THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MATERIALS SCIENCE AND ENGINEERING

METHODOLOGY OF MEASURING POLAR DISPLACEMENTS IN COMPLEX OXIDES WITH ABERRATION CORRECTED STEM

LEIXIN MIAO
SPRING 2017

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Materials Science and Engineeing
with honors in Materials Science and Engineering

Reviewed and approved* by the following:

Nasim Alem
Assistant Professor of Materials Science and Engineering
Thesis Supervisor

Allen Kimel
Assistant Professor of Materials Science and Engineering
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

In this work, we have developed statistical methods that can be applied to analyze and quantify the distortion and polarization effects in a variety of materials and nanostructures. In this study, we have performed statistical analysis on at the domain walls in perovskite ferroelectric materials including $LiNbO_3$ and $Ca_3Ru_{1.9}Ti_{0.1}O_7$ using self-developed MATLAB codes. The atomic structure of the ferroelectrics were previously investigated using aberration corrected Scanning Transmission Electron Microscopy imaging. The developed statistical analysis in this thesis was conducted to make the precise measurement of the atom displacements down to 5 pm near the domain walls to determine the domain wall types.

Bright filed STEM images were analyzed in $LiNbO_3$ to measure the relative distance between nobium and oxygen in the sample. Statistical analysis was performed in both parallel and perpendicular to wall directions and the result showed there are both in-plane rotation and gradual change in magnitude of the polarization, which confirmed the mixed Icing and Néel feature of the domain wall.

The same statistical method was applied on high-angle annular dark-field STEM images (HAADF) in $Ca_3Ru_{1.9}Ti_{0.1}O_7$ to measure Ca and Ru/Ti atom distortions. The statistical analysis results show an average of 25 pm displacement of Ca atoms in opposite directions in two adjacent domains and a polarization switching behavior at the domain wall. In addition, the polarization maps show the head-to-head configuration of the 90-degree domain walls and the head-to-tail configuration of the 180-degree domains.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank Professor Nasim Alem for giving me the opportunity to join this wonderful research group and work on this amazing project. Furthermore, I am grateful that she offered me the chance to learn so much in the field of transmission microscopy and I appreciate all the help she has been providing me. I want to thank my mentor Mr. Debangshu Mukherjee for capturing and providing the STEM imaging and his help with the data analysis, as well as the rest of the group for their amazing help on this project and the guidance they provided me.

Finally, I would like to thank my family for supporting my study abroad. Their support and love enabled me to study at Penn State, and because of this, I learned and grew so much in these four years and had so many unforgettable memories here in the US.

Chapter 1 INTRODUCTION

1.1. Introduction

Ferroelectric materials are non-centrosymmetric crystals with a spontaneous, switchable dipole moment, and are one of the four known ferroic orders. These materials find applications in piezoelectrics, sensors, optical switches to name a few. Research on ferroelectrics is a fast developing field in materials science, with a significant focus in recent years towards understanding the quantum mechanical origins of ferroic orders. Ferroelectric domain walls have been observed to possess some unique properties such as electrical conductivity and photocurrent generation. In order to determine and predict some of the resulting properties that domain walls offer, it is important to determine the domain wall structure and chemistry. Therefore, imaging of the structure and quantification of the structural distortions and bond displacements with Picometer accuracy is required. [1]

The development of aberration corrected Scanning Transmission Electron Microscopy (AC-STEM) has allows the direct imaging of atom positions. Thus, AC-STEM can directly reveal atomic position with a sub-angstrom resolution, correlating atomic structure with bulk properties, across interfaces like thin films and domain walls. By employing the Titan^3 scanning/transmission electron microscopy, images with the resolution of approximately 7 pm per pixel have been previously obtained. This thesis explores statistical methods using self-developed MATLAB codes that enable determination of the ferroelectric polarization from the atomic displacements extracted from STEM images.

Thin film samples of $LiNbO_3$ and $Ca_3Ru_{2-x}Ti_xO_7$. (x=0.1) were previously prepared with focused ion beam (FIB). The samples were observed under Aberration Corrected STEM (Titan^3) at Penn State Materials Characterization Lab, and images of samples are analyzed using custom-built software routines. The focus of this thesis is on the statistical analysis on the atom displacement across the domain walls to help determine the types of the domain walls and quantitatively measure the displacement of atoms. Using the statistical analysis methods developed in this study, this thesis led to the first direct observation of mixed domain walls in $LiNbO_3$, along with imaging of 90^o and 180^o domain walls in $Ca_3Ru_{2-x}Ti_xO_7$. Additionally, the observed structure of the domain walls provides direct evidence that domain walls in ferroelectrics possess emergent properties that are absent in the bulk, like enhanced electronic conductivity, correlated electronic liquids and topological insulators.

Chapter 2

BACKGROUND AND LITERATURE REVIEW

2.1. Ferroelectricity in perovskite oxides

Ferroelectrics materials exhibit spontaneous polarization, which is the polarization in the absence of an electric field. Permanent electric dipoles exist in ferroelectric materials. The spontaneous polarization originates from the arrangement of atoms in the crystal structures and their positions, and ferroelectricity can only be found in materials in polar space groups. Additionally, polarization is required to be switchable between more than two discrete states. As a result, many materials with crystal structures of polar space group cannot be ferroelectric. [2]

The atom displacements in crystals break down the centrosysmmetry at high temperatures and leads to multiple discrete polarized states. The perovskite crystal structure serves as an example. This type of crystals typically possesses the chemical formula of ABO_3, with eight A atoms locate in the corner of the cell and a B atom sits in the center of the cell. The oxygen atoms are on the face center of the cell and form an octahedral cage surrounding the central B atom, as shown in Figure 2-1 and Figure 2-2. In a centrosymmetric structure with no structural polarization, the B atom is at the center of 6 oxygen octahedral, arranged at the corners of a regular octahedron. The octahedra are linked at their corners and form a simple cubic network. This simple cubic network creates large holes to accommodate A atoms and thus each A atom has closest neighboring 12 O atoms. [3]

