mathbf{k}$ and an outgoing wave $\mathbf{k}'$. Defining the scattering vector as $\Delta\mathbf{k} = (\mathbf{k}' - \mathbf{k})$, the phase factor between the waves is $\exp(-i\Delta\mathbf{k} \cdot \mathbf{r})$. We introduce a scattering density F, defined as

$$F = \int_{\text{crystal}} \rho(\mathbf{r}) e^{-i\Delta\mathbf{k}\cdot\mathbf{r}} dV, \tag{2.6}$$

which describes the amplitude of the scattered wave with scattering vector $\Delta\mathbf{k}$.

Inserting Equation 2.5 into Equation 2.6, we get

$$F = \sum_{hkl} \int_{\text{crystal}} \rho_{\mathbf{g}} e^{i(\mathbf{g}_{hkl} - \Delta\mathbf{k})\cdot\mathbf{r}} dV. \tag{2.7}$$

Note that F can be complex, but as the intensity $I \propto F^* \cdot F$ this will only appear as a phase term, which is subsequently lost as we only measure the amplitude of the intensity.

Integrating Equation 2.7 over an infinite lattice makes F non-zero only for certain values of $\mathbf{k}'$. These give the diffraction condition

$$\Delta\mathbf{k} = \mathbf{g}_{hkl}. \tag{2.8}$$

Multiplying both sides of Equation 2.8 by the translation vectors, we get the Laue equations:

$$\mathbf{a}_i \cdot \Delta\mathbf{k} = 2\pi v_i \tag{2.9}$$

for integers v_i.

The diffraction condition $\Delta\mathbf{k} = \mathbf{g}$, the Laue equations, and Bragg's law are all equivalent formulations of the same phenomenon. One interpretation is that Bragg's law is the real-space formulation of the diffraction condition, whereas the Laue equations are the reciprocal space formulation.

The diffraction condition in reciprocal space is nicely visualized using Ewald's sphere. Since elastic collisions are assumed, all possible $\mathbf{k}'$ lie in a sphere around $\mathbf{k}$, due to the conservation of energy. Imposing this sphere onto the reciprocal lattice, the points at which $\Delta\mathbf{k}$ equals a reciprocal lattice vector is readily observed by finding intersections of the sphere and the lattice. A 2D example Ewald sphere construction can be seen in Figure 2.2.1.

In an infinite lattice, the points in the Ewald sphere construction are mathematically perfect points. With finite crystals, diffraction is more lenient. This is due to a finite crystal being the product of an infinite lattice and a box function, meaning the sum in Equation 2.5 needs to be replaced by an integral over the new reciprocal lattice. This can be accounted for by introducing an excitation error $\mathbf{s}$:

$$\Delta\mathbf{k} = \mathbf{g}_{hkl} + \mathbf{s}. \tag{2.10}$$

Applying the convolution theorem, the Fourier transform of the finite lattice is the convolution of a reciprocal lattice, and the Fourier transform of the box function. As the Fourier transform of a box function is sinc-shaped, which is close to 0 everywhere except near its center, Equation 2.5 is a good approximation for macro-sized crystals. Due to the inverse proportional relationship between spatial and reciprocal extent, a crystal which is thin (e.g. about 100 nm) will exhibit relrods. These are a significant widening of the points in the reciprocal lattice, making diffraction more lenient. More mathematically, the excitation error in Equation 2.10 is larger in directions where the crystal is smaller.

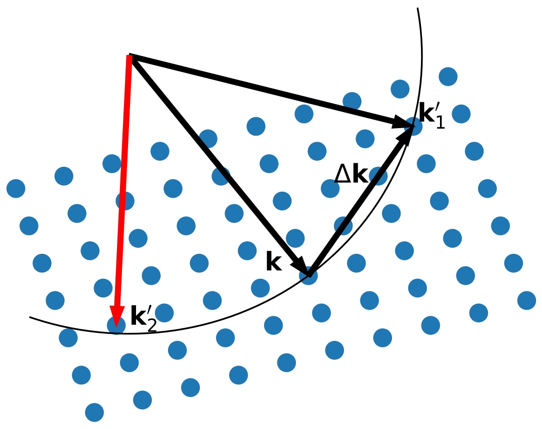


Figure 2.2.1: An arc of Ewald's sphere for $\mathbf{k}$. Only $\Delta\mathbf{k}$ that coincide with a lattice vector exhibit diffraction, as is the case for $\mathbf{k}_1'$. Drawn in red is $\mathbf{k}_2'$, which fulfills the diffraction condition with a lenient enough maximal excitation error $\mathbf{s}$.

2.2.3 Structure Factor

If the diffraction condition is met, then Equation 2.7 simplifies down to $F = NS_\mathbf{g}$ where $S_\mathbf{g}$ is the structure factor, i.e. the scattering density contribution for a single cell, and N is the amount of cells in the volume. Written out,

$$S_\mathbf{g} = \int_{\text{cell}} \rho(\mathbf{r}) e^{i\mathbf{g}\cdot\mathbf{r}} dV. \tag{2.11}$$

If ρ can be decomposed to a superposition, e.g. the charge density contribution from multiple atoms in a multi-atom unit cell, then one can write

$$\rho(\mathbf{r}) = \sum_{j=1}^{s} \rho_j(\mathbf{r} - \mathbf{r}_j) \tag{2.12}$$

for s atoms in the basis, at positions $\mathbf{r}_j$. Defining $\mathbf{r}' = \mathbf{r} - \mathbf{r}_j$, and inserting Equation 2.12 into Equation 2.11, $S_\mathbf{g}$ can now be written as

$$S_\mathbf{g} = \sum_j e^{i\mathbf{g}\cdot\mathbf{r}_j} \int_{\text{cell}} \rho_j(\mathbf{r}') e^{i\mathbf{g}\cdot\mathbf{r}'} dV.$$

Introducing the atomic form factor f_j as

$$f_j = \int_{\text{cell}} \rho_j(\mathbf{r}') e^{i\mathbf{g}\cdot\mathbf{r}'} dV, \tag{2.13}$$

we can write

$$S_\mathbf{g} = \sum_j f_j e^{i\mathbf{g}\cdot\mathbf{r}_j}.$$

With $\mathbf{r}_j = x_j\mathbf{a}_1 + y_j\mathbf{a}_2 + z_j\mathbf{a}_3$, this simplifies down to

$$S_{\mathbf{g}_{hkl}} = \sum_j f_j e^{2\pi i(x_jh + y_jk + z_jl)}. \tag{2.14}$$

Taking cesium chloride as a simple example, with a two-atom basis in a cubic P lattice. Defining the origin at a cesium atom, the chlorine atom sits at [0.5 0.5 0.5]. Choosing $\mathbf{g}_{100}$, inserting into Equation 2.14 yields

$$\begin{aligned} S_{\mathbf{g}_{100}} &= \sum_j f_j e^{2\pi i(x_jh + y_jk + z_jl)} \\ &= f_{\text{Ce}} e^0 + f_{\text{Cl}} e^{2\pi i(0.5\cdot 1 + 0.5\cdot 0 + 0.5\cdot 0)} \\ &= f_{\text{Ce}} + f_{\text{Cl}} e^{\pi i} \\ &= f_{\text{Ce}} - f_{\text{Cl}}. \end{aligned}$$

Similarly, $S_{\mathbf{g}_{200}} = f_{\text{Ce}} + f_{\text{Cl}}$. Notice that, if $f_{\text{Ce}} = f_{\text{Cl}}$, then $S_{\mathbf{g}_{100}}$ would be zero. If the atoms are identical, then this would be the case. However, if the atoms are identical, with one atom at the origin and one in the center of the unit cell in a cubic lattice, then it would not be a two-atom basis in cubic P, but a one-atom basis in cubic I. Therefore, cubic I has the "rule" where $S_{\mathbf{g}_{hkl}} = 0$ for $h + k + l$ being an odd number, and $S_{\mathbf{g}_{hkl}} = 2f$ for even $h + k + l$. This mathematical condition for the structure factor is called an extinction rule, and results in a missing diffraction spot even when the diffraction condition is satisfied. Both centerings, glide planes and screw axes can lead to extinction rules, all of which are tabulated in [1, page 397].

Using the structure factor formalism, describing the unit cell of a crystal is enough to predict the scattering properties of the material. Conversely, measuring the scattering properties of a material allows for reconstruction of the unit cell. The calculation of $S_\mathbf{g}$ requires a description of the atoms in the basis, their positions and form factors, as well as the reciprocal lattice of the crystal. Since $F^*_{hkl} = F_{\bar{h}\bar{k}\bar{l}}$, and therefore $I_{hkl} \propto F^*_{hkl} \cdot F_{hkl} = F_{hkl} \cdot F^*_{\bar{h}\bar{k}\bar{l}}$, then I must be centrosymmetric ($I_{hkl} = I_{\bar{h}\bar{k}\bar{l}}$). For the measurement of the intensity pattern, the point group of the reciprocal lattice is therefore reduced to its centrosymmetric Laue group if it did not already have this symmetry. This is known as Friedel's law, and breaks the direct relationship between scattering and crystal structure for non-centrosymmetric crystals. For example, a kinematic diffraction measurement would not be enough to determine polarization of a ferroelectric crystal, as the polarization is determined by the direction of the non-centrosymmetric unit cell of the crystal.

In practice, handling the unit cell parameters, position and type of all atoms in a crystal can be cumbersome, especially for large complex crystals like zeolites or protein crystals. Therefore, crystallographers often make use of `.cif`-files, an abbreviation of Crystal Information File. These text files contain lattice parameters, atoms and their positions in the basis, and all valid symmetry operations for a certain crystal. Additionally, the format supports a wide range of metadata, such as the name of the structure and scientific sources for the data.

The kinematic diffraction theory presented in this chapter functions as an approximation of electron scattering in crystalline materials. For the weak interactions of X-rays and matter, the accuracy is high. For the comparably strong electron-matter interactions, however, the accuracy quickly declines with increased sample thickness. This is due to the increased likelihood of dynamic scattering, where electrons participate in more than one scattering event on their path through the sample. The advantage of a strongly interacting wave is the ability to reduce the probe size compared to more weakly interacting waves, but as discussed, it comes at the cost of a less kinematic diffraction. Methods to alleviate this, other than making the sample thinner, are discussed in Chapter 2.3.4.

2.3 Transmission Electron Microscopy

Transmission electron microscopes (TEMs) are a central instrument in the study of materials, with a spatial resolution down to the sub-nm-range. Broadly, a TEM accelerates electrons to energies in the 60 keV to 400 keV range, which then hits a thin (100 nm-range) sample. These electrons travel through the sample, thereby interacting with it and changing their paths, before hitting a detector on the other side. The electron beam can be static, broad, parallel, or convergent to a point on the sample, categorized into conventional TEM (CTEM) for parallel beam operation and convergent-beam electron diffraction (CBED) for convergent beam operation. A user can either capture spatial data, e.g. using high resolution TEM (HRTEM) or high-angle annular dark-field (HAADF), or diffraction data, the latter being the focus of this section.

2.3.1 Working Principle

A TEM, whether scanning or not, consists of the same general parts: a column with an electron gun at the top, electromagnetic condenser-, objective-, and intermediate lenses for focusing and for switching between reciprocal- and real-space, a sample holder, various apertures in e.g. the image-plane and/or the back focal plane, and detectors. Electrons are accelerated along the optical axis, which they should remain close to to avoid aberrations, before being collected by the condenser lenses. They then pass between the condenser aperture, and into the scanning coils which can move the focal point around on the sample. The condenser system is responsible for collecting and focusing the beam, and the scanning coils move the beam focal around on the sample. Next, the beam is passed through the upper objective lens, the sample, and the lower objective lens. An objective aperture and a selected area aperture may be placed in the back focal plane and image plane, respectively. Finally, the beam is passed through intermediate lenses and projector lenses, before entering the viewing chamber or directly to the detector. The projector system allows the user to switch between imaging mode and diffraction mode, and change the magnification. As the system totals around 1 m to 2 m in length, and electrons interact strongly with matter, the whole column is vacuum-pumped. A schematic overview of a TEM can be seen in Figure 2.3.1.

When the projector lenses projects the back focal plane to the detector, the diffraction pattern can be collected. Different detectors allow for different imaging techniques, as e.g. a bright-field (BF) or HAADF. These detect electrons in certain angular ranges. With a pixelated detector, e.g. a direct electron detector (DED) or simply a camera aimed at a fluorescent screen, one can capture the scattering intensity as a grid, instead of a single value for a range. This results in a 4D dataset, where a 2D diffraction pattern image is captured for each sample scan position. The technique is known as scanning electron diffraction (SED), which is a CBED technique. With a pixelated detector, one can still extract the same data an e.g. BF-detector would, as this data is present in the dataset, by azimuthally integrating over the correct radial range. For BF, the pixelated detector-version is then called virtual bright-field (VBF).

Compared to conventional TEM techniques, SED allows for more automated data collection and processing. An example of data captured with this technique can be seen in Figure 2.3.2(a), where the total integral of intensity in the diffraction pattern is used to reduce the 4D data to a comprehensible 2D format. Note that this is not VBF, but a integral (sum) of the entire detector area.

2.3.2 Selected Area Electron Diffraction

Selected area electron diffraction (SAED) is a CTEM technique used to attain high quality diffraction patterns from a small region of a sample [2, ch. 2.3.2]. The region selection is performed in the image plane, using the Selected Area Aperture as shown in Figure 2.3.1. The detector is in diffraction mode, and the images captured with SAED normally have small diffraction spots compared to CBED techniques. The spatial resolution for the 'selected area' is limited to around 1 µm in diameter, a restraint which can be improved by convergent techniques [2, p. 73].

2.3.3 Nano-beam Diffraction

Nano-beam diffraction (NBD) is a CBED technique, where the electron beam is focused to a point with nanometer diameter. The size of this point controls the volume of the sample which the electrons interact with on their path towards the detector, and it also controls the size of the diffraction spots on the captured image. As opposed to SAED, the diffraction spots from NBD are disks, rather than points,

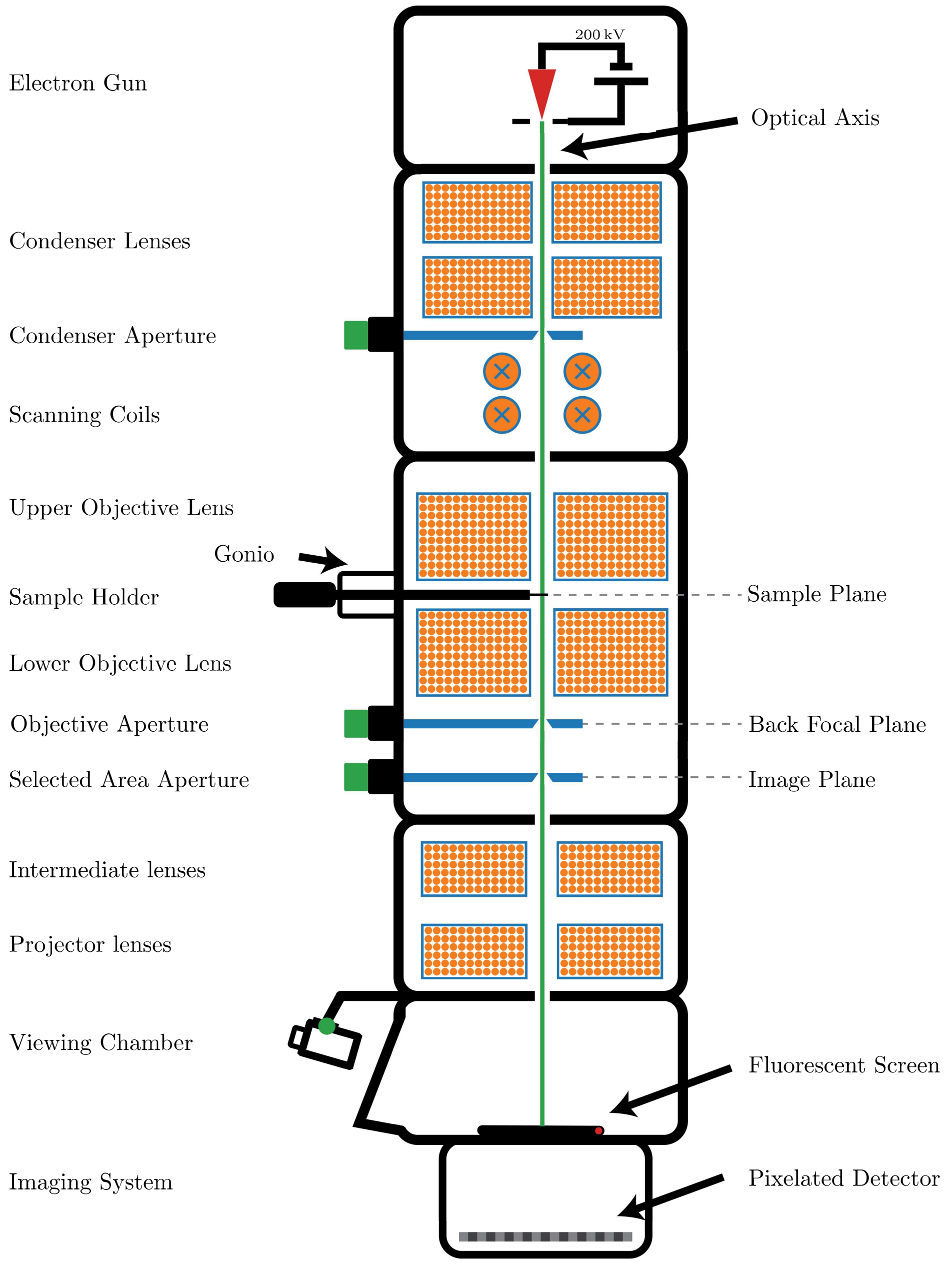
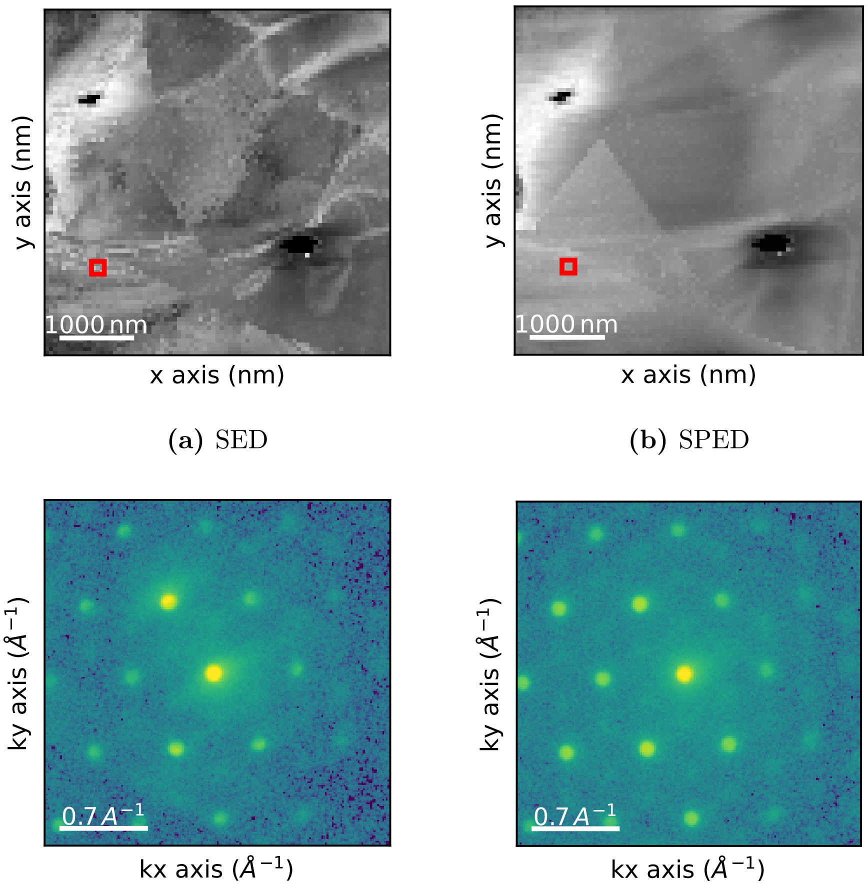


Figure 2.3.1: Schematic representation of a TEM, with important components labeled. Reprinted with permission from [49].



(a) SED **(b)** SPED

(c) NBD **(d)** PED

Figure 2.3.2: 256×256 scanning electron diffraction (SED) data, with and without precession, taken of FCC Ag. (a), (b) Integrated intensity micrographs for SED and SPED, respectively. (c), (d) Diffraction patters taken from the red square in (a) and (b), respectively.

due to the convergence angle of the beam being non-zero. Modern TEMs are capable of sub-nm probe sizes, and sub-mrad convergence angles [2, p. 78]. NBD is often used in conjunction with scanning of the probe, i.e. SED.

2.3.4 Scanning Precession Electron Diffraction

The theory presented on diffraction in Chapter 2.2 assumes elastic, kinematic diffraction. A theory which accounts for a more realistic, dynamical, diffraction is certainly possible, but it would necessarily be much more involved and demanding to simulate. Steps can instead be taken to reduce dynamical effects in measurements. One such approach is to angle the convergent beam slightly (e.g. 1°) off the optical axis, precessing around the optical axis, and measuring the average diffraction intensities over one or more precessions. This CBED technique is called precession electron diffraction (PED), and results in a more kinematic-like diffraction pattern. To avoid aliasing, one must ensure the detector period is an integer multiple of the precession period, e.g. 50 ms detector period for a 100 Hz precession would make the beam precess 5 times per diffraction image.

The decreased dynamical effect comes at the price of resolution, as the tilted electron beam interacts with a larger volume of the sample, thus decreasing spatial resolution. Furthermore, the optical system responsible for the precession comes with its own optical challenges, such as alignment and aberrations. Note that precession is performed in the opposite direction after the beam has passed through the sample, to obtain a static PED pattern.

The effect of precession on Ewald's sphere can be seen in Figure 2.3.3, showing how more reflections are in the diffraction pattern due to the excitation error **s** from Equation 2.10.

When a precessing beam is scanned over the sample, i.e. using SED and PED simultaneously, the technique is called scanning precession electron diffraction (SPED). An example of SED vs. SPED can be seen in Figure 2.3.2, where the blurring of the diffraction spots along certain directions in the SED diffraction pattern is noticeably reduced in the SPED diffraction pattern. The effect is also apparent in the integrated intensity images, with the SPED image being much less noisy and less segmented than the SED image.

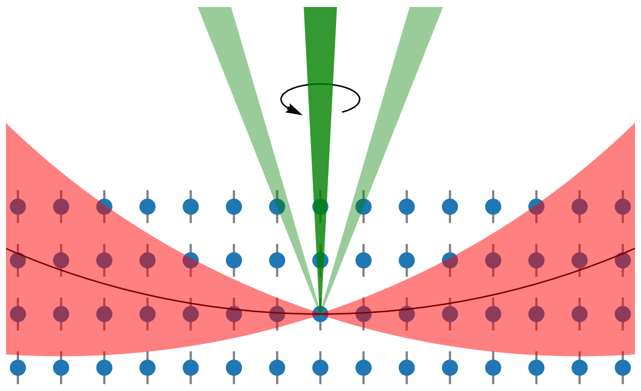


Figure 2.3.3: Schematic representation of the Ewald's sphere construction for PED. The convergent electron beam, shown in green, would only hit the sample as the central, darker green beam in NBD, resulting in the spherical Ewald's sphere shown as a black arc. As the beam precesses, shown in the left- and rightmost extrema with the other two lighter green electron beams, more reflections are hit. This is represented with the intersections of the red PED Ewald's sphere volume and the blue reciprocal lattice. The lattice is shown including relrods, represented with black bars.

2.3.5 Laue Zones and the Laue Circle

In TEM, $\Delta\mathbf{k}$ is normally much larger than the reciprocal lattice vector. As such, Ewald's sphere can usually be approximated as flat. The region where the sphere is approximately flat is called zeroth-order Laue zone (ZOLZ), and gives rise to most reflections in a zone axis pattern. Further out in reciprocal space, the curvature of Ewald's sphere becomes significant enough as to no longer intersect with the relrods, which show up as a dim radial region in SAED patterns. Eventually, with large enough camera lengths, Ewald's sphere once again intersects the relrods, this time in the layer of unit cells above the ones in the ZOLZ. These are the higher-order Laue zones (HOLZs). [2, p. 278]

SAED zone axis patterns are radially symmetric, as Ewald's sphere is aligned with the crystal. Small deviations from a zone axis will generally keep the same reflections, due to the extent of relrods, but intensities change. Specifically, the first-order Laue zone (FOLZ) gives rise to a 'circle', or arc, of more intense reflections. This is known as the Laue circle. By calculating the angle between the center of the Laue circle with the direct beam (using Equation 2.3 to translate between vectors in reciprocal space and angles), one gets the angular deviation of the crystal orientation to the zone axis.

2.3.6 Sample Holder

To insert a sample into the TEM, it needs to be placed in a sample holder. These keep the sample in place in a known location, and are designed to facilitate some sample manipulation when inside the vacuum column. Sample holders come in many varieties, e.g. with heating or cooling, and most allow movement and tilting. Multiple different sample holder designs exists for tilting, e.g. double-tilt and tilt-rotate. Both of these share a tilt axis along the sample holder, often called the gonio tilt, which can be seen in Figure 2.3.1 at the 'Gonio' annotation. The difference between double-tilt and tilt-rotate sample holders lie in the position of their second tilt axis. The double-tilt holder has another tilt axis in the sample plane, whereas the tilt-rotate has an axis parallel to the optical axis. Note that, in both cases, these axes are attached to the gonio axis, such that they move along with the sample if a gonio tilt is applied.

An important difference between double-tilt and tilt-rotate sample holders are the tilt ranges. Along a given tilt axis, only certain angles are possible to tilt to, limiting the orientations the sample can be put in. Commonly, the gonio tilt is limited to $\pm 30°$. Double-tilt holders are commonly limited to $\pm 20°$ or $\pm 30°$ in the second tilt axis, whereas tilt-rotate holders might have the full $\pm 180°$ range inside its limits. More specialized holders exist, e.g. specialized tomography holders might have $\pm 60°$ for the gonio tilt.

2.4 Coordinate Systems and Orientations

When describing a TEM, the relationship between positions on the sample, positions on the detector, and the position and orientation of the crystal(s) in the sample can be cumbersome to work with in conjunction. Depending on the focus of study, different reference frames are used and/or considered. The relationship between these will be discussed in this section.

2.4.1 Orientations and Misorientations

Orientations are a description of the way an object is oriented in space. An interpretation of orientations is that they represent the coordinate transform between the object's coordinate system and the coordinate system the object occupies. As an example, if a cube is aligned with a Cartesian coordinate system, and then rotated 45° around one of the axes, then its orientation changed.

Orientations can also have a notion of symmetry. If the cube were to be rotated another 45° around the same axis as before, then its orientation would be the same as in the initial state, since cubes have 90° rotational symmetry.

Misorientations can be seen as a generalization of orientations. Where orientations represent the operation from the reference frame an object exists in to the reference frame of the object, misorientations represent the operation transforming one orientation to another. Mathematically, this can be expressed as $q_{AB} = q_{AO}q_{OB}$, where q_{ij} represents the transformation from i to j, and O is the reference frame both objects occupy. An orientation q_A would then be the same as a misorientation q_{OA}, meaning $q_{AB} = q_{AO}q_{OB} = q_A^{-1}q_B$. The same notion of symmetry applies to misorientations as with orientations, but since a misorientation entails two separate orientations, the symmetry of a misorientation consists of two possibly distinct parts. For example, the misorientation between a monoclinic and a hexagonal crystal can certainly be constructed, but care must be taken to account for the different symmetries in the two crystals. For the generalization of orientations as misorientations, this means that the space O in $q_A = q_{OA}$ has point group 1.

2.4.2 Orientation Representations

To represent coordinate transforms and orientations, one can employ Euler angles. These describe a sequence of rotations along the cardinal directions. The rotations can either be intrinsic, i.e. the chained rotations follow the new axes from the previous rotations, or extrinsic, i.e. chained rotations follow static external axes. If only a single rotation is applied, the two are equivalent. Two chained rotations (along two different axes) are different in extrinsic and intrinsic rotations, as can be seen in Figure 2.4.1.

The transform from one coordinate system to another can also be represented with a matrix, which, when multiplied by a vector in one coordinate system, gives the same vector represented in the other coordinate system. When working with Euler angles, this matrix is straightforward to construct. One constructs a matrix for each axis, and multiply them either to the left or right depending on whether they are extrinsic or intrinsic. A rotation matrix is mathematically more convenient to work with than Euler angles, as the representation of any vector in either reference frame can easily be converted to the other. Rotation matrices do not contain more or different information than Euler angles, as they are both equivalent representations of a orientation.

The rotation matrices for each axis are as follows:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \tag{2.15}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \tag{2.16}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.17}$$

Using Figure 2.4.1 as an example, the rotation matrix transforming a vector in the blue rotated coordinate system in Figure 2.4.1(b) back to the fixed coordinate system is $R_x(30°)R_z(60°)$, whereas for the intrinsic rotation in Figure 2.4.1(a) the rotation matrix is $R_z(60°)R_x(30°)$. To transform the other way around, i.e. a vector in the fixed coordinate system to the rotated one, one multiplies instead with the inverse of the total rotation matrix. For any rotation matrix, its inverse is its transpose, making this operation trivial to compute.

As a concrete example, the unit vector along x_e in Figure 2.4.1(b) can be transformed back into the fixed coordinate system as follows:

$$R_x(30°)R_z(60°)\begin{bmatrix}1\\0\\0\end{bmatrix}_e = \begin{bmatrix} 0.5 & -0.866 & 0 \\ 0.75 & 0.433 & -0.5 \\ 0.433 & 0.25 & 0.866 \end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}_e = \begin{bmatrix}0.5\\0.75\\0.433\end{bmatrix}.$$

Note that this is the same vector, but represented in different coordinate systems.

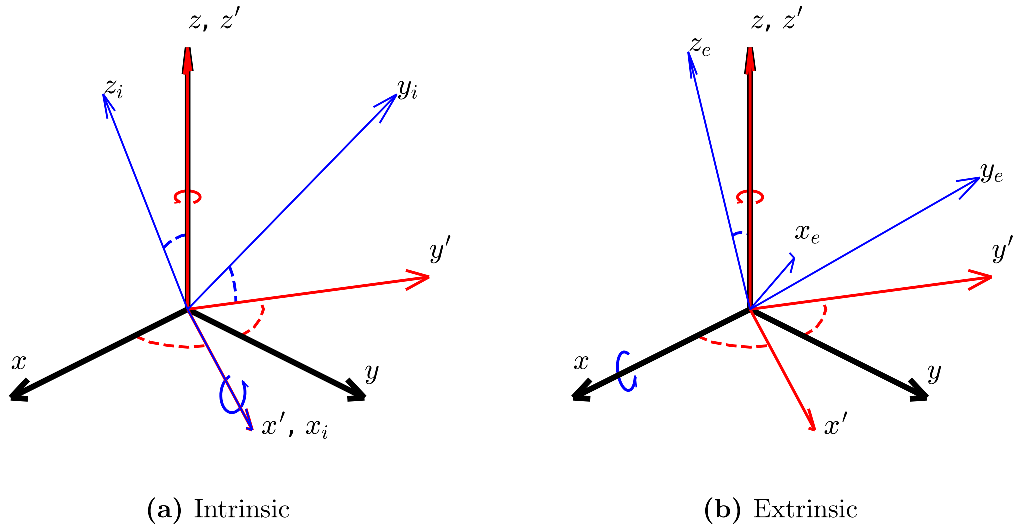


Figure 2.4.1: Schematic representation of two consecutive Euler rotations, intrinsic in (a) and extrinsic in (b). A 60° z-rotation followed by a 30° x-rotation is shown for both. The z-rotation is shown in red, transforming x to x' etc. The intrinsic x-rotation is shown in blue in (a), transforming x' to x_i etc. by rotating around x', while (b) shows the extrinsic x-rotation in blue, transforming x' to x_e etc. by rotating around x.

Euler angles suffer from a few drawbacks, notably Gimbal lock, as well as the need to very clearly specify the axis convention used [50]. While crystallographers commonly use the intrinsic zxz (Bunge) convention [47, ch. 2], somewhat mitigating the convention issue, both issues can be mitigated by a change in representation. Rotation matrices is one way, but as they use nine different numbers in their representation, most orientation software uses quaternions for internal calculations. These use four numbers to represent an orientation, which can save memory for large calculations, while simultaneously mitigating issues of numerical instability that can be encountered with rotation matrices [50].

Quaternions also lend themselves nicely for easy conversion to the final orientation representation to be discussed, namely the axis-angle representation. This representation consists of a vector as an axis of rotation, and an angle to rotate around that axis. This representation is easier to interpret, but is not commonly used for rotation/orientation calculations directly. The axis-angle representation is particularly useful when describing crystal twins. As an example, FCC crystals can exhibit $\Sigma 3$ twinning, with a 60° rotation around the $\langle 1\,1\,1 \rangle$ axis.

Note that all orientation representations are equivalent, as they describe the same operation, and as such they can be converted to and from each other.

Calculating the mean orientation of a set of orientations is non-trivial. For quaternions, for example, simply taking the mean of each of the four elements is not sufficient, as this trivial algorithm does not preserve the norm. A paper by Markley et al. [51] define an algorithm (equation 13) with the proper properties for a mean quaternion.

2.4.3 Uniform Sampling of Orientation Space

Orientations in 3D form a space known as SO(3). Uniformly sampling this space can be tricky, especially when different expressions of the same space can give different measures of uniformity [52–54].

SO(3) is a fairly large group; with the unit quaternion interpretation [54, ch. 4], and a resolution of 1, 23,207,680 unique elements exist. This can be reduced by symmetries [31], e.g. imposing the point group m$\bar{3}$m onto these reduces the 23 million down to 2,907,105. Further reduction can be achieved if the orientations need only span the subspace of a projection along a given direction. Specifically for crystal orientations in the TEM, only the orientations with a unique projection along the optical axis needs to be considered. These can easily be sampled with only two Euler angles, and further reduce the previously discussed orientation set down to 485,147 unique elements. With Euler angle sampling of 1° resolution, only 300 unique z-projections exist for m$\bar{3}$m.

2.4.4 TEM Coordinate Systems

A tilt-rotate and a double tilt holder are fundamentally the same, with the main differences being the position of the second rotation axis. For both, the sample can be rotated around the axis connecting the holder to the column, and subsequently an orthogonal axis which moves along with the first rotation. In a tilt-rotate holder, this second axis is parallel to the optical axis in the default position, whereas the second tilt axis in a double tilt holder is in the sample plane. The first axis, called a gonio-tilt or α-tilt, is extrinsic, whereas the second rotation is intrinsic. An overview of the different coordinate systems and their relations may be found in Figure 2.4.2.

To calculate how tilting affects the sample and, more importantly, the crystal(s) in the sample, the orientation relations of each of these reference frames must be established. The crystal-sample relation is found by orientation mapping, e.g. using template matching (TM), which is the subject of Chapter 2.5.3. The S frame is related to the L frame through a simple rotation around the Lz-axis, by the chosen

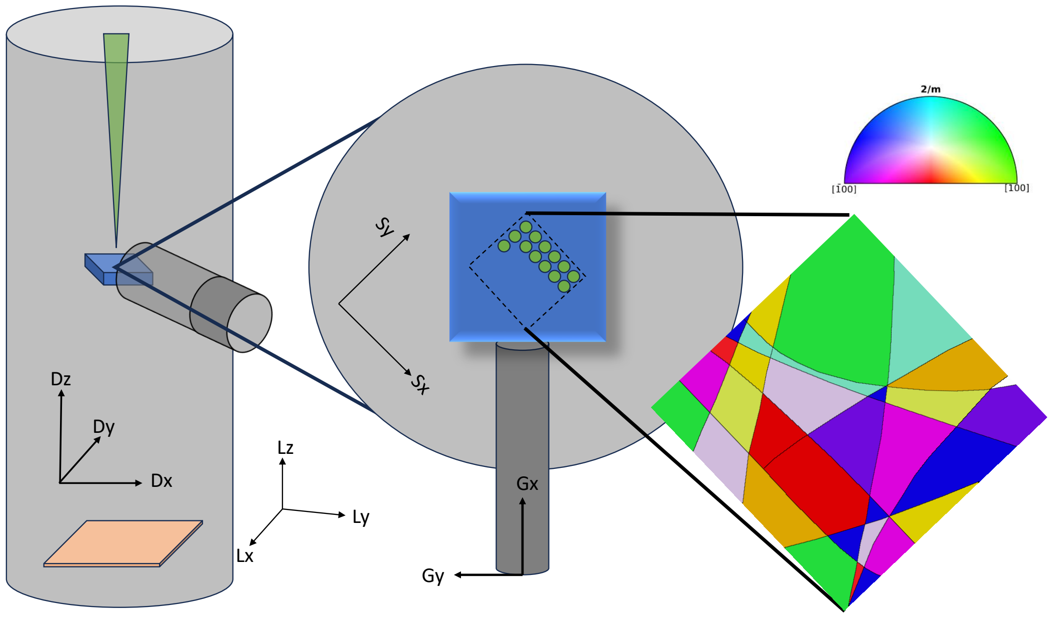


Figure 2.4.2: An overview of the different coordinate systems used when describing a sample holder. The electron beam is parallel to the lab-z (Lz). The rest of the lab frame, i.e. Lx and Ly, are arbitrary. The detector is aligned with the lab frame in z, i.e. Dz = Lz. The x- and y-directions of the detector are static compared to the lab frame. The zoom-in shows the scan directions Sx and Sy, which can be rotated compared to Dx and Dy, but the beam is always along Sz = Dz = Lz. Example scan positions are indicated with green dots on the blue sample. The sample holder, i.e. the dark gray cylinder, can tilt and rotate the sample. The Gx axis is static compared to Lx/Ly, and consequently Dx/Dy, but a rotation around Gx changes the positions of Gy and Gz. With no rotations of either sample axes, Gz coincides with all other z-axes so far. The sample's Cartesian coordinate system is the gonio (G) frame, i.e any rotation of the axes do not change the sample's internal coordinates. However, a crystalline sample often has different orientations, quantified in an orientation map. The final zoom-in is a inverse pole figure (IPF) of an orientation map, where a IPF-x would be the orientation between Sx and the crystal at each point in the scan, and so on. Above the inset is a colormap translating the color visualization to a direction, here in the $\frac{2}{m}$ point group. Note that the IPF relate crystal directions, which can require careful considerations when translating to Cartesian space depending on the Bravais lattice.

scan rotation parameter. The same is normally true for the G reference frame, although it is certainly possible to construct TEMs with tilt axes outside of the Lxy-plane. Therefore, the tilt axes (which are aligned with Gx and e.g. Gy) can be expressed in the S frame by multiplying with Equation 2.17 twice; once to transform the G frame to the L frame, and once more to transform the L frame to the S frame. This requires careful consideration of the order, as with all misorientations, as one of the operations is inverse. When tilting, the G frame moves compared to the L frame, by properly chained extrinsic and intrinsic rotations around the axes. The D frame does not require too much consideration, as it cannot move in any way. It can be assumed to be perfectly aligned with the L frame. Note that others might choose differently, e.g. the Python package Hyperspy defines the positive Dz direction to be -Lz.