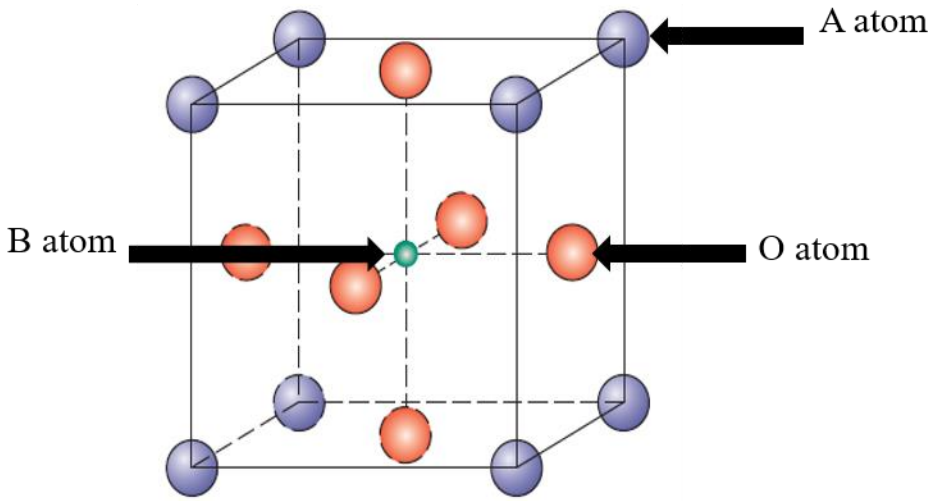

Figure 2-2: Schematics of the perovskite structure. Positions of A, B and O atoms are shown by arrows [2]

An empirical criterion for the stability of the perovskite structure was proposed by Goldschmidt (1926), based on the concept of ionic radius. This model follows two rules, (i) a cation needs to be surrounded by as many anions as possibly being in touch. (ii) all the anions must touch the cations and anion-cation distance is sum of their ionic radii. Following those rules, the ideal condition in perovskite should be: $r_A + r_O = \sqrt{2}\,(r_B + r_O)$, where r_A, r_B, and r_O are the radius of A ion, B ion and O ion respectively.

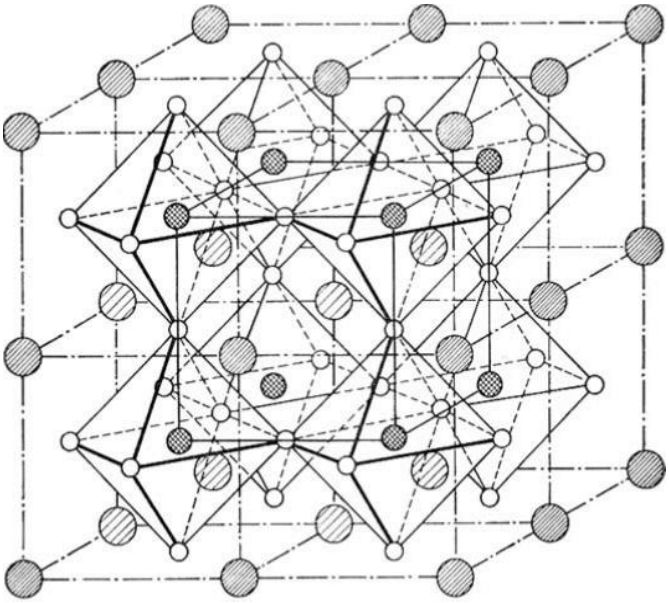

Figure 2-1: Another view of the ABO3 ideal cubic perovskite structure from [4]

To test this criterion in real conditions, a tolerance factor t is created and defined as follows: $t = \frac{r_A + r_O}{\sqrt{2}\,(r_B + r_O)}$. Perfect perovskite structure is formed when the condition satisfies t ≈ 1. When t > 1, the structure is held by the A–O distance and the B atom is too small to fit into the oxygen octahedron so that the structure will develop a small polar distortion, as in $BaTiO_3$. Whereas, when t < 1, the A atoms are too small for the hole created by octahedral. As a result, A atom is not able to bond to all 12 neighboring O atoms. If the t is only slightly smaller than 1, the structure will be likely to slight tilt and enable A atoms to bond with O atoms sufficiently. If the t is even smaller, as Li atoms in $LiNbO_3$, the compound will favor a strongly distorted structure with only 6 neighboring O atoms for A. [4].

The perovskite oxides can form solid solutions, and many of them show complete miscibility. For example, Ba/Sr and Ti/Zr are especially relevant to the study of ferroelectricity, because the formal valence counting can produce insulating behaviors in the phase diagram. The change of cations can result in the shift in transition temperatures. For example, the substitution of Sr into $BaTiO_3$ lowers the transition temperature. [5]

2.2 Examples of ferroelectric perovskite materials

2.2.1 $BaTiO_3$

$BaTiO_3$ is the first perovskite compound to be identified as ferroelectric material. The valence are +2 and +4 for Ba and Ti respectively, perfectly matching the total valence of Oxygens. At high temperature, it has a paraelectric cubic perovskite structure ($Pm\bar{3}m$). At 393K, it transforms into a ferroelectric orthorhombic phase (P4mm). At 278K, there is a second phase transition into a ferroelectric orthorhombic sysmmetry (Amm2). At 183, the last transition undergoes as it transforms into ferroelectric rhombohedral (R3m). In each transition, there are small atomic displacements undergoing. It is dominated by that of Ti ions relative to O networks and macroscopic strain. Different phases of the $BaTiO_3$ is shown in Figure 2-3. In these phase transitions, polar axes aligns along <100>, <110> and <111> family of directions. [6]

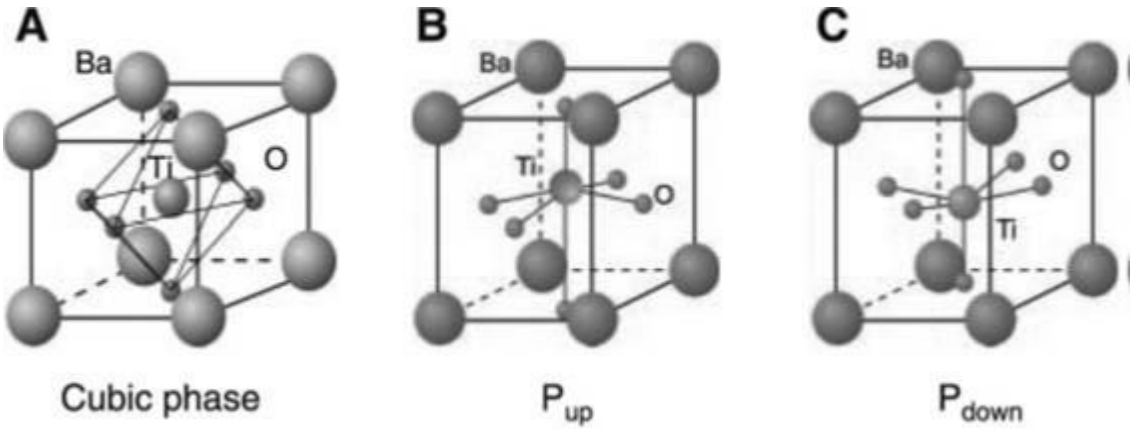

Figure 2-3. (A) High temperature cubic paraelectric phase. (B) and (C) Room temperature ferroelectric phase with up and down polarization. [4]

2.2.2 $LiNbO_3$

$LiNbO_3$ is a special case for ferroelectric perovskite oxides. It has a trigonal paraelectric structure, although it theoretically can form cubic perovskite structure, but due to the small size of Li atoms, it needs huge distortion to achieve it. So it is sometimes not considered to be a perovskite, even though it has the same crystallographical arrangement as R3c $BiFeO_3$. [7] A demonstration of the crystal structure of $LiNbO_3$ is shown in Figure 2-4. The threefold axis is formed by a chain of equidistant A and B atoms. B is surrounded by 8 oxygen atoms, which forms a octahedral cage. However, relative to perovskite

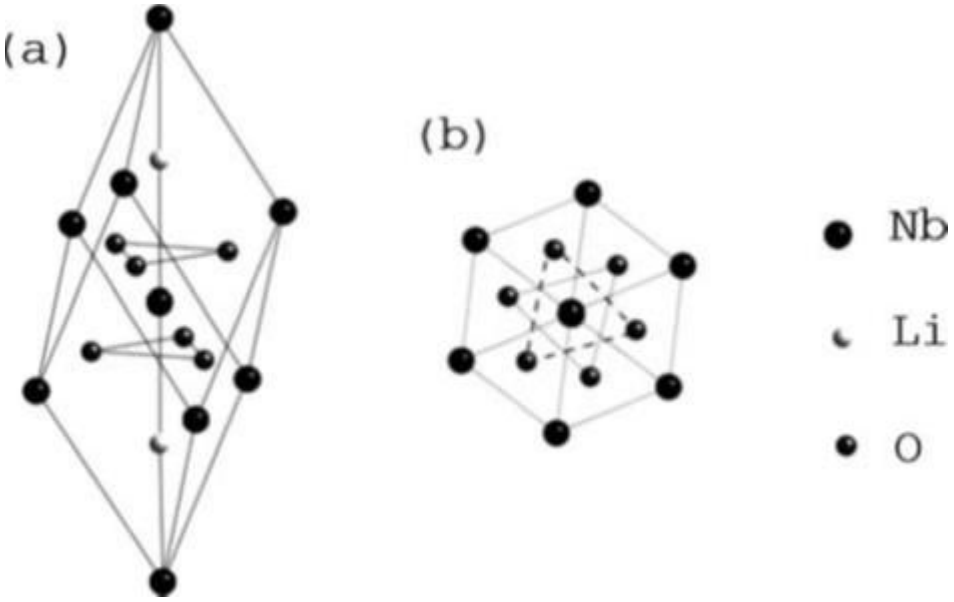

Figure 2-4. Primitive unit cell of the R3c trigonal structure of LiNbO3. Views (a) perpendicular and (b) parallel to the three fold axis [3]

structure, the octahedron is rotated around [111] axis and A atoms have only 6 nearest neighboring oxygen atoms. The low temperature ferroelectric phase is obtained by the displacement of cations along [111] direction and the transition happens at 1483K. [4]

2.2.3 Layered Oxide Ferroelectrics

The Ruddenlesden-Popper (RP) family of transition metal oxides is an example of stacking perovskite blocks. This family has the formula of $A_{n+1}B_nO_{3n+1}$. The structure between perovskite blocks is rocksalt layers. The members of this family are indexed by n, and n describes the thickness of the perovskite layer. The structure is terminated on both ends by AO rocksalt layers, which was originated from a lateral shift of perovskite layer by ½ (a_0,a_0,0). [8] [9] The schematic of their crystal structures is shown in Figure 2-5.

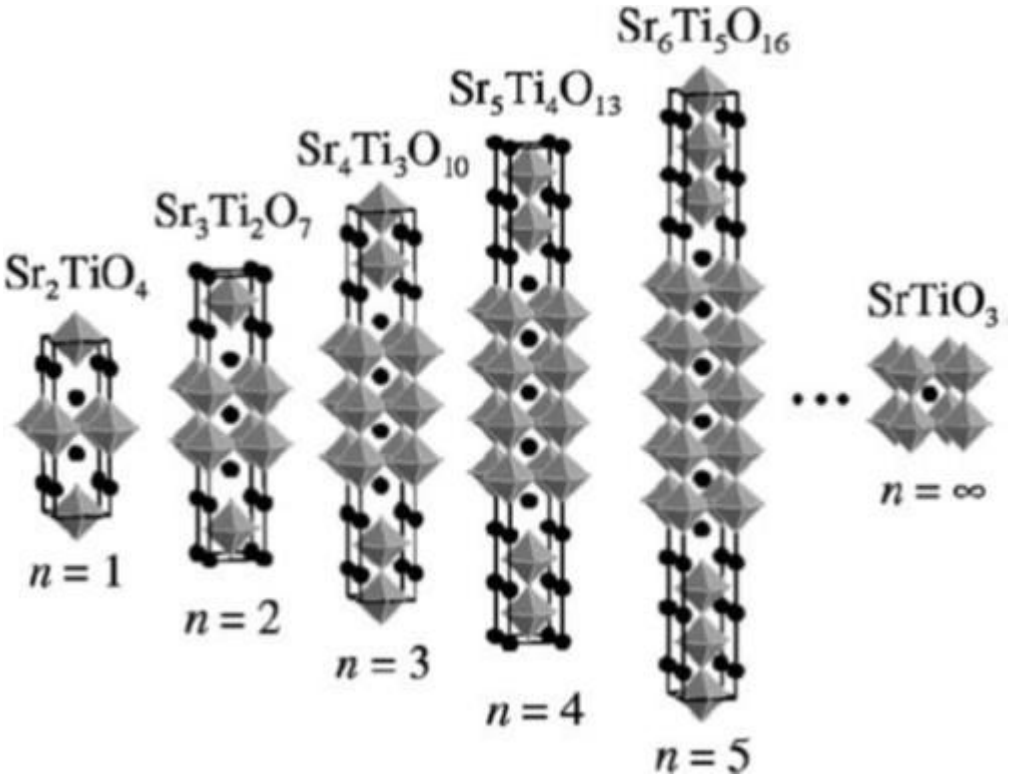

Figure 2-5. Schematic of the crystal structure of a unit cell of the n = 1, 2, 3, 4, 5 and ∞ members of the $Sr_{n+1}Ti_nO_{3n+1}$ series. [4]