Additionally, eucentric height can affect tilting. If the eucentric height is not set correctly, the sample can drift significantly during tilting. This can be especially confusing if the scan rotation is not set as to align the S frame with the G frame. Manual calibration is easily performed by rocking the sample with the gonio whilst adjusting the eucentric height; the height is correct when the sample stops moving (tilting will still cause shearing and stretching, but no translation).

2.4.5 Sample Holder Orientation

With the preliminary definitions of rotations, as well as the relationships between coordinate systems shown in Figure 2.4.2, the effect of tilting on orientation maps can be deduced. This entails defining the orientation of each coordinate system compared to the others, defining the rotation operations of the tilt/rotate axes, and combining these in a chain of e.g. rotation matrices. Extra care must be taken to ensure intrinsic and extrinsic relationships are properly accounted for, as Figure 2.4.1 shows, getting this wrong can produce different results. Additionally, the orientations in the orientation map must be considered, as the reference frame which is transformed to and from might be different.

2.5 Orientation Mapping with Template Matching

Orientation mapping is the process of determining the crystal orientation(s) of a sample. A common approach is using electron back scattered diffraction (EBSD) based on Kikuchi band patterns, but this comes with the limitation of a spatial resolution in the 20 nm to 80 nm-range. Using diffraction spot patterns from a TEM gives a much higher spatial resolution, but orientation maps based on these are typically accurate to around 1.1°, compared to 0.3° using Kikuchi patterns from TEM, and 0.6° with EBSD [39]. While the accuracy of diffraction spot-based orientation mapping is lower than other methods, its advantage is the possibility of automated crystal orientation mapping (ACOM). A common approach to determining crystal orientation from a diffraction spot pattern is template matching (TM), where a library of diffraction simulations are compared to the diffraction pattern, and the simulation that matches the best is assumed to be correct. A schematic representation of the workflow can be found in Figure 2.5.1

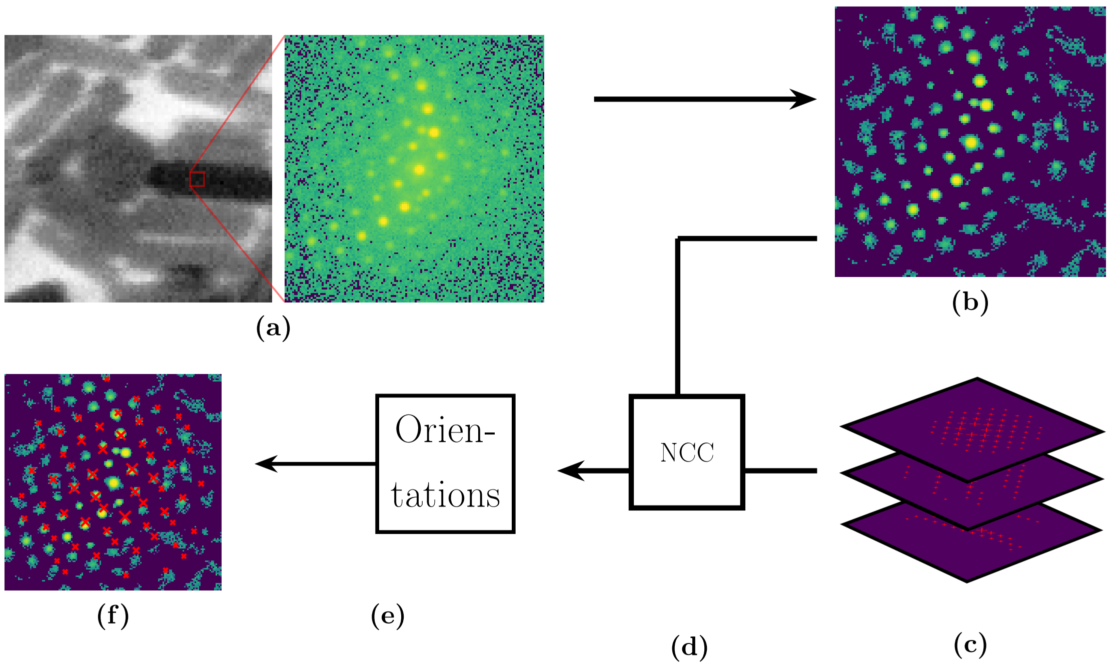


Figure 2.5.1: Schematic representation of template matching workflow. (a) Record diffraction data. (b) Pre-process it to center and remove noise. (c) Simulate a template bank. (d) Correlate the patterns with the template. (e) The output of TM: orientations. (f) Verify the solutions. The data is from a SPED scan of gold nanoparticles, with a 144×144 detector. The reader might note that these are poor results, as can be seen by the poor fit between the simulated and measured spots in (f).

2.5.1 SPED Data Pre-Processing

SPED data is often imperfect, being subject to noise effects. For high-index zone axes, the reflections can be comparable in intensity to the background noise. As such, SPED data often needs pre-processing before a good TM result can be achieved.

The pre-processing steps required for good TM results vary between instruments, scan settings, and samples. Calibrating the scale, both in real and reciprocal space, is often necessary. To this end, two techniques are presented for reciprocal scale calibration. Real-space calibration is not discussed.

The first, more manual, approach is to navigate to a diffraction pattern with a known zone axis. From this diffraction pattern, reflections can be manually indexed. Dividing the reciprocal distance between reflections, which can be calculated as the inverse of Equation 2.2, by the amount of pixels between the reflections gives the calibration value in reciprocal units per pixel.

A more automatic approach, and thus less prone to human error, begins by again selecting a known zone axis. Then, one performs simulations of the same zone axis, but for a range of different calibrations. The calibration yielding the simulation with the highest correlation is then selected as the true calibration.

Data centering can be crucial for TM, especially with a large ($\sim$100 µm^2) scan area that forces the scan to be far ($\sim$5 µm) from the optical axis, causing significant drift in the direct beam. One can mitigate this with proper instrument setup, but it becomes more difficult with larger scan areas. Instead, the data can be centered on the computer, for example by moving the brightest pixel to the center of each diffraction image. A more robust approach is to calculate the data center of mass (CoM) in each diffraction pattern, and moving the data such that the CoM is in the center. Moving the diffraction patterns will make some of the data outside the borders of the image, and introduce areas without data. A common approach to address this is to ignore the data that gets moved out of the image, and setting the areas without data to zero intensity.

After scaling and centering SPED data, one can perform background removal. One approach is the difference of Gaussians (DoG) method. This is based on blurring the image with two different Gaussians with different sigmas, and comparing the results. Peaks will have a clear difference for the two blurs, indicating that the area in question is not a part of the background. Similarly, for areas that are a part of the background, blurring with different sigmas is unlikely to have a large difference, provided the blurring is not strong enough as to include intensity from the peaks. The formula is

$$\begin{cases} d - g(d, \sigma_{\max}), & g(d; \sigma_{\min}) > g(d; \sigma_{\max}) \text{ and } d > g(d; \sigma_{\max}) \\ 0, & \text{else} \end{cases}, \tag{2.18}$$

where d is the pixelated diffraction data, and $g(x; \sigma)$ is a Gaussian filter, i.e. a convolution of x with a (2D) Gaussian. The conditions are pixel-wise.

A simpler approach for background removal is to threshold the data, i.e. setting all values below a chosen value to zero. Mathematically, this looks like

$$\begin{cases} d, & d > t \\ 0, & \text{else} \end{cases} \tag{2.19}$$

for a threshold t. The condition is pixel-wise.

2.5.2 Crystal Diffraction Simulation

To simulate crystal diffraction, one must define the beam with a given **k**, a crystal with a given lattice and atomic form factors, and Equations 2.10 and 2.14. The reflections are given by the intersections of Ewald's sphere with the reciprocal lattice (Equation 2.10), and the intensities of the reflections are given by the atomic form factors and their positions in the lattice, i.e. the basis of the crystal (Equation 2.14).

The orientation between **k** and the reciprocal lattice is crucial for determining the reflections that contribute to the diffraction pattern. Using rotation matrices (Equations 2.15, 2.16 and 2.17), the transformation between crystal space and reciprocal space, as well as the transformation between Cartesian and crystal space, one can transform a **k** in Cartesian space to its equivalent vector in reciprocal space, and simultaneously rotate it (or, equivalently, rotate the reciprocal lattice) to any orientation.

Note that, due to crystal symmetry, multiple orientations are equivalent, and yield the same diffraction pattern. For example, in a cubic crystal, the beam being parallel to [1 0 0] is equivalent to it being parallel to [0 1 0]. As such, to simulate all unique directions, one needs only consider the symmetry reduced zone of the crystal, as described in the end of Chapter 2.1.3. Depending on the way in which the orientation map is further analyzed, care must be taken when sampling direction space, as the smallest possible misorientation is limited by the angular distance between the sampled directions. If, for example, the aim is to accurately determine misorientations with an angular resolution in the 0.1°-range (which is practically unattainable for normal TM), then a library with angular resolution of e.g. 1° would be useless, as the smallest possible misorientation angle would be 1°.

The simulations produced by the described method are kinematic, as this is the assumption in the physical foundation of the equations. To account for dynamical effects, one may employ the multi-slice solution [55]: a discretized approach simulating diffraction with quantum-mechanical probabilistic propagation of the wave through the sample, available with e.g. `py_multislice`[56]. This is significantly more computationally intensive.

2.5.3 Template Matching

To quantify correlation between two datasets, a common approach is the (zero)normalized cross-correlation (NCC). NCC is commonly used to correlate 2D data, e.g. images. The formula is

$$\mathrm{NCC} = \frac{\sum_i \left(I_{a,i} - \bar{I}_a\right)\left(I_{b,i} - \bar{I}_b\right)}{\sqrt{\sum_j \left(I_{a,j} - \bar{I}_a\right)^2 \sum_k \left(I_{b,k} - \bar{I}_b\right)^2}}, \tag{2.20}$$

where I_a, I_b are the two datasets, i, j and k are indices in the data, and $\bar{I}$ is the mean value of the dataset. The zero-normalized NCC value ranges between -1 and 1.

A typical TM workflow looks as follows, with a visual representation shown in Figure 2.5.1:

Data collection

Collect a diffraction dataset, preferably (S)PED to make the patterns more kinematic-like.

Data Pre-processing

Scale, center, and remove noise in the diffraction data.

Template Simulation

Simulate a diffraction template library, with the same scale, acceleration voltage, and other TEM parameters as used in data collection. The crystal must be defined beforehand, and the crystal directions to simulate for must be chosen.

Template Indexation

Correlate all diffraction measurements with all diffraction templates, using e.g. NCC. The template with the highest correlation score is assumed to be the correct crystal orientation for the given diffraction measurement.

Result verification

Manually check if the results look reasonable, e.g. plot the simulated diffraction template on top of the diffraction measurement or plot the IPF colormap, to verify if the results look reasonable.

2.5.4 The Pyxem Suite

Pyxem is an open-source Python software suite, consisting of four packages: Pyxem [27], Orix [57], Diffsims [58], and Kikuchipy [59]. Kikuchipy is an EBSD-focused package, and is not used for this project.

Orix is an orientation- and crystal symmetry-analysis package. It lets users analyze misorientations between grains, draw plots with the stereographic projection, and visualize orientation maps with IPFs. (Mis)orientations can easily be mapped to the symmetry reduced zone of the given space group, and the symmetry reduced region can be sampled for further use with Diffsims for diffraction simulations. Orix uses quaternions for its internal orientation calculations, but has functionality for Euler angles in the Bunge convention, as described in Chapter 2.4.2. The mean quaternion algorithm from Markley et al. [51] is also implemented.

Diffsims is a specialized electron diffraction simulation package. It lets the user set TEM parameters, such as acceleration voltage, precession angle, and crystal symmetries, and then simulates the diffraction pattern for given crystal orientation(s) as described in Chapter 2.5.2. The simulations are stored in a sparse format, specifying the coordinates and intensities of reflections, rather than the full dense array format that SPED data is captured with. As such, the simulated diffraction spots are pixel-precise points, and are not disks like the actual NBD data. If this is not desired, one can convolve with a Gaussian to give the diffraction spots a more disk-like shape. The benefit of the sparse format, apart from a reduced memory usage, is that it allows for a much more efficient correlation score calculation. The implicit zeros at all the positions that are not specified can be ignored in the correlation score calculation, since they would not contribute to the score. Concretely for NCC (Equation 2.20), the indices i (and k, assuming the simulations are dataset b) are limited to only the coordinates of simulated diffraction spots, even if the diffraction data is non-zero elsewhere. Ignoring overhead, with around 65 simulated diffraction spots and a detector resolution of 256×256 pixels, the sparse format necessitates only $\frac{65}{256\times256} \approx 0.1\%$ of the calculations that a dense format would. The expected number of diffraction spots vary between crystals and camera lengths, but is often in the order of 65 or less.

The Pyxem package is an extension of the Hyperspy [60] package, with specialized data classes for e.g. diffraction data. Pyxem implements the CoM data centering algorithm, and DoG background subtraction with Equation 2.18, which makes it a useful tool for data (pre-)processing. As it extends Hyperspy, the useful plotting functionality is exposed as well, allowing for easy data exploration during the pre-processing process. Additionally, it has implemented Equation 2.20 for template indexing, letting users easily perform ACOM with TM.

When using Pyxem v0.16, which was used for TM in this thesis, the full TM workflow as described in Chapter 2.5.3 would look something like this:

- Load the data with Hyperspy and/or Pyxem
- Pre-process the data with e.g. Pyxem's centering and DoG algorithms
- Define the sample crystal with a .cif-file, and Diffsim's dependency Diffpy
- Use Diffsims to sample the symmetry reduced zone for Euler angles to simulate
- Create a library of diffraction simulations with Diffsims
- Index the pre-processed data with the library with Pyxem's `index_dataset_with_template_rotation`

- Visualize the results as a IPF by exporting the TM results to a Orix `CrystalMap` and calling its `plot` method.

Even though the Pyxem suite has extensive support for orientations through Orix, the results from TM are Euler angles in the Bunge convention, i.e. intrinsic zxz-rotations as mentioned in Chapter 2.4.2. These are easily loaded as Orix's `Orientation`-objects, with the `from_euler` class method.

With the new refactors in Pyxem v0.19, the output and the workflow changes slightly:

- Load the data with Hyperspy and/or Pyxem
- Pre-process the data with e.g. Pyxem's centering and DoG algorithms
- Transform the $\mathbf{k}_x$, $\mathbf{k}_y$ signal dimensions to polar coordinates with `Diffraction2d.get_azimuthal_integral2d`
- Define the sample crystal with Orix's `Phase` class
- Use Orix to sample the symmetry reduced zone for directions to simulate
- Create a library of diffraction simulations with Diffsims
- Index the polar pre-processed data with `PolarDiffraction2d.get_orientation`
- Visualize the results with e.g. `OrientationMap.plot_over_signal`

The grain reconstruction features in Orix is limited to a DBSCAN-based clustering algorithm, and is not strictly a part of the package but rather provided as an example[1] using scikit-learn [61]. A dedicated orientation map-based grain reconstruction algorithm is implemented in the MATLAB-package MTEX [31]. To export orientation maps from Orix to MTEX, one can export to a .ang-file. This format is readable in MTEX as a EBSD orientation map with `EBSD.load`, which allows for grain reconstruction with the method `calcGrains`. When translating back and fourth between Orix and MTEX, one must pay attention to the coordinate system definitions. For example, where Orix stores orientations as lab-to-crystal, MTEX stores them as crystal-to-lab. The 'lab', in both cases, is not the L frame as defined in Chapter 2.4.4, but rather the S frame.

2.5.5 Template Matching Optimizations

In Pyxem, the TM algorithm contains more steps than the general algorithm outlined in Chapter 2.5.3, to increase the speed of orientation mapping. This comes in addition to hardware- and data structure optimizations, e.g. using a GPU and sparse simulations. The optimization assumes that the user only wants a few (e.g. < 50) best matches. With this assumption, if one can rule out certain orientations or simulations before applying NCC, then one should see a speedup in execution if the process of ruling out is faster than the NCC. In Pyxem, this is performed by a one-dimensional correlation of simulations and diffraction patterns, before the full

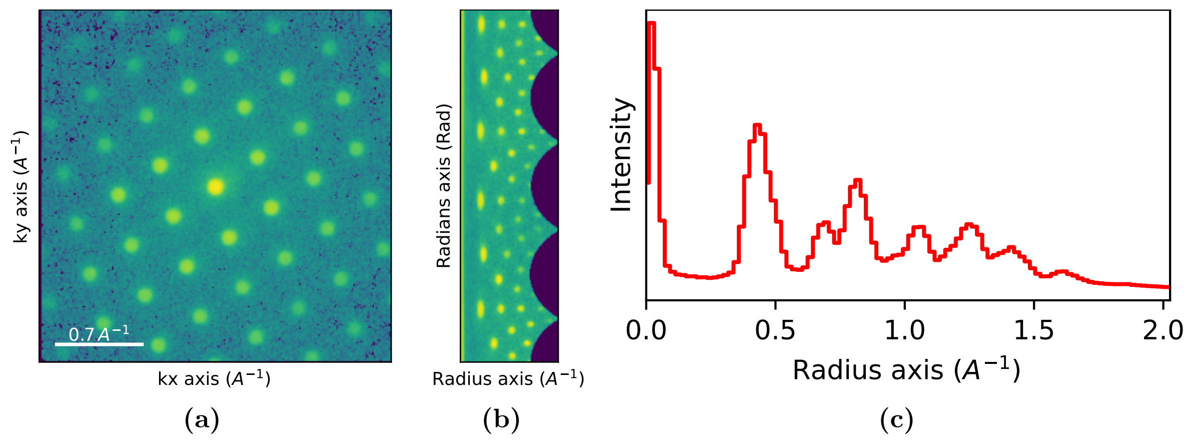


Figure 2.5.2: Diffraction pattern processing for template matching correlation. (a) the original diffraction pattern. (b) the diffraction pattern transformed to polar coordinates, to ease in-plane rotation correlation. (c) sum of the polar diffraction pattern along the polar axis, for use in Pyxem's pre-selection step.

two-dimensional correlation. The algorithm in Pyxem is first described in [29], and seemingly independently in [62].

First, the diffraction patterns and simulations are converted from Cartesian coordinates k_x, k_y to polar coordinates r, ϕ. This is shown schematically in Figure 2.5.2(a) and 2.5.2(b). To avoid confusion, ϕ is used here for the angular dimension, as the more common θ is already used in Equation 2.3. θ can be measured along r when combined with the camera length.

Next, the patterns and simulations are summed along the ϕ axis, resulting in a radial profile like the one in Figure 2.5.2(c). This profile is then correlated with the simulations, which is very fast.

When all simulations have been correlated in this polar sum form, certain simulations are discarded for full correlation. These are selected by the user, as either a fraction or an integer number of simulations. The simulations with the lowest correlation scores are discarded accordingly. Note that, in the current implementation, this pre-selection is performed regardless of whether the user wants to discard anything, which adds a slight overhead.

With this pre-selection, one can discard e.g. 90% of the simulations, avoiding the need to perform full 2D correlation on patterns which do not fit even in one dimension. Since the processing to perform pre-selection is run regardless of the specified discarding strategy, any discarding should yield a reduction in runtime.

[1] `https://orix.readthedocs.io/en/stable/tutorials/clustering_orientations.html`

CHAPTER

THREE

METHOD

In this chapter, the methodology for both data collection and processing is presented. Both samples are subjects of other theses, namely the preliminary work[45] and Aune's Master's thesis[63], so the section regarding those is brief. Next, the structure, function and development of `tiltlib` is presented. Finally, an algorithm for increased performance for TM is presented.

3.1 Sample Overview

Two samples were used in this project: polycrystalline Ag ($Fm\bar{3}m$) and lithium manganese nickel oxide (LMNO or $LiMn_{1.5}Ni_{0.5}O_4$) ($Fd\bar{3}m$). The Ag sample was used as part of the preliminary work[45], and the LMNO sample is the subject of Aune's Master's thesis[63]. These samples and TEM data acquisition will be briefly presented here. For more details, refer to the respective theses.

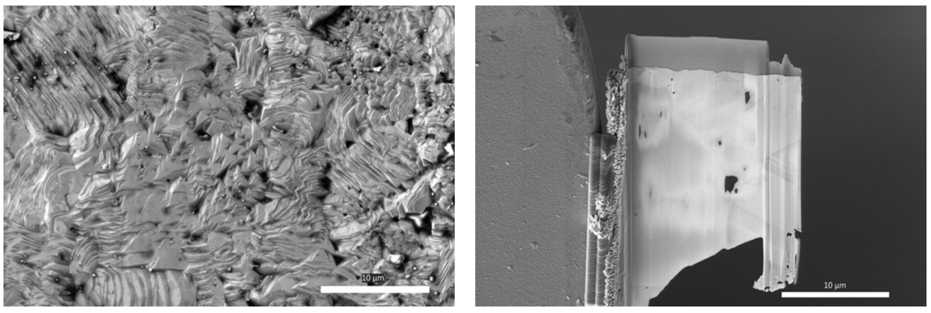
3.1.1 The Samples

The Ag sample was prepared by Tina Bergh on a FEI Helios G4 UX focused ion beam (FIB). Overview scanning electron microscope (SEM) images of the sample before and after TEM preparation is shown in Figure 3.1.1. The preparation entailed laying a capping layer of amorphous C, before milling, extracting and thinning the lamella. The LMNO sample was prepared similarly by Ruben Bjørge.

3.1.2 TEM Data Collection

All data of Ag was collected on a JEOL JEM 2100F TEM, using a Quantum Detectors Merlin 256×256 Medipix DED. Either SED or SPED was used. The collection was performed by Tina Bergh and Emil Christiansen. The parameters of the scans are summarized in Table 3.1.1. The sample holder was a tilt-rotate holder with ±30° range in the Gx tilt.

Three TEMs were used for the LMNO sample: a JEOL JEM 2100F, a JEOL JEM 2100, and a JEOL JEM ARM200F. SPED was used in the 2100F, and NBD, SAED, and HRTEM was used in the other two. The SPED scans were used for orientation mapping, whereas the others were used for verification of tilt angles. Collection was performed by Ton van Helvoort, Ruben Bjørge, and Inger-Emma



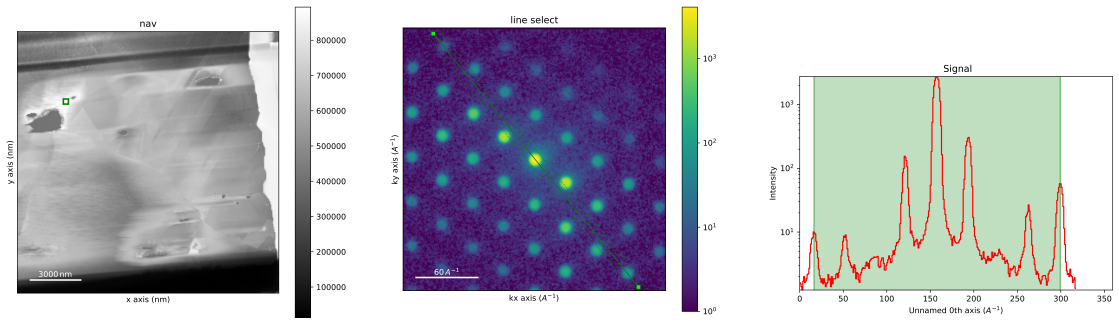
(a) (b)

Figure 3.1.1: SEM images taken during the Ag sample preparation, by Tina Bergh. (a) The sample surface, before the capping layer was deposited. (b) The lamella after being thinned. The C capping layer is visible on the top of the lamella. Scalebars are 10 µm.

Nylund. Five SPED scans of LMNO were used in this thesis, and some of the parameters are given in Table 3.1.2. The sample holder was a double-tilt holder, both axes with a range of ±30°. As mentioned, more details can be found in Aune's thesis[63].

3.1.3 Ag SPED Data Processing

Data processing was performed using Pyxem version 0.16. A more complete list of software versions can be found in Chapter 3.6. The data was centered using `ElectronDiffraction2D.center_of_mass`, and the reciprocal scale was found to be 0.0113 Å^{-1} using the method described in Chapter 2.5.1. The process is shown in Figure 3.1.2.



(a) (b) (c)

Figure 3.1.2: Data scaling on Ag was performed by navigating to a [1 0 1] zone axis in the green square in (a), followed by identifying (3 3 5) and ($\bar{3}\,\bar{3}\,\bar{5}$) in (b), before measuring the distance between them in (c). In this case, the scale was found to be 0.0113 Å^{-1} per pixel. Figure is taken from [45].

The tilt series data was pre-processed before running TM. After testing multiple different parameters and possible steps, the following steps were chosen, as the resulting orientation maps seemed homogeneous within each grain. The steps were:

Table 3.1.1: An overview of the scan parameters of each TEM scan for the Ag sample. (a) Parameters that differ for the different scans. (b) Parameters which are the same for all scans. Rocking angle and frequency is for precession, t is exposure time, XYZ are the coordinates of the sample holder, TX is the tilt angle, and TY is the sample holder rotation angle. E is the beam energy, the mode is nano-beam diffraction (NBD), and α is the convergence angle of the beam. Cl is the camera length, nx/ny is the detector size in pixels, dx/dy is the distance between probe positions, and Θ is the scan rotation. The first block of rows is SPED and SED, and the second is a tilt series.

a

Scan	Rocking angle	Rocking frequency	t	X	Y	Z	TX	TY
[°]	[Hz]	[ms]	[µm]	[µm]	[µm]	[°]	[°]	
1	1.0	100	10	180.5	55.9	-63.5	-8.5	-61.0
2	0.0	0	1	180.5	55.9	-63.5	-8.5	-61.0
3	1.0	100	10	-56.4	121.0	-63.5	0.0	-0.1
4	1.0	100	10	-56.4	121.0	-63.5	5.0	-0.1
5	1.0	100	10	-56.4	121.0	-63.5	10.1	-0.1
6	1.0	100	10	-56.4	121.0	-63.5	15.0	-0.1

b

E	Mode	Spot	α	Cl	nx	ny	dx	dy	Θ
[keV]		[nm]	[°]	[cm]	[pixels]	[pixels]	[nm]	[nm]	[°]
200	NBD	1.000	5	10	256	256	61.075	61.075	0

1. Mask out the direct beam
2. Perform DoG background subtraction (Equation 2.18) with $\sigma_{\text{min}} = 4.68$ and $\sigma_{\text{max}} = 10.58$
3. Perform thresholding with $t = 1$ (Equation 2.19)
4. Blur with a Gaussian with $\sigma = 2$
5. Threshold the data with $t = 2$
6. Change all zeros to -20

Orientation mapping was performed using Pyxem's TM functionality. Since this was performed in October 2023 as part of the preliminary work[45], older software versions were used for creating the orientation maps. In Pyxem version 0.19, the application-programmer interface (API) will change significantly compared to 0.16, which was used here.

The diffraction library parameters were set to the same values found in Table 3.1.1, and the `minimum_intensity` was set to 10^{-6}. Precession was approximated with a Lorentzian, as described in [64], by setting `approximate_precession=True`.

Table 3.1.2: An overview of the scan parameters of each SPED scan for the LMNO sample. TX and TY are the tilt angles of Gx and Gy, respectively. E is the beam energy, and Θ is the scan rotation. More details are available in [63].

Scan	1	2	3	4	5
TX [°]	0	5	10	15	0
TY [°]	0	0	0	0	5
E [keV]	200	200	200	200	200
Θ [°]	28	28	28	28	28
nx [pixels]	273	273	273	273	273
ny [pixels]	511	511	511	511	511

The orientation space was sampled according to the two Euler angle method described in Chapter 2.4.3. This resolution and sampling resulted in 11476 unique patterns. A .cif-file for crystalline Ag was used for specifying the phase, from Suh et al. [65].

Template indexation was performed with `index_dataset_with_template_rotation` from `pyxem.utils.indexation_utils`, on the GPU.

To verify the precision, twin misorientation relations were used. As the FCC Ag crystals exhibit Σ3 twinning along $\langle 1\,1\,1 \rangle$, with a known misorientation angle of 60°, these can be used to quantify the precision of TM. A crop of the scans containing twins were extracted from the orientation maps. Two orientation populations were identified by thresholding the misorientation angle at 5° and 55°, and those in-between were considered misindexed. The misorientation angle between the means of the populations were calculated, and compared with the expected 60°.

3.1.4 LMNO SPED Data Processing

A similar workflow to the Ag scans was used to create the initial orientation maps for LMNO. These were subsequently used in a refinement step, where the orientations at each navigation position was compared in a tilt series. By using multiple `n_best` orientations and comparing across the tilt series, the most likely candidates were chosen for each navigation position. This aims to combat speckling, and is an important part of Aune's thesis [63].

3.2 Tilt Axis Identification

To predict how tilting affects the crystal, the relationship between the G frame and the S frame must be established, as discussed in Chapter 2.4.4. To this aim, a fitness function was used, and subsequently minimized as a function of tilt axis position. The fitness function used all possible combinations of the scans in the tilt series, and all pixels in each scan was compared to the corresponding pixels in the other scans. Three tilt series were used in this project: scans 3-6 inclusive of the Ag sample for Gx, scans 1-4 inclusive of the LMNO sample for Gx, and scans 1 and 5 of the LMNO sample for Gy. All scans were virtually tilted back to 0° before comparing. No attempt was made to account for the change in pixel position by physical shift of the sample during tilting.

For the Ag sample, the total dataset for optimization contained $6 \times 256 \times 256 = 393,216$ misorientation angles. As the LMNO sample had more probe positions, the datasets consisted of $6 \times 511 \times 273 = 837,018$ and $511 \times 273 = 139,503$ for the Gx and Gy axes, respectively. From these, the lowest third misorientation angles were selected, and their mean was the final output of the fitness function. By using only the small angles, the effect from grain displacement by tilting is reduced. As the overlaps between grains between scans are likely to have a large misorientation angle, these will be ignored by only considering the bottom third. Similarly, regions with noise, non-crystalline regions, or other sources of less meaningful TM results can be less impactful on the fitness score.

The fitness function has only one input: the angle between Sx and the tilt axis, in the Sxy-plane. The tilt axis is therefore assumed to lie in the Sxy plane, as described in Chapter 2.4.4.

Minimization was performed using Scipy's `minimize`-function [66], as well as a visual inspection of a graph of the fitness function. The complete fitness function code is listed in Listing 3.2.1.

Verification of the position was performed by using a different scan rotation Θ for the Ag and LMNO scans. As the SPED scans were performed on the same microscope, the tilt axis should lie in the same place in the L frame, but offset in the S frame by the difference in scan rotation ($0° - 28° = -28°$).

A Jupyter notebook outlining the entire process of tilt axis identification can be found in the GitHub repository for the project[1], under the `examples`-folder. This notebook uses a fabricated tilt series as an example, and was used as a baseline for tilt axis identification in this thesis.

3.3 `tiltlib`

With the geometry of the TEM identified, and using the sample-crystal relation found by TM, a program was developed to predict sample holder tilt angles which would align a region of the sample to a given zone axis. The structure and implementation of `tiltlib` is presented in this section.

3.3.1 Design

The code was written as a standalone Python module, dependent on, but not a part of, Orix and Hyperspy. A schematic overview of the structure of `tiltlib` is shown in Figure 3.3.1. The intended workflow for a user of the program is to first create a `CrystalMap` from Orix, and determine the tilt axis position of the TEM. Then, the user would create a `Sample`-object from those. Grain selection, or region selection in general, would be performed using one of Hyperspy's ROIs. The user would then create a `Miller`-object from Orix, containing the desired $[u\,v\,w]$ zone axis (or $[u\,v\,t\,w]$ for hexagonal crystals), and supply it to the `Sample.find_tilt_angles`-function. The result of this function is a tuple of tilt angles, corresponding to the

[1]`https://www.github.com/viljarjf/tiltlib`

```python
import tiltlib

import numpy as np
from itertools import combinations

x = orix.Vector3d.xvector()
y = orix.Vector3d.yvector()
z = orix.Vector3d.zvector()

xmaps: list[orix.CrystalMap]  # TM results
tilt_angles: list[float]  # original tilt angle of each sample


def fitness_function(axis_angle: float):
    axis_direction = x.rotate(z, np.deg2rad(axis_angle))

    samples: list[tiltlib.Sample] = []
    for xmap, angle in zip(xmaps, tilt_angles):
        axis = tiltlib.Axis(
            axis_direction, min=-30, max=30, angle=angle, degrees=True
        )
        s = tiltlib.Sample(xmap, [axis])
        s.rotate_to(0)
        samples.append(s)

    misorientation_angles = []

    for sample_a, sample_b in combinations(samples, 2):
        oris_a = sample_a.orientations
        oris_b = sample_b.orientations
        angs = oris_a.angle_with(oris_b, degrees=True)
        misorientation_angles.append(angs)
    misorientation_angles = np.array(misorientation_angles).flatten()

    k = misorientation_angles.size // 3
    lowest_k = np.partition(misorientation_angles, k, axis=None)[:k]
    return np.mean(lowest_k)
```

Listing 3.2.1: The fitness function used to find the tilt axis position, by minimizing the fitness function.

tilt axes, which minimizes the misorientation angle(s) between the desired zone axis and the predicted zone axis of the sample.

Imprecision is inherent to multiple steps in the determination of tilt angles, and as such cannot be expected to be perfect. In practice, the angles are therefore unlikely to perfectly align the crystal. Imprecision of TM contributes with around 1°[39]. As ALPHABETA suggests 1° to 2° precision[32], and have more precisely determined orientations (±0.25°), `tiltlib` should predict angles with around 5° precision. The aim is to be close enough to see the Laue circle, and align more precisely using that. This will be evaluated, and is described in Chapter 3.4.

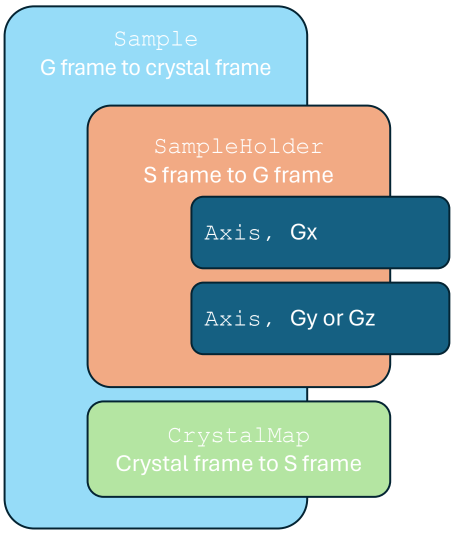


Figure 3.3.1: An overview of the structure of `tiltlib`. The `SampleHolder`-class takes a list of `Axis` as input, which handle the rotations around tilt axes of the sample holder. A `Sample` is an extension of a `SampleHolder`, which additionally takes in a `CrystalMap` from Orix that defines the sample-crystal orientations.

3.3.2 Implementation

As new code and functionality is developed within this work, below the implementation structure is explained in detail.

Apart from general project setup files, such as the `pyproject.toml`-file defining the module, the code is organized into three separate folders. The `examples`-folder contains Jupyter notebooks, which serve as rudimentary tutorials for how to identify the tilt axis position from a tilt series, and how to predict tilt angles for zone axis alignment. These also serve as integration tests. The `tests`-folder contains unit tests. Finally, the `src`-folder contains the source code for the project.

The source code consists of three files, each containing a single class. These are: the `Axis`-class, the `SampleHolder`-class, and the `Sample`-class.