2.3 Electronic conduction

When a crystal is formed by a large number of atoms, quantum mechanics determines that there are bands of discrete levels of energy exist closely in the energy spectrum. Generally, bands are separated by a 'forbidden zone', which is a zone where electrons cannot exist. With the help of Fermi statistics,

people find that at absolute zero of temperature, some bands are completely filled while some are empty, and some are partially filled. To consider a conduction process, it is only required to discuss those bands in the outer electronic shells since deeper core electrons have no influence over this process. The electrons in the bands that are partially filled can move to higher energy levels, and thus when an electric field is applied, those electrons can have additional kinetic energy. In other words, the current will flow. For electrons in the completely filled bands, electrons cannot acquire energy from electric fields, so no current flows. However, if there is a narrow energy gap, at room temperature, some electrons can be thermally excited and enter an empty band, leading to conduction. Those thermally excited electrons leave empty electronic states in valence bands, which are called positive holes. Positive holes can also accept energy from the field. [10] As a result, some ceramic materials can conduct current like metals, and those ceramics can also be identified as metals when it comes to electronic conductivity.

2.4 Non-Centrosymmetric Metal

In 1965, P. W. Anderson and E. I. Blount proposed the theory that second-order transitions usually involve some change in internal symmetry other than the mere strain, may be "ferroelectric" in the sense of the appearance of a polar axis, or possibly at least involve the loss of a center of symmetry. [11] It means that if metals undergo second order phase transition, a non-centrosymmetric structure can be obtained, which leads to the possibility of ferroelectricity. To be specific, the appearance polar axis and disappearance of the inversion center accompanies this structural transition [12]. In normal perovskite ferroelectric materials, the polarity usually results from B atom displacement with respect to center of oxygen octahedron. This theory indicated that the tilting of those octahedrons could also lead to polar displacement.

This type of ferroelectric materials had not been discovered for 50 years after this theory is proposed. Until 2013, Youguo Shi et al. reported a centrosymmetric ($R\bar{3}c$) to non-centrosymmetric (R3c) transition in metallic $LiOsO_3$. They used high-pressure-synthesized material $LiOsO_3$, and discovered a continuous shift in mean Li+ ion positions under 140K. The resulting structure has a crystal structure similar to $LiNbO_3$. They firstly applied neutron diffraction for temperatures between 10K and 300K, and found that the phase transition is continuous and it behaves like a secondary order phase transition. The data also indicated that Li displacement along Z direction increased significantly under T_s. They confirmed the result by performing convergent-beam electron diffraction (CBED) on the sample at

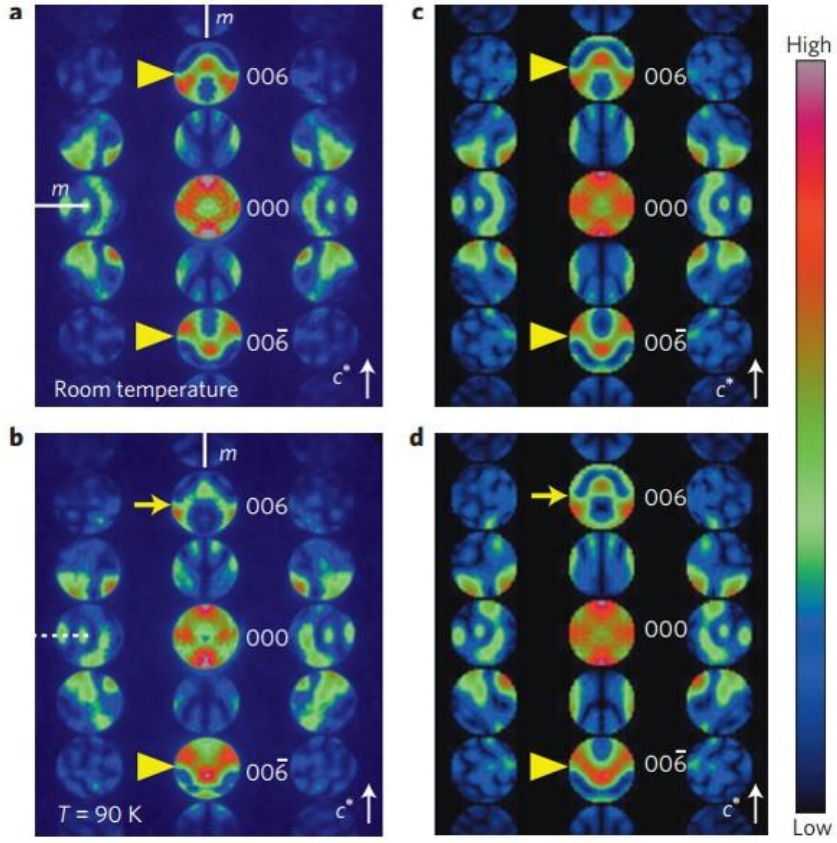

Figure 2-6. Experimental and simulated CBED patterns for LiOsO3 taken along [120] zone axis. (a) is experiment CBED pattern and (b) CBED simulation, are taken at room temperature. (c) is experiment CBED pattern and (d) CBED simulation, are taken at 90K temperature. [12]

different temperatures. As shown in Figure 2-6. The CBED pattern confirmed with certainty that at room temperature, its structure is centrosymmetric. In contrast, CBED pattern at 90K shows conclusively the

disappearance of centrosymmetry and those results agree with simulations. It also indicates that the polar axis is along the c axis of the unit cell. [12]

Additionally, if the strain is the driving force of the transition, the transition will be discontinuous, so that the magnetic ordering might be the driving force. However, from the neutron diffraction analysis, magnetic Bragg peaks from long range magnetic order cannot be found at T_s, and as a result, no evidence that magnetic order accompanies the structural transition. The lack of magnetic feature indicates $LiOsO_3$ behaves like a normal metal without strong electron correlations. [12]

Moreover, the data shows that the transition in $LiOsO_3$ is structurally equivalent to those in $LiNbO_3$ and $LiTaO_3$, two technologically important and extensively studied ferroelectrics. Since the transition in $LiOsO_3$ is unlikely to be due to collective electron dynamics, the structure data suggests that the transition mechanism is the same as that in $LiNbO_3$, which is most likely driven by a displacive or order-disorder process involving a shift in the mean positions of the Li atoms along the c axis. [12]

2.5 Hybrid Improper Ferroelectric

In order to discuss the origin of ferroelectricity in perovskites, it is useful to divide ABO_3 into two groups according to tolerance factor, t, which is created and defined as follows: $t = \frac{rA+rO}{\sqrt{2}\,(rB+rO)}$. Perovskites with t < 1 including the classical ferroelectric systems, like BaTiO3, are called B-site driven, because the B-site cation (i.e., Ti) is nominally too small for its site, and may tend to displace. [13] The ferroelectricity originates from the displacement of the B atom with regard to the octahedron formed by 6 O atoms around it. On the other hand, t < 1 or so-called A-site driven materials are generally not ferroelectric at all, and instead find favorable bond lengths by tilts and rotations of the BO_6 octahedron. [13]

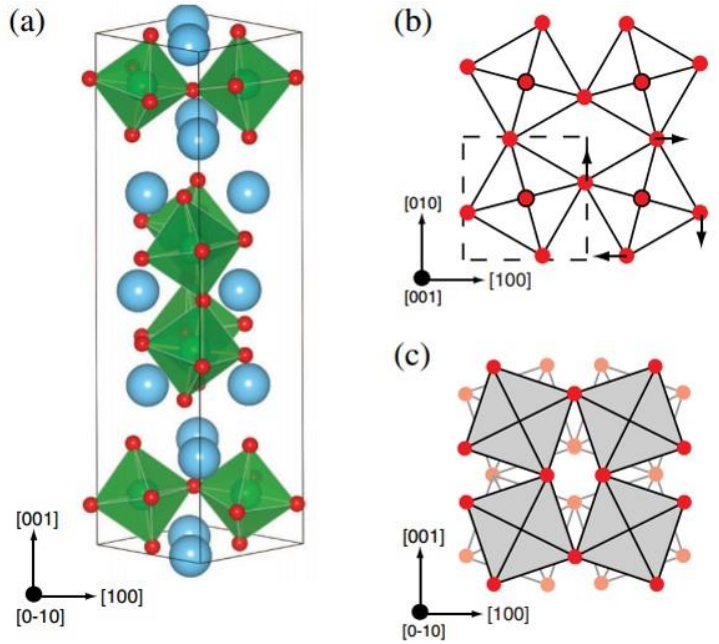

Figure 2-7. $Ca_3Mn_2O_7$ structure and rotation distortions. (a) The $A2_1am$ ferroelectric ground state structure. Large (blue) spheres correspond to Ca ions. (b) Schematic of the atomic displacements corresponding to the X_2^+ rotation. The dashed square denotes the unit cell of the I4/mmm parent structure. (c) Schematic of the X_3^- tilt mode. All axes refer to the coordinate system of the I4/mmm parent structure. [15]

However, in 2008, Bousquet, et al. [14] discovered that by layering perovskite in an artificial superlattice, a polarization can arise from the coupling of two rotational modes, and introduced the concept of improper ferroelectricity. Nicole A. Benedek et al. did further investigation on the mechanism by applying the First Principle calculations. They performed calculations on $Ca_3Mn_2O_7$, a layered perovskite as a member of the Ruddlesden-Popper series ($A_{n+1}B_nO3_{n+1}$), which occurs in the nature in bulk. [15] For $Ca_3Mn_2O_7$ (n = 2) the experimental shows the sequence of phase transitions from the paraelectric I4/mmm phase to the ferroelectric $A2_1am$ phase. In their calculations, the oxygen octahedron is significantly tilted with the structure. Their results indicated that the rotation mode and the tilt mode are the primary modes driving the transition to the ferroelectric phase. In contrast of the conventional improper ferroelectrics, in hybrid improper ferroelectric polarization may shift due to more than one lattice distortion, and it will result in two polar domains. [15] The two tilting mode is shown in Figure 2-7, and the hybrid improper ferroelectricity is the result of such lattice distortion. In short, if the octahedral

are tilted, and the tilting is in layers, then also the center of symmetry can be altered, as shown in Figure 2-8.

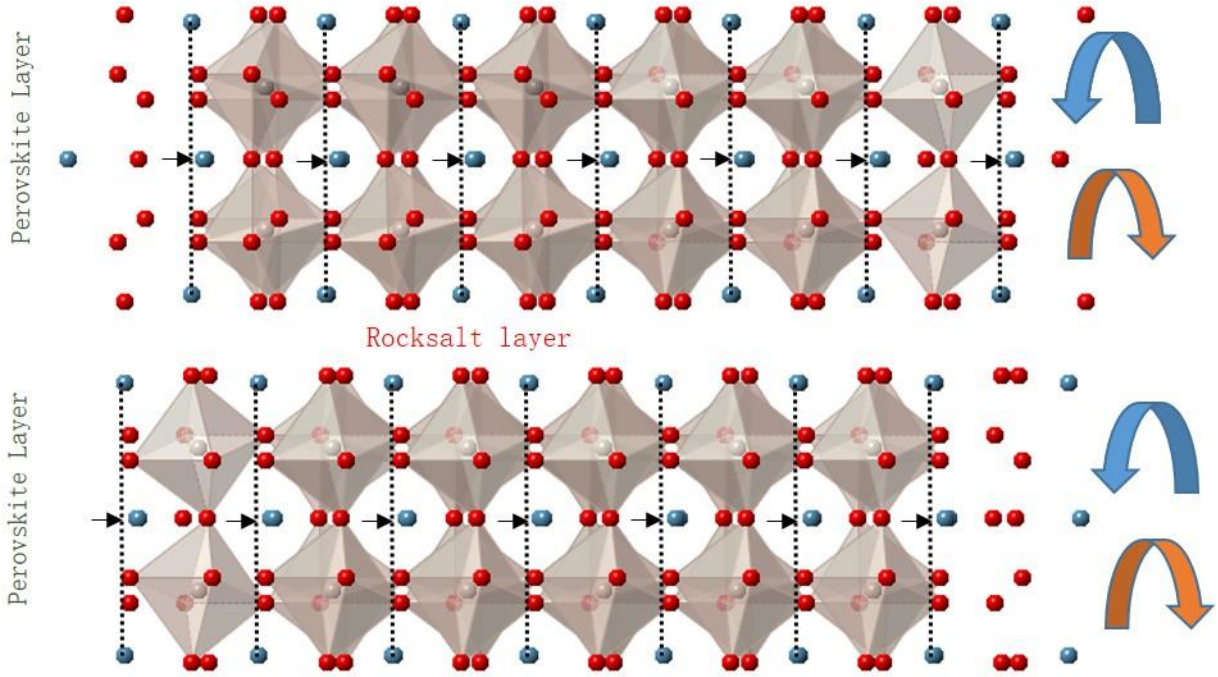

Figure 2-8. Schematics of the tilting in layered perovskites yielding the hybrid improper ferroelectricity. (Credit: Debangshu Mukherjee)