The `Axis`-class is a simple data container, which handles the position, tilt, and range of tilt axes. It is initialized with the direction it is pointing, in the sample reference frame, the range of tilt angles it is capable of, and its current tilt position, if applicable. Additionally, one can specify whether the supplied angles were given in degrees or radians, and whether the axis is intrinsic or extrinsic.

The `SampleHolder`-class is responsible for calculating the `Rotation`-object transforming the scan (S) frame to the gonio (G) frame, as defined in Chapter 2.4.4. It is initialized with a list of `Axis`-objects, which define the tilt axes of the sample holder. The order of the list matters in two ways: the first `Axis` in the list is extrinsic, regardless of what was specified in its creation, and the list of angles returned by member functions of both this and the `Sample`-class correspond to the axes in that order.

Two important member functions of the `SampleHolder` class is `rotate` and `rotate_to`. They both take angles as input, which can be specified as either degrees or radians, and compute internally the `Rotation`-object corresponding to a rotation of each tilt axis by the given angle. The difference between the two functions lies in whether they rotate additively, i.e. the given angles are added to the current position, or if the given angles are the target rotations for the sample holder. `rotate` works additively, whereas `rotate_to` rotates to the given angles regardless of the current ones. Note that, in both cases, the resulting position cannot be outside the range of any of the tilt axes, and the amount of angles supplied to the function must correspond to the amount of tilt axes. The class also exposes some helpful functions for converting a vector back and forth between the sample and TEM reference frame, and a rotation matrix representation of the rotation.

Finally, the `Sample`-class handles the change in orientation from tilting the sample. It inherits from the `SampleHolder`-class, meaning all functionality of the parent class is available in the child as well. A `Sample`-object is initialized with a `CrystalMap` from Orix, and either a list of `Axis` or a `SampleHolder`. The pixel-wise `Orientations` are available with the `Sample.orientations` member, which are automatically kept up to date when tilting. This is one of the main functionalities of the class, and is how the updated orientations should be accessed. Plotting functionality is also available, e.g. as an interactive window allowing the user to change the tilt angle(s) in real time and see the effect on the IPFs as shown in Figure 3.3.2. The `to_navigator` member function returns a IPF-z color map of the sample as a `Signal2D` object from Hyperspy, intended for use as a navigator when plotting. A new `Sample`-object is returned when a Hyperspy ROI is supplied to the `crop` member function, containing only the orientations within the region of interest. Note that plotting a `Sample` cropped with a `CircleROI` is unlikely to be helpful, as the data is flattened to a 1D array.

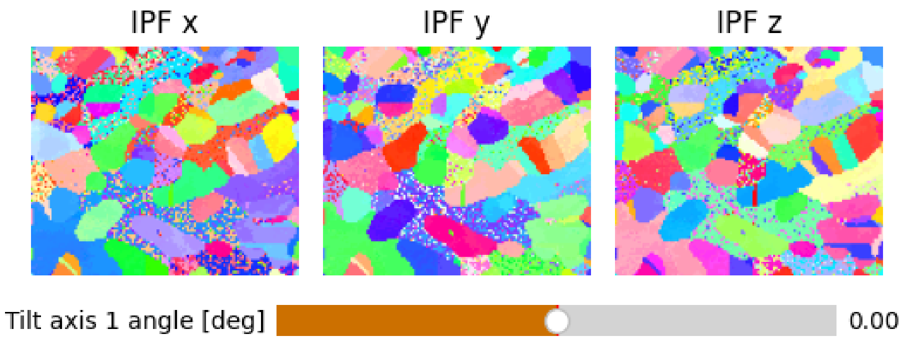


Figure 3.3.2: Example of an interactive plot of a sample using `tiltlib`. The tilt position is controlled with the slider, updating the colors of the IPFs. Data from an Austenite EBSD orientation map supplied by Orix, from `orix.data.sdss_austenite`.

The most important functionality of the `Sample` class lies in the `find_tilt_angles` member function. A listing of the function can be found in Listing 3.3.1. The function takes a `Miller` object from Orix, which defines a desired zone axis, and returns the best tilt angles to align the sample to that zone axis. The algorithm is based on optimization, similar to the tilt axis identification algorithm seen in Listing 3.2.1, rather than analytical determination of optimal tilt angles. First, the optical axis is expressed, here as a `Miller` object, which is [0 0 1]. Next, the function to be optimized is defined. Its inputs are tilt angle(s), and the output needs to attain its minimum at the best tilt angles. For the output, `Sample.angle_with` is used, which calculates the angle between the optical axis of each pixel and the given zone axis. As the optimization needs to output only a single number, the mean of this set of angles is used. The tilt angles are used by rotating the sample before calling `angle_with`. The function to be optimized is similar to the tilt axis identification function, but rather than discarding a fraction of the angles, the mean of *all* of the angles is calculated. Optimization is performed using Scipy[66], and the result is returned as a tuple of floats in the same order as the tilt axes used of the `Sample`.

```python
def find_tilt_angles(
    self, zone_axis: Miller, degrees: bool = True
) -> tuple[float, ...]:
    """Calculate the tilt angle(s) necessary to align the sample
    with a given optical axis

    Args:
        zone_axis (Miller): desired zone axis
    Returns:
        tuple[float, ...]: Tilt angles for each axis
    """

    def optimize(angles) -> float:
        self.rotate_to(*angles, degrees=degrees)
        aw = self.angle_with(zone_axis, degrees=degrees)
        return np.mean(aw)

    bounds = [(ax.min, ax.max) for ax in self.axes]
    angles = self.angles
    if degrees:
        bounds = np.rad2deg(bounds)
        angles = np.rad2deg(angles)

    res = minimize(
        optimize,
        angles,
        bounds=bounds,
        method="Nelder-Mead",
    )
    self.reset_rotation()
    return res.x
```

Listing 3.3.1: The member function of the `Sample` class responsible for calculating the optimal tilt angle(s) for aligning the sample to a zone axis.

3.3.3 Code Testing

All parts of the code were tested individually, by comparing the output of functions to the expected output. For rotations, simple 90° rotations around coordinate axes in the coordinate system(s) were chosen, to ensure the expected output could be determined manually, by e.g. physically rotating a Rubik's cube around the given axes. Combinations of extrinsic and intrinsic chained rotations, handling of angles and radians, properly transforming between coordinate systems and properly resetting rotations were tested. Extrinsic and intrinsic rotations were thoroughly tested by generating thousands of random chained rotation objects using `tiltlib` and asserting that they all compare equal to a `Rotation` object from Scipy with the same settings. To ensure correctness of the rotation matrix, random tilt angles were supplied to Orix, Scipy, `tiltlib`, and the analytical expression found by Qing et al. [9]. These all compared equal for each set of random tilt angles.

Unit tests were written alongside the code, ensuring that at each step of development the code gave the expected output, and that any changes to the code left the outputs unchanged. The tests were automatically run whenever the code was changed on GitHub, using the testing library Pytest and GitHub Actions.

Integration testing was performed by running the example Jupyter notebooks. These were written at the end of development, and run manually as opposed to the automatic unit tests.

3.4 Zone Axis Alignment

The two samples were both used for zone axis alignment. As the Ag sample was only scanned in one session, verification was performed by aligning to a zone found at a different tilt. For the LMNO sample, the predictions were verified by re-inserting the sample in the TEM, and setting the tilt angles to the predicted values.

3.4.1 Ag Sample

Zone axis alignment with real data was performed with the example notebooks as a baseline, first using crystal maps from the Ag tilt series. Tilt axis identification was performed with the `tilt_axis_identification.ipynb` notebook as a baseline, but using the tilt series crystal maps rather than the fabricated tilt series in the example file. The identified axis was subsequently used in the `zone_axis_alignment.ipynb`-notebook to test the `find_tilt_angles` function. The axis position was verified by selecting a grain from two different scans, and finding the tilt angles to align one grain with the other. The expected output would then be the tilt angle at which the other scan was taken at. Additionally, the zone axes for grains which were not used for alignment should still align with the corresponding zone axes in the other scan.

3.4.2 LMNO Sample

A more robust verification test was performed with the LMNO sample, as the predicted tilt angles were verified in two different microscopes. The tilt axes of the sample were identified similarly as the Ag sample, using the two tilt series for Gx and Gy. The `CrystalMap` from the scan at TX = TY = 0°, as well as the tilt axis positions found using the tilt series, was used to initialize a `Sample`-object from `tiltlib`. Tilt angles for zone axis alignment were predicted using this `Sample`.

Three regions of interest were identified from the orientation map. Area 1 is a $\Sigma 3$ twin, and the target zone was [1 1 0]. Area 2 had two grains, with targets [3 2 3] and $[0\,\bar{1}\,3]$. Area 3 had [1 1 2] set as target. The naming of the areas align with Aune [63]. Predicted tilt angles were noted, before the sample was re-inserted into the TEM. Two microscopes, the JEOL 2100 and the JEOL ARM200F, were used to test zone axis alignment. Testing was performed by rotating the sample holder to the predicted angles, before using the Laue circle for final correction. The angles at the optimal manual alignment were noted. In the JEOL 2100, SAED patterns were taken at default, predicted, and final tilt angles.

To ensure the tilt axes found by the tilt series would be valid for the other microscopes, the sample was inserted into the sample holder at the same orientation as in the 2100F. This preserves the sample orientation in the G frame, but as the alignment is performed manually it introduces a possible misorientation. Photographs of the process is shown in Figure 3.4.1.

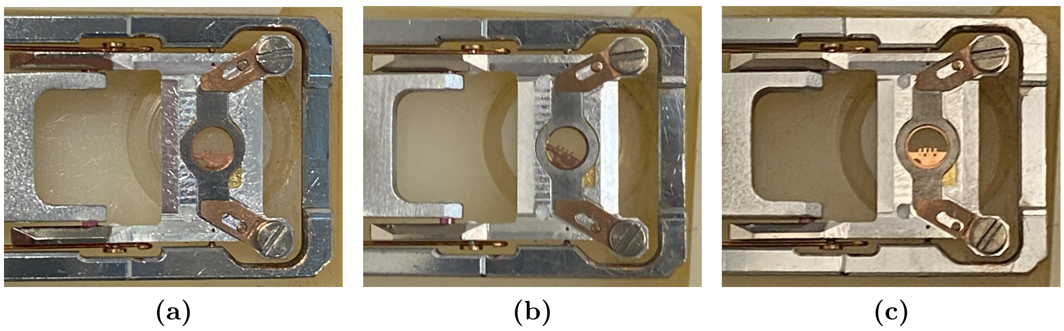


Figure 3.4.1: Photographs of the sample holder with the LMNO sample inserted. The alignment of the sample in the holder needed to be identical as with the 2100F, to ensure the predicted tilt angles were valid. (a) 2100F. (b) 2100, incorrectly aligned. (c) 2100, correctly aligned. Images taken by Kaja Eggen Aune.

Note that the tilt angles used on the microscopes was predicted by visual inspection of the SP IPF, using the program Recipro rather than `tiltlib`. The predictions align with `tiltlib`. This was done as `tiltlib` was in an unfinished state at this point. The measurements of the final tilt angles are unaffected by this.

3.5 Template Matching Pre-selection Algorithm

In order to improve runtime of TM, an algorithm to select which templates to use was developed, similar to the one used in Pyxem and outlined in Chapter 2.5.5. Rather than reducing the dimensionality of the problem, this new approach relies on the continuity of correlation scores in orientation space. By sampling the correlation scores of orientations and interpolating between them, one can discard orientations near regions with low correlation score. The aim of the method is to give the same answer as a single TM sweep with a dense bank of simulations, but in a much shorter time. In this section, the developed algorithm will be detailed, along with some specific theory beyond the general theory given in Chapter 2.

3.5.1 Algorithm Outline and Implementation

This method consists of three steps:

1. Full TM with a rough orientation grid
2. Interpolate correlation scores
3. Refined TM with promising orientations

An outline of the workflow for a single diffraction pattern is shown schematically in Figure 3.5.1.

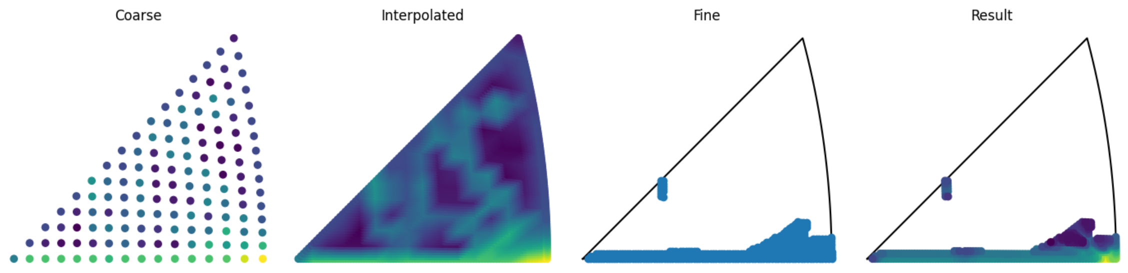


Figure 3.5.1: Outline of the pre-selection step, for a single diffraction pattern. A coarsely sampled template bank is used for TM, and the correlation scores are interpolated. The best 500 interpolated scores were chosen in this case, and used for a second TM run.

The biggest difference between this and other TM techniques, is the fact that the set of simulations in the second sweep is different for each diffraction pattern. This stems from the correlation values for each of the coarsely sampled simulations being slightly different for each diffraction pattern, consequently making the interpolated correlation scores different, which in turn makes the set of simulations for the second sweep different. If a 300×300 probe position scan is used, and the user wants to use the best 500 templates from interpolation, then this would require $300 \times 300 \times 500 = 45$ million templates. To simulate all of these would take considerable time, and would be mostly unnecessary as the vast majority would likely be duplicates. Therefore, the implementation only simulates the unique orientations, and distributes the simulations with clever indexing as shown later in this section. This implementation also allows for the use of a different, possibly more accurate and computationally demanding, simulation generator.

Depending on the orientation resolution for the initial TM sweep, and the amount of orientations requested for refinement, the interpolation step can take up considerable time. Initial testing revealed that a naive implementation, using Scipy's `interpolate.griddata`, spends half the runtime on interpolating the correlation scores. By restricting ourselves to a linear interpolation, and by observing that both the data coordinates and the resampled coordinates are the same for all probe positions, then we can optimize this step considerably.

The interpolation algorithm works as follows:

1. Calculate the Delaunay triangulation of the coordinates for the initial TM sweep

2. Identify which triangle each new sample point resides in

3. Calculate the Barycentric coordinates for all new sample points in their respective triangles

4. When interpolating: Sum the product of correlation scores and the Barycentric weights

This way, only the final step needs to be performed for all different sets of correlation scores. All other steps need only be run once, saving significant overhead in the interpolation step.

```python
# Based on https://stackoverflow.com/a/20930910
from scipy.spatial import Delaunay
import numpy as np

# Known correlation scores' coordinates:
xy: np.ndarray[n_points, 2]

# Points to interpolate the correlation score
new_xy: np.ndarray[n_new_points, 2]

# Perform triangulation
tri = Delaunay(xy, incremental=False)

# Find the vertices for all new points
simplex = tri.find_simplex(new_xy)
vertices = np.take(tri.simplices, simplex, axis=0)

# Calculate the Barycentric coordinates
temp = np.take(tri.transform, simplex, axis=0)
delta = new_xy - temp[:, -1]
bary = np.einsum("njk,nk->nj", temp[:, :-1, :], delta)

# Use the Barycentric coordinates as weights for linear interpolation
weights = np.hstack((bary, 1 - bary.sum(axis=1, keepdims=True)))


# Actual interpolation function.
# NOTE: The values to interpolate must be ordered in the same way as `xy`
def interpolate(values: np.ndarray[n_points]) -> np.ndarray[n_new_points]:
    return np.einsum("nj,nj->n", np.take(values, vertices), weights)
```

Listing 3.5.1: The implementation of an efficient multilinear interpolation using Delaunay triangulation and Barycentric coordinates, used for interpolating correlation scores. Note that the type annotations are for clarity, and will throw an error if run as-is.

Without going into too much detail, Delaunay triangulation is a tessellation of the convex hull of a set of points, using triangles that maximize the smallest angle in all the triangles [67]. Barycentric coordinates are essentially a decomposition of a linear interpolation on a polygon, quantifying the contribution from each vertex to a given interior point [68].

The implementation used in the project can be found in Listing 3.5.1, which adapts Jaime [69]. What is not shown is the calculation of the coordinates `xy` and `new_xy`. For this, the rotations are projected onto the optical (z) axis, before the stereographic projection is used to convert to planar xy-coordinates used in `xy`.

The interpolated correlation scores can then be calculated for each diffraction pattern, using the first TM sweep, and the orientations at the top correlation scores are used for the second TM sweep. The implementation details regarding the calculation of indices in the simulation bank can be seen in Listing 3.5.2.

```
1  oris: orix.Orientation  # Resampled orientations
2  simgen: diffsims.SimulationGenerator
3  orientationmap: pyxem.signals.OrientationMap
4  num_oris: int  # amount of orientations to keep for final TM
5
6
7  def get_top_indices(orientationmap_result):
8      indices, correlations, _, _ = orientationmap_result.T
9      indices = indices.astype(int)
10
11     # Re-order correlations, they need to be in the same order
12     # as the Delaunay triangulation was made with
13     sorted_indices = np.argsort(indices)
14     data = interpolate(correlations[sorted_indices])
15
16     # Only keep the top `num_oris`.
17     # Use argpartition, since we don’t care about the order
18     # (arg since we want indices, partition since we dont care about order)
19     k = data.size - num_oris
20     inds = np.argpartition(data, k)[k:]
21     return inds
22
23
24 indices_signal = orientationmap.map(
25     get_top_indices, inplace=False, lazy_output=False
26 )
27
28 # Reduce total simulation count by only using the unique orientations
29 unique_inds, reconstruct_inds = np.unique(indices_signal, return_inverse=True)
30 indices = reconstruct_inds.reshape(indices_signal.data.shape)
31
32 # Perform the simulations, only use unique orientations
33 simulations = simgen.calculate_ed_data(
34     orientationmap.simulation.phases,
35     oris[unique_inds],
36 )
```

Listing 3.5.2: The calculation of indices corresponding to the orientations chosen from interpolating correlation scores. The `interpolate`-function is the one defined in Listing 3.5.1. Note that only the unique orientations are used for simulations, handled in lines 29 and 35.

The result is a numpy array `indices`, containing indices into the simulation array. The indices are ordered such that, for each diffraction pattern index, there are `num_oris` indices corresponding to this amount of simulations with the best interpolated correlation scores. As such, accessing the simulations with `simulations[indices]` would result in an array of simulations with shape (n_y, n_x, `num_oris`), where each diffraction pattern index only has the `num_oris` most promising simulations. For the example with 300×300 probe positions and `num_oris = 500`, this is an array of 45 million simulations, but the amount of simulations performed is at most `oris.size`.

When performing TM for the second time (step 3), each probe position effectively has a unique simulation library associated with it. As the size of these is controlled by the user specifying `num_oris`, the specialized simulation libraries can be much smaller than the full orientation space, which should dramatically reduce runtime compared with using a full set of simulations. This is where runtime can be saved, given that the initial TM step (step 1) and interpolation (step 2) are fast enough.

3.5.2 Performance Testing

To evaluate the performance of the new algorithm, both it, the full dense correlation, and the current pre-selection algorithm was run. The runtime was measured for multiple runs of the same parameter set, and the resulting orientation maps were kept for similarity comparisons. Three different metrics were used to measure the similarity of the orientation maps:

Correlation score similarity The Euclidean norm of the difference in correlation scores with the dense orientation map, for each navigation position.

Orientation similarity The angle between each orientation and the corresponding index (x, y, `n_best`) in the dense orientation map, measured in degrees.

Equality length The fraction of solutions present in an orientation map, which are also present in the dense orientation map. Measured per navigation position.

The implementation of these algorithms is shown in the `TemplateMatchingComparison`-class in Appendix A.4. For each of these metrics, the mean and standard deviation was calculated for the entire dataset. These results, along with extensive metadata regarding the test parameters, were stored as `.json`-files, which were subsequently used for further analysis. An example of a comparison file can be found in Appendix A.1.

Parameter sweeps for the angular resolution of orientations of the first TM sweep (`coarse_resolution`), the resolution for the second sweep (`fine_resolution`), and the number of orientations to keep for the second sweep (`n_keep`, or `frac_keep`) were performed, measuring the runtime and similarity scores for both the new and old pre-selection algorithms. Tests were run on four datasets, one experimental and three simulated from different point groups. The experimental dataset was a crop of Scan 1 of the Ag sample, without background subtraction, whereas the simulated datasets were from the three phases: copper ($Fm\bar{3}m$), graphite ($P6_3/mmc$), and AgAuTe$_4$ ($P2/c$). These phases were chosen as they are the archetypes of their respective space groups [70, p. 124-127]. The point groups were chosen to evaluate the performance for a variety of crystals and sizes of the symmetry reduced zone. `.cif`-files used to simulate the three phases were found on `crystallography.net`, using Swanson and Tatge [71], Hassel [72], and Pertlik [73], respectively.

The runtime and similarities of the results were measured by performing TM on the four datasets, and subsequently compared. Runtime was measured for 6 iterations of each method, to increase statistical significance. The similarities were calculated using the previously discussed metrics. TM was performed with `n_best` $= 10$ for all methods and all datasets.

For scan 1, the sweep of the `n_keep`-parameter was performed with 2° resolution in the coarse simulation bank (300 simulations) and 0.4° resolution in the fine bank (6555 simulations). The sweep of the coarse resolution parameter had a resolution of the fine simulation bank was 0.5° (4186 simulations) and `n_keep` $= 400$. Finally, the sweep of the fine resolution parameter had a coarse simulation bank of 2° resolution (300 simulations) and `n_keep` = 400.

The same resolutions and `n_keep` values were used for the simulated datasets as well, but due to the difference in size of the symmetry reduced zones for the different point groups, the amount of simulations were different. For $AuAgTe_4$, 2° resolution corresponds to 3687 simulations, 0.5° to 58,065 and 0.4° to 90,631. For graphite, 2° corresponds to 900 simulations, 0.5° to 13,456 and 0.4° to 21,025. Copper has the same point group as Ag, $m\bar{3}m$, and as such the same numbers of simulations are used for TM on the simulated copper dataset as for scan 1.

The simulated databanks were made with the same parameters as the simulations used for TM, apart from the precession. For both, the precession angle was 1°. Differently to where TM was performed with an analytical Lorentzian approximation[64], the simulated datasets were performed with numerical integration of a full revolution. The resolution of the $AuAgTe_4$ bank was 3°, corresponding to 1743 simulations. For graphite, the resolution was 2°, which was 1770 simulations. Finally, for copper, 1° i.e. 1081 simulations was used. The crop of scan 1 was taken of a region containing multiple grains, and every fourth diffraction pattern was chosen, resulting in a total dataset of 720 diffraction patterns.

3.6 Computing Resources

All data processing was run on a Windows 10 computer, with 80 GB of RAM, and a 12-core AMD Ryzen 3900x CPU. TM was run on a Nvidia GeForce RTX 3080 GPU, with 10 GB of VRAM, which sped up the TM process significantly compared to using CPU. The software versions are listed in Table 3.6.1. Developer versions are listed, as these were versions which received contributions as part of this thesis. Specifics regarding open-source contributions during this thesis are listed in Appendix C.

Table 3.6.1: List of software versions used for this project. Dev-versions of packages have their Git commit SHA-1 specified.

Name	**Version**	**Git commit SHA-1**
Python	3.10.12	
TM		
Pyxem	0.17.dev0	`48c83c65acb52aa8ff2f2ea8c28306399ae1cff1`
Orix	0.11.1	
Diffsims	0.5.2	
Hyperspy	2.0rc0	`2c841044a6ac232edc23f73d44bdb66bacfd9ea1`
Analysis		
Pyxem	0.19.dev0	`c52505f49349058ad3bce38bc82f42729923a3e1`
Orix	0.12.1	
Diffsims	0.6rc1	`870d6323ea940e9639655f2b8ed2f89feded0582`
Hyperspy	2.1	
`tiltlib`	0.0.5	

CHAPTER

FOUR

RESULTS

In this chapter, the results of orientation mapping, tilt axis identification, zone axis alignment, and testing the pre-selection algorithm are shown. First, the samples are presented, and the data acquisition and processing is outlined. Next, results from zone axis alignment are presented, before the final section shows results from the TM pre-selection algorithm.

4.1 Ag Sample

The Ag sample was scanned using both SED and SPED, and subsequently orientation mapped using TM. Next, the tilt series was used to identify the tilt axis position of the TEM. This chapter presents an overview of the sample, an analysis of the precision, and tilt axis identification.

4.1.1 Overview

A SPED scan of the Ag sample, scan 1, can be found in Figure 4.1.1. Some points are indicated, notably Figure 4.1.1(a) showing the [1 0 1]-orientation used for calibration as mentioned in Chapter 3.1.3. The other three patterns are notable as they do not contain the reflections expected of diffraction patterns of Ag, e.g. Figure 4.1.1(d) being too thick. Most of the scan shows crystalline diffraction. Additionally, the region of Σ3 twins used for TM precision estimation is shown in the black dashed box.

4.1.2 Twin Grains

Σ3 twin grains with known theoretical misorientation angle were used to quantify the precision of TM. An overview of the process carried out for a single scan, scan 4, is shown in Figure 4.1.2. A crop of the scans containing only twins was used, where two populations o1 and o2 were identified by thresholding at 5° and 55°. The mean orientations of these populations were calculated for each scan, and the misorientation angles between these means were calculated. A table of these population misorientation angles can be seen in Table 4.1.1.

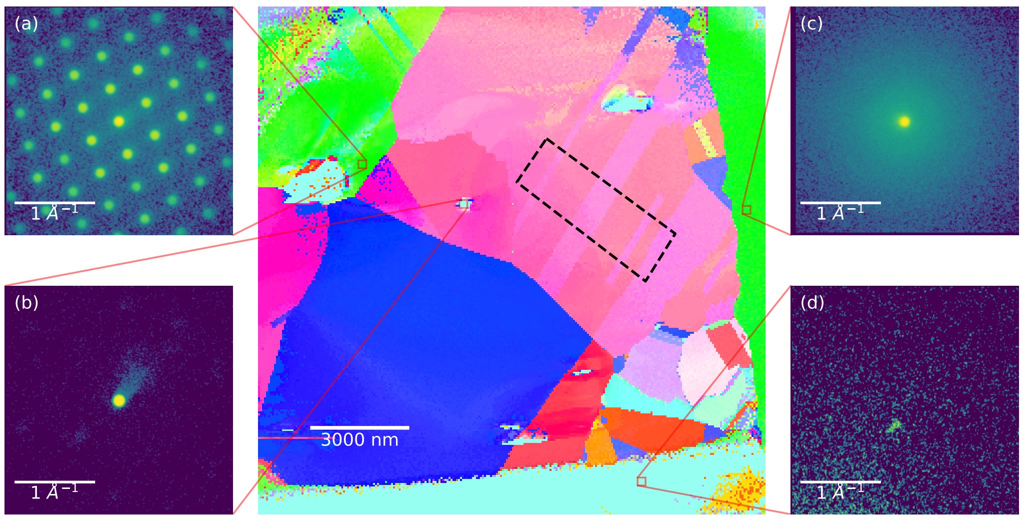


Figure 4.1.1: An overview of the sample, showing a IPF-z from scan 1 with insets of different diffraction patterns. (a) [1 0 1] diffraction pattern used as a reference for scaling, as mentioned in Chapter 3.1.3. (b) The direct beam through a hole in the sample. (c) Amorphous capping layer diffraction pattern. (d) A region with too large thickness for transmission. The black dashed box contains twins, later used for TM precision evaluation.

Figure 4.1.2 shows the general case, scan 4, where the deviation from the expected misorientation is 2.15°. The histogram and IPFs show three distinct populations, two of which where considered for calculations. This is not the case for all scans, as some scans had only two distinct populations present. Similar plots as was made for scan 4 in Figure 4.1.2 is shown for the two extrema, i.e. scans 5 and 6, in Appendix B.1.

Table 4.1.1: Deviation from the expected misorientation angle (60°) between Σ3 twins in the Ag sample.

Scan	1	2	3	4	5	6
Angle [°]	2.79	4.29	3.50	2.15	11.06	0.97

4.1.3 Tilt Axis Identification

To identify the tilt axis in the JEOL JEM 2100F, the workflow outlined in Chapter 3.2 was performed. The result of the optimization indicated a tilt axis position in the xy-plane, 30° counter-clockwise from the x-axis. A fitness score of 1.4 was the minimum. As can be seen in Listing 3.2.1, the score is the mean of the bottom third of misorientation angles, i.e. over 130 thousand angles.

To ensure the optimization resulted in the global minima, a sweep of the fitness function over tilt angles was performed. The results are plotted in Figure 4.1.3(a), and indicate a well-behaved fitness function. The six different combinations of scans, and the pixel-wise misorientation angle between them, are plotted for the optimal tilt axis position in Figure 4.1.3(c).

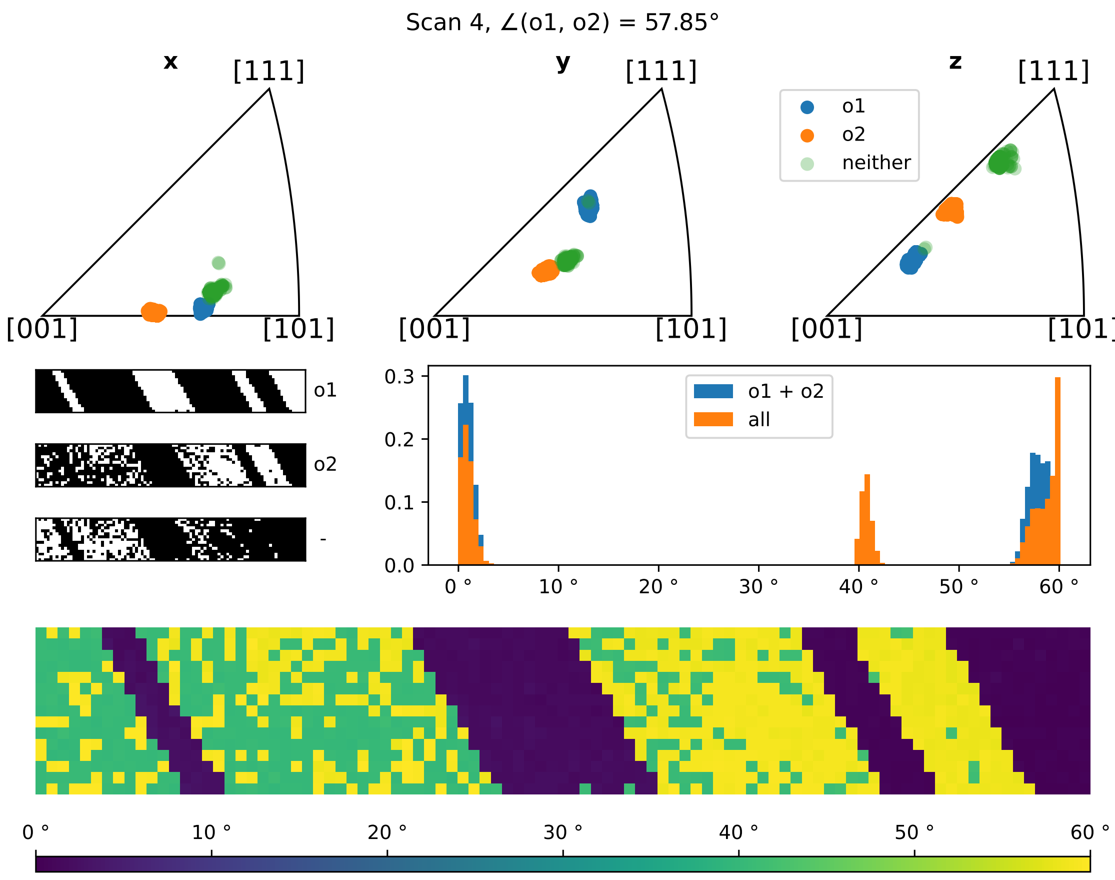


Figure 4.1.2: Twin misorientation of scan 4. The top shows IPFs of the orientations, where the two populations are indicated by color. The left side shows the masks of where the orientations corresponding to the two populations are. A histogram of the misorientation angle of all orientations with all other orientations, both with and without the discarded orientations, is shown in the center right. The misorientation angle is shown along the bottom. As the title indicates, the misorientation angle between the mean orientations of the two populations is 57.85°.

Histograms for the around 390 thousand misorientation angles can be seen for a tilt axis position of 30° and 100° in Figure 4.1.3(b). The threshold between kept and discarded data is marked with blue and red dashed vertical lines. Both the variance and the mean is clearly lower for the tilt axis position of 30° compared to 100°.

With the tilt axis identified, as well as the sample being orientation mapped, the full orientation relation between the sample holder and the crystals in the sample is known. This is subsequently used in `tiltlib` to predict tilt angles for zone axis alignment in the next section.

4.2 Zone Axis Alignment

The zone axis alignment functionality of `tiltlib` was tested in two ways: first, on the Ag sample used for the tilt series, and next on the LMNO sample. Only the latter was verified by physical testing in the TEM.

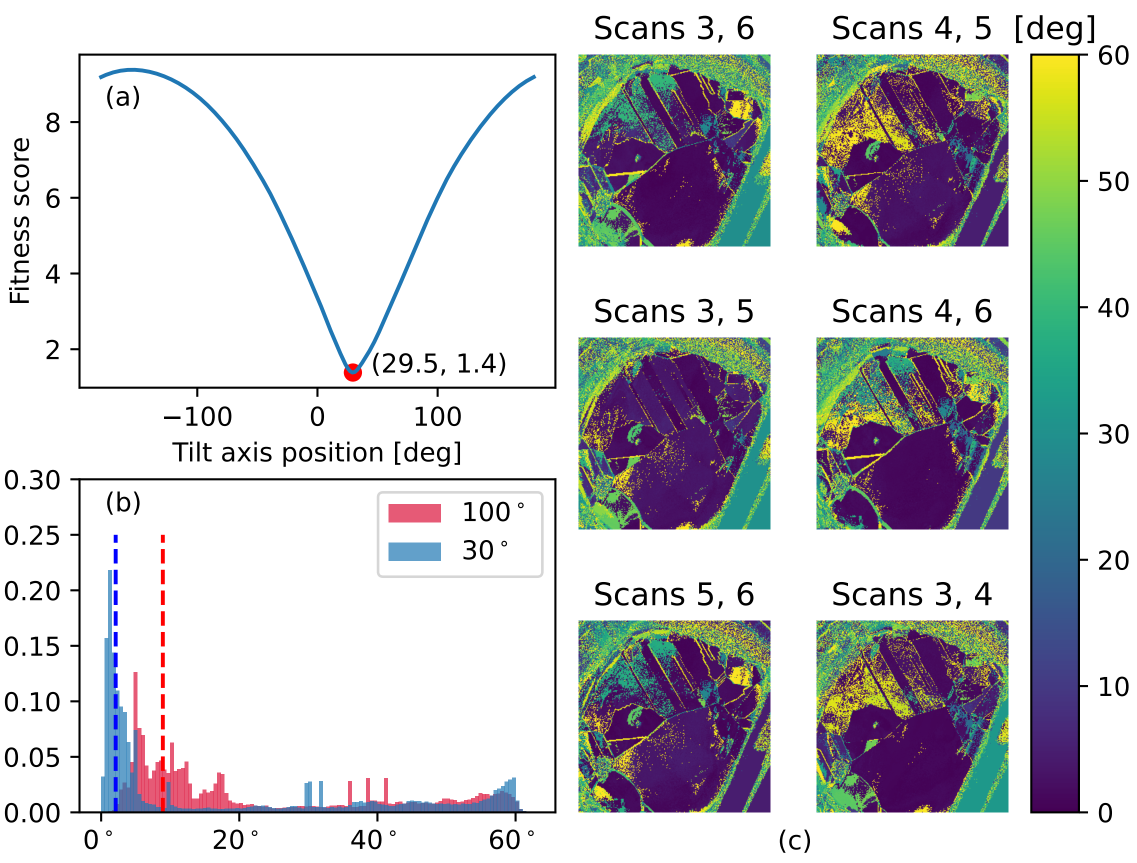


Figure 4.1.3: Tilt axis identification on Ag. The tilt axis lies 29.5° from the x-axis, as seen in (a). (a) The optimization landscape for all axis positions in the xy plane, measured with an angle from the x-axis. (b) Histograms of the full misorientation angle dataset, for two different tilt axis positions; 30° (the optimum) and 100°. The cutoff value of $\frac{1}{3}$ is indicated with corresponding vertical dashed lines. (c) The six misorientation angle sets between scans.

4.2.1 Ag Sample

With the position of the tilt axis determined, the zone axis alignment procedure was tested. First, a virtual test was performed, where one of the scans in the tilt series was aligned with the zone axis of the same grain in a different scan. The expected output would then be the tilt angle of the other scan, and the initial scan would align well with the target scan for all grains. The test is virtual in the sense that both the initial and the target scans were made before attempting to align either to a certain zone axis.

The results are shown in Figure 4.2.1, where the green rectangle was used both for finding the mean zone axis target, and for optimization of tilt angle. After tilting to the predicted 16.1°, which is only 1.1° from the expected 15°, the scans seem to align well both qualitatively from looking at the colors, and from the mean zone axes in the black rectangles.