2.6 Domain walls in the ferroelectric materials

There are different polarities in the ferroelectric materials and they are energetically equivalent, so in theory, it is possible for them to appear simultaneously as the sample cools down from the paraelectric phase. As a result, zero-field cooled ferroelectrics can spontaneously divide into small regions of different polarities, which are called “domains”, and the boundaries between two domains are called “domain walls” or “domain boundaries”. An ideal infinite crystal of the ferroelectric phase that is in the single-domain state should be the most energetically stable, but the domain walls can be considered as a finite size effect that can help minimize the surface energy. The need to minimize the surface energy can overcome the barrier

to form domain walls, and thus the domains appear. Consequently, the overall behavior of the films may be different from bulk because it may be dominated by the domain walls. [16]

The domain size is determined by the competition between an energy of domain and the energy of the domain walls. The energy of the domains is proportional to the domain size, and we have equation E = Uw, where U is the volume energy density and w is the width of the domain. On the other hand, the energy cost for forming the domain wall is inverse proportional to the domain size (1/w), and thus the domain wall energy is E = σd/w. σ refers to the energy density per unit area. By adding up the energy of domains and energy cost of forming domain walls, famous square root equation is proposed: $w = \sqrt{\frac{\sigma}{U} d}$. This is called the Kittel's law, and this equation can be applied to the ferroelectrics with 180° domain walls, ferroelastic think films and epitaxial films with both ferroelectricity and ferroelasticity. The square root equation is thus a general property of all ferroics, and it holds for other domain patterns. [16]

In ferroelectric is induced by the broken space inversion. In this case, the domain walls separating regions of opposite polarizations are called 180° domain walls. 180° walls tend to be parallel to the polar axis, so as to avoid head-to-head convergence of the dipoles. But head-to-head domain walls are possible, and it has been recently directly observed using aberration-corrected TEM and it was found to be 10 times wider than the neutral walls. [17]

An 180° magnetic wall in the ferromagnetic material is typically Bloch or Néel-type as shown in Figure 2-9. The Bloch wall is defined as the polarization rotate in a plane parallel to the wall, whereas the Néel wall is defined as the polarization in a plane perpendicular to the wall. [18] Unlike in the ferromagnetics, ferroelectric polarization is not quantized, and thus it is allowed in different magnitude. Consequently, at domain walls, the polarizations can stay the same orientation and just reduce the magnitude, change sign and then increase magnitude. Such

nonchiral walls are called Ising-like walls. Otherwise, if the polarization is rotated across the domain walls, the domain walls are chiral. [16]

The chiral walls are generally not favored because the rotation of polarization means a large elastic cost; and a change in polarization perpendicular to the walls will lead to accumulation of charges at walls, leading to electrostatic cost. [16] As a result, 180° walls are traditionally viewed as Ising-like, but it is challenged by the report showing 180° walls in perovskite ferroelectric can be partially chiral. [18] It happens when two order parameters involved, such as ferroelectric and ferroelastic in perovskite.

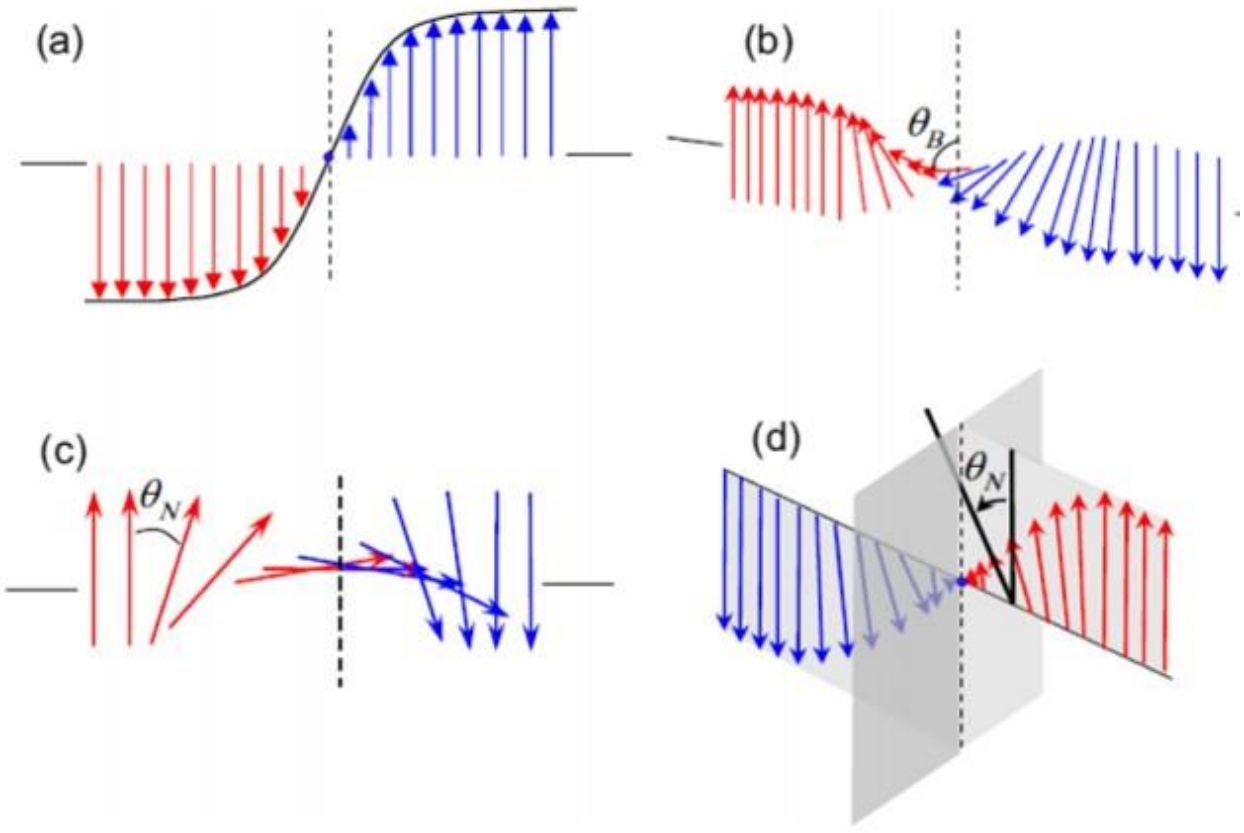

Figure 2-9. (a) Ising wall, (b) Bloch wall, (c) Neel wall, and (d) mixed Ising-Neel wall. Recent calculations show that domain walls in perovskite ferroelectrics tend to be of mixed character. [18]