4.2.2 LMNO Sample

Tilt axis identification using the tilt series of LMNO indicated an x-tilt axis rotated 0.6° from the x-axis, and a y-tilt axis $-90°$ from the x-axis. These were used in the

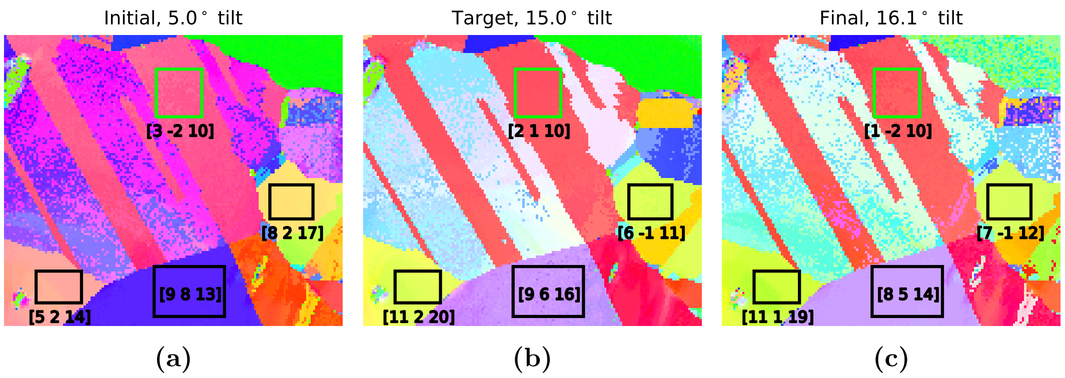


Figure 4.2.1: Zone axis alignment performed on a cropped region of scan 4 (a), using the green rectangle for optimization. The target was set to the mean zone axis of the same region (green rectangle) in scan 6 (b). (c) scan 4 rotated to the angle found by the zone axis optimization, which was 16.1°. A selection of regions and their mean zone axes are also annotated. All images are IPF-z.

`Sample`-object used for zone axis alignment. An overview of the orientation map, and the three chosen areas, is shown in Figure 4.2.2. Tilt angles predicted with `tiltlib`, and the manually corrected tilt angles, are tabulated in Table 4.2.1.

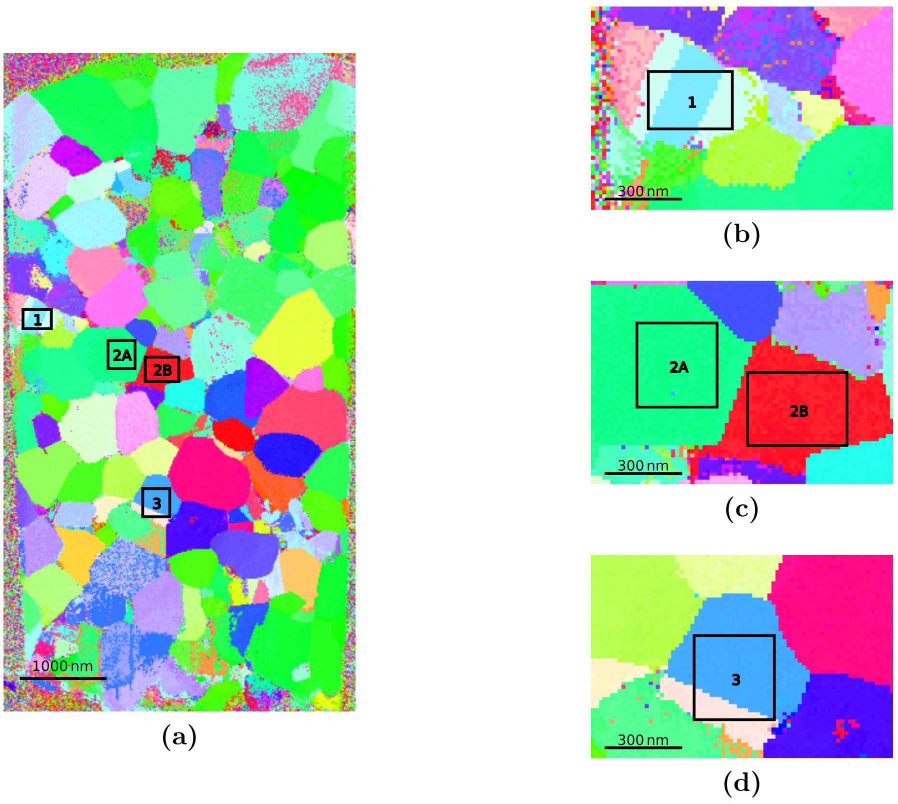
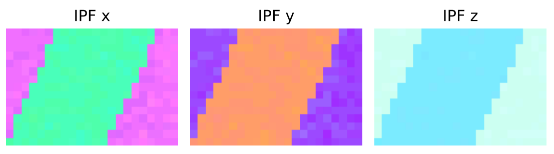


Figure 4.2.2: Overview of the LMNO orientation map used for zone axis alignment, shown as z-IPFs. (a) Full orientation map, with regions of interest marked and annotated. (b), (c), (d) Zoom-ins of the three regions. The lower regions of (a) show some speckling in the crystals. The multi-colored strip along the border is likely not crystalline. The scalebar in (a) is 1000 nm, the rest are 300 nm.

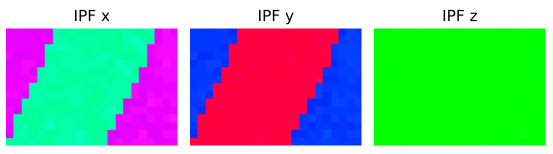
Table 4.2.1: Predicted and found angles to align grains to zone axes. The actual tilt angles were measured with two microscopes: the JEOL ARM200F and the JEOL 2100. Note the two different sets of angles predicted for area 2B, corresponding to different initial tilts for optimization.

Area	**Target zone**	**Description**	**Tilt angles [$x°$, $y°$]**	
Area 1	[1 1 0]	Predicted	17.0°,	−4.49°
		Actual, ARM	17°,	−7.1°
		Actual, 2100	16.4°,	−8.9°
Area 2A	[3 2 3]	Predicted	−10.6°,	12.3°
		Actual, ARM	−9.3°,	12.0°
		Actual, 2100	−8.3°,	12.9°
Area 2B	[0 $\bar{1}$ 3]	Predicted	13.9°,	1.00°
		Predicted	−9.37°,	12.9°
		Actual, ARM	−7.8°,	13.3°
		Actual, 2100	−6.6°,	13.8°
Area 3	[1 1 2]	Predicted	3.76°,	−12.9°
		Actual, ARM	3.2°,	−13.8°
		Actual, 2100	1.4°,	−14.0°

The crops which were used for predicting the tilt angles in each area are shown in Figure 4.2.3, 4.2.4, 4.2.5, and 4.2.6, for areas 1, 2A, 2B, and 3, respectively.



(a)



(b)

Figure 4.2.3: LMNO orientation map, Area 1 IPFs. (a) before tilting to predicted angles for zone [1 1 0]. (b) after tilting to the predicted angles.

Area 1 is a region with a twin, and the crop includes orientations from both grains. Tilt angles to align both to [1 1 0] were successfully found, as indicated by the homogeneous green color in the IPF-z in Figure 4.2.3(b).

Area 2 was aligned to [3 2 3] and [0 $\bar{1}$ 3], for 2A and 2B, respectively. As the two grains were aligned to different zones, the optimization was performed separately for each grain. The first attempt yielded different tilt angles for the two grains, but different initial parameters for optimization gave more similar results. The two different tilt angle sets found for 2B both align the crystal with the target zone axis, as can be seen in Figure 4.2.5(b) and 4.2.5(c).

For area 3, the alignment with [1 1 2] was performed successfully, as indicated by the homogeneous purple color in the IPF-z in Figure 4.2.6(b), corresponding to [1 1 2] in the IPF colormap for m$\bar{3}$m. Interestingly, the two other directions also show homogeneity in the color, implying the twins are aligned in all three directions. By extracting the mean orientation from each grain in area 3 after alignment to

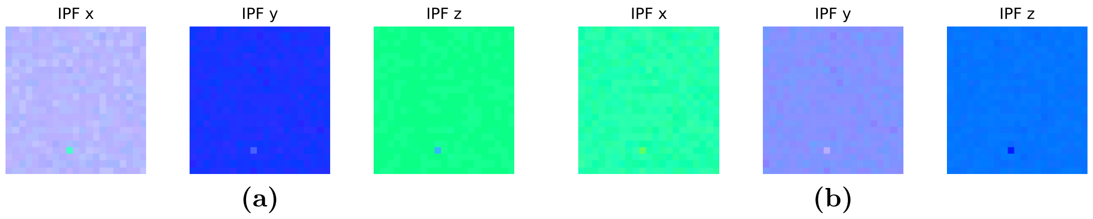


Figure 4.2.4: LMNO orientation map, Area 2A IPFs. (a) before tilting to predicted angles for zone [3 2 3]. (b) after tilting to the predicted angles.

[1 1 2], the misorientation angle between them was calculated to 59.67°, even though the IPFs show them as being identical.

The grains are fairly homogeneous, as can be seen by the even coloring in the different IPF colormaps in Figures 4.2.3 - 4.2.6. Therefore, the SP IPFs are also plotted, shown in Figure 4.2.7 for all areas. This makes nuances in color more easy to differentiate, e.g. the vibrant red and blue of the IPF-y in Figure 4.2.3(b) seems closely aligned to [1 0 0] and [1 1 1] by only looking at the color. Figure 4.2.7(a) shows this alignment is a couple degrees off.

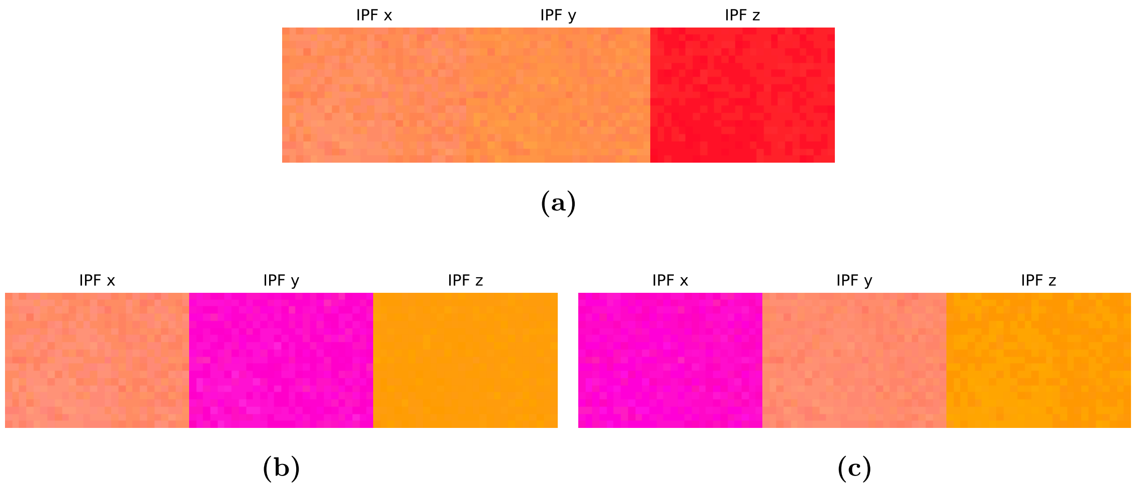


Figure 4.2.5: LMNO orientation map, Area 2B IPFs. (a) before tilting to predicted angles for zone [0 $\bar{1}$ 3]. (b), (c) after tilting to the two sets of predicted angles.

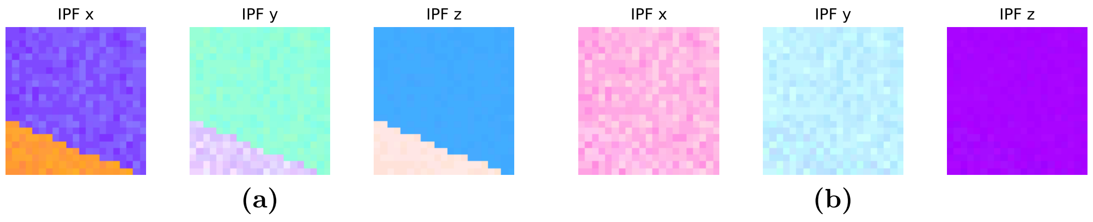
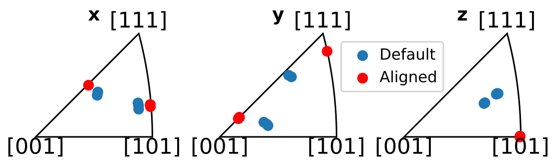
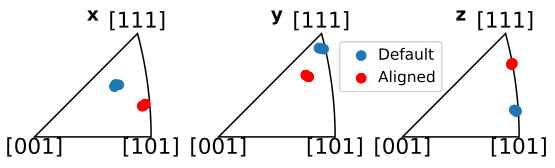


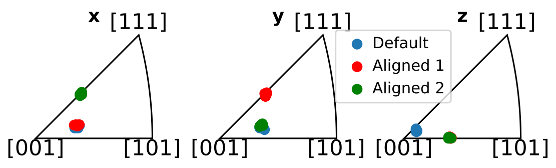
Figure 4.2.6: LMNO orientation map, Area 3 IPFs. (a) before tilting to predicted angles for zone [1 1 2]. (b) after tilting to the predicted angles.



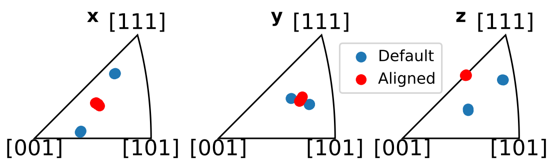
(a)



(b)



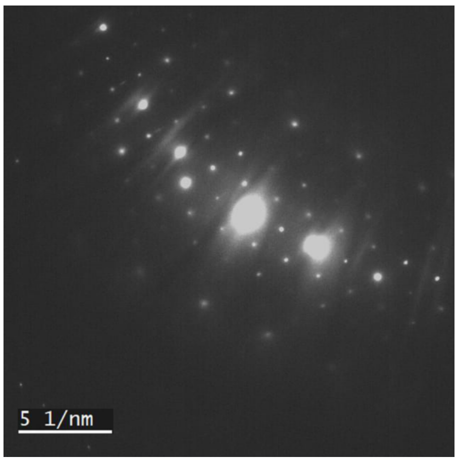
(c)



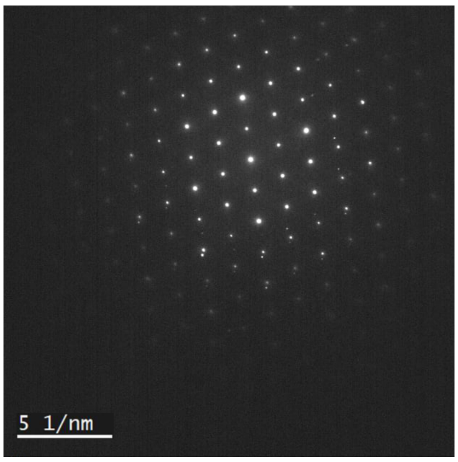
(d)

Figure 4.2.7: SP IPF of the areas in the LMNO sample. (a) Area 1. (b) Area 2A. (c) Area 2B. (d) Area 3.

Figure 4.2.8 shows SAED diffraction patterns of area 1 from the JEOL 2100 TEM. In Figure 4.2.8(a), the Laue circle is clearly visible, as a ring of greater intensity centered outside the image to the top right. This was used for the final manual alignment, by tilting in such a way as to move the center of the Laue circle to coincide with the direct beam. As is shown in Table 4.2.1, this difference is around 2° to 5°.



(a)



(b)

Figure 4.2.8: SAED of area 1, for predicted tilt angles (a), and actual tilt angles (b). Note the clear Laue circle in (a), centered near the top-right corner, easily allowing the operator to correct the final alignment.

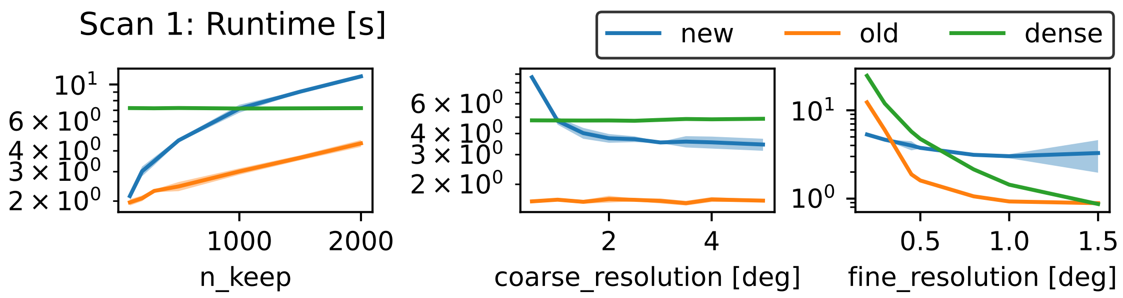
4.3 Template Matching Pre-selection Algorithm

The new pre-selection algorithm was tested against the existing (old) algorithm on four different datasets: a crop of scan 1 from the Ag sample, and simulated datasets for copper, graphite and $AuAgTe_4$. The results are shown in Figure 4.3.1, 4.3.2, 4.3.3 and 4.3.4, respectively. Each figure shows the runtime and correlation score similarity, plotted as a function of the `n_keep`-parameter, the angular resolution of the coarse template bank, and the angular resolution of the fine template bank. The remaining similarity scores, i.e. the orientation similarity and the equality length, can be found in Appendix B.2. The other scores show qualitatively the same trends as the correlation score similarity.

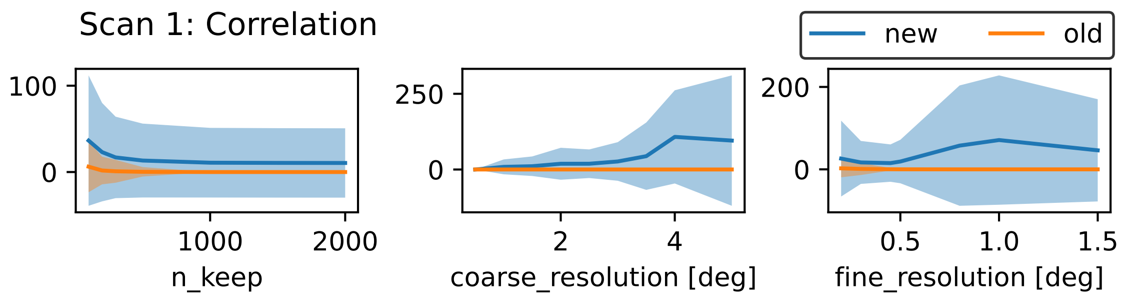
The runtime of the dense algorithm is constant for sweeps of `n_keep` and coarse resolution. The coarse resolution does not affect the old algorithm either, but changes with `n_keep`. Correlation scores generally converge quickly for the old algorithm, and both the variance and mean of the new algorithm stays at a higher level throughout. The exception is $AuAgTe_4$, as Figure 4.3.4(b) shows, where the new algorithm seems to outperform the old in terms of correlation score. Regarding runtime, the new algorithm only consistently outperforms the old for graphite, as Figure 4.3.3(a) shows. However, the corresponding correlation similarity scores for graphite are much worse for the new versus the old algorithm.

In total, each parameter sweep took between 30 minutes for $m\bar{3}m$ to around 18 hours for $2/m$. With limited computational resources, finer step sizes and combined parameter sweeps was down-prioritized in favor of testing more point groups.

An attempt was made to achieve a situation where the new algorithm comprehensively outperformed the old. A similar strategy to the tilt axis identification and zone axis alignment was performed, where a score function was minimized using Scipy[66]. This was performed on the experimental dataset of scan 1. The scoring function, in this case, was a weighted mean of the three similarity metrics, as well as the total runtime. Both the runtime, orientation similarity and correlation similarity attain their optimal values at 0, so they were used as-is. The final similarity score, the equality length, is optimal at a value of 1, so the absolute difference of equality length with 1 was used. The weighting prioritized runtime, and the remaining scores were weighted roughly equally. Only a roughly equal weighting was used, since the scores are not normalized. The tests on the experimental dataset did not converge within 24 hours, and the attempt was abandoned.

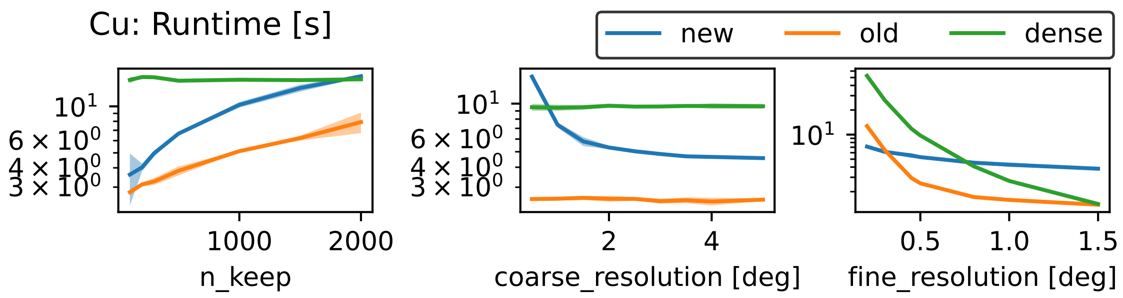


(a)

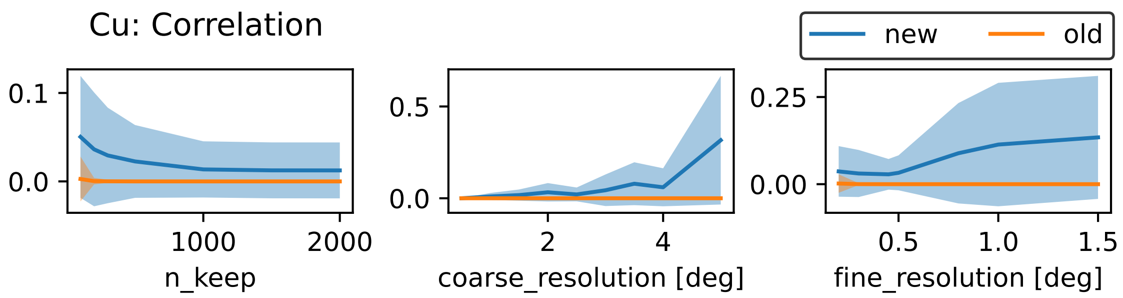


(b)

Figure 4.3.1: Performance evaluation of the new and old TM pre-selection algorithms, tested on scan 1. (a) Mean runtime for 6 runs, with standard deviation, plotted for the parameter sweeps. (b) The correlation score similarity, with standard deviation, for the parameter sweeps.

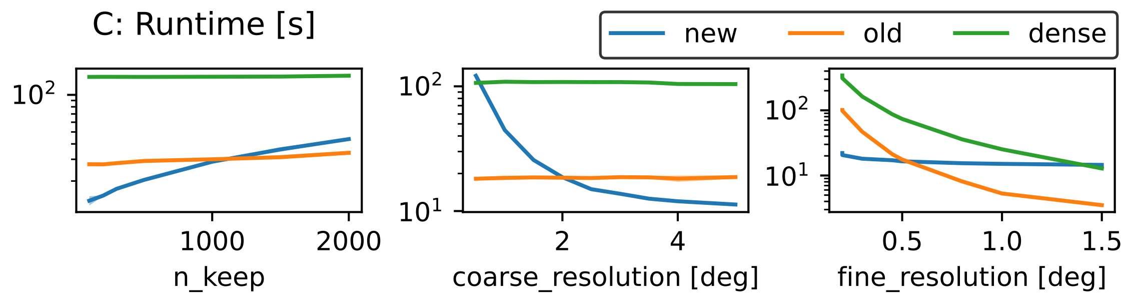


(a)

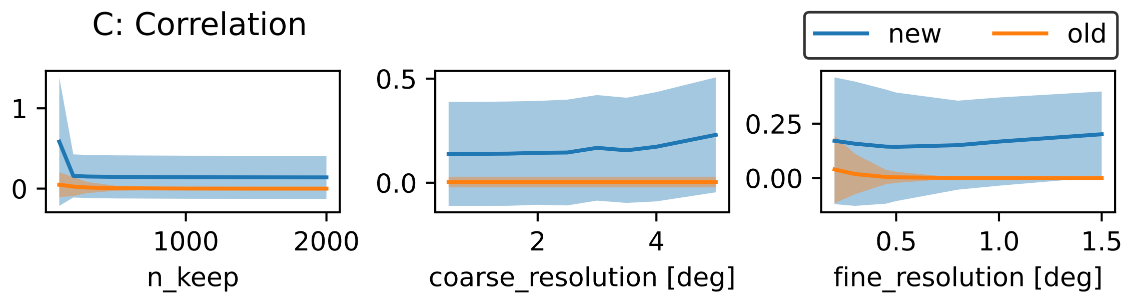


(b)

Figure 4.3.2: Performance evaluation of the new and old TM pre-selection algorithms, tested on simulated data from copper. (a) Mean runtime for 6 runs, with standard deviation, plotted for the parameter sweeps. (b) The correlation score similarity, with standard deviation, for the parameter sweeps.

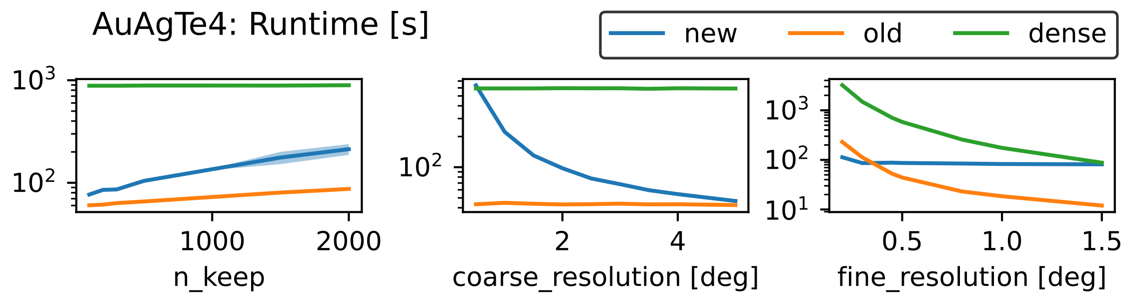


(a)

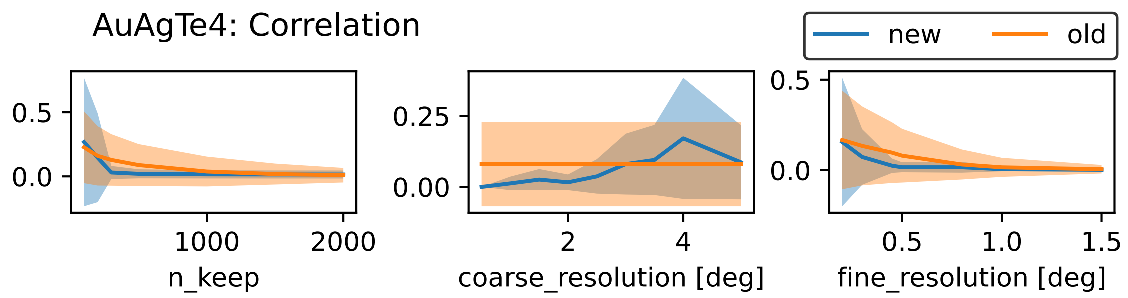


(b)

Figure 4.3.3: Performance evaluation of the new and old TM pre-selection algorithms, tested on simulated data for graphite. (a) Mean runtime for 6 runs, with standard deviation, plotted for the parameter sweeps. (b) The correlation score similarity, with standard deviation, for the parameter sweeps.



(a)



(b)

Figure 4.3.4: Performance evaluation of the new and old TM pre-selection algorithms, tested on simulated data for $AuAgTe_4$. (a) Mean runtime for 6 runs, with standard deviation, plotted for the parameter sweeps. (b) The correlation score similarity, with standard deviation, for the parameter sweeps.

CHAPTER

FIVE

DISCUSSION

In this chapter, the methods and results are discussed. First, the structure of `tiltlib` is discussed, before following the same structure as the results chapter. Possible explanations for the observations are brought up, as well as ways to potentially mitigate the issues.

5.1 `tiltlib`'s Design

The aim of the navigator was to extend the usability of orientation maps from TM given a complete description of the TEM reference frames as outlined in Chapter 2.4.4. In broad terms, this entailed expressing the tilt axis rotations and crystal orientations in the same reference frame. This is conceptually simple, but requires careful considerations in implementation. When writing `tiltlib`, decisions were made regarding the design and implementation of the code. These will be discussed in this section.

5.1.1 Project Structure

As discussed in Chapter 3.3.1, the code was written as a standalone Python module. This makes the code more portable, and makes the lead time of new features shorter than being a part of a bigger framework, e.g. Pyxem. Instead of taking weeks or months to get a new feature out, typical for new versions of Pyxem, a new version could be released with new features whenever it was finished. Such a development cycle puts increased requirements on automatic testing, as extensive user testing for quality assurance takes time. This is useful and practical for established software, where most or many features are already available and thoroughly tested, but impractical for initial development.

The project being a separate module rather than a part of e.g. Pyxem was also chosen due to the lack of an obvious place where it fits in. The Pyxem module is more centered on analysis of data, rather than data acquisition and instrument control. Diffsims and Kikuchipy are focused elsewhere as well, namely diffraction simulation and EBSD analysis. `tiltlib` might fit as a part of Orix, which is focused on plotting and crystallography, especially since it is the only dependency of `tiltlib` from the Pyxem suite. However, the developed tool is very focused on practical S(P)ED rather than general crystallography, which is the focus of Orix, which is in turn heavily inspired by MTEX. As such, even though `tiltlib` could fit as a submodule of Orix, the decision was made to keep it as an independent project. The ownership of the repository on GitHub could instead be transferred to Pyxem, incorporating it into the Pyxem suite. This would depend on if the quality of the project is good enough, and if it is deemed useful and general enough for a wider audience. If the project is not transferred to Pyxem, it can be transferred to the TEM Gemini Centre at NTNU, which was the initial scope of the userbase. The project will remain open-source, and hopefully have users in a broader community.

`tiltlib` is an extension of the work by Mathisen [49], which uses Pyxem orientation maps and grains identified by MTEX[31] to predict tilt angles for zone axis alignment. The extensions include e.g. support for spatially distinct orientations in an orientation map rather than MTEX grains, plotting functionality, and an improved tilt axis calibration method. He discussed whether to add his code to Orix in an issue on GitHub[1], but no conclusion was reached and the discussion went stale. The Orix developers seemed positive to an inclusion of a tutorial for zone axis alignment based on Mathisen's work, indicating an inclusion of `tiltlib` in Orix or the Pyxem suite might be accepted as well.

The folder structure is quite standard, used for several Python projects, and is used in the official tutorial for creating Python modules[74]. The Pyxem suite is set up a little differently, where the `tests`-folder is a subfolder of the sourcecode-folder, and the contents of the `src`-folder is moved up one level. Recently, it was discussed whether Diffsims should move the tests out of the sourcecode folder, which concluded by keeping them, since this allows users to run the tests themselves[2]. The reason for the tests being in a completely separate folder in `tiltlib`, is to unambiguously separate them from the module code itself. The tests are only meant to test the code, and not meant as a part of the distribution. Therefore, the decision was made to move them out into their own folder in the project directory.

The decision to have a `src`-folder, instead of simply moving its contents up one layer and deleting the (now empty) `src`, was less considered. A project structure with `src`-folder(s) is quite common in other programming languages, such as C and C++, and is familiar to many programmers. Regardless, changing the structure to align more with the Pyxem packages is simply done, and does not affect the end user of `tiltlib`.

5.1.2 Non-code Parts of the Project

Effort was put into setting up tools which would ease development work. These are mainly GitHub Actions, which were set up to automatically run all the tests on multiple versions of Python every time new code was added or edited. Another action was to ease deployment of new versions of the code, where the package was built and published to PyPI whenever a new release was made in GitHub. PyPI (Python Package Index) is a Python package repository, and uploading a package there facilitates download/installation with `pip`. Similar actions can be set up for automatically building and publishing new versions of documentation, if such were to be written. These automatic actions were very helpful during development, and will likely continue to be helpful if more features are added later. The automatic

[1]`https://github.com/pyxem/orix/issues/452`
[2]`https://github.com/pyxem/diffsims/issues/212`

testing is especially useful, as merging code into the main branch is automatically blocked if the tests do not pass.

The `examples`-folder contain Jupyter notebooks which serve as tutorials and, inadvertently, integration tests. These currently show how to identify the tilt axis position from a tilt series, and how to calculate the tilt angles necessary to align the sample to a given zone axis. Both of these are quite specific and complex features. The project can benefit from more tutorials with a simpler scope, showing how to initialize `Axis` and `Sample` instances, and showing the different plotting functionality available. Additionally, documentation of the API should be easily accessible to users and developers.

These can both be achieved by using e.g. Sphinx[75] to generate a website, similar to what is done for all modules in the Pyxem suite. This website can show installation instructions, the discussed examples, simpler tutorials, and an API reference. As most functions have docstrings, and all functions have type annotations, an auto-generated API reference is likely quite presentable and useful without much work. A home page for the project could also attract more attention to it, both for users and developers. The work required for creating such a website is low, as Sphinx does most of it automatically, and as all of Pyxem use Sphinx to generate their websites. Additionally, as discussed, GitHub Actions can be set up to automatically generate and publish new versions of the documentation, lessening the workload further.

As mentioned in Table 3.6.1, the version of `tiltlib` used in this thesis is 0.0.5. When finalizing the thesis, version 0.1 was released, which implemented a documentation website as outlined in this section. The website is available on `https://viljarjf.github.io/tiltlib`.

5.1.3 The Classes

The `Axis` class is a data container, keeping track of the direction of rotation, the current angle of rotation, and the limits of said angle. For some usecases, setting the limits to lower values than what the sample holder is capable of can be beneficial, e.g. to limit the use of a secondary tilt axis, or to only allow tilting towards a secondary detector in the column. Excluding this, what is relevant to discuss is the unit of measurement for the angles. Upon initialization, the user may specify their angles either in degrees or radians. At creation, the constructor will convert all these to radians, as all internal calculations use radians by default. As such, when accessing the member variables, one will get radians, regardless of whether the object was created with degrees or radians. This might lead to downstream bugs, if the programmer assumes their angles and units were kept as they were initialized. A workaround exists, as the `degrees` member variable, which describes whether the angles are measured in degrees (`True`) or radians (`False`), is set correctly after initialization. A user may access this variable, and act accordingly.

The `SampleHolder` class is relatively straightforward, and the scope of its features is currently limited to only handling the rotation object transforming from the detector reference frame to the sample reference frame. The reason for keeping this functionality in its own class, as opposed to having it in the `Sample`-class, is the possibility for creating other subclasses. The physical behavior of the sample holder is the same if using S(P)ED or e.g. rotation electron diffraction (RED), but the format of the data is very different. As such, by keeping the rotation logic separate, more features can more easily be added at a later stage.

A useful feature which is currently not implemented in the `SampleHolder` is to account for the shift in position which comes with a rotation. The tilt axes might not be perfectly aligned with the studied region of the sample, requiring the sample to be shifted around to find the same scan region after tilting. In practice, this realignment might take up a considerable part of the valuable time at the microscope with an inaccurate calibration of the eucentric height, but the mathematics behind the shifting is simple. The difficulty of predicting the sample movement after tilting lies not in the mathematics, but in the sample holder geometry. Tilt axis determination can be used to determine the *direction* of a tilt axis, but the technique described in this thesis does not determine the *position* of the tilt axis relative to the sample. As shown in Table 3.1.1, the sample position in the sample holder (X,Y,Z) was unchanged between scans in the tilt series. Still, the scans are quite well aligned, indicating that the eucentric height intersects or is close to the sample. However, this might not be the case in general, and the different tilt/rotation axes are likely to lie at different positions making simultaneous calibration difficult.

For orientations, the position of the tilt axes do not matter, only the direction, but the tilt axis positions have a large impact on where the sample is at a given scan coordinate. If the positions of the axes were known, then the physical displacement of the sample could easily be predicted, saving time at the TEM. This would simply entail calculating the distance of a point on the sample to the tilt axis, i.e. the radius of rotation, and multiplying with the change in tilt angle, to get the arc length. Projecting this into the scan frame should resolve the shifting from any eucentric height, and for multiple tilt axes at different heights.

To achieve this, the position of the tilt axes must be known. The necessary accuracy of the position has not been investigated, but it could be in the µm - nm range as this is the typical length scales of a scan. However, the relative change in coordinates between pixels can be calculated without knowing the position of the tilt axes. The shifting will need to be accounted for manually at the microscope, but accounting for relative squeezing/stretching is as simple as multiplying with the `Rotation`-object calculated by the `SampleHolder`. The relative squeezing/stretching is identical when projected onto the scan frame, regardless of how far from the tilt axis the scanned region lies, making it possible to implement in `tiltlib`.

The `Sample` class is responsible for handling the change in orientation from a rotated sample holder. The scope of other features is limited to mostly plotting. A lacking feature is, as discussed, the physical movement of the orientations. The current implementation only accounts for the change in orientation, not in position, meaning the orientations are changed in the positions they started in. This might be confusing to users of `tiltlib`, and can be mitigated by expanding the `SampleHolder`-class with proper scan coordinate handling as discussed.

`Sample` is a specialized class for handling the change of a `CrystalMap` from Orix. To make the user experience more streamlined, the `Orientation` from the `CrystalMap` is accessed using the private `_rotation`-member. This is due to the way the crystallographic data is stored in the `CrystalMap`, where all arrays are flattened. Some bugs were encountered during development, where `CrystalMaps` sometimes had transposed the data arrays before flattening them. When reshaping the orientations back into a 2D array, the data was the transpose of the expected. When working with square arrays, this might not be noticed, as the relative positions of orientations with their neighbors do not change. However, for non-square arrays, the orientations were unrecognizably ordered. No satisfying explanation was found when investigation this issue, other than recognizing the transpose.

Python does not enforce private/public variables. A popular convention is to prepend an underscore before private members of a class, but there is nothing stopping anyone from accessing these. As such, a user might break some functionality of the classes by accessing and modifying the private members of the classes. The `_original_rotations` of `Sample` is especially vulnerable, as it is used every time the tilt angles change. The orientations of the `Sample` might be implemented in another way to circumvent this issue. This was down-prioritized during development, as the implementation works, and accessing private members is discouraged for users. In version 0.1, a newer version of `tiltlib` not used in this thesis (0.0.5 was used), the implementation is changed to no longer rely on private members of `CrystalMap`.

5.1.4 `find_tilt_angles`

With the given implementation, the accuracy of zone axis alignment can be estimated. TM is precise to around 1°, which when comparing with other TM results to find the tilt axis would compound to an around 2° precision. However, as this is averaged over hundreds of thousands of orientations, the tilt axis identification should provide little decrease in precision. Additionally, the effect of an imprecisely determined tilt axis position is more noticeable at high tilts and less noticeable at low tilts. The remaining sources of imprecision is then the TM results used for prediction, which as discussed have a precision of 1°. Re-inserting the sample into a sample holder might introduce misorientation as well; a precision of e.g. 3° can be expected from visual inspection and manual alignment. The goniometer might introduce some imprecision, especially if they are stepping-motor based rather than a closed-loop control system with an actual reading of the tilt angle. The gears in the goniometer might be more worn at low tilts, possibly improving the precision at high tilt angles. Cautaerts et al. [32] observes the opposite: a worse precision at large tilt angles. Finally, as the grain orientation is not necessarily homogeneous, using a crop of the grain might introduce more variance. Commonly, intra-grain orientation variance in TM is around 0° to 2°. In total, a precision of around 5° is expected, but by careful sample alignment and good TM-results, one can expect around 1°. This should be close enough to the target to allow the operator to observe the Laue circle, which can be used for final alignment. ALPHABETA suggests their predictions are accurate to 1° to 2°, but the uncertainty of orientations seems to be $\pm 0.25°$, much lower than that of TM[32].

As can be seen in Listing 3.3.1, the general algorithm for `find_tilt_angles` is similar to the one used to identify the tilt axis position. A score function, based on a set of angles, is optimized to find a minimum. The biggest difference lies in this set of angles, which for `find_tilt_angles` is the angle between the target zone axis and the current zone axis of the sample. Another difference is what is done to reduce this set of angles to a fitness score, in this case taking the mean of *all* the misorientation angles, whereas the tilt axis identification algorithm only used a third of the misorientation angles.

By using all angles in the optimization, the user is required to crop their scan to a single grain for the result to have any reasonable accuracy in alignment. Otherwise, the optimization will only find the angle which fits best for all grains in the scan simultaneously. With undesired pixels or grains contributing to the score, the predicted angles are unlikely to actually align closely with the desired zone axis. However, in certain cases, multiple grains can simultaneously be aligned to the same zone axis. This is the case for FCC $\Sigma 3$ twins along $\langle 1\,1\,1 \rangle$, where the twin grains share $[1\,1\,1]$ and $[1\,1\,2]$, which is shown in Figure 4.2.3 and 4.2.6, respectively. The alignment might also be possible by chance for grains without a pre-determined orientation relation. Cropping the scan in such a way as to include both grains will make `tiltlib` attempt to align both simultaneously. This can be very useful for studying twins, as the likelihood of such an alignment being present in a scan by chance is low, and manually aligning both simultaneously can be very difficult. Alignment like this facilitates studying effects related to grain boundaries, e.g. diffusion, as is explored in [63].

The current implementation of the algorithm first finds the angle between each vector in the z-projection (the optical axis) of the orientations in the sample and the desired vector, before taking the mean of these angles. Alternatively, one could use the mean orientation of the sample, which would circumvent the need for a separate function for finding angles and a fitness score. The separation was initially used to allow for a percentile selection of angles, similar to the tilt axis identification, but this was later changed to use 100% of the data, deprecating the separation of the functions.

Calculating a mean orientation comes with its own challenges, as discussed in Chapter 2.4.2. The algorithm by Markley et al. [51] is the one used in Orix, so using the mean orientation should provide little numerical difficulties. This is already what is used for the `mean_zone_axis` member function of the `Sample` class, and e.g. using the output of this function as a basis for the fitness function instead should provide little issue.

If the code is refactored to use the mean orientation, instead of the mean angle, the possibility of a selection similar to the one used in the tilt axis fitness function is lost. Initially, this was used in `find_tilt_angles` too, but it was removed during development. By using all pixels for optimization, the result is more likely to generalize to the whole scan, provided the crop is monocrystalline. If not, e.g. if the grain is difficult to crop with either a circle or a rectangle, then the results might be better if the user was allowed to choose a fraction of the pixels to discard. Implementing such a choice does not entail much work, and would expand the usability of the library. Additionally, one could then add the option of using the mean orientation instead of the mean angle. `tiltlib` version 0.1, a newer version developed after finalizing the thesis, supports this option for zone axis alignment.

5.1.5 Code Testing

The code of `tiltlib` was tested mainly in the form of unit tests. These can be found in the `tests`-folder of the project. Unit tests entail testing some functionality on its own, rather than the interaction between different parts of the code. For `tiltlib`, this mostly meant testing whether the rotations were handled correctly, as discussed in Chapter 3.3.3. Although all the tests that were constructed gave the correct output, they were mostly simple 90° rotations around the Cartesian axes. This made the tests easy to construct, seeing as the outputs were straightforward to calculate separately. However, the simplicity of the tests might not capture all dynamics of the rotations. As an example, if rotating twice around the same axis, then it does not matter if the second rotation is intrinsic or extrinsic. The tests needs to capture all possible cases that might be encountered when using the program, and ensuring this is indeed the case is a difficult task.

One way this issue was mitigated was by keeping the scope of each test small, and keeping the scope of functionality for each part of the code small. This makes it easier to construct tests which span the expected input space of the functions. As an example of keeping the scope of tests small, separate tests were written for checking if an object could be initialized at all.

The unit tests were written alongside the development of `tiltlib`, sometimes being modified to fit the new expected output. A better method would be to plan out the API of the module, and write all the tests before starting to develop the actual module. While this development cycle might ensure less bugs are introduced during development, it is difficult to implement in practice. It would ensure a clear plan was made, and that the functionality was carefully considered. This is common practice in enterprise-level software development, indicating it is good practice. The biggest issue is time, as the planning and considerations of necessary features is rather time-consuming. Furthermore, when planning a project to a level of detail precise enough as to be able to write encompassing tests, then one might as well simply write the functions to test simultaneously. Additionally, planning out the entire API can be difficult, as the requirements for the project might be poorly understood in the beginning. By simply starting to write some code, one can get a better understanding of what a user might want, how to structure the code, what should be available for the user and what should be hidden *ect.* This more dynamic approach leads to a quickly developed project, but it can be more prone to bugs as the tests are written alongside the code that is being tested.

The greatest weakness of the testing suite is the lack of automatic integration tests. As opposed to unit tests, integration tests are tests which test how different parts of the codebase interact. For `tiltlib`, this was done using the example notebooks, as mentioned in Chapter 3.3.3. These were written after the development was assumed to be finished, but multiple bugs were found thanks to these notebooks, showcasing the importance of proper testing. The notebooks can be run automatically, similarly to the unit tests, but verification becomes more difficult as they are currently checked manually. As discussed, this can be addressed by running the notebooks and displaying them as part of online documentation.

5.2 Tilt Axis Identification

The tilt axis position was determined to lie 30° from the x-axis in the xy-plane of the sample, as can be seen in Figure 4.1.3(a). From a manufacturing standpoint, this makes sense, as 30° is a round and reasonable angle to use when designing the instrument.

The method developed and used in this thesis differs significantly from established methods. Often, other papers only concerns themselves with tilt axis identification in the detector frame. This is the case for e.g. ALPHABETA[32] and PETS2.0[64]. Tomography-focused microscopists are often interested in the tilt axis in the sample frame, and employ a more similar method to what was done in this thesis, namely a tilt series[76]. In this case, the implementation is image-based, rather than diffraction-based as used in this thesis. Brown et al. [77] use SED for to identify and compensate for tilting, but do not identify the position as they use annular detectors. For the purpose of orientation mapping, especially with SED data, Mathisen [49] suggests a method using a SPED tilt series, where grains orientations are compared for multiple tilts. Additionally, the preliminary work for this thesis[45] attempted to perform tilt axis identification in a similar way. However, both of these rely on grain segmentation, using MTEX[31], which complicates the workflow. A standalone method using a SPED tilt series was therefore developed. The methodology used, its downsides and potential improvements, are discussed in this section.

5.2.1 Pixel Coordinate Misalignment

The position of the tilt axis was determined through pixel-wise comparison between orientations of the sample. As such, there are multiple factors which can contribute to misidentification.

The largest contribution to misidentification is the misalignment between pixel coordinates and true sample coordinates for different scans. The fact that tilting the sample causes a displacement, making a given pixel correspond to a different position on the sample, is not corrected for in the fitness function. The six plots of the pixel-wise misorientation angle at the optimal fitness function value, shown in Figure 4.1.3(c), clearly show how the grain boundaries do not perfectly coincide between scans. This adds a large contribution of non-representative misorientation angles to be part of the fitness function, where pixels from one grain near the boundary are compared with the neighboring grain in a different scan. The contribution becomes larger for larger tilt angles, as the physical displacement of the sample is greater. Thus, the comparison between scan 3 and 6 with 15° tilt angle difference, has a high amount of incorrect comparisons. This could be somewhat detrimental to the accuracy of the optimization, since a larger difference in tilt angle would decrease the effect imprecise orientation mapping has.

To combat this, one could crop the data of each scan before comparing them. The cropping could be aligned in such a way as to minimize the overlap of the grains along boundaries. Simultaneously, one could choose a region where the sample is crystalline, avoiding e.g. holes and amorphous regions. The data was not cropped in this project, as the process is rather time-consuming.

Instead of cropping the data, one could use the known tilt angle to predict how the sample coordinates of each pixel transform under rotation. Hyperspy does seem to support such an action, as the navigation and signal axes coordinates can be updated manually. The transformation to apply is the same as the one applied to the orientations, which suggests this being possible to implement. Additionally, if the coordinate transform feature is implemented in `tiltlib`, no additional work is required to use the same framework for zone axis alignment. Both of these considerations might improve the accuracy of the comparisons, but as the results without these measures work well, this was not prioritized.

Apart from coordinate misalignment in real-space, the reciprocal space might be misaligned. Two factors can contribute to this; centering and scaling. Additional factors might be present with different detectors, e.g. shearing and stretching with camera imaging of a fluorescent screen, but the DED used in this thesis does not suffer from these. Centering was performed with subpixel-precise estimation of the CoM of the central spot, which should be precise enough for accurate TM. The scaling, however, might be incorrect, as it was done with the assumption of an alignment on [1 0 1]. TM revealed the alignment of the crystal to [1 0 1] was off by a few degrees, although it was certainly close enough to index the spots manually. However, the deviation of a few degrees is unlikely to significantly move the positions of the reflections, but rather change their intensity with a differing excitation error. Additionally, by manual inspection of the TM results with overlaid simulated spots, the calibration seems correct, as is shown in Figure 5.2.1.

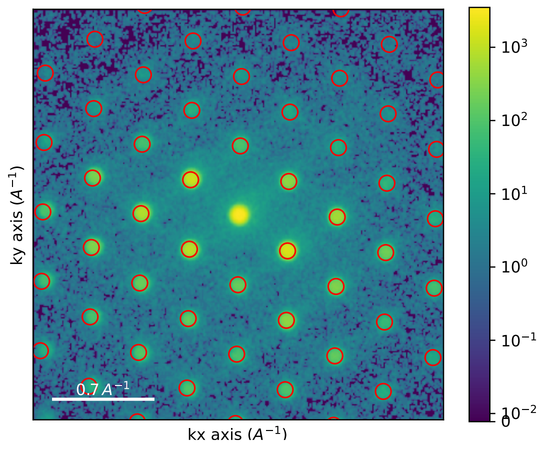
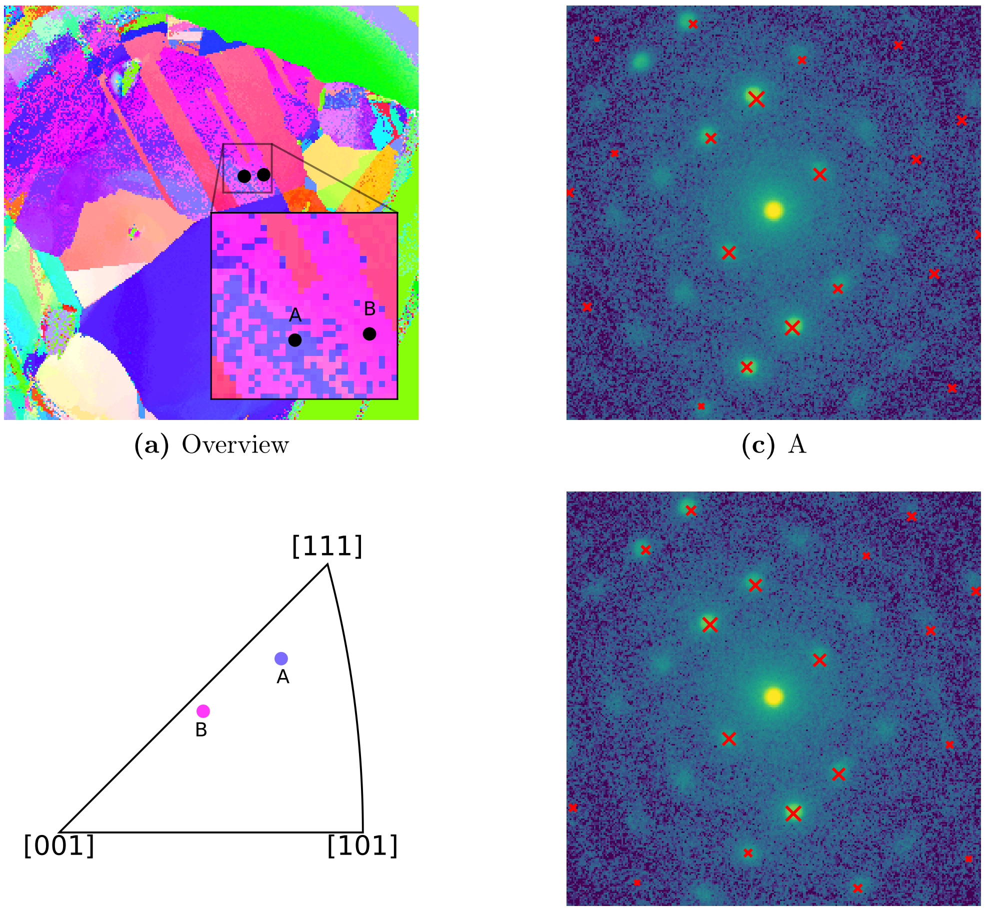


Figure 5.2.1: The near-[1 0 1] zone axis pattern used for reciprocal calibration of the Ag sample in Chapter 3.1.3, with the TM results overlaid as red circles. The spots align well, both near and far from the central spot, indicating a fitting scale calibration.

5.2.2 Misindexation

Another factor which might contribute to misidentification of the tilt axis is the effect known as 'speckling'. This is a misindexation caused by two orientations, or more accurately, two regions of orientation space, having similar-looking diffraction patterns. If the orientations are sufficiently far apart, one would see them in a IPF as pixel-wise speckling, where two different colors are present in a single grain. Speckling is present to some extent in all four scans in the Ag tilt series, and a specific example is shown in Figure 5.2.2. The misorientation angle between the two orientations shown is around 15°, and a significant fraction of the grain is misindexed in this way.

Aune [63] attempts to mitigate this effect by means of a tilt series. Tilting the sample can break the similarity between the regions, which can be used to determine which of the two speckled colors are more likely to be the true orientation.



(a) Overview **(c)** A **(b)** IPF **(d)** B

Figure 5.2.2: Two orientations in the same grain in scan 4, which were identified as two orientations which are around 15° from each other. (a) Locations of the two points in the grain. (b) Location in the symmetry reduced zone of the two orientations, with a misorientation angle of about 15°. (c), (d) Diffraction spots of the labeled points A and B respectively, with the simulated diffraction spots marked as red crosses. Figure from [45].

A third factor to consider which might cause an incorrectly identified tilt axis, is a consistent misindexation of a grain. This would essentially be the same as speckling, but instead of just affecting some pixels, entire grains would be identified as the incorrect orientation. Although unlikely, the impact of such an occurrence would be much larger than simple speckling.

5.2.3 Data Selection

All three of the factors discussed so far can be addressed by considering only some, and not all, pixels. One must then decide which pixels to choose. Manually correcting for speckling and/or consistent misidentification by removing those pixels is an option. One could also choose a cutoff point, discarding any misorientation angles above a certain value. By inspecting Figure 4.1.3(b), a cutoff of 20° seems a good choice. However, as only two tilt axis positions are shown, it is unlikely to be a good choice for all tilt axis positions. If, for example, some tilt axis position existed where all grains have a 30° misorientation angle, then a simple cutoff at 20° would discard so much of the data that the fitness score no longer holds much weight. The aim of the fitness function is to calculate how effective a certain virtual tilt axis position aligns with the real one, so by discarding most of the data the fitness score loses meaning.

Another option for a discarding strategy, the one used for the fitness function, is to choose a fraction of the pixels with the lowest misorientation angle. The $6 \times 256 \times 256$ misorientation angles were sorted in ascending order, and the top $\frac{2}{3}$ of those were discarded, leaving the bottom $\frac{1}{3}$ for further processing. Note that the actual implementation is algorithmically more efficient, as it uses the introselect algorithm[3] instead of actually sorting each element. The selection is unsupervised, meaning it has no regard for the physical position of the pixels or which scans were compared in a given misorientation angle.

As the grains were relatively large, the overlap between grains along the boundaries still left a good amount of pixels which were compared to the correct corresponding grain. This can be seen in Figure 4.1.3(c), where grain boundaries are only 5-10 pixels wide, meaning their misorientation angle of about 60° does not constitute a large fraction of the data. The ratio $\frac{1}{3}$ to keep was chosen to keep a safe margin, allowing some correctly compared pixels to be discarded rather than opening for the possibility of including incorrect comparisons. The kept/discarded misorientation angles are shown for two different tilt axis positions, 30° and 100°, in Figure 4.1.3(b). By inspection, it seems a larger fraction of misorientation angles could safely be included in the optimization without issues with e.g. convergence. However, the aim of ensuring a safe margin seems to have been fulfilled.

By varying the fraction to keep/ discard, the fitness score changes accordingly. Keeping more data results in a larger fitness score, as the score takes the mean value of the kept angles. Similarly, keeping a smaller fraction results in a lower fitness score.

By lowering the fraction of kept angles enough, one will eventually start overfitting to the data, and one might unknowingly only consider a single grain, or a single scan combination. This might result in a tilt axis position which does not generalize well to the whole dataset. Conversely, considering a too large fraction might introduce more variance, which might lower the precision of the answer. Furthermore, for the scans in this thesis, multiple regions of the scans are not crystalline (holes, the amorphous capping layer, and the thick region), but still have an assigned orientation from TM. By considering a larger fraction, these regions are eventually included in consideration. This is unwanted, as the crystallographic orientation assigned to these regions do not correspond to a physical orientation. Holes and thick regions are likely to be assigned an orientation at random, as the data for these regions are essentially random. For amorphous materials, however, the diffraction pattern is radially symmetric. Therefore, due to the implementation of TM in Pyxem, the zone axis in x- and y scan directions are spatially uncorrelated. The z-zone axis, however, will correspond to the diffraction pattern which is the densest in reciprocal space, in terms of reflections. For cubic crystals, this is usually the $\langle 1\,1\,0 \rangle$-direction, as can be seen in Figure 5.2.3, but other crystals may have other directions of maximal diffraction spot density. Regardless of which direction is the one being picked, the assigned orientations of amorphous diffraction patterns are not indicative of the actual orientation of the sample, and as such have a detrimental

[3]`https://numpy.org/doc/stable/reference/generated/numpy.partition.html`

impact on the reliability and accuracy of the tilt axis position if those orientations are included in the fitness score.

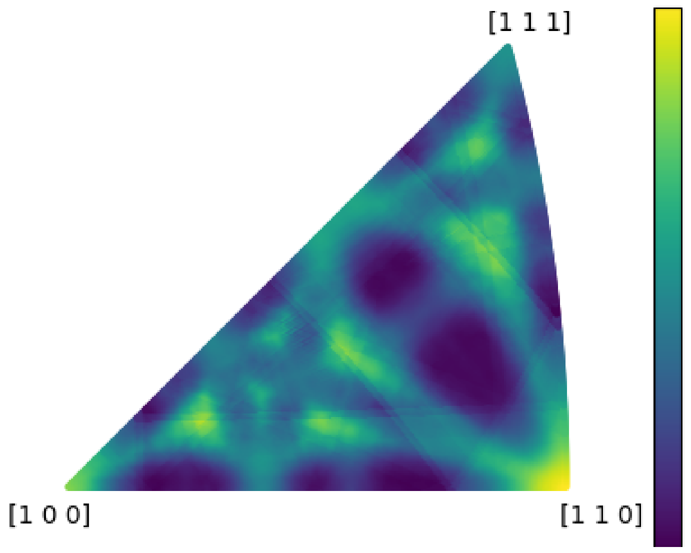


Figure 5.2.3: The NCC values of templates with a uniform diffraction pattern, giving a measure of the density of diffraction spots in reciprocal space. The highest correlation, and therefore density, is at [1 1 0]. Figure from [45].

5.2.4 Parameter Space Restrictions

The plot in Figure 4.1.3(a), showing the tilt axis fitness score as a function of tilt axis position, is one dimensional, as the tilt axis was assumed to be in the xy-plane. This assumption removes a degree of freedom, and is reasonable given the knowledge of the physical layout of the TEM. However, it might not be entirely accurate, as a small deviation from the plane might be present. Optimizing in the complete direction space, instead of the planar subspace chosen for the fitness function, might increase accuracy. The computation time would increase, but as the tilt axis position would only need to be determined once per microscope, this consideration bears little weight.

Note that the minimum of the fitness function in Figure 4.1.3(a) is not 0, but 1.4. The misorientation angle is never negative, so taking the mean of these is very unlikely to ever reach 0, due to e.g. the $1°$ expected precision of TM.

One issue to consider is the parameter space increasing in size, which might lead to local minima which are not the global minimum. Clearly, the single-parameter optimization shown in Figure 4.1.3(a) is very well-behaved, appearing to have only a single minimum. Adding another parameter to the optimization could introduce local minima which 'trap' the optimization algorithm.

Symmetry might also cause local minima, or even degenerate global minima, if the tilt axis is aligned with an axis of symmetry in the crystal. This is observed for area 2B for zone axis alignment, which uses a similar optimization function. Along one of the $\langle 1\,0\,0 \rangle$-directions in a cubic crystal, for example, one would get four minima separated by $90°$. This effect is easily prevented by including multiple grains in different orientations. The total fitness function would then act as a superposition of the fitness functions considering each grain separately, and the global minimum would only occur at the minimum valid for all grains. This can be seen in Figure 4.1.3(c), where all grains (excluding non-crystalline regions, speckling, and overlapping grains) have a low misorientation angle. Other tilt axis positions might have some grains with low misorientation angles, e.g. Figure 4.1.3(b) shows a peak at around 5° for the tilt axis position of 100°, but only where all grains simultaneously have a low misorientation angle can the true tilt axis lie.

A parameter restriction enforced by the data collection strategy is the limited tilt range in the scans. Only 15° is spanned by any of the three tilt series used in this thesis, and the Gy-tilt series of LMNO was only 5°. As a larger tilt span would cause a larger difference in orientation, this could potentially give a better precision. The eventual limitation, given that the increased distortion in sample coordinates is handled, will be the range of tilt available with the given sample holders. In this thesis, the limits were ±30°, but specialized tomography holders can go much higher. More scans will also increase the precision, as the increase in data will give a better estimate. The current implementation uses all possible combinations of scans, making the misorientation angle count scale with the square of the number of scans. This is a rapidly increasing scaling, which can quickly impose a computation resource limitation on the optimization. Other strategies can easily be implemented by modifying the function as listed in Listing 3.2.1.

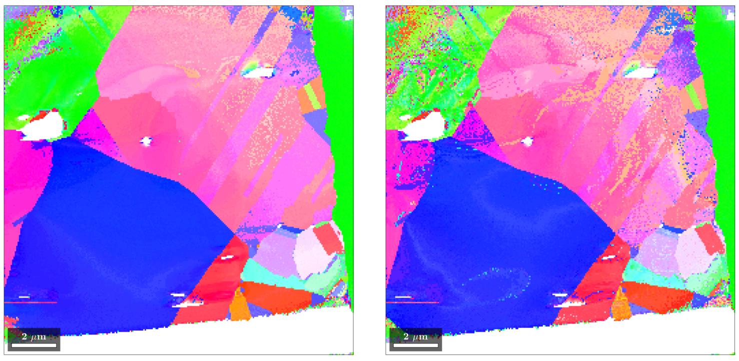
5.2.5 Grain Segmentation and Statistics

By introducing grain segmentation, instead of considering pixel-by-pixel orientations, one could circumvent a lot of the points discussed so far. Considering only the mean orientation of a grain, and ignoring the physical position as well as speckling, would remove the need to align and filter the misorientations. This comes at a cost of decreased data size, as e.g. 20 grains in each scan would give a total dataset of 240 angles, whereas the pixel-approach had a total dataset of almost 400 thousand angles, where over 120 thousand of them were used for the Ag sample.

Grain segmentation is a complicated problem, where several aspects, criteria and models have to decided on, and there is no dedicated functionality for this in Pyxem. Exporting the TM results to e.g. MTEX[31] is an option, as the package has dedicated grain segmentation algorithms. This was done in the preliminary work[45], but one then needs to consider how exporting and importing the data between packages works, and ensuring the different conventions are accounted for. Additionally, the inclusion of MTEX and Matlab in the pipeline makes it much more complicated and less streamlined.

Packages exist for grain segmentation in Python, e.g. DefDap[78], so it should be possible to have a grain-based workflow while keeping to the Python framework. Staying within one framework is desired, both for the user and for the programmer, as having to switch programs or write interfaces between systems is tedious.

Assuming the data alignment between scans is mitigated, and ignoring speckling and misindexation, then the pixel-by-pixel comparisons should be very similar to a grain-based comparison. The difference would lie in the variance within grains, which is ignored when using only the mean orientation of grains. If variance is the only difference between using grains and pixels, then one could argue that using SED should be sufficient to determine the tilt axis. Inspecting Figure 5.2.4, it seems using SPED instead of SED decreases intra-grain orientation variance, but even



(a) IPF-z, SPED **(b)** IPF-z, SED

Figure 5.2.4: IPFs-z of orientation maps for a SPED (scan 1) and a SED (scan 2). The color map can be found in Figure 2.1.4(c). (a) SPED IPF-z. (b) SED IPF-z.

with increased variance the fitness function should still attain its minimum at the correct tilt axis position. As such, one could collect data at a much faster rate, and avoid precession alignment challenges, while still determining the tilt axis position with sufficient precision. If tilt axis identification is possible without precession, it also opens up for using this method for tilt axis identification in microscopes without a precession system, making `tiltlib` useful for these as well. Bergh et al. [43] compares the effects of precession on orientation maps, and it seems a shorter acquisition time can be detrimental, but orientation maps created from SED data with longer acquisition times seem to be of high enough quality to determine the tilt axis. This possibility was not explored in this thesis, as SED was only used for a single scan rather than a tilt series.

5.3 Zone Axis Alignment

The predicted tilt angles for alignment to a given zone axis generally seems to align well with the actual tilt angles. This is shown in e.g. Table 4.2.1, where the predicted and actual angles are up to around 4° apart. Both for the Ag sample and the LMNO sample, `tiltlib` show alignments within a few degrees, which is within the expected range of 5° outlined in Chapter 5.1.4.

Both tests were performed on cubic materials, with the point group $m\bar{3}m$. This might hide some bugs in the code, as cubic crystals have an orthogonal basis. `tiltlib` could have defined some operation where an orthogonal basis is assumed, which would yield inaccurate results for e.g. hexagonal and monoclinic crystals. Care was taken during development to outsource as much crystallography as possible to Orix, which is extensively tested to ensure correct handling of all crystals. Undiscovered errors might still be present in `tiltlib`, and further testing is necessary to ensure correctness for other crystal systems.

5.3.1 Ag Sample

Qualitatively, Figure 4.2.1(b) and 4.2.1(c) show a large degree of similarity, although simple visual color inspection is not very precise. As Figure 4.2.1(c) is virtually

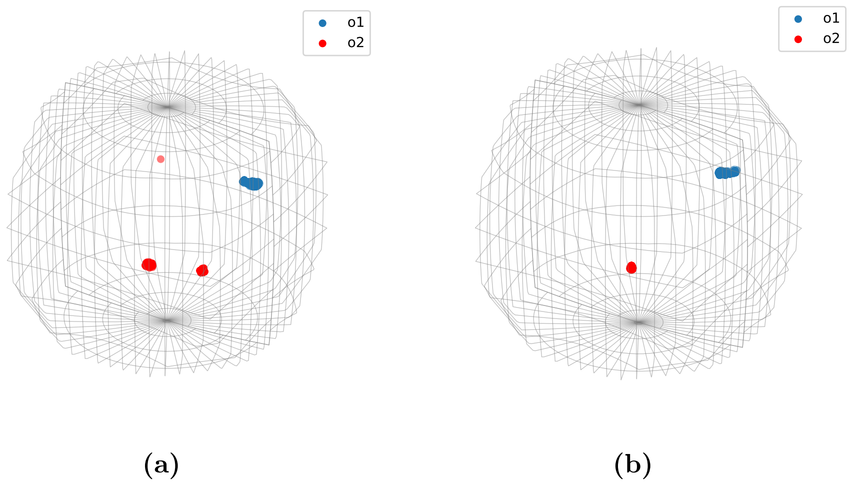


Figure 5.3.1: 3D axis-angle representation[31] of the two populations used in twin analysis in Chapter 4.1.2. (a) The twins of scan 5 of the Ag sample, showing how o2 consists of two distinct populations. Therefore, the mean of o2 is not representative of the population as a whole. The single orientation of o2 in the upper 'hemisphere' is considered an outlier. (b) Scan 6, shown as an example of homogeneous populations found in the remaining scans

tilted to 16.1°, only 1.6° from Figure 4.2.1(b), their visual similarity indicates a good fit of the tilt axis. Furthermore, as the tilt angle was calculated by aligning the grain marked with a green rectangle in Figure 4.2.1(a) to the mean zone axis of the same grain in Figure 4.2.1(b), the apparent similarity indicates a working zone axis alignment in `tiltlib`. The remaining indicated zone axes also show a good alignment, with angles between the mean zone axes between 0.9° and 1.7°.

As Table 4.1.1 shows, the deviation from the known expected misorientation angle of the $\Sigma3$ twin domains is generally around 2° to 4°. This deviation is a little higher than expected, as comparisons with two populations of 1° precision should have around 2° precision. Generally, the results of zone axis alignment with these orientation maps should be within 4°, if the twin misorientation deviation is representative of the true deviation.

A clear outlier is present in Table 4.1.1: scan 5. This is due to the population selection by thresholding, as two sub-populations are present within one. This is shown in Figure 5.3.1(a), where o2 is shown to consist of two separate populations when viewed in the axis-angle representation. The mean orientation of o2 is not representative of the population as a whole, and the angle between the means of o1 and o2 is therefore not indicative of the precision of TM for this scan. For comparison, Figure 5.3.1(b) shows the axis-angle plot of the two twin populations in scan 6, the twins with the lowest deviation from 60°. This distribution, where neither population have a bimodal distribution, is the case for all scans except scan 5.

One limitation of attempting zone axis alignment on the Ag sample is the fact that all scans were taken before the orientation maps were made. Therefore, the data is already used for tilt axis identification, and the results are technically part of the data. However, the Ag sample is prone to oxidization once removed from the vacuum-pumped TEM column, which would affect the crystals. The LMNO sample does not suffer from this drawback, and the structure was stable for the weeks between the SPED session for orientation mapping and the verification session. If orientation maps were produced without removing the sample, and zone axis alignment predictions with `tiltlib` was made on the fly, then an increased precision is expected due to no longer having a sample-sampleholder misalignment possibility.

The results for the Ag sample show how `tiltlib` can accurately predict tilt angles, with a precision of 1° to 2°. One might question the generality of these results, as already stated above, one could argue the expected result was a part of the data used for obtaining the tilt axis position. This is analogous to machine learning, where one avoids testing a model on the same dataset as it is trained on[79]. Furthermore, the search for a tilt axis was limited to only the x-tilt, which might artificially impose a bias towards the expected answer. In general, the results for the Ag sample are promising, but more robust testing, and validation after prediction, is necessary to properly evaluate the accuracy.

5.3.2 LMNO Sample

The LMNO sample, which is part of Aune's thesis[63], was used in a more robust test. The sample was scanned, orientation mapped, and tilt angles were predicted for alignment on multiple grains, before verifying the predictions by re-inserting the sample into the TEM. This provides the necessary results to determine if `tiltlib` is accurate and precise enough to be of use.

The predicted tilt angles align well for all investigated grains, zone axes, and combinations of axes, both on the JEOL ARM and the JEOL 2100 microscopes, as shown in Table 4.2.1. All sets of predicted angles were within a couple degrees, which made final alignment using the Laue circle method easy for the operator, as can be seen in Figure 4.2.8. The accuracy is within the expected range of 5°, as mentioned in Chapter 5.1.4, despite the sample being removed and re-inserted into the sample holder, and into different TEMs.

Area 1 and 3 show how `tiltlib` can be used for aligning multiple grains simultaneously to the same zone axis. This is especially useful for HRTEM grain boundary imaging, where the grains should be aligned to a low-index zone axis for good contrast in lattice imaging. Figure 4.2.3 and 4.2.6(a) show two distinct populations of orientations, i.e. the two twin grains in each of the two scans, meaning the mean orientation is unlikely to be representative for the crop as a whole. This is the case in general for distinct orientation populations; the mean of disjunct populations of similar size are not representative for either population. As such, using the mean orientation for the optimization function would not allow for this usecase of `tiltlib`.

Predicted angles for alignment of LMNO area 2B initially differed from the ones predicted for area 2A, as is shown in Table 4.2.1 where there are two sets of predicted angles for area 2B. This is caused by symmetry and the optimization algorithm. As can be seen in Figure 5.3.2(a), the optimization landscape in the case of area 2B contains four degenerate minima. A more typical example with only one minimum (that is, one minimum within ±20° in this case) is shown in Figure 5.3.2(b), showing the optimization landscape for area 2A. This makes the optimization more sensitive to the initial tilt angles, as was the case in this instance. Starting optimization of area 2B from tilt angles (0°, 0°) yielded the tilt angles (13.9°, 1.0°). By starting at (10.6°, 12.3°) (the results from aligning area 2A, see Table 4.2.1), the prediction instead converged to (−9.37°, 12.9°). This behavior is an advantage; by specifying the limits for the tilt axes, one can force the optimization to e.g. heavily favor one tilt axis, or to only tilt towards a secondary detector.

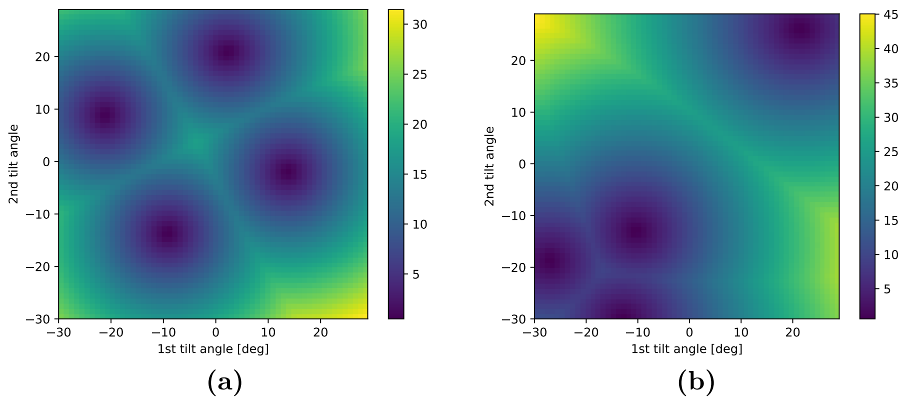


Figure 5.3.2: The zone axis alignment optimization landscape of area 2 in the LMNO sample when aligning to zone axes. (a) Area 2B, aligned on $[0\,\bar{1}\,3]$, showing the presence of four degenerate minima within the tilt ranges. (b) Area 2A, aligned on $[3\,2\,3]$, with a single minimum within ±20°.