2.7 Measurement and statistical Analysis

In 2016, Greg Stone et al. reported the first direct observation of the competing polar states in a Ruddlesden–Popper layered oxide. [19] They conducted DFT calculation and direct observation of the atom position with Picometer precision using STEM on $Sr_{n+1}Ti_nO_{3n+1}$ (with n=6) thin films. The combination of the DFT calculation and the measurement done on the STEM images are shown in Figure 2-10. In (a), Cation u_C and oxygen u_O interatomic spacing was measured in STEM image and also calculated with DFT method. The statistical method was applied on the measurement on STEM images, and the average value of such measurement matches the calculation almost perfectly. The STEM experiment was only post processed with linear drift correction [1], which makes the agreement more impressive. To show that this agreement is not a coincidence or only limited in the ferroelectric structure,

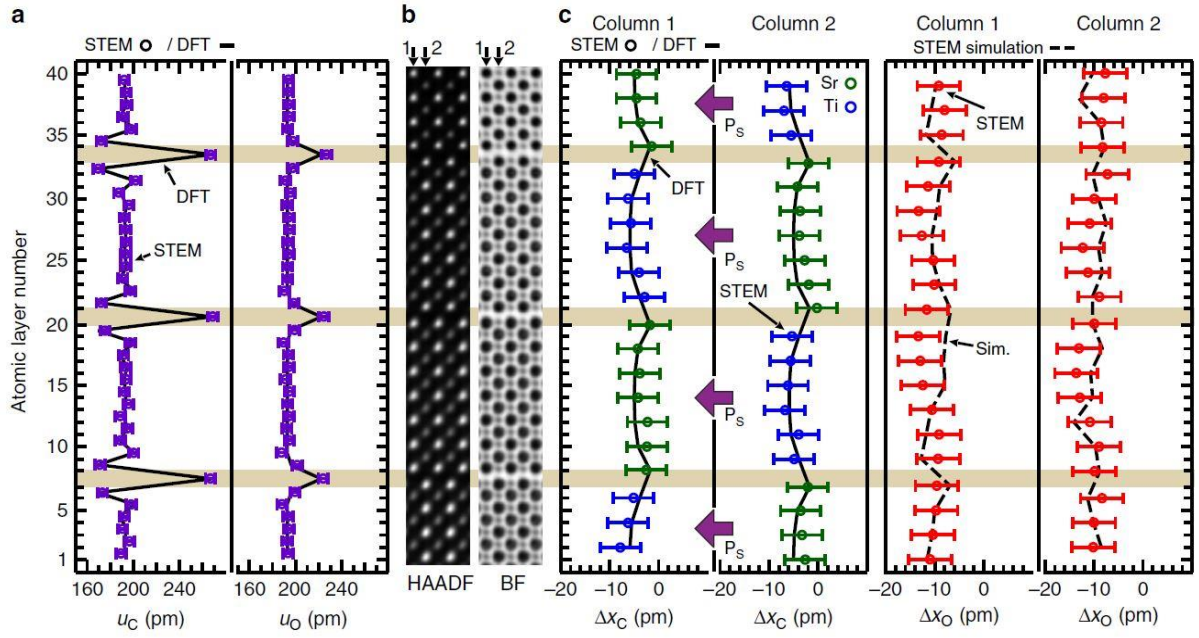

Figure 2-10 (a) Cation u_C and oxygen u_O interatomic spacing (purple open circle) measured on the drift-corrected HAADF and BF STEM images superimposed on the DFT calculated values (solid grey line). Exceptional agreement between experiments and DFT is seen. (b) Average experimental HAADF and BF STEM slices from the sample. (c) Cation Δx_c (blue and green open circles) and oxygen Δx_O (red open circles) displacements along the [100] along with the superimposed FE DFT calculated positions (solid black line) and FE DFT simulated oxygen positions (dashed black lines). The error bars are taken to be the root mean s.e. from the positions of a best fit lattice determined by cross-validation (CV) approach. [19]

they performed the same measurement on paraelectric $Sr_{n+1}Ti_nO_{3n+1}$ (with n=4) thin films, and the result

proves the nearly exact matching between statistically analyzed experimental data and the DFT calculation. [19]

Subsequently, the measurement on both HAADF and BF STEM images Figure 2-10 (b) were conducted to investigate the displacement of the cation atoms Δx_c (blue and green open circles) and the oxygen atoms Δx_O (red open circles). The result is shown in Figure 2-10 (c). As shown in the comparison, the experimental measurement data shows excellent agreement with the DFT calculation, and according to the experimental Δx_c data, the region is determined as ferroelectric. Also, considering the DFT calculation is done at 0 K in temperature, the experimental data obtained at near 300 K suggests the cation atom columns retained displacement similar to the predicted low-temperature phase. This work proves the accuracy of the statistical measurement on STEM images.

2.8 Engineering Consideration

Several considerations must be made in this transmission electron microscopy research on ferroelectric materials including the resources used as well as the implications of the work on the future applications. The combination of nanoscience and energy technology might lead to the next industrial revolution. From the economic and stainability standpoint, the TEM research is important in characterizing the atomic structure in the materials, which helps make nano-devices that are more powerful. [20] The US Department of Energy's Grand Challenge report includes the remarkable advances in exotic nanomaterials useful for energy research, from separation media in fuel cells, to photovoltaics and nano-catalysts which might someday electrolyze water under sunlight alone. To achieve this goal, it is crucial to master the techniques to control the materials synthesis at the atomic level. Additionally, the environmental issues in TEM research can be minimized, because the focused ion beam technique (FIB) allows minimum use of the

materials for sampling. As a result, multiple TEM observations are enabled to be conducted on an extremely small size sample. [2] Moreover, the research involves no ethical issues.

In terms of the manufacturability, LiNbO3 has already been massively produced in the industry for various applications. It is one of the most widely used ferroelectrics currently with applications ranging from surface active waveguides to Q-switched laser waveguides. [4]

In terms of social and political impact, the characterization at atomic level can lead to major breakthrough in science and technologies. In order to investigate the exact structure of the new materials being made, and the quality of these new materials, the synthesis must be accompanied with atomic scale analysis, which is the exact advantage of the TEM. TEM can produce atomic-resolution images of the materials and their defects, along with the exact composition and the diffraction pattern from sub-nanometer regions. [20] With the help of the TEM, more efficient, sustainable and economically important devices will be developed.