As Figure 2.4.2 shows, the scan rotation affects the real-space coordinates of the scan. This in turn affects the orientation of the tilt axes in the scan reference frame. To account for this, the orientation maps made from the SPED tilt series were used to identify the tilt axes of the sample as a safety measure, given that the same microscope (JEOL 2100F) was used for both this and the Ag tilt series. The samples were inserted differently in the sample holder; the Ag sample was randomly oriented, whereas the LMNO sample was intentionally aligned with the tilt axes. Scan rotation was used to additionally align the sample with the scan. Since the scan rotation for the LMNO scans is known to be 28°, it can be presumed that the tilt axes sit at $30° - \Theta$ and $-60° - \Theta$ for the x- and y-tilt axis, respectively, for a scan rotation Θ. This might be different for other microscopes, depending on the internal definitions of reference frames. However, as Figure 2.4.2 shows, the scan rotation only affects the S reference frame, rather than e.g. crystal orientation relationships.

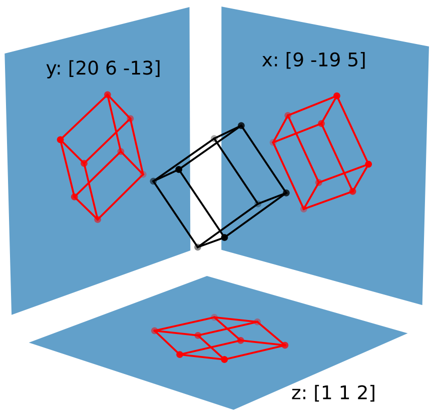
By careful consideration of the sample orientation in the sample holder, as well as the sample holder axes in the TEM, one can significantly reduce the mental workload for the operator when searching for and aligning on a zone axis. If the sample is aligned well with the sample holder before insertion, e.g. aligning an edge of the sample to the Gx axis before inserting the holder, then the physical (image mode) effect of tilting is consistent between instruments, and predictable for the operator. Inspecting Figure 3.4.1, this was done for the LMNO sample. Furthermore, if the scan rotation is set to the position of the tilt axis in the TEM, i.e. $\Theta = 30°$ for the JEOL 2100F used in this project, then the tilt axes would be aligned with the scan axes Sx, Sy. Aligning both the sample and scan rotation with the tilt axes allows the operator to more easily navigate the sample, both to identify regions of the sample between scans or microscopes, *and* to more easily understand and predict the effects of tilting.

Scan rotation alignment was used when the LMNO sample was re-inserted into the microscope, as shown in the photographs of the sample holder in Figure 3.4.1. A consistent alignment between the sample and the sample holder between TEM sessions is necessary for the predicted tilt angles to be of any use, as they are predicted based on the tilt axes' positions relative to the initial orientation map. A slight discrepancy, e.g. $\pm 3°$, of the sample orientation is to be expected, given the manual insertion of the sample into the sample holder. Inspecting Table 4.2.1, one might observe the consistent trend of a slightly smaller actual tilt angle in the x-tilt, and a slightly larger actual y-tilt, for the 2100 compared to the ARM. The consistency of this trend implies the cause is due to the presence of a misorientation of the sample in the sample holder between the two microscopes. This might account for some of the misalignment observed between the predicted and actual tilt angles.

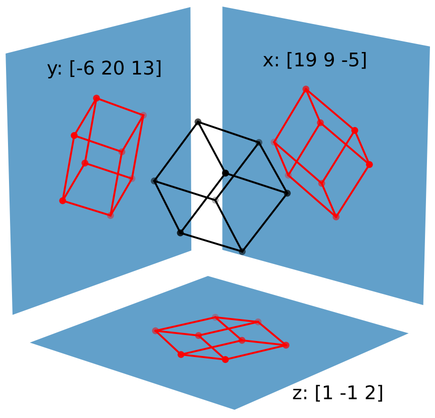
The main reason for having to take the sample out of the holder is the need for time-consuming data analysis (e.g. TM) to create an orientation map for use in `tiltlib`. During this time, occupying the microscope and/or holder without doing any data collection is unproductive in terms of efficient use of resources, which can rather be spent by someone else in the meantime. If enough sample holders are available, or if some automatic sample storage and loading system is in place, then one could keep or accurately reproduce the sample orientation for subsequent analysis after `tiltlib` has produced its output.

Alternatively, one could identify the regions of interest while operating the TEM, and either take a small SPED scan or simply a single PED or NBD pattern. This would reduce the data analysis time, enabling analysis to take place on the fly during the TEM session. Currently, single PED patterns are not directly supported in `tiltlib`, but by creating a 1×1 `CrystalMap` from its orientation it should work without much trouble. A small scan (in terms of probe positions) would not only reduce the time spent on TM, but also the time spent on finding the tilt angles for zone axis alignment. Optimally, the whole procedure of SPED, TM, and tilt axis alignment, should be possible to perform whilst at the microscope in an interactive fashion. Similar processes are already possible with programs such as KSpaceNavigator[33] and τompas[34], exemplifying the feasibility of an interactive in-line `tiltlib`.

The twinned grains in Figure 4.2.6, i.e. area 3, is an interesting special case where it seems both grains are aligned in all three directions x, y, and z. As mentioned in Chapter 4.2.2, the misorientation angle between the grains was 59.67°. This aligns well with the expected 60° from $\Sigma 3$ twins around $\langle 1\,1\,1 \rangle$, and suggests that they are indeed different domains even when the homogeneous color in all three IPFs suggests otherwise. Normally, even when aligned in z, twin domains are often differently colored in x and y, which is seen in e.g. Figure 4.2.3. The fact that the grains from area 3 align so well in the two other directions could just be a lucky coincidence (apart from the desired $[1\,1\,2]$ in the z-direction, which is expected). The zone axes for x and y, $[13\,6\,20]$ and $[9\,5\,19]$, are seemingly arbitrary high indices and do not share any factors, suggesting they do not hide some hidden meaning. Yet, the orientation of the two crystals is clearly different, as is understood by the different cubes in Figure 5.3.3(a) and 5.3.3(b). This example shows that, if possible, multiple regions at different (initial) orientations should be analyzed.



(a)



(b)

Figure 5.3.3: 3D construction showing the zone axes for the mean orientation of the twin grains in area 3. The zone axes in all three Cartesian directions are symmetrically equivalent. (a) The lower grain, as viewed in Figure 4.2.6. (b) The upper grain.

5.3.3 Computational Performance Considerations

Predicting tilt angles for zone axis alignment with `tiltlib` is time-consuming, taking a couple minutes for a double-tilt holder and a approximately 30×30 orientation map crop. Profiling the code indicated initialization of Orix's `Orientation` was responsible for significant throttling of the runtime. Orix might not be designed for this usecase, and certain methods and operations might be inefficient. The underlying data structure is `numpy_quaternions`, which are very fast to compute, so by bypassing some of Orix's data handling one might see a performance increase in `tiltlib`.

Another possible avenue for increased performance is the optimization routine. Currently, Scipy's minimization algorithm is used without much consideration for optimal arguments. By tuning these, one might observe increased performance. The parameter which likely has the largest effect on runtime is the `method` parameter. This controls the underlying solver, which in turn can have a large impact on the number of times the optimization function is called. Ideally, the optimization function should be called as little as possible, seeing as it is computationally expensive with the current Orix-based implementation. The currently used solver, 'Nelder-Mead', could possibly be improved by supplying the initial simplex, rather than the initial tilt angles. Other solvers might see increased performance by manually supplying the Jacobian, which might reduce the need for gradient estimation by means of optimization function evaluation.

The underlying problem of decomposing an orientation into one or two orientations with known axes, does not strictly necessitate an optimization-based approach. ALPHABETA was also designed to use numerical calculations, but state that an analytical solution is theoretically possible[32]. This would reduce the problem to 3×3 matrix factorization (assuming the matrix representation of orientations), which would be many orders of magnitude faster than the current algorithm. The mean orientation of the grain can be easily extracted from the `Sample`-object with `get_mean_orientation`, which could be supplied to the analytical solver along with the rotation axes. As opposed to the numerical approach, an analytical solver is likely to require the existence of an analytical solution. This might not always be the case, as guaranteeing the existence of a solution would require the tilt axes to span orientation space (or rather, the symmetry reduced subspace of SO(3) with unique z-projections). With highly symmetric crystals, a double-tilt holder, and grains in multiple orientations, the orientation space is likely to be spanned. However, for low-symmetry crystals, limited tilt ranges, and e.g. a single tilt axis, not all zone axes can necessarily be reached. This does not pose an issue with the optimization approach; it will simply return the angles which most closely align the crystal to the zone axis. Whether this makes the crystal 0.02° or 20° from the target makes no difference, so long as no other closer orientation is available. To avoid confusion for users of `tiltlib`, a visualization showing the available region of orientation space (i.e. reduced zone IPF) should be implemented in a future version, to allow users to verify if the target is reachable.

Using the program Recipro, this visualization is what Aune [63] used to predict tilt angles for zone axis alignment. She entered the orientations found by TM and saw what low-index zone axes were within the tilt ranges. As she found similar angles to `tiltlib`, this another independent confirmation that `tiltlib` gives reasonable predictions.

5.3.4 Other Considerations

The accuracy of `tiltlib` zone axis alignment is within a few degrees, but could likely be improved further. Fundenberger et al. [6] saw increased accuracy of orientation mapping by increasing the camera length, albeit by using Kikuchi lines for indexation. From a template matching perspective, this is likely to help as well, as a larger camera length would increase the amount of visible reflections. As Figure 5.2.2 shows, diffraction patterns of different orientations might only show differences further out in reciprocal space, so increasing the camera length to capture these differences might also reduce speckling as discussed in Chapter 5.2.2.

Kikuchi-based indexation is difficult with SPED, as Kikuchi patterns arise from dynamical effects, but using SED could work. However, as multiple sources state that Kikuchi-based orientation mapping is significantly more precise[6, 17, 44], it might be worth looking into a possible combination of dynamical Kikuchi indexation and kinematical TM indexation.

Shi et al. [80] present a precision estimate of TM-based Kikuchi pattern indexation for EBSD, where the quaternion representation of the orientations of two $\Sigma3$ twins are used in multiple ways to estimate precision. Their analysis methods can be applied to `tiltlib` as well, especially as the orientations in Orix are stored as quaternions. The twin analysis used in this project, i.e. Figure 4.1.2, is easily expanded to include the different metrics presented in [80], which would facilitate an increased understanding in the precision of `tiltlib`.

Zone axis alignment software that interface directly with the TEM can achieve arbitrary precision, with a closed-loop control system. ALPHABETA, which similarly to `tiltlib` is off-line,state a zone axis alignment precision of around 2° [32]. Cautaerts et al. present a thorough error analysis for their precision estimate, much more in-depth than was presented in this thesis, which in the future can be adapted for `tiltlib` to improve the error estimate. Some software do a correction step, based on the Laue circle, to achieve final alignment and a closed-loop control [33, 34]. For both `tiltlib` and ALPHABETA, this is not currently possible, as neither interface directly with the TEM.

5.4 Template Matching Pre-selection Algorithm

One of the downsides of TM is how time consuming the process is, enough to make it unfeasible to perform during a TEM session. One of the aims of this work was to improve TM runtime, primarily on angular resolution, without becoming even slower due to the use of a too large template bank. Angular resolution of the bank becomes a larger issue for low-symmetry phases, as the template banks become increasingly large. In this work, the full brute-force TM with a large template bank is compared to two alternatives: 1) the existing polar-based pre-selection and ii) new 2-step alternative pre-selection.

In general, the current algorithm for pre-selection templates in Pyxem performs better than the new proposed algorithm in this thesis. In almost all test cases, the runtime is lower for the old algorithm compared with the new, and both pre-selection algorithms are much faster than the default brute-force approach. When this is not the case, e.g. in Figure 4.3.3(a), the results of the new algorithm usually disagree with the results without any pre-selection. In cases where the new algorithm has a better agreement with the results without pre-selection, the runtime is higher than the old algorithm. As such, the general conclusion is that this new proposed algorithm does not give the desired improvement compared to the current polar pre-selection algorithm. Note that, for algorithm evaluation, all TM was performed on a 12-core CPU rather than a GPU, which was used for orientation mapping on the Ag sample.

5.4.1 Runtime and Correlation Evaluation

There is one case where the new algorithm might prove useful and better than the old. Inspecting the runtimes of the fine resolution sweeps, e.g. in Figure 4.3.4(a), the runtime of the new algorithm is mostly constant as a function of the fine resolution. This is expected, as the number of simulations to keep (the `n_keep`-parameter) was kept constant, and this is what controls the number of simulations included in calculations. The trend for different crystal systems seems to be a greater reduction in runtime compared to the runs without filtering when the crystal is less symmetric. There seems to be a point for $AuAgTe_4$ around 0.3° fine resolution where both the runtime and accuracy of the new algorithm is better than the old, as can be seen in Figure 4.3.4. As such, it seems highly un-symmetric crystals (or rather, large template banks) might get better results faster with the new pre-selection algorithm than the old. This could be due to the computational complexity of the algorithm, which is constant for all fine resolutions with a constant coarse resolution and `n_keep`. While this is the case for a constant `n_keep`, this value will eventually become too small to be useful. A constant `frac_keep` would likely perform better, but it would increase runtime with a finer fine resolution. A dedicated test, sweeping the fine resolution for a constant `frac_keep`, should be performed on a low-symmetry crystal.

Graphite seems to be the crystal with the best runtime performance of the new algorithm compared with the old, as Figure 4.3.3(a) shows. Both the `n_keep` sweep and the coarse resolution sweep show an reduced runtime for the new in around half the sweep. Taken on its own, this could indicate that the new algorithm is more suited for hexagonal crystals. However, by taking the correlation similarity scores in Figure 4.3.3(b) into consideration, the new algorithm seems less suitable. All three of the similarity scores, the correlation similarity, orientation similarity and equality length, show a considerably worse performance for the new algorithm compared with the old, as can be seen in Appendix B.2, where all results are given. The similarity scores for the new algorithm are consistently worse and have higher variance. Additionally, they converge to values which differ significantly from the expected values, a trend not seen with the polar algorithm. This could indicate a problem with the algorithm itself, or an unfit parameter set for the static parameters.

Disregarding graphite, all datasets converge to optimal correlation values for the coarse resolution sweep. This is expected, as when the coarse resolution approaches the fine resolution, the new algorithm essentially performs TM with the same template bank twice. However, this adds further confusion around the results for graphite, as they do not converge to the correct value. A more in-depth analysis on hexagonal crystals could be fruitful to understand the inconsistent results compared to the remaining crystals.

An interesting feature in the correlation graphs for the two m$\bar{3}$m datasets is the dip in the fine resolution sweep. The increase towards very low resolution, corresponding to very large template banks, is observed for all datasets, and could indicate that the `n_keep` parameter is too small or the coarse resolution being too large. In either case, the correct (i.e. the ones the brute-force dense run chose) templates are being discarded erroneously, which is undesired. The polar algorithm does not seem to have this issue, and stays at a very low level throughout the sweep. As to why the correlation similarity score increases again after the dip with increasing fine resolution, this could be an effect of the linear correlation interpolation being inadequate at predicting the correlation scores of the fine orientation when they approach the resolution of the coarse orientations.

Another interesting find with the two m$\bar{3}$m datasets is their qualitative similarity. While they have the same symmetry, only one of the datasets contain the noise and dynamical scattering effects from real TEM data, which should differentiate them substantially. The graphs are less similar if the y-scale is considered, as for all three sweeps the correlation similarity scores are multiple orders of magnitude different between the two datasets. The runtimes also differ slightly, especially for the `n_keep` sweep, where the crossover point between the new and dense happens earlier for scan 1 than for simulated copper. However, this could be attributed to the dataset sizes rather than the data source, as the copper dataset is larger (1081 diffraction patterns, scan 1 has 720) and has longer runtimes consistent with this discrepancy. The overhead with the new algorithm seems to be the limiting factor for its runtime, as they are practically the same for the two datasets.

5.4.2 Similarity metrics

The goal of the pre-selection algorithms is to reduce the runtime, while still yielding the same results as if they were not used. In this thesis, the three similarity scores, described in Chapter 3.5.2, were utilized to quantify the similarity. The first two, correlation score similarity and orientation similarity, should be 0 when the two datasets are identical, while the equality length should be 1. Three metrics were used, since only using a single one is unlikely to capture the similarity well. A drawback of the correlation score similarity, for example, is that if a single pattern is discarded that would have ended up in the top `n_best`, then all patterns with lower correlation score than that would contribute to the total correlation dissimilarity for that pixel. Additionally, the correlation score measure only considers the correlation score, ignoring e.g. pseudosymmetries and systematic rows, which are common causes for misindexation as discussed previously in Chapter 5.2.2. The orientation similarity score uses the element-wise angle between the orientations of each match, which would capture these misindexations and show them as a high score. However, the orientation score suffers from the same issue regarding missing templates, which cause all subsequent matches to contribute to the score even if they are correctly ordered. The final metric, the equality length, aims to combat this shortcoming by not considering the order of the matches, but rather the fraction of simulations which appear in both sets. This way, one out of ten missing simulations would give a 90% similarity, whereas the orientation similarity and correlation score similarity would likely indicate a more dissimilar result.

In TM, a commonly displayed metric is the confidence score [17], which was not used in this project for similarity evaluation. Confidence scores is a metric for the accuracy of TM, based on the misorientation between the `n_best` simulations. For similarity evaluation, this internal metric is unsuited, as it does not entail comparisons between datasets. The aim of the pre-selection algorithm is not to achieve better results than a full correlation, but rather to achieve the same results. Evaluating the precision using the confidence score is therefore not helpful to the aim of evaluating if the pre-selection algorithms give the same results as the brute-force approach.

For the simulated datasets, additional similarity metrics are possible where the two pre-selection algorithm results can be compared to the true value for the given simulation. This is not possible for the real dataset, as the orientation (i.e. the ground truth) is not known from beforehand. Even for the simulated datasets, this was not done, as in the best case, the pre-selection algorithms give the same results as without pre-selection. The underlying problems of NCC, and the TM algorithm as a whole, cannot be addressed by careful selection of simulations to correlate in this way. The metrics were therefore all comparing with the dense simulation bank rather than the ground truth, as comparing with the ground truth is not directly relevant to determining if the pre-selection algorithms performed correctly.

5.4.3 Algorithmic Considerations

The two-step approach for TM is suggested in recent work; Corrêa et al. [44] proposes using TM for rough orientation mapping before using residue optimization on (dynamical) intensities of indexed reflections. What is new with the proposed pre-selection algorithm is using correlation scores in orientation space to predict unknown correlation values.

The most efficient optimization for the implementation of the pre-selection algorithm was to pre-compute weights for multilinear interpolation. This imposes a first-degree (linear) interpolation on the correlation scores. As such, the first sampling of orientation space needs to be fine enough as to make this assumption accurate. Alternatively, one could implement a higher-degree interpolator, e.g. using Scipy's `SmoothBivariateSpline`. This would add significant computational complexity, but if it allows for a coarser sampling for the first TM sweep then it might lead to lower computation times overall. A bigger impact should be expected for crystals with low symmetry, as these have a larger reduced zone IPF. If changing from a linear to a cubic interpolation allowed for 1° higher angular distance between simulations, then a high-symmetry crystal might go from e.g. 2000 to 1000 simulations in the coarse bank, whereas a low-symmetry crystal might go from 40000 to 20000. A difference of 1000 simulations has a low impact in the long run, whereas a difference of 20 thousand simulations would save hours of runtime.

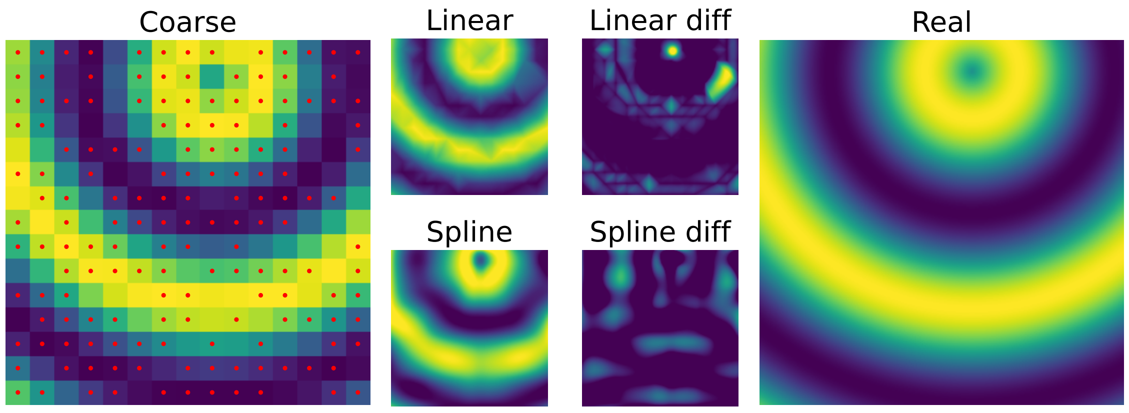


Figure 5.4.1: A comparison between linear and 2nd degree spline interpolation. The left image shows the data to be interpolated, with the sampled points marked with dots. The center shows the resulting linear and spline interpolations, with the deviation from the target. The right shows the actual values for the chosen function. While both interpolations have around the same magnitude of deviations, the spline is much smoother.

Higher-order interpolation also opens up for a different algorithm altogether; only run TM once, and use the interpolated correlation scores directly as the result. For linear interpolation, the largest value can only ever occur at one of the sampled points, meaning higher-order interpolation is the only option for this approach. However, as Figure 5.4.1 shows, spline interpolation does not seem to necessarily be much better at capturing the dynamics of the data. Furthermore, as the coarse sampling is relatively fine, the interpolated values are unlikely to be much different when using spline interpolation compared to linear. Further testing is needed to see if this approach can be fruitful for orientation refinement.

A situation where one might see additional speedup is in a sample with clear orientation relations, e.g. a substrate with precipitates. In this case, only a fraction of the total dense bank is likely to be used for the second sweep. Since the simulations for the second sweep are made on-the-fly, using only the set of unique orientations chosen for the second sweep, this can save time. This would be especially noticeable for more computationally expensive dynamical simulations. Currently, with kinematic simulation and Lorentzian approximation for precession, simulating a bank of tens or hundreds of thousands of simulations take around the same time than the NCC step (using CPU, 90 000 simulations took 4 minutes, TM with this bank on a 256×256 dataset took 6 minutes). The effect should therefore be noticeable even with simple default simulation parameters, given a sample with such orientation an relation.

It is worth noting that increased accuracy of simulations is not necessarily the correct measure to increase the precision of TM. Cautaerts mentioned in a discussion post[4] on this very topic that he observes improved results by binarizing the diffraction patterns, e.g. performing background subtraction and setting all non-zero intensities to 1. As the NCC score (Equation 2.20) heavily favors the strong reflections (near the direct beam), binarizing will give much increased relevance to the weak reflections further out. However, this effectively deletes the potentially valuable intensity information in the patterns, which would reduce the best-case precision significantly. This is due to minute variations in orientation having a large impact on the intersection of Ewald's sphere with the relrods, increasingly so further out in the ZOLZ or HOLZs. These variations in intensity is what allowed Corrêa et al. [44] to achieve their impressive precision, and this information is lost by binarizing. It is especially detrimental in less symmetric crystals, where diffraction patterns are less unique when intensities are disregarded.

During the final stages of development, a bug was found in the interpolation code. This bug might be the cause for some of the discrepancies in results with the new algorithm, where the correlation scores did not always converge to the optimum. While simple to fix, it was not discovered early enough to have time for new runs. The bug is related to the triangle assignment of resampled points. The interpolation is based on finding which triangle *within the convex hull* of the coarse points each fine point resides in. No issues occurred in the interior, but along the edges of the reduced zone, points were at risk of being outside the convex hull. Along straight edges, the assignment is correct, but along edges with a bend there will be resampled points on the outside. The bug is this: the `tri.find_simplex`-function in line 15 in Listing 3.5.1, which normally returns the index of the triangle the requested point resides in, simply returns -1 instead. A fix is shown in Listing 5.4.1, intended to slot in between lines 15 and 16 in Listing 3.5.1.

```
# Account for simplices outside the convex hull
outside = simplex < 0
# Just bump the points a little closer to the origin,
# and assume this moves them within a close triangle
simplex[outside] = tri.find_simplex(new_xy[outside] * 0.99)
```

Listing 5.4.1: A possible fix for the interpolation bug with points outside of the convex hull. These lines are intended to be slotted in between lines 15 and 16 in Listing 3.5.1.

This behavior of returning -1 is described in the documentation, but is not displayed during runtime as a warning or error. The programmer must handle this

[4]https://github.com/pyxem/pyxem/pull/1076#issuecomment-2117352228

themselves. The issue is that, in Python, -1 is a valid index, and returns the last element of the iterable. This made all points along a bend receive values as if they were part of the last triangle in the list, rather than e.g. extrapolating the closest triangle. For $m\bar{3}m$, and the sampling with Orix, the final triangle in the list is the one containing the [1 0 1] vertex. This region often has a large correlation value, which was then incorrectly used for all the points along the edge between [1 0 1] and [1 1 1].

CHAPTER

SIX

CONCLUSION

The overall aim of this thesis was to improve orientation analysis with SPED, by writing new code based on the Pyxem suite that establish the full gonio-sample-crystal geometry and use it for the navigator tool `tiltlib`. The tool allows users to easily traverse crystallographic space given the sample and gonio layout. Additionally, reducing the runtime of TM is required for improving practical use, and addressing more demanding cases such as low-symmetry phases without compromising on angular resolution.

The first aim, developing a tool to predict tilt angles for zone axis alignment, was performed and culminated in `tiltlib`. Accuracy testing based on SPED scans of a polycrystalline FCC Ag sample, taken at four 5° intervals and orientation mapped with Pyxem, had convincing results indicating a sufficient accuracy. The tilt axis was identified based of a limited SPED tilt series, comparing all combinations of scans at different tilt angles to attain the position. From the combined set of over 400,000 misorientation angles, the bottom $\frac{1}{3}$ were selected, allowing the tilt axis identification algorithm to accurately determine the position even with the limited precision of TM. In the JEOL JEM 2100F TEM used in this work, the tilt axis was determined to lie 30° - Θ from the x-axis in the scan frame, where Θ is the scan rotation.

With the tilt axis identified, and the orientations mapped using TM with Pyxem, a test was performed to verify the accuracy of `tiltlib`. A region of the Ag sample was identified in two scans at different tilts, and the zone axis of this region was extracted from the orientation map of one scan. This zone axis was set as the target of the same region in the other scan. The alignment was 1.6° off when restricting the search to a single tilt axis. By comparing more different regions between the two scans, the alignment was between 1° and 2° for multiple differently oriented grains.

The polycrystalline Ag sample used in this thesis had many clear $\Sigma3$ twins, which have a known 60° misorientation around $\langle 1\,1\,1 \rangle$. These were used to estimate the accuracy of the orientation maps. The TM orientations were within 2° to 4° of the known 60° misorientation, which is higher than expected (1°). Many of the grains within the scanned region and at different tilts, displayed misindexation, which is detrimental to accurate analysis. Despite both misindexation and the relatively imprecise orientation maps, the tilt axis identification algorithm was robust enough to accurately and precisely determine the tilt axis, and thereby the sample-gonio relation.

Additional experimental tests were performed using another cubic sample, here polycrystalline LMNO. The tests include re-inserting the sample into the sample holder, and were performed on two different TEMs by different operators. The results of these tests on three different regions show an accuracy of `tiltlib` of 2° to 5°. This is well within the range required to observe the Laue circle, easily allowing the operator to manually align the final few degrees. Tilt axis identification by means of a tilt series gave good results, and seems to be a robust method without requiring cumbersome manual zone axis alignment or manually following Kikuchi bands (which are suppressed in SPED) by experienced operators.

The tool, `tiltlib`, is available to install through PyPI, and on GitHub[1]. It was successfully used for predicting tilt angles for aligning both single grains and multiple grains simultaneously to given zone axes. Multiple visualization tools were developed, such as IPFs scatterplots and colormaps, and interactive versions exploration of the data. Additionally, the mean zone axis is easily calculated with `mean_zone_axis`, which is useful to get a more precise estimate of the orientation than e.g. the color maps.

The second aim, decreasing the runtime of TM without changing the angular resolution or accuracy, i.e. the returned values, was evaluated by running TM on four datasets, with three different methods. The methods were 1) brute-force dense template bank, 2) Pyxem's template pre-selection, using azimuthal integration and radial correlation, and 3) the new two-step correlation interpolation approach developed in this work. The four datasets were chosen to cover a variety of point groups and sizes of the symmetry reduced zone, as well as testing perfect simulated data and realistic experimental data with noise effects. The datasets were an experimental SPED scan of Ag ($m\bar{3}m$), and simulated datasets of copper ($m\bar{3}m$), graphite ($6/mmm$), and $AuAgTe_4$ ($2/m$). Three metrics were used to evaluate the similarity of the pre-selection algorithms (2 and 3) with the dense results (1). These were the correlation score similarity, orientation similarity, and equality length.

The results indicated little to no improvement of the developed two-step approach compared to the existing polar algorithm. The new proposed algorithm generally had longer runtimes, and was less similar to the brute-force dense template bank than the current algorithm was. One possible usecase where the new algorithm might outperform the existing one, is for very large template banks, e.g. when using crystals with low symmetry and small angular distance between simulations. With the developed two-step algorithm, the simulated hexagonal graphite dataset did not conform to the expected output for any of the three evaluated metrics, indicating some error happened somewhere in the new algorithm. A possible cause, and fix, for this discrepancy is outlined in the discussion (Listing 5.4.1).

The navigation tool was successfully developed and tested. With the future possibility of automatic control of the TEM for more efficient automatic data aquisition, and faster and more accurate orientation mapping with TM all combined with `tiltlib`, automatic investigation of materials is more accessible and convenient, facilitating further research on crystals using the TEM.

[1]`https://www.github.com/viljarjf/tiltlib`

CHAPTER
SEVEN

FUTURE WORK

The aim of this work was to improve crystal orientation analysis with SPED and TM, and led to two main findings: i) the navigation tool `tiltlib` that extend the use of TM results beyond orientation maps and further orientation analysis, and ii) that in general, the proposed 2-step TM-approach with pre-selection based on a course sampling of orientation space before fine sampling around interpolated correlation scores. For the latter, there are some suggestions for future work made in the discussion chapter (for example considering other interpolation routines and dedicated studies for low-symmetry classes). In this chapter, the suggestions for future work are mostly related to i) the navigation tool.

`tiltlib` is a program meant to facilitate easier exploration of crystalline materials in the TEM, and as such one could be very general and say that more different materials with different symmetries can be explored with it. This chapter will instead focus on specific usecases and expansions of `tiltlib`, and orientation mapping as a whole.

7.1 Automation

The step from predicting tilt angles to automatically aligning to those angles is quite small. Automatic TEM control with open-source software should be fairly straightforward to implement with e.g. PyJEM [37], SerialED and PyED[38].

However, as discussed, the accuracy of `tiltlib` is only within a few degrees; enough to see the Laue circle, but not enough for lattice imaging in HRTEM imaging for grain boundary analysis. Automatic alignment on the Laue circle has been proposed before [11], and is implemented in e.g. KSpaceNavigator [33], and is as such a possible avenue for a closed-loop automatic alignment control system. Note that the technique is recently patented[13–15] despite being known for half a century[81, p. 18], so care must be taken to avoid legal action.

To facilitate automatic alignment, orientation mapping should be made possible when at the microscope. Currently, the TM process is too time-consuming for this to be feasibly performed at scale, hence why speedup of TM is important. By limiting the scan to fewer probe position, and larger step sizes, the dataset can become manageable. Automatic grain segmentation, reducing the number of orientations to consider from thousands of pixels down to e.g. 50 mean grain orientations, would allow the program to choose which grain to align to any given zone axis, similarly to Mathisen's zone axis alignment program [49]. A fully automatic approach, where the user simply provides a sample, a `.cif`-file, and a zone axis, seems highly possible within a few years.

An alternative approach is to use SED or SAED rather than SPED, to get a single diffraction pattern. TM to determine this single orientation is certainly possible while at the microscope, meaning `tiltlib` could be used in-place even without automatic TEM control. This would likely increase the accuracy as well, since re-inserting the sample into the sample holder is observed to cause misalignment.

7.2 `tiltlib` Expansion

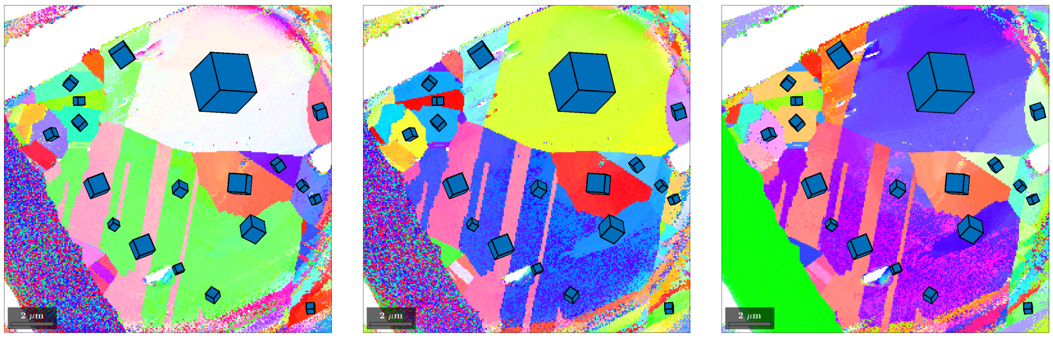
Apart from the already discussed support for single diffraction pattern samples from e.g. SAED, `tiltlib` could be improved and expanded further. One major limitation is the computation time of zone axis alignment, which can take minutes with the current implementation. As discussed in Chapter 5.3.3, this could be mitigated by optimizing on the mean orientation, at the cost of losing the possibility to align on multiple grains simultaneously. Apart from algorithmic considerations, the code itself should be looked more into, as Orix is a performance bottleneck at the moment. By doing more low-level operations on the quaternions directly, a significant speedup should be possible. With a faster optimization, the tool is even more usable directly while using the TEM.

`tiltlib` has presently been tested on $m\bar{3}m$ only. Simple tests on e.g. hexagonal data should be performed to ensure the conventions are followed. Non-centrosymmetric crystals could also be interesting to test, as this would confirm if the positive z-direction is aligned correctly. The $ErMnO_3$ orientation maps from Mathisen [49] seems like a promising candidate, as $ErMnO_3$ has the hexagonal non-centrosymmetric point group 6mm, and both experimental tilt series, finished orientation maps, and multi-slice simulations are available.

More visualization tools could be beneficial for `tiltlib`, e.g. a IPF without the symmetry reduced projection, where low-index zone axes are shown alongside a rectangle showing the range of the tilt holder. A similar visualization is available in multiple existing tools for zone axis alignment, indicating this is a practical addition.

Currently, zone axis alignment is the main functional feature of `tiltlib`. Other uses are possible with the full description of the orientation relations in the TEM. One possibility is to allow the user to supply an axis of rotation, e.g. a `Miller`, and the angle to rotate around that axis. `tiltlib` could then compute the necessary tilt angles to move the sample equivalently. This could be useful for boundary alignment, or tilting along a boundary.

Finally, `tiltlib` should be tested with other sample holders, notably the tilt-rotate holder. This unique sample holder geometry has seen little usage for zone axis alignment tools, and operators usually find them difficult to wrap their heads around. In theory, as the rotation axis has a much larger angular range, the available orientation space should be much larger for a tilt-rotate holder than a double-tilt holder. `tiltlib` is designed with support for arbitrary tilt axes in mind, meaning it should work nicely. Preliminary investigations were performed in this work and the preliminary study[45], using the orientation map shown in Figure 7.2.1. Although the mathematics are the same, in practice the usage of a tilt-rotate holder is more complicated than a double tilt. The difficulty comes mainly from the displacement of the sample when rotating. This can be somewhat mitigated by changing the scan rotation with the rotation, but the sample will still move around in the frame if the rotation axis does not coincide with the center of the frame. `tiltlib` supports using a tilt-rotate holder, but more work is needed to confirm if it works correctly.



(a) **(b)** **(c)**

Figure 7.2.1: IPFs from a SPED scan of the Ag sample, with unit cells displayed on large grains. The unit cells were added using MTEX[31]. This scan is part of a rotation series, i.e. a tilt series using the rotation axis of the tilt-rotate sample holder. (a), (b) and (c) are IPF-x, y and z, respectively.

7.3 Orientation Refinement

The pre-selection algorithm developed (Chapter 3.5), results presented (Chapter 4.3) and discussed (Chapter 5.4) in this thesis were mainly focused on decreasing the runtime of TM rather than increasing the precision/accuracy. If orientation maps are more accurate, `tiltlib` is more accurate. As such, the approach of Corrêa et al. [44] seems promising, as they achieved orders of magnitude better precision than is expected from TM [18]. Orientation refinement on intensities is very promising, but also time-consuming, as Corrêa et al. spent multiple days on their computations (for comparison, TM often takes mere minutes, for much larger datasets). Method- and code-optimization is necessary to feasibly use this for datasets of significant sizes. This could be implemented as part of Pyxem, to keep the code open-source and accessible, and to increase the likelihood of someone seeing possible optimizations.

Corrêa et al. manually index the diffraction patterns before refinement, which is undesirable from an automation perspective. They suggest using TM for initial indexation, before employing the residual optimization for refinement. For this, one might employ the binarization step Cautaerts suggests, to increase the initial accuracy without increasing computation time much. However, as the discussion suggests, this is case dependent, and can worsen the confidence off-zone as intensities are not taken into account. Another approach to increase the importance of reflections further out, is to simply mask out the inner reflections. This is made easy with the new API for TM in Pyxem v0.19, where the present work has contributed to (see Appendix C), as shown in Listing 7.3.1. The output of running this script is shown in Figure 7.3.1.

```python
from hyperspy import api as hs
from pyxem.signals import Diffraction2D


def mask_inner(data, r):
    out = data.copy()
    out[:r, :] = 0
    return out


data: Diffraction2D = hs.load("path/to/your/data.hspy")

# Use npt=140 as this is 100 * sqrt(2),
# so we know the (approximate) radius in pixel coordinates
polar = data.get_azimuthal_integral2d(npt=140, mean=True)

polar.map(mask_inner, r=40, inplace=True)

polar.plot(norm="symlog", cmap="viridis")
```

Listing 7.3.1: An example showing how one can easily mask a radial range for TM using Pyxem v0.19. The output is shown in Figure 7.3.1.

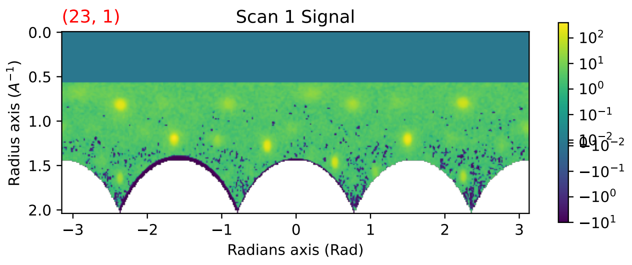
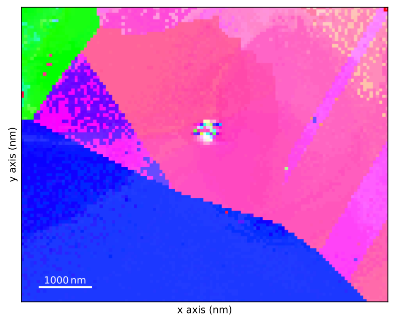


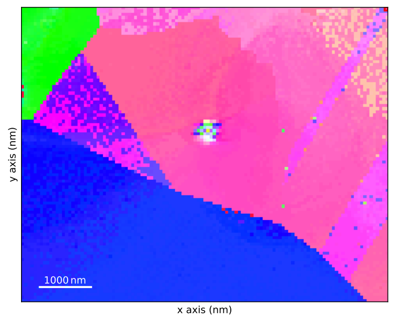
Figure 7.3.1: The output of running the script in Listing 7.3.1, showing how an inner radial range is masked out.

Alternatively, one can increase the relative intensity (and consequently the relative importance for NCC) of far reflections by simply using `mean=False` in `Diffraction2D.azimuthal_integral2d`, which is the default in Pyxem 0.19. As the regions near the center have much more pixels in polar space than regions further out, the intensity in polar space is much reduced when the integrated intensity is preserved. This increases the relative intensity of reflections proportional to r^2, giving increased significance to far reflections in the NCC. Previous versions of Pyxem performs polar unwrapping as a part of the TM process, which the user cannot control, but it effectively used `mean=True`. The difference between using `mean=True` and `mean=False` is shown with IPF-z colormaps in Figure 7.3.2. No obvious improvement can be seen by visual inspection of the IPF colormaps. However, the pre-processing was simple, using only `PolarDiffraction2D.subtract_diffraction_background` with the `polar_median` method. More considered pre-processing might yield other results.

Corrêa et al.'s approach is certainly possible to implement in Pyxem. In fact, much of the functionality is already in place for the refinement step, and all that should be necessary is to combine existing methods and classes. A good place



(a)



(b)

Figure 7.3.2: IPF-z of a crop of scan 1, using TM with Pyxem 0.19 with different polar unwrapping strategies. (a) using `mean=False`. (b) using `mean=True`.

to start would be the `DiffractionVectors2D`-class, and the various peak- and intensity-finding functionality associated with it, as well as the different indexation generators. A large optimization can be made with clever use of Diffsims, by assuming 1. small deviance between the refined orientation and the initial orientation found by TM and 2. precession. If these two conditions are met, then the reflections are likely to only change in intensity rather than changing in indices. The hkl-values are therefore constant, and only the intensity needs re-calculating. This skips a significant computational bottleneck in the diffraction simulation, namely finding the intersection between Ewald's sphere and all reflections. With these optimizations, this facilitates very fast diffraction simulation, which is necessary for refinement; a pre-simulated template bank is unlikely to contain the very best possible simulation for a given list of reflections. The optimization algorithm for refinement, i.e. difference between simulated and measured intensity for each hkl, is fairly straightforward to implement after Corrêa et al.

One limitation with their approach is the simulations, which are based on a dynamical two-beam approximation. These are more accurate to the data, as electron diffraction is indeed dynamical, but comes at a significant cost in terms of computation time, compounded with the vastly expanded parameter space. As briefly mentioned in Chapter 2.5.2, Python packages for this exists, e.g. `py_multislice`[56], and writing an interface to Diffsims should allow these to be used with TM in Pyxem. Minimizing the size of the parameter space is crucial, as e.g. the thickness parameter was optimized to a constant throughout the sample in [44]. Parameter space minimization is also used to good effect in Pyxem, where the analytical precession estimation by a Lorentzian is possible by setting the extinction length to π times the thickness, which makes the expression independent on thickness[64].

As dynamical electron diffraction simulations are much more computationally expensive than kinematic ones, optimizations in the simulation step should be sought. When simulating a bank with precession, regardless of whether they are dynamical or not, one could achieve faster computation times by avoiding recomputing previous orientations. The algorithm would look something like this:

Simulate a bank without precession Using any simulation method, e.g. simple kinematic or multislice.

Sum simulations in a ring As this is essentially what precession entails.

The radius of the ring is determined by the rocking angle. A visual outline of the algorithm is outlined in Figure 7.3.3 for a single orientation.

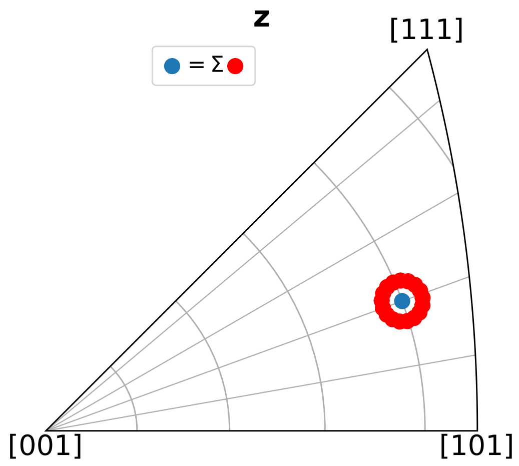


Figure 7.3.3: A visual representation of an approximation of precession, where a precessed simulation at the blue orientation can be seen as a sum of the red unprecessed simulations. Shown as IPF-z.

Another possible avenue for refinement, in the form of lowering the misindexation rate, is to automatically determine if the orientation found by TM is likely to be misidentified. This is commonly done by looking at the reliability index. If the orientation is difficult to correctly index, `tiltlib` could suggest a different tilt, where the orientations are expected to no longer be as difficult, and re-index at the new orientation. Aune [63] proposes and investigates a method for automatically discarding misindexed orientations from a tilt series, which could be integrated into `tiltlib` for this purpose. The process is outlined in Figure 7.3.4, showing how a slight tilt can make difficult orientations easier to index correctly. The results from the tilt with easily indexed diffraction patterns could be used to make a new template bank, containing only the orientations that are possible given the easily indexed orientation.

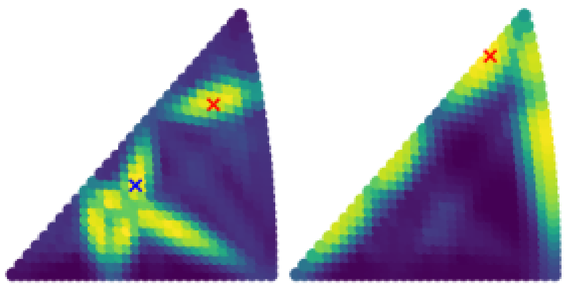


Figure 7.3.4: The correlation score plotted in orientation space for two orientations, marked with red crosses. The blue cross in the left map is the point with the highest correlation score. This misindexation is mitigated with a slight tilt, shown in the right map.

REFERENCES

[1] Christopher Hammond. *The Basics of Crystallography and Diffraction.* 3rd ed. Oxford, United Kingdom: International Union of Crystallography and Oxford University Press, 2009.

[2] Brent Fultz and James Howe. *Transmission Electron Microscopy and Diffractometry of Materials.* 4th ed. Springer Berlin, Heidelberg, 2012.

[3] Charles Kittel. *Introduction to Solid State Physics.* 8th ed. Wiley, 2004. ISBN: 9780471415268.

[4] "Theory of Electron Diffraction". In: *Transmission Electron Microscopy: Physics of Image Formation.* New York, NY: Springer New York, 2008, pp. 270–325. ISBN: 978-0-387-34758-5. DOI: `10.1007/978-0-387-40093-8_7`.

[5] S.I. Wright and D.J. Dingley. "Orientation imaging in the transmission electron microscope". In: *Materials Science Forum* 273-275 (1998), pp. 209–214. DOI: `10.4028/www.scientific.net/msf.273-275.209`.

[6] J.-J. Fundenberger et al. "Polycrystal orientation maps from TEM". In: *Ultramicroscopy* 96.2 (2003), pp. 127–137. ISSN: 0304-3991. DOI: `10.1016/S0304-3991(02)00435-7`.

[7] Qing Liu. "A simple and rapid method for determining orientations and misorientations of crystalline specimens in TEM". In: *Ultramicroscopy* 60.1 (1995), pp. 81–89. ISSN: 0304-3991. DOI: `10.1016/0304-3991(95)00049-7`.

[8] Qing Liu. "A simple method for determining orientation and misorientation of the cubic crystal specimen". In: *Journal of Applied Crystallography* 27.5 (1994), pp. 755–761. DOI: `10.1107/S0021889894002062`.

[9] Liu Qing, Meng Qing-Chang, and Hong Bande. "Calculation of tilt angles for crystal specimen orientation adjustment using double-tilt and tilt-rotate holders". In: *Micron and Microscopica Acta* 20.3 (1989), pp. 255–259. ISSN: 0739-6260. DOI: `10.1016/0739-6260(89)90058-0`.

[10] Liu Qing. "An equation to determine the practical tilt angle of a double-tilt specimen holder and its application to transmission electron microscopy". In: *Micron and Microscopica Acta* 20.3 (1989), pp. 261–264. ISSN: 0739-6260. DOI: `10.1016/0739-6260(89)90059-2`.

[11] J. Jansen, M.T. Otten, and H.W. Zandbergen. "Towards automatic alignment of a crystalline sample in an electron microscope along a zone axis". In: *Ultramicroscopy* 125 (2013), pp. 59–65. ISSN: 0304-3991. DOI: `10.1016/j.ultramic.2012.09.010`.

[12] Daliang Zhang et al. “Atomic-resolution transmission electron microscopy of electron beam–sensitive crystalline materials”. In: *Science* 359.6376 (2018), pp. 675–679. DOI: 10.1126/science.aao0865.

[13] Daliang Zhang et al. “Transmission electron microscope sample alignment system and method”. US 10067078 B1. 2018.

[14] John J. Flanagan et al. “Method and system for automatic zone axis alignment”. US 11211222 B2. 2019.

[15] Zhenxin Zhong. “Method and system for zone axis alignment”. US 11024480 B2. 2018.

[16] E F Rauch and L Dupuy. “Rapid spot diffraction patterns identification through template matching”. eng. In: *Archives of metallurgy and materials* 50.1 (2005), pp. 87–99. ISSN: 1733-3490.

[17] A. Morawiec and E. Bouzy. “On the reliability of fully automatic indexing of electron diffraction patterns obtained in a transmission electron microscope”. In: *Journal of Applied Crystallography* 39.1 (Feb. 2006), pp. 101–103. DOI: 10.1107/S0021889805032966.

[18] Edgar F. Rauch and Laurent Dupuy. “Comments on ‘On the reliability of fully automatic indexing of electron diffraction patterns obtained in a transmission electron microscope’ by Morawiec & Bouzy (2006)”. In: *Journal of applied crystallography* 39.1 (2006), pp. 104–105. ISSN: 1600-5767.

[19] John Francis Flanagan. “System for orienting a sample using a diffraction pattern”. US 9978557 B2. 2016.

[20] Gary W. Paterson et al. “Fast Pixelated Detectors in Scanning Transmission Electron Microscopy. Part II: Post-Acquisition Data Processing, Visualization, and Structural Characterization”. In: *Microscopy and Microanalysis* 26.5 (2020), pp. 944–963. DOI: 10.1017/S1431927620024307.

[21] Colin Ophus. “Four-Dimensional Scanning Transmission Electron Microscopy (4D-STEM): From Scanning Nanodiffraction to Ptychography and Beyond”. In: *Microscopy and Microanalysis* 25.3 (2019), pp. 563–582. DOI: 10.1017/S1431927619000497.

[22] Benjamin H. Savitzky et al. “py4DSTEM: A Software Package for Four-Dimensional Scanning Transmission Electron Microscopy Data Analysis”. In: *Microscopy and Microanalysis* 27.4 (2021), pp. 712–743. DOI: 10.1017/S1431927621000477.

[23] Daliang Zhang et al. In: *Zeitschrift für Kristallographie* 225.2-3 (2010), pp. 94–102. DOI: doi:10.1524/zkri.2010.1202.

[24] Bin Wang, Xiaodong Zou, and Stef Smeets. “Automated serial rotation electron diffraction combined with cluster analysis: an efficient multi-crystal workflow for structure determination”. In: *IUCrJ* 6.5 (Sept. 2019), pp. 854–867. DOI: 10.1107/S2052252519007681.

[25] Chunfei Li and David B. Williams. “Application of automated crystallography for transmission electron microscopy in the study of grain-boundary segregation”. In: *Micron* 34.3 (2003), pp. 199–209. ISSN: 0968-4328. DOI: 10.1016/S0968-4328(03)00026-X.

[26] P. Moeck et al. “High spatial resolution semi-automatic crystallite orientation and phase mapping of nanocrystals in transmission electron microscopes”. In: *Crystal Research and Technology* 46.6 (2011), pp. 589–606. DOI: 10.1002/crat.201000676.

[27] Duncan Johnstone et al. *pyxem/pyxem: v0.18.0*. Version v0.18.0. May 2024. DOI: 10.5281/zenodo.11162592.

[28] E.F. Rauch and M. Véron. “Automated crystal orientation and phase mapping in TEM”. eng. In: *Materials characterization* 98 (2014), pp. 1–9. ISSN: 1044-5803.

[29] Niels Cautaerts et al. “Free, flexible and fast: orientation mapping using the multi-core and GPU-accelerated template matching capabilities in the python-based open source 4D-STEM analysis toolbox Pyxem”. In: (2021). eprint: 2111.07347.

[30] Duncan N. Johnstone et al. “Density-based clustering of crystal (mis)orientations and the *o*rix Python library”. In: *Journal of Applied Crystallography* 53.5 (Oct. 2020), pp. 1293–1298. DOI: 10.1107/S1600576720011103.

[31] Robert Krakow et al. “On three-dimensional misorientation spaces”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science* 473 (Oct. 2017), p. 20170274. DOI: 10.1098/rspa.2017.0274.

[32] N. Cautaerts, R. Delville, and D. Schryvers. “ALPHABETA: a dedicated open-source tool for calculating TEM stage tilt angles”. In: *Journal of Microscopy* 273.3 (2019), pp. 189–198. DOI: 10.1111/jmi.12774.

[33] T. Duden, A. Gautam, and U. Dahmen. “KSpaceNavigator as a tool for computer-assisted sample tilting in high-resolution imaging, tomography and defect analysis”. In: *Ultramicroscopy* 111.11 (2011), pp. 1574–1580. ISSN: 0304-3991. DOI: 10.1016/j.ultramic.2011.08.003.

[34] Rui-Xun Xie and Wen-Zheng Zhang. “τompas: a free and integrated tool for online crystallographic analysis in transmission electron microscopy”. In: *Journal of Applied Crystallography* 53.2 (Apr. 2020), pp. 561–568. DOI: 10.1107/S1600576720000801.

[35] Matthew Olszta and Kevin Fiedler. “Nanocartography: Planning for success in analytical electron microscopy”. In: *Microscopy and Microanalysis* (May 2022).

[36] Kevin Fiedler and Matthew Olszta. “NanoCartography: Mathematics of Crystal Orientation in Double Tilt Holders”. In: *Microscopy and Microanalysis* 26.S2 (2020), pp. 252–253. DOI: 10.1017/S1431927620013963.

[37] JEOL. *PyJEM*. 2024. URL: https://pyjem.github.io/PyJEM/.

[38] Martin Schorb et al. “Software tools for automated transmission electron microscopy”. In: *Nature Methods* 16.6 (June 2019), pp. 471–477. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0396-9.

[39] A. Morawiec et al. “Orientation precision of TEM-based orientation mapping techniques”. In: *Ultramicroscopy* 136 (2014), pp. 107–118. ISSN: 0304-3991. DOI: 10.1016/j.ultramic.2013.08.008.

[40] Kevin R Fiedler et al. “Evaluating Stage Motion for Automated Electron Microscopy”. In: *Microscopy and Microanalysis* 29.6 (Oct. 2023), pp. 1931–1939. ISSN: 1431-9276. DOI: 10.1093/micmic/ozad108.

[41] R. Vincent and P.A. Midgley. “Double conical beam-rocking system for measurement of integrated electron diffraction intensities”. eng. In: *Ultramicroscopy* 53.3 (1994), pp. 271–282. ISSN: 0304-3991.

[42] D. Dingley. “Orientation Imaging Microscopy for the Transmission Electron Microscope”. In: *Microchimica Acta* 155 (Jan. 2006), pp. 19–29. DOI: 10.1007/s00604-006-0502-4.

[43] Tina Bergh et al. “Scanning Electron Diffraction: To Precess or not to Precess?” In: *Microscopy and Microanalysis* 29.Supplement 1 (July 2023), pp. 2101–2102. ISSN: 1431-9276. DOI: 10.1093/micmic/ozad067.1088.

[44] Leonardo M. Corrêa et al. “High precision orientation mapping from 4D-STEM precession electron diffraction data through quantitative analysis of diffracted intensities”. In: *Ultramicroscopy* 259 (2024), p. 113927. ISSN: 0304-3991. DOI: 10.1016/j.ultramic.2024.113927.

[45] Viljar Johan Femoen. “Relating holder axes and template matching for grain orientation analysis”. In: *Institute of Physics, NTNU* (Dec. 2023). Project thesis.

[46] Marc De Graef. *Introduction to Conventional Transmission Electron Microscopy.* Cambridge University Press, 2003. DOI: 10.1017/CBO9780511615092.007.

[47] Hans-Joachim Bunge. *Texture Analysis in Materials Science: Mathematical Methods.* Digital Edition. 2015.

[48] Anthony D. Rollett and Katayun Barmak. *Physical Metallurgy.* Ed. by David E. Laughlin and Kazuhiro Hono. Fifth Edition. Oxford: Elsevier, 2014. ISBN: 978-0-444-53770-6. DOI: 10.1016/B978-0-444-53770-6.00011-3.

[49] Anders Christian Mathisen. “Scanning Precession Electron Diffraction of Ferroelectric Polycrystalline h-ErMnO3”. eng. MA thesis. 2023. URL: https://hdl.handle.net/11250/3092879.

[50] Mike Boyle et al. *moble/quaternion: Release v2022.4.4.* Version v2022.4.4. Dec. 2023. DOI: 10.5281/zenodo.10237609.

[51] Landis Markley et al. “Averaging Quaternions”. In: *Journal of Guidance, Control, and Dynamics* 30 (July 2007), pp. 1193–1196. DOI: 10.2514/1.28949.

[52] 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 (Oct. 2014), p. 075013. DOI: 10.1088/0965-0393/22/7/075013.

[53] Anna Yershova, Steven M. LaValle, and Julie C. Mitchell. “Generating Uniform Incremental Grids on SO(3) Using the Hopf Fibration”. In: *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics.* Ed. by Gregory S. Chirikjian et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 385–399. ISBN: 978-3-642-00312-7. DOI: 10.1007/978-3-642-00312-7_24.

[54] Steven M. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006. ISBN: 0521862051.

[55] John M Cowley. *Diffraction physics*. eng. 3rd rev. ed. Amsterdam: Elsevier, 1995. Chap. 11. ISBN: 0444822186.

[56] Hamish Galloway Brown and Thomas Aarholt. *HamishGBrown/py_multislice: For publication with MeasureIce*. Version v1.0.1. Dec. 2021. DOI: 10.5281/zenodo.5762736.

[57] Håkon Wiik Ånes et al. *pyxem/orix: orix 0.12.1.post0*. Version v0.12.1.post0. Apr. 2024. DOI: 10.5281/zenodo.11068626.

[58] Duncan Johnstone et al. *pyxem/diffsims: diffsims 0.5.2*. Version v0.5.2. May 2023. DOI: 10.5281/zenodo.7962969.

[59] Håkon Wiik Ånes et al. *pyxem/kikuchipy: kikuchipy 0.9.0*. Version v0.9.0. Nov. 2023. DOI: 10.5281/zenodo.10069324.

[60] Francisco de la Peña et al. *hyperspy/hyperspy: v2.0rc0*. Version v2.1.0. May 2024. DOI: 10.5281/zenodo.11148112. URL: https://zenodo.org/records/11148112.

[61] Olivier Grisel et al. *scikit-learn/scikit-learn: Scikit-learn 1.3.2*. Version 1.3.2. Oct. 2023. DOI: 10.5281/zenodo.10039710.

[62] Garrett Baucom et al. "Nanoscale Phase and Orientation Mapping in Multiphase Polycrystalline Hafnium Zirconium Oxide Thin Films Using 4D-STEM and Automated Diffraction Indexing". In: *Small methods* (May 2024), e2400395. DOI: 10.1002/smtd.202400395.

[63] Kaja Eggen Aune. "Optimizing orientation analysis of polycrystalline $LiNi_{0.5}Mn_{1.5}O_4$ cathode material using scanning precession electron diffraction". eng. Unpublished. MA thesis. 2024.

[64] Lukáš Palatinus et al. "Specifics of the data processing of precession electron diffraction tomography data and their implementation in the program PETS2.0". In: *Acta Crystallographica Section B* 75.4 (2019), pp. 512–522. DOI: 10.1107/S2052520619007534.

[65] I -K Suh, H Ohta, and Y Waseda. In: *Journal of Materials Science* 23.High-temperature thermal expansion of six metallic elements measured by dilatation method and X-ray diffraction Sample: at T = 293 K (1988), pp. 757–760.

[66] 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.

[67] Robin Sibson. "Locally Equiangular Triangulations". In: *Comput. J.* 21 (1978), pp. 243–245.

[68] Juyong Zhang et al. "Local barycentric coordinates". In: *ACM Trans. Graph.* 33.6 (Nov. 2014). ISSN: 0730-0301. DOI: 10.1145/2661229.2661255.

[69] Jaime. *Speedup scipy griddata for multiple interpolations between two irregular grids*. 2014. URL: https://stackoverflow.com/a/20930910.

[70] Allen G. Jackson. *Handbook of crystallography: for electron microscopists and others*. Springer-Verlag, 1991.

[71] H E Swanson and E Tatge. "Standard X-ray diffraction powder patterns". In: *National Bureau of Standards (U.S.), Circular* 359 (1953), p. 1.

[72] O Hassel. "Ueber die Kristallstruktur des Graphits". In: *Zeitschrift fuer Physik* 25 (1924), pp. 317–337.

[73] F Pertlik. "Kristallchemie natuerlicher Telluride. I: Verfeinerung der Kristallstruktur des Sylvanits, Au Ag Te4". In: *Tschermaks Mineralogische und Petrographische Mitteilungen* 33 (1984), pp. 203–212.

[74] The Python Packaging Authority. *Packaging Python Projects*. Accessed: 30th of May, 2024. Apr. 2024. URL: `https://packaging.python.org/en/latest/tutorials/packaging-projects/`.

[75] Takeshi KOMIYA et al. *sphinx-doc/sphinx: Sphinx 7.3.7*. Version v7.3.7. Apr. 2024. DOI: `10.5281/zenodo.10995501`.

[76] Atsuko Kobayashi et al. "Technical note: A tool for determining rotational tilt axis with or without fiducial markers". In: *Ultramicroscopy* 110.1 (2009), pp. 1–6. ISSN: 0304-3991. DOI: `10.1016/j.ultramic.2009.08.007`.

[77] H.G. Brown et al. "A new method to detect and correct sample tilt in scanning transmission electron microscopy bright-field imaging". In: *Ultramicroscopy* 173 (2017), pp. 76–83. ISSN: 0304-3991. DOI: `10.1016/j.ultramic.2016.11.024`.

[78] Michael Atkinson et al. *MechMicroMan/DefDAP: v0.93.6*. Version v0.93.6. Nov. 2023. DOI: `10.5281/zenodo.10160238`.

[79] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. Chap. 1.2.3.

[80] Qiwei Shi et al. "Accuracy assessment of crystal orientation indexations by EBSD". In: *Measurement Science and Technology* 35.4 (Jan. 2024). DOI: `10.1088/1361-6501/ad204d`.

[81] J.W Edington. *Electron Diffraction in the Electron Microscope*. eng. 1st ed. 1975. Philips Technical Library, Monographs in Practical Electron Microscopy in Materials Science. London: Macmillan Education UK : Imprint: Red Globe Press, 1975. ISBN: 1-349-02595-X.

APPENDICES

The first chapter lists the code used for performance evaluation of the TM-preselection algorithm. The second chapter show extended results and figures supplementary to the thesis. The third chapter lists and comments upon contributions to open-source software made as part of the work done during this project. The final chapter contains the abstract accepted for oral presentation at EMC 2024, to be held by Antonius van Helvoort.

APPENDIX
A

TEMPLATE MATCHING PRE-SELECTION ALGORITHM CODE

This chapter contains listings of the code used to run tests for the TM-pre-selection algorithm discussed in this thesis. The font size is kept small to fit more of the code in fewer pages.

A.1 Results Data File

This is an example of the file written as output after testing, containing the results and metadata from the test.

```json
{
    "dense": {
        "runtime_mean": 7249603716.666667,
        "runtime_std": 104926601.59386683,
        "runtimes": [
            7155551100,
            7445231500,
            7332224800,
            7211154000,
            7179539000,
            7173921900
        ],
        "total_runtime": 43.4976223,
        "runs": 6
    },
    "refined": {
        "correlation_mean": 85.22341485373194,
        "correlation_std": 114.2972650689875,
        "orientation_mean": 9.834190280111782,
        "orientation_std": 16.310914001091085,
        "runtime_mean": 2231131333.3333335,
        "runtime_std": 1182358903.0371683,
        "runtimes": [
            4860438800,
            1623325600,
            1631349100,
            1971628100,
            1686737700,
            1613308700
        ],
        "total_runtime": 13.386788,
        "runs": 6,
        "equality_length_mean": 2.736111111111111,
        "equality_length_std": 4.4581169902510895,
        "equality_length_set_mean": 7.873611111111111,
        "equality_length_set_std": 2.249536560639618
    },
    "old": {
        "correlation_mean": 16.84738847403998,
        "correlation_std": 53.60347776437754,
        "orientation_mean": 3.4698019224642622,
        "orientation_std": 11.256141732474601,
        "runtime_mean": 1794140966.6666667,
        "runtime_std": 49612047.37946442,
        "runtimes": [
            1776323400,
            1740644100,
            1737157200,
            1839436200,
            1873359800,
            1797925100
        ],
        "total_runtime": 10.7648458,
        "runs": 6,
        "equality_length_mean": 7.555555555555555,
        "equality_length_std": 4.297573245736381,
        "equality_length_set_mean": 9.248611111111112,
        "equality_length_set_std": 1.8523628345946845
    },
    "coarse_resolution": 2,
    "coarse_oris_count": 300,
    "fine_resolution": 0.4,
    "fine_oris_count": 6555,
    "n_best": 10,
    "frac_keep": null,
    "n_keep": 50,
    "simulation_generator_parameters": {
        "reciprocal_radius": 1.7,
        "with_direct_beam": false,
        "max_excitation_error": 0.01,
        "accelerating_voltage": 200,
        "approximate_precession": true,
        "precession_angle": 1.0,
        "minimum_intensity": 1e-20,
        "shape_factor_model": "lorentzian"
    },
    "dataset_parameters": {
        "title": "Scan 1",
        "space_group": "SpaceGroup #225 (Fm-3m, Cubic). Symmetry matrices: 192, point sym. matr.: 48",
        "phase_name": "Ag",
        "device": "CPU",
        "num_diffraction_patterns": 720
    },
    "comment": "n_keep sweep"
}
```

A.2 get_data.py

This file contains functions to load the real dataset, and simulate datasets for a given phase.

```python
from hyperspy import api as hs
from pyxem.signals import ElectronDiffraction2D
from orix.crystal_map import Phase
from orix.quaternion import Orientation
from diffsims.generators.simulation_generator import SimulationGenerator
from pathlib import Path

ROOT = Path(__file__).parent


def get_real_data() -> ElectronDiffraction2D:
    data = hs.load("data/centered_calibrated_1_crop.hspy", lazy=True)
    data.metadata.General.title = "Scan 1"
    data = data.inav[::4, ::4]
    return data


def get_simulated_data(phase: Phase, oris: Orientation) -> ElectronDiffraction2D:
    gen = SimulationGenerator(
        minimum_intensity=1e-20, precession_angle=1, approximate_precession=False
    )
    sim = gen.calculate_diffraction2d(
        phase=phase,
        rotation=oris,
        reciprocal_radius=1.7,
        with_direct_beam=False,
        max_excitation_error=0.01,
    )
    get_diffraction_pattern_kwargs = {
        "shape": (128, 128),
        "calibration": 0.02,
        "sigma": 2,
    }
    sig = ElectronDiffraction2D(
        [
            s.get_diffraction_pattern(**get_diffraction_pattern_kwargs)
            for s in sim.irot
        ]
    )
    sig.set_diffraction_calibration(get_diffraction_pattern_kwargs["calibration"])
    sig.calibrate(center=None)
    sig.metadata.General.title = f"Simulated data: {phase.name}"
    return sig
```

A.3 refinedsimulation.py

This file contains the source code for the new algorithm, as a subclass of 'Simulation2D' from Diffsims.

```python
from pyxem.signals.indexation_results import OrientationMap
from diffsims.simulations import Simulation2D
from diffsims.generators.simulation_generator import SimulationGenerator
from scipy.spatial import Delaunay
from orix.quaternion import Orientation
from orix.vector import Vector3d
import numpy as np
from orix.projections import StereographicProjection
from pyxem.utils.indexation_utils import _get_max_n


s = StereographicProjection()


def ori2xy(o) -> tuple[np.ndarray, np.ndarray]:
    y, x = s.vector2xy(o * Vector3d.zvector())
    y = -y
    return x, y


class RefinedSimulation2D(Simulation2D):
    def __init__(
        self,
        orientationmap: OrientationMap,
        simgen: SimulationGenerator,
        oris: Orientation,
        n_keep: int = None,
        frac_keep: int = None,
        simgen_kwargs: dict = None,
    ) -> None:
        if simgen_kwargs is None:
            simgen_kwargs = {}

        num_oris = _get_max_n(oris.size, n_keep, frac_keep)

        ## Set up coordinates for interpolation.
        # Known correlation scores:
        x, y = ori2xy(orientationmap.simulation.rotations)
        xy = np.vstack(
            [
                x.flatten(),
                y.flatten(),
            ]
        ).T

        # Points to interpolate the correlation score
        new_x, new_y = ori2xy(oris)
        new_xy = np.vstack(
            [
                new_x.flatten(),
                new_y.flatten(),
            ]
        ).T

        ## Set up interpolation.
        # Since the data coordinates and interpolated coordinates
        # are always the same, and since we use linear interpolation,
        # we can pre-calculate the barycentric weights for Delaunay triangulation
        # https://stackoverflow.com/questions/20915502
        tri = Delaunay(xy, incremental=False)
        simplex = tri.find_simplex(new_xy)
        vertices = np.take(tri.simplices, simplex, axis=0)
        temp = np.take(tri.transform, simplex, axis=0)
        delta = new_xy - temp[:, -1]
        bary = np.einsum("njk,nk->nj", temp[:, :-1, :], delta)
        weights = np.hstack((bary, 1 - bary.sum(axis=1, keepdims=True)))

        def interpolate(values):
            return np.einsum("nj,nj->n", np.take(values, vertices), weights)

        # Map func to calculate indices
        def get_top_indices(orientationmap_result):
            indices, correlations, _, _ = orientationmap_result.T
            indices = indices.astype(int)

            # Re-order correlations, they need to be in the same order
            # as the Delaunay triangulation was made with
            sorted_indices = np.argsort(indices)
            data = interpolate(correlations[sorted_indices])

            # Only keep the top `num_oris`.
            # Use argpartition, since we don't care about the order
            # (arg since we want indices, partition since we dont care about order)
            k = data.size - num_oris
            inds = np.argpartition(data, k)[k:]
            return inds

        indices = orientationmap.map(
            get_top_indices, inplace=False, lazy_output=True
        )

        # Reduce total simulation count by only using the unique orientations
        unique_indices, reconstruct_indices = np.unique(indices, return_inverse=True)
        self.indices = reconstruct_indices.reshape(indices.data.shape)

        # Perform the simulations, only use unique orientations
        simulations = simgen.calculate_diffraction2d(
            orientationmap.simulation.phases, oris[unique_indices], **simgen_kwargs
        )

        # Store the flattened unique simulations,
        # for use in `self.polar_flatten_simulations`
        self._original_simulations = simulations

        Simulation2D.__init__(
            self,
            simulations.phases,
            simulations.coordinates[self.indices],
            simulations.rotations[self.indices],
            simulations.simulation_generator,
            simulations.reciporical_radius,
        )

    def polar_flatten_simulations(self, radial_axes=None, azimuthal_axes=None):
        """Flattens the simulations into polar coordinates for use in template matching.
        The resulting arrays are of shape (n_simulations, n_spots) where n_spots is the
        maximum number of spots in any simulation.


        Returns
        -------
        r_templates, theta_templates, intensities_templates
        """

        (
            r_templates,
            theta_templates,
            intensities_templates,
        ) = self._original_simulations.polar_flatten_simulations(
            radial_axes, azimuthal_axes
        )
        return (
            r_templates[self.indices],
            theta_templates[self.indices],
            intensities_templates[self.indices],
        )
```

A.4 simulation_comparison.py

The main workhorse of the comparisons. The class `TemplateMatchingComparison` stores the parameters, runs the tests, and calculates statistics from those.

```python
from refinedsimulation import RefinedSimulation2D
from orix.sampling import get_sample_reduced_fundamental
from pyxem.signals import PolarDiffraction2D
from pyxem.signals.indexation_results import OrientationMap
from diffsims.generators.simulation_generator import SimulationGenerator
from orix.crystal_map import Phase
import numpy as np
from tqdm import tqdm

from time import perf_counter_ns
from dataclasses import dataclass, field

DEFAULT_SIMGEN_KWARGS = {
    "reciprocal_radius": 1.7,
    "with_direct_beam": False,
    "max_excitation_error": 0.01,
}


@dataclass
class TemplateMatchingParameters:
    pol: PolarDiffraction2D
    gen: SimulationGenerator
    coarse_res: float
    fine_res: float
    phase: Phase
    n_best: int
    frac_keep: float = None
    n_keep: int = None
    simgen_kwargs: dict = field(default_factory=lambda: {**DEFAULT_SIMGEN_KWARGS})

    def __post_init__(self):
        self.coarse_oris = get_sample_reduced_fundamental(
            self.coarse_res, point_group=self.phase.point_group
        )
        self.fine_oris = get_sample_reduced_fundamental(
            self.fine_res, point_group=self.phase.point_group
        )

        self.coarse_sim = self.gen.calculate_diffraction2d(
            self.phase, self.coarse_oris, **self.simgen_kwargs
        )
        self.fine_sim = self.gen.calculate_diffraction2d(
            self.phase, self.fine_oris, **self.simgen_kwargs
        )

    def to_dict(self) -> dict[str]:
        return {
            "coarse_resolution": self.coarse_res,
            "coarse_oris_count": self.coarse_oris.size,
            "fine_resolution": self.fine_res,
            "fine_oris_count": self.fine_oris.size,
            "n_best": self.n_best,
            "frac_keep": self.frac_keep,
            "n_keep": self.n_keep,
            "simulation_generator_parameters": self.compile_simulation_generator_parameters(),
            "dataset_parameters": self.compile_dataset_parameters(),
        }

    def compile_simulation_generator_parameters(self) -> dict[str]:
        return {
            **self.simgen_kwargs,
            "accelerating_voltage": self.gen.accelerating_voltage,
            "approximate_precession": self.gen.approximate_precession,
            "precession_angle": float(self.gen.precession_angle),
            "minimum_intensity": self.gen.minimum_intensity,
            "shape_factor_model": self.gen.shape_factor_model.__name__,
        }

    def compile_dataset_parameters(self) -> dict[str]:
        return {
            "title": self.pol.metadata.General.title,
            "space_group": str(self.phase.space_group),
            "phase_name": self.phase.name,
            "device": "GPU" if self.pol._gpu else "CPU",
            "num_diffraction_patterns": len(self.pol),
        }


class TemplateMatchingComparison:
    def __init__(self, params: TemplateMatchingParameters) -> None:
        self.params = params

        self._dense_orientation_map = None
        self._refined_orientation_map = None
        self._old_orientation_map = None
        self.dense_runtimes = []
        self.refined_runtimes = []
        self.old_runtimes = []

    @property
    def is_test_complete(self) -> bool:
        return all(
            len(times) > 0
            for times in [
                self.dense_runtimes,
                self.refined_runtimes,
                self.old_runtimes,
            ]
        )

    @property
    def refined_orientation_map(self) -> OrientationMap:
        if self._refined_orientation_map is None:
            self._refined_orientation_map, runtime = self._run_tm_refined()
            self.refined_runtimes.append(runtime)
        return self._refined_orientation_map

    @property
    def old_orientation_map(self) -> OrientationMap:
        if self._old_orientation_map is None:
            self._old_orientation_map, runtime = self._run_tm_old()
            self.old_runtimes.append(runtime)
        return self._old_orientation_map

    @property
    def dense_orientation_map(self) -> OrientationMap:
        if self._dense_orientation_map is None:
            self._dense_orientation_map, runtime = self._run_tm_dense()
            self.dense_runtimes.append(runtime)
        return self._dense_orientation_map

    def collect_runtime_data(self, num_runs: int = 1):
        if num_runs < 1:
            return

        # Ensure the properties are set as well
        if not self.is_test_complete:
            _ = self.refined_orientation_map
            _ = self.dense_orientation_map
            _ = self.old_orientation_map
        tqdm_total = sum(
            num_runs - len(t)
            for t in (self.dense_runtimes, self.refined_runtimes, self.old_runtimes)
        )
        with tqdm(
            total=tqdm_total, leave=False, desc="Collect data", position=1
        ) as pbar:
            while len(self.dense_runtimes) < num_runs:
                _, runtime = self._run_tm_dense()
```

```python
                self.dense_runtimes.append(runtime)
                pbar.update()

            while len(self.refined_runtimes) < num_runs:
                _, runtime = self._run_tm_refined()
                self.refined_runtimes.append(runtime)
                pbar.update()

            while len(self.old_runtimes) < num_runs:
                _, runtime = self._run_tm_old()
                self.old_runtimes.append(runtime)
                pbar.update()

    def _run_tm_refined(self) -> tuple[OrientationMap, float]:
        ### START OF TIMER
        start_time = perf_counter_ns()
        orient = self.params.pol.get_orientation(
            self.params.coarse_sim,
            n_keep=None,
            n_best=self.params.coarse_oris.size,
            normalize_templates=True,
            frac_keep=1,
        )
        ori_inds = RefinedSimulation2D(
            orient,
            self.params.gen,
            self.params.fine_oris,
            frac_keep=self.params.frac_keep,
            n_keep=self.params.n_keep,
            simgen_kwargs=self.params.simgen_kwargs,
        )
        orient_best = self.params.pol.get_orientation(
            ori_inds,
            n_keep=None,
            n_best=self.params.n_best,
            normalize_templates=True,
            frac_keep=1,
            lazy_output=True,
        )
        end_time = perf_counter_ns()
        ### END OF TIMER

        timedelta = end_time - start_time

        return orient_best, timedelta

    def _run_tm_dense(self) -> tuple[OrientationMap, float]:
        return self._run_tm_old(dense=True)

    def _run_tm_old(self, dense: bool = False) -> tuple[OrientationMap, float]:
        frac_keep = 1 if dense else self.params.frac_keep
        n_keep = None if dense else self.params.n_keep

        ### START OF TIMER
        start_time = perf_counter_ns()
        orient = self.params.pol.get_orientation(
            self.params.fine_sim,
            n_best=self.params.n_best,
            n_keep=n_keep,
            frac_keep=frac_keep,
            normalize_templates=True,
        )
        end_time = perf_counter_ns()
        ### END OF TIMER

        timedelta = end_time - start_time

        return orient, timedelta

    @staticmethod
    def correlation_score_similarity(
        ori_map_1: OrientationMap, ori_map_2: OrientationMap
    ) -> np.ndarray:
        def similarity(result_1, result_2):
            _, scores_1, _, _ = result_1.T
            _, scores_2, _, _ = result_2.T
            return np.linalg.norm(scores_2 - scores_1)

        return ori_map_1.map(
            similarity, result_2=ori_map_2, inplace=False
        ).data.squeeze()

    @staticmethod
    def orientation_similarity(
        ori_map_1: OrientationMap, ori_map_2: OrientationMap
    ) -> np.ndarray:
        oris_1 = ori_map_1.to_single_phase_orientations()
        oris_2 = ori_map_2.to_single_phase_orientations()
        return oris_1.angle_with(oris_2, degrees=True)

    @staticmethod
    def equality_length(
        ori_map_1: OrientationMap, ori_map_2: OrientationMap
    ) -> np.ndarray:
        # Assume equal correlation score => same simulation
        def get_inequality_index(result_1, result_2):
            _, scores_1, _, _ = result_1.T
            _, scores_2, _, _ = result_2.T
            eq = scores_1 == scores_2
            # if full overlap, argmax would return 0
            if all(eq):
                return eq.size / scores_1.size
            return np.argmax(eq) / scores_1.size

        return ori_map_1.map(
            get_inequality_index, result_2=ori_map_2, inplace=False
        ).data.squeeze()

    @staticmethod
    def equality_length_set(
        ori_map_1: OrientationMap, ori_map_2: OrientationMap
    ) -> np.ndarray:
        def get_equality_set_length(result_1, result_2):
            _, scores_1, _, _ = result_1.T
            _, scores_2, _, _ = result_2.T
            return len(set(scores_1).intersection(set(scores_2))) / scores_1.size

        return ori_map_1.map(
            get_equality_set_length, result_2=ori_map_2, inplace=False
        ).data.squeeze()

    def compile_statistics(self) -> dict[str, dict[str, float]]:
        if not self.is_test_complete:
            raise Exception(
                "Tests have not been completed; no statistics to compile"
            )

        refined_out = {}
        dense_out = {}
        old_out = {}

        refined_correlation = self.correlation_score_similarity(
            self.refined_orientation_map, self.dense_orientation_map
        )
        old_correlation = self.correlation_score_similarity(
            self.old_orientation_map, self.dense_orientation_map
        )

        refined_out["correlation_mean"] = float(np.mean(refined_correlation))
        refined_out["correlation_std"] = float(np.std(refined_correlation))

        old_out["correlation_mean"] = float(np.mean(old_correlation))
        old_out["correlation_std"] = float(np.std(old_correlation))

        refined_orientation = self.orientation_similarity(
            self.refined_orientation_map, self.dense_orientation_map
        )
        old_orientation = self.orientation_similarity(
            self.old_orientation_map, self.dense_orientation_map
        )

        refined_out["orientation_mean"] = float(np.mean(refined_orientation))
        refined_out["orientation_std"] = float(np.std(refined_orientation))

        old_out["orientation_mean"] = float(np.mean(old_orientation))
        old_out["orientation_std"] = float(np.std(old_orientation))

        refined_out["runtime_mean"] = float(np.mean(self.refined_runtimes))
        refined_out["runtime_std"] = float(np.std(self.refined_runtimes))
        refined_out["runtimes"] = self.refined_runtimes
        refined_out["total_runtime"] = sum(self.refined_runtimes) / 1e9
        refined_out["runs"] = len(self.refined_runtimes)

        old_out["runtime_mean"] = float(np.mean(self.old_runtimes))
        old_out["runtime_std"] = float(np.std(self.old_runtimes))
        old_out["runtimes"] = self.old_runtimes
        old_out["total_runtime"] = sum(self.old_runtimes) / 1e9
        old_out["runs"] = len(self.old_runtimes)

        dense_out["runtime_mean"] = float(np.mean(self.dense_runtimes))
        dense_out["runtime_std"] = float(np.std(self.dense_runtimes))
        dense_out["runtimes"] = self.dense_runtimes
        dense_out["total_runtime"] = sum(self.dense_runtimes) / 1e9
        dense_out["runs"] = len(self.dense_runtimes)

        refined_equality_length = self.equality_length(
            self.refined_orientation_map, self.dense_orientation_map
        )
        refined_equality_length_set = self.equality_length_set(
            self.refined_orientation_map, self.dense_orientation_map
        )

        old_equality_length = self.equality_length(
            self.old_orientation_map, self.dense_orientation_map
        )
        old_equality_length_set = self.equality_length_set(
            self.old_orientation_map, self.dense_orientation_map
        )

        refined_out["equality_length_mean"] = float(np.mean(refined_equality_length))
        refined_out["equality_length_std"] = float(np.std(refined_equality_length))
        refined_out["equality_length_set_mean"] = float(
            np.mean(refined_equality_length_set)
        )
        refined_out["equality_length_set_std"] = float(
            np.std(refined_equality_length_set)
        )

        old_out["equality_length_mean"] = float(np.mean(old_equality_length))
        old_out["equality_length_std"] = float(np.std(old_equality_length))
        old_out["equality_length_set_mean"] = float(np.mean(old_equality_length_set))
        old_out["equality_length_set_std"] = float(np.std(old_equality_length_set))

        return {
            "dense": dense_out,
            "refined": refined_out,
            "old": old_out,
        }
```

A.5 run_tests.py

The file which was actually run. Parameter sweeps, optimization with Scipy, and data collection and storage to `.json`-files.

```python
from simulation_comparison import (
    TemplateMatchingComparison,
    TemplateMatchingParameters,
)
from get_data import get_real_data, get_simulated_data, ROOT
from diffsims.generators.simulation_generator import SimulationGenerator
from orix.crystal_map import Phase
import json
from hyperspy import api as hs
from tqdm import tqdm
from glob import glob
from pathlib import Path
from orix.sampling import get_sample_reduced_fundamental

RESULTS = ROOT / "results"
DATA = ROOT / "data"


def store_test_results(comment: str, data: TemplateMatchingComparison):
    res = data.compile_statistics()
    res.update(data.params.to_dict())
    res["comment"] = comment

    files = (Path(file) for file in glob(str(RESULTS / "*.json")))
    filenums = (int(file.with_suffix("").name) for file in files)
    max_filenum = max(filenums, default=0)
    filename = RESULTS / f"{max_filenum + 1:06}"

    if not filename.parent.exists():
        filename.parent.mkdir()
    with open(filename.with_suffix(".json"), "w") as f:
        json.dump(res, f, indent=4)


def main():
    hs.preferences.General.show_progressbar = False
    hs.set_log_level("ERROR")

    phase = Phase.from_cif(str(DATA / "Cu.cif"))
    gen = SimulationGenerator(
        minimum_intensity=1e-20, precession_angle=1, approximate_precession=True
    )

    oris = get_sample_reduced_fundamental(1, point_group=phase.point_group)
    data = get_simulated_data(phase, oris)
    # data = get_real_data()
    print("Starting azimuthal integration")
    pol = data.get_azimuthal_integral2d(npt=100)
    pol.compute()
    print("Starting sweep")
    for n_keep in tqdm(
        [100, 200, 300, 500, 1000, 1500, 2000], desc="n_keep", position=0
    ):
        params = TemplateMatchingParameters(
            pol, gen, 2, 0.4, phase, 10, n_keep=n_keep
        )
        comp = TemplateMatchingComparison(params)
        comp.collect_runtime_data(6)
        store_test_results("n_keep sweep", comp)
    for coarse_res in tqdm(
        [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 5.0], desc="coarse_res", position=0
    ):
        params = TemplateMatchingParameters(
            pol, gen, coarse_res, 0.5, phase, 10, n_keep=400
        )
        comp = TemplateMatchingComparison(params)
        comp.collect_runtime_data(6)
        store_test_results("coarse_resolution sweep", comp)
    for fine_res in tqdm(
        [0.2, 0.3, 0.45, 0.5, 0.8, 1.0, 1.5], desc="fine_res", position=0
    ):
        params = TemplateMatchingParameters(
            pol, gen, 2, fine_res, phase, 10, n_keep=400
        )
        comp = TemplateMatchingComparison(params)
        comp.collect_runtime_data(6)
        store_test_results("fine_resolution sweep", comp)


def optimize():
    hs.preferences.General.show_progressbar = False
    hs.set_log_level("ERROR")

    phase = Phase.from_cif(str(DATA / "Ag.cif"))
    gen = SimulationGenerator(
        minimum_intensity=1e-20, precession_angle=1, approximate_precession=True
    )

    data = get_real_data()
    print("Starting azimuthal integration")
    pol = data.get_azimuthal_integral2d(npt=100)
    pol.compute()
    print("Starting optimization")

    from scipy.optimize import minimize
    import numpy as np

    def to_optimize(params):
        coarse_res, fine_res, n_keep = params
        n_keep *= 100
        n_keep = int(n_keep)
        params = TemplateMatchingParameters(
            pol, gen, coarse_res, fine_res, phase, 10, n_keep=n_keep
        )
        comp = TemplateMatchingComparison(params)
        comp.collect_runtime_data(6)
        store_test_results("optimization", comp)

        # Quantify the difference
        res = comp.compile_statistics()
        runtime = res["refined"]["runtime_mean"] / 10
        correlation = res["refined"]["correlation_mean"] / 50
        orientation = res["refined"]["orientation_mean"] / 10
        equality = 1 - res["refined"]["equality_length_set_mean"] / 10
        return np.linalg.norm([runtime, correlation, orientation, equality])

    opt = minimize(to_optimize, [2, 0.2, 5])
    print(opt)


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

APPENDIX

B

SUPPLEMENTARY FIGURES

This chapter contains supplementary figures for this thesis.

The first section contains twin misorientation plots, similar to Figure 4.1.2, for scans 5 and 6 of the Ag sample. The second section contains complete results for the TM pre-selection algorithm.

B.1 Twin Misorientation Figures

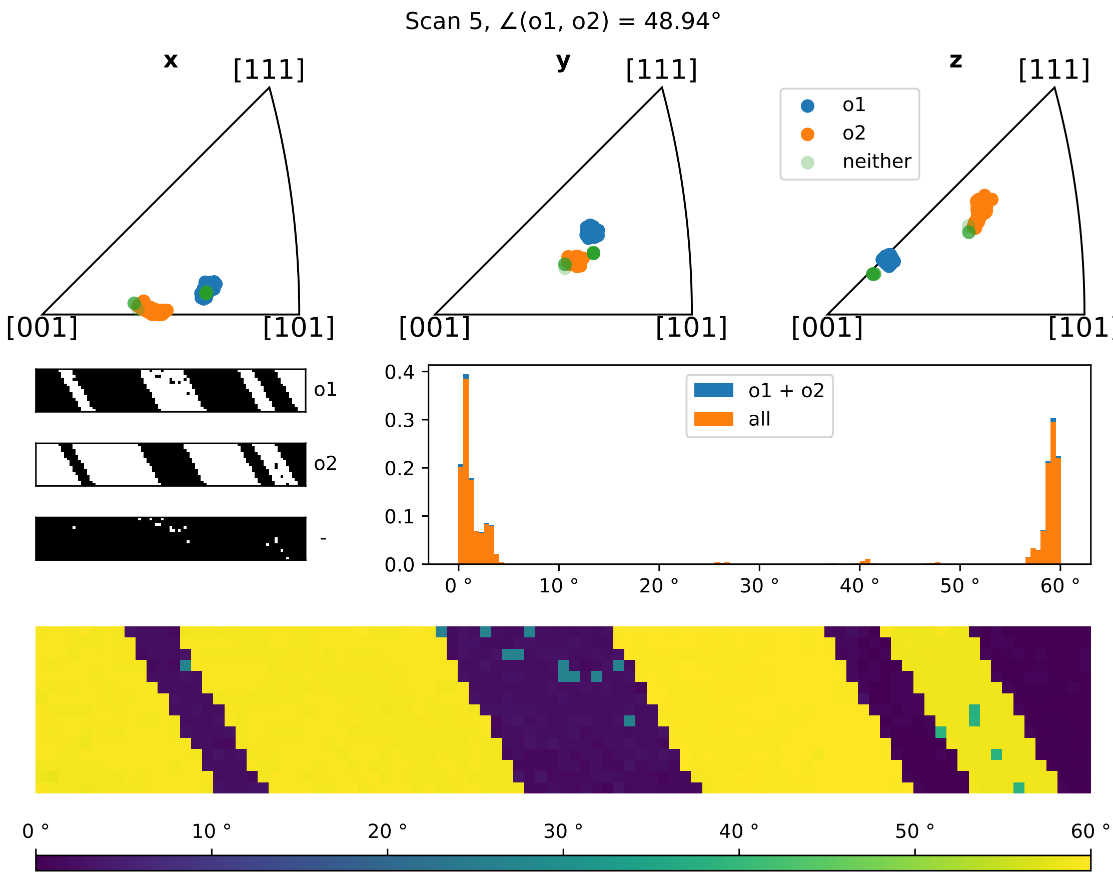


Figure B.1.1: Twin misorientation of scan 5, which had the largest deviation from the expected (11.06°). For more details on the figure layout, see Figure 4.1.2.

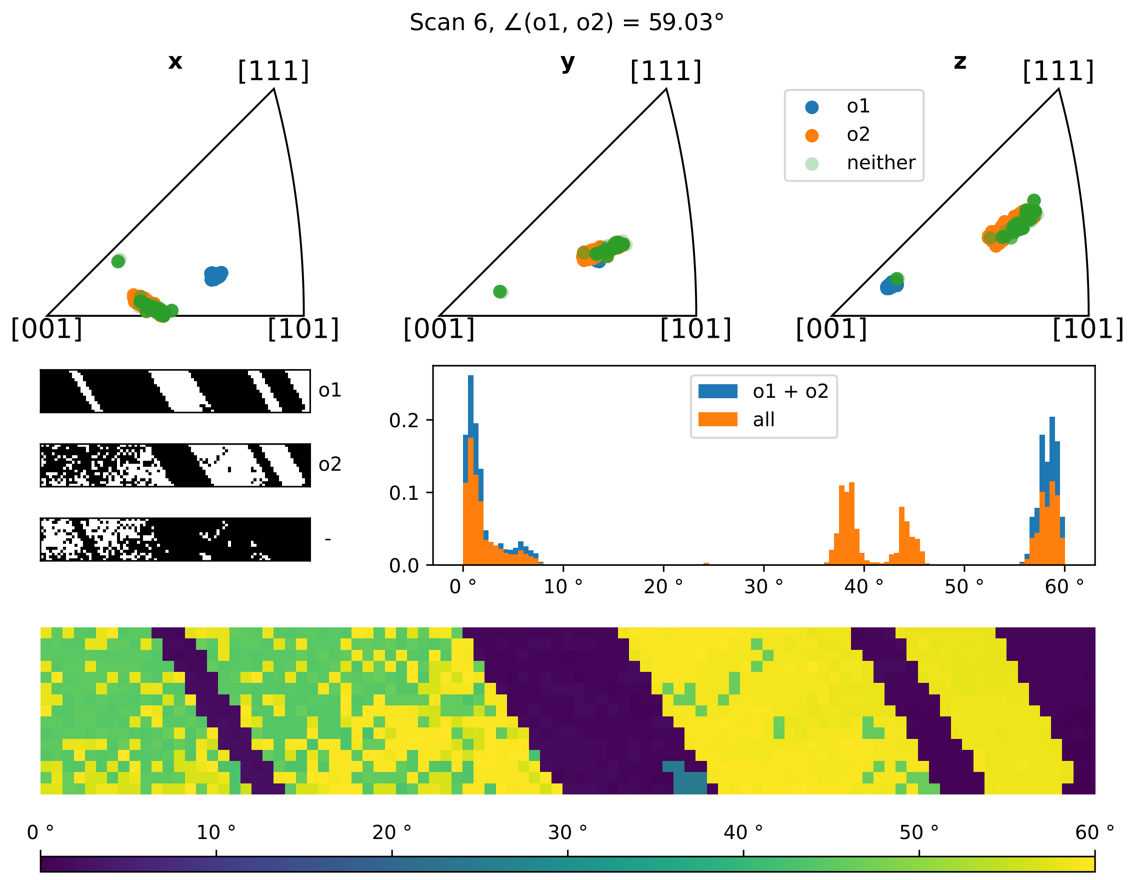
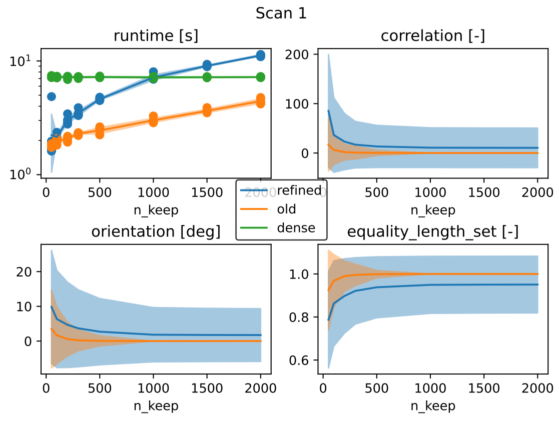


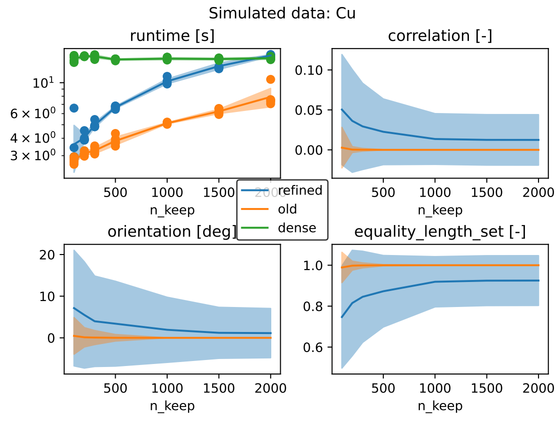
Figure B.1.2: Twin misorientation of scan 6, the scan which conformed the best with the expected (0.97° off). For more details on the figure layout, see Figure 4.1.2.

B.2 Template Matching Pre-selection Algorithm Results

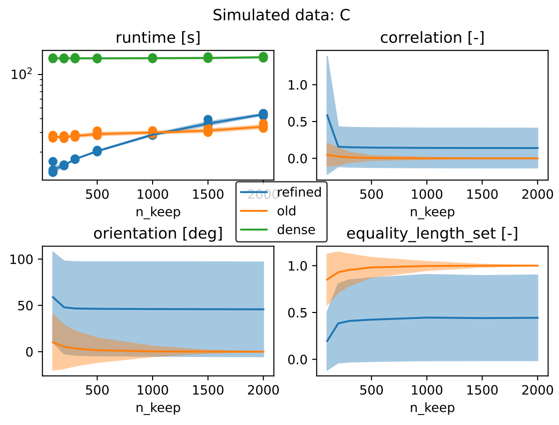
This section contains plots of the results from testing the new TM pre-selection algorithm against the current and no filtering. To fit more plots per page, they are grouped according to the parameter along the x-axes. This makes them small, so a print might not show all details, but the pdf is zoomable to any desired level.



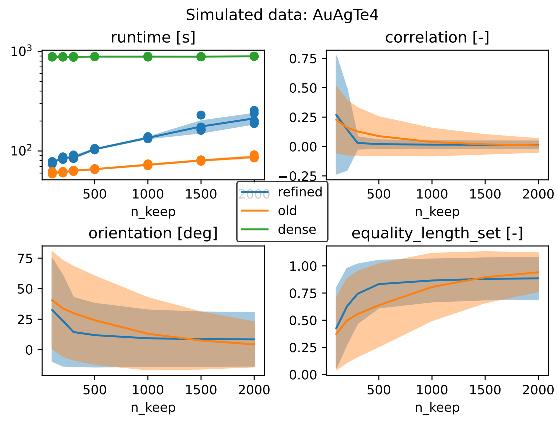
(a)



(b)

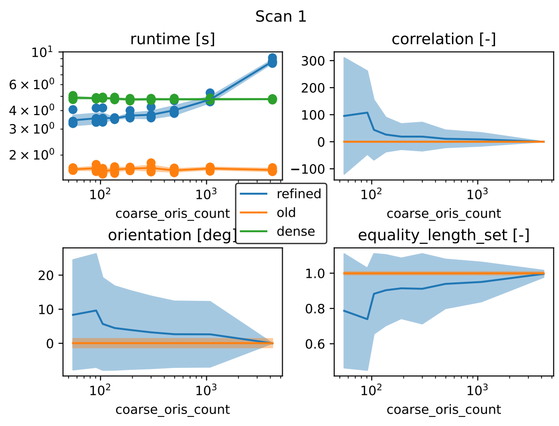


(c)

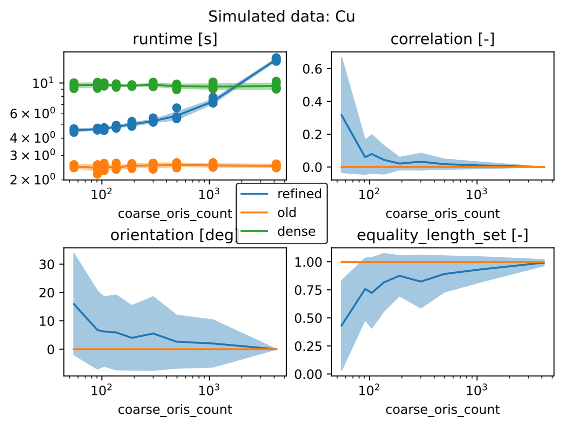


(d)

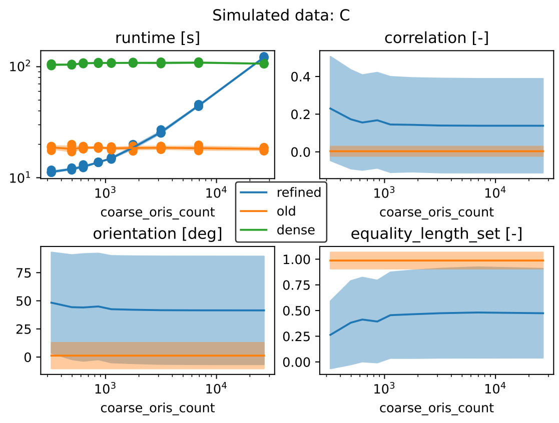
Figure B.2.1: n_keep sweep. (a) Scan 1. (b) Copper. (c) Graphite. (d) AuAgTe$_4$.



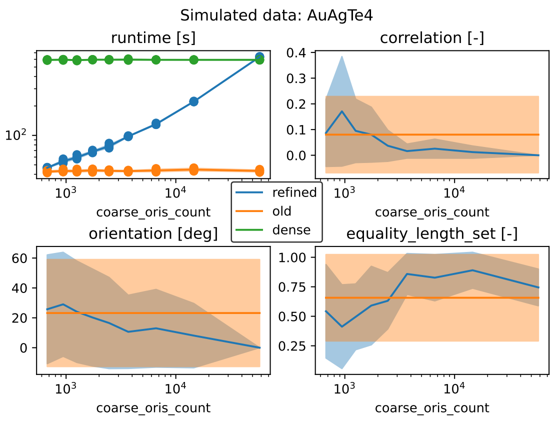
(a)



(b)

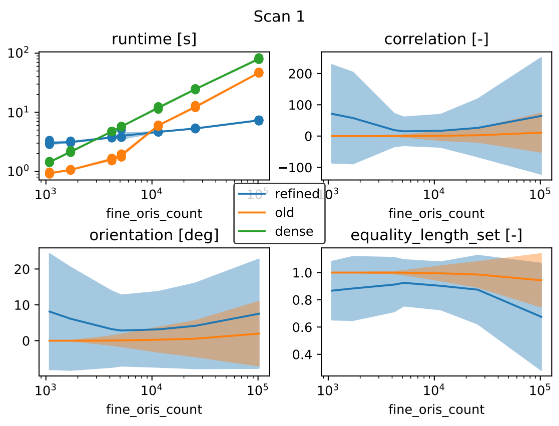


(c)

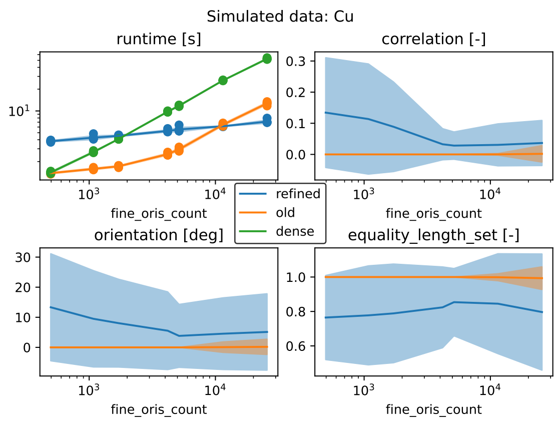


(d)

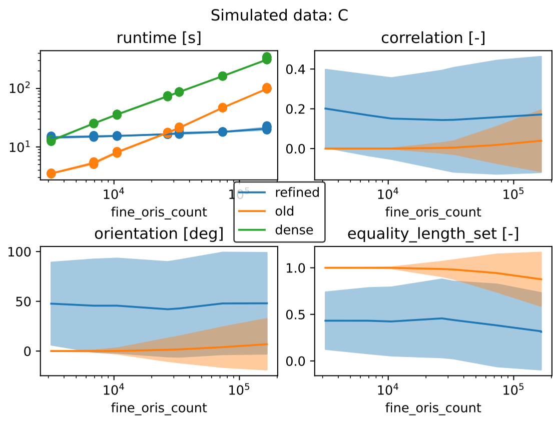
Figure B.2.2: Coarse resolution sweep. (a) Scan 1. (b) Copper. (c) Graphite. (d) $AuAgTe_4$.



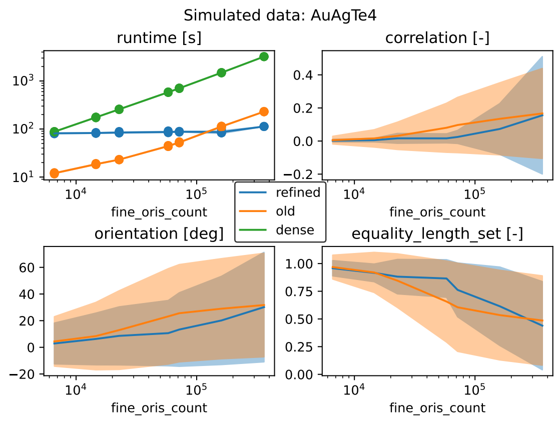
(a)



(b)



(c)



(d)

Figure B.2.3: Fine resolution sweep. (a) Scan 1. (b) Copper. (c) Graphite. (d) $AuAgTe_4$.

APPENDIX

C

CODE CONTRIBUTIONS

This chapter lists the contributions to open-source software made as part of this project.

C.1 Pyxem

`https://github.com/pyxem/pyxem/pull/1058` Fix an off-by-one error in azimuthal integration, dropping the final row and column from the original data.

`https://github.com/pyxem/pyxem/pull/1060` Properly support azimuthal range in azimuthal integration. This is important for a consistent definition of the in-plane angle when template matching.

`https://github.com/pyxem/pyxem/pull/1061` Expands upon the new template matching result class, by implementing multiple plotting utility functions. Of note is the `to_marker` function, which generates markers to add to a plot representing the simulated diffraction spots.

`https://github.com/pyxem/pyxem/pull/1062` Implemented some background subtraction methods for polar data, specifically radial mean and radial percentile.

`https://github.com/pyxem/pyxem/pull/1076` In favour of `https://github.com/pyxem/pyxem/pull/1061`

Additionally contributed with discussions and code review.

C.2 Orix

`https://github.com/pyxem/orix/pull/469` Fix a bug in phase initialization, where atom positions were not expressed in the correct basis.

C.3 Diffsims

`https://github.com/pyxem/diffsims/pull/205` Contributed to large restructuring of simulations, with code review, discussions, and testing. See also `https://github.com/pyxem/diffsims/pull/201`.

C.4 Hyperspy

`https://github.com/hyperspy/hyperspy/pull/3362` Fix discrepancy in axes ticks between plotting a signal and plotting several images.

APPENDIX

D

EMC ABSTRACT

Below is the abstract submitted and accepted as an oral presentation during the European Microscopy Congress 2024, to be held by my supervisor Antonius van Helvoort.

Abstract submitted and accepted for an oral presentation at the European Microscopy Congress 2024 in the session IM-06 - Diffraction Techniques and Structural Analysis, Copenhagen, Denmark, 25 - 30 August 2024

Scanning precession electron diffraction tilt series for orientation analysis

A.T.J. van Helvoort[1], V.J. Femoen[1], A.C. Mathisen[1], K.E. Aune[1], E.F. Christiansen[1], I.-E. Nylund, T. Bergh[1,3], R. Bjørge[1,4]

[1] *Department of Physics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.*
[2] *Department of Materials Science and Engineering, NTNU, Trondheim, Norway.*
[3] *Department of Chemical Engineering, NTNU, Trondheim, Norway.*
[4] *Materials and Nanotechnology, SINTEF Industry, Trondheim, Norway.*

Identifying the orientation of crystalline phases at the nanometer scale is relevant for understanding materials properties. Here we will demonstrate that collecting and processing scanning precession electron diffraction (SPED) datasets collected at a few different specimen tilts can improve the accuracy of template-based orientation mapping [1,2]. In addition, the tilt series allows complete determination of the relation between the specimen crystallographic setting and the goniometer axes. This insight, combined with the orientation map, can be used in a convenient semi-automatic approach to predict the tilts required to reach a target specimen orientation for further structure analysis.

SPED of different polycrystalline systems (Si, Ag, and oxides) were recorded in a JEOL JEM2100F with a NanoMegas DigiStar precession system and a Quantum Detectors MerlinEM direct electron detector. Scans were taken over areas up to 15 x 15 μm containing 10's of grains using a nominal precession angle of 1˚. Tilts series using one or two axes contained 3-5 tilts in the range 0 - 20˚ from the initial flat specimen position. For data analysis and visualization, we used primarily the open-source python library pyxem [3].

The indexed frames, taken at different tilts, were compared, after manually aligning the frames, using the set tilt as expected misorientation. Together with considering as well the best 5 to 25 normalized cross correlations between the experimental patterns and the simulated pattern bank, orientation-dependent misindexations can be identified and the orientation estimate refined. Compared to the standard approach of collecting SPED using only a single specimen tilt and the best correlation scores for each pattern, the tilt series approach reduces indexation variations within grains. This gives a more uniform representation of the grains in the final orientation maps.

The accuracy of the refined orientation analysis can be determined with a known orientation relation, here for example Σ3 twins in face-centered cubic and diamond crystal systems. The misorientation deviation between the measured and expected misorientation between twin domains is used as the metric [4]. The found accuracy is below the used precession angle.

The refined orientation mapping based on a small tilt series has a further practical use. From a single axis tilt series, the position of the two perpendicular tilt axes can be determined. As the tilt series is small and over a limited angular range, sufficient probe positions must be used to accurately determine the tilt axis position. Misaligned between frames and areas with overlap, such as grain boundary areas, were excluded through thresholding. A second tilt axis series or grains correctly indexed in different frames can be used to verify the found axes positions. Using the determined orientation of a grain together with the deduced axes positions, the tilts to reach a target zone for a grain can be predicted. Based on tests on different TEMs and holders, the target zone was within 2˚. In the tests the specimen was placed at approximately the same rotation relative to the holder axes. However, should the specimen be differently placed compared to where the orientation was mapped out, an additional transformation matrix can be included in the navigation tool to recalculate the target tilts for the given specimen placing. This correction is based on a manual estimation of misorientation from the tilts to the actual target zone, diffraction (using the Laue circle), or imaging (assuming in-plane rotation).

To conclude, template matching based on multiple SPED scans at a few varying specimen tilts improves the accuracy and the final orientation visualization. In addition, the approach is used to make a practical navigator tool that widens to use of template matching results for subsequent lattice imaging and further crystallographic analysis. With the advancements in automatic scan controls, faster detectors and optimized transparent open-source routines, the benefits gained will more than compensate for the drawbacks of acquiring and processing multiple scans.

Keywords
Texture, diffraction, open-source, template-matching, ACOM

References:
[1] E.F, Rauch et al., Z. für Kristall. 225 (2010), p. 103. doi:10.1524/zkri.2010.1205
[2] N. Cautaerts et al., Ultramicr. 237 (2022), p. 113517. doi:10.1016/j.ultramic.2022.113517
[3] D. Johnstone et al., Zenodo (2024), 10551678. https://zenodo.org/records/10551678
[4] Q. Shi et al., Meas. Sci. Technol. 35 (2024), 045030. doi:10.1088/1361-6501/ad204d
[5] The Research Council of Norway is acknowledged for support to the Norwegian Center for Transmission Electron Microscopy, NORTEM (197405).

-

NTNU

Norwegian University of
Science and Technology