Chapter 3

EXPERIMENTAL METHODS

3.1. Sample Preparation and Imaging

Samples for this investigation were previously prepared using the Focused Ion Beam and were further imaged using the aberration-corrected S/TEM, Titan^3, at Penn State. The incident electrons from the TEM have the strong interaction with materials. The typical mean free path of an incident electron is only around 100 to 200 nm [21]. In order to achieve electronic transparency under the TEM, the sample thickness should not surpass 100 nm. In order to achieve better imaging quality on small atoms such as oxygen atoms, the sample thickness should be even smaller (about 30 nm).

The focused-ion beam (FIB) instrument is now widely used for TEM sample preparation. The fundamental mechanism of FIB is using energetic ions or atoms to bombard the surface of the sample, and sputter materials until the sample is thin enough. In the FIB instrument, voltage, temperature, types of ions and the geometry of incidence are all variables that can be controlled. [20] The accelerating voltage is usually 4 to 6 KeV, and the incident angle can be adjusted according to needs. Noble gas ions are often used as incident ions because they are more stable and less likely to react with the sample.

STEM observation was conducted at 200 keV in a probe corrected FEI Titan^3 G2*. On the $LiNbO_3$ sample, incident beam was parallel to the $[1\bar{1}00]$ axis of the crystal, and Simultaneous bright field (BF) STEM and annular dark-field (HAADF) STEM images were acquired. Two image sets are combined and drift-corrected according to recently developed drift-correction procedure. [1]

3.2 Detailed process of STEM image processing

3.2.1 Load the STEM image data and finding approximate atom positions

The STEM images obtained from FEI Titan^3 were saved as .ser files, which include spatial intensity for each pixel on the images and the calibration values for calculating the size of each pixel. The files were processed using self-build MATLAB scripts. The second step is to obtain a rough approximation of the atom positions in the image. The image processing code Realspacelattice01, written by Collin Ophus from NCEM, is used to find the approximate position. In short, this code helps identify the pixel with the highest intensity value within the radius of an atom, and it prompts the user to select the unit cell of the crystal and assign each

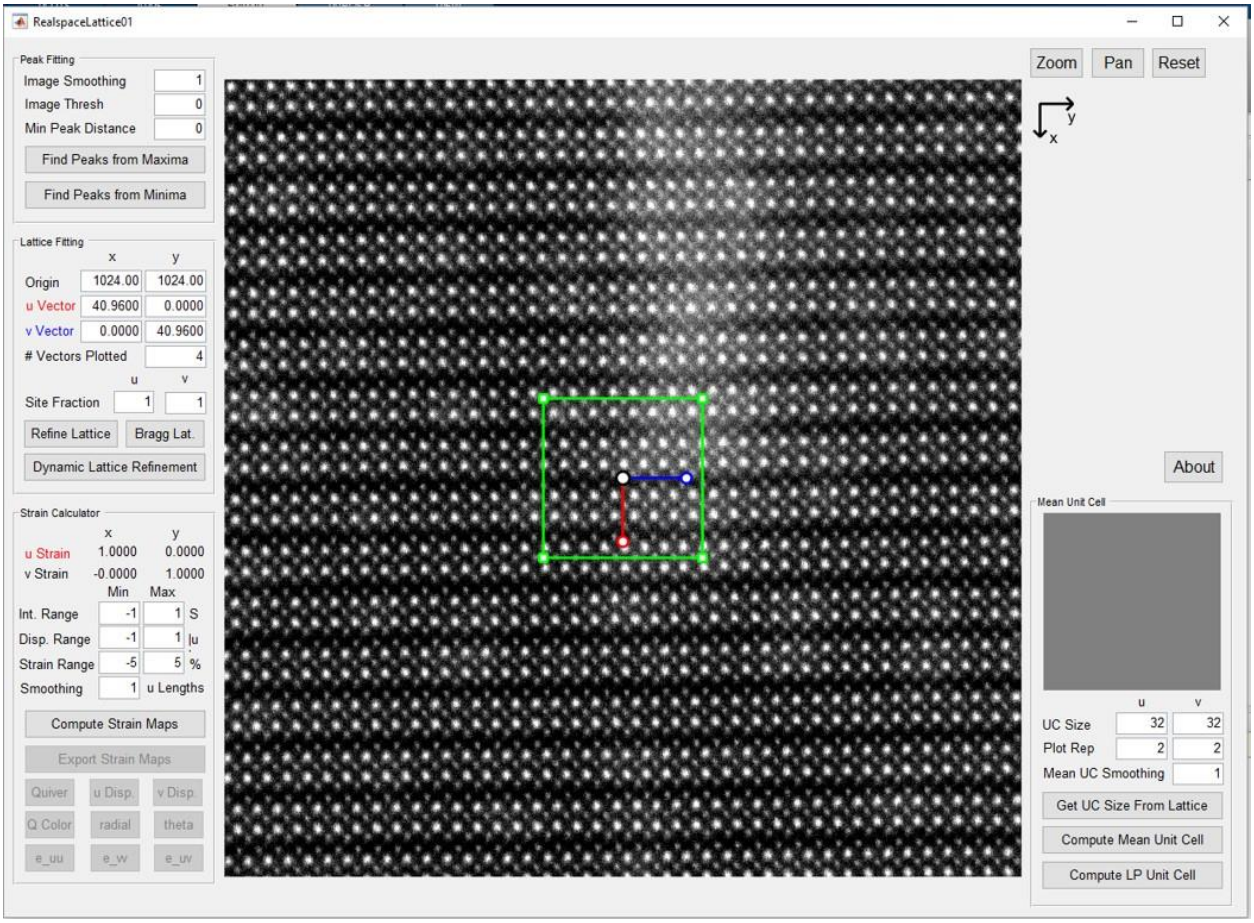

Figure 3-1: The user interface of the code 'Realspacelattice01'

atom in the unit cell a unique u and v unit-cell vector value. The u and v unit-cell vector values are crucial for later processing.

First step is to enter the value on the upper left corner, which is the configuration of the peak fitting. The ‘Image Smoothing’ prompts user to input a value used to reduce noise within the image and get a less pixelated image. [22] The ‘Image Thresh’ is a value used for image thresholding, which is an effective way of divide image into a foreground and a background. [23] The ‘Min Peak Distance’ is the value for user to enter that defines the minimum distance between two adjacent peaks. User can try different combination of those numbers to achieve the best position results. The next step is to enlarge the green square overlapped on the original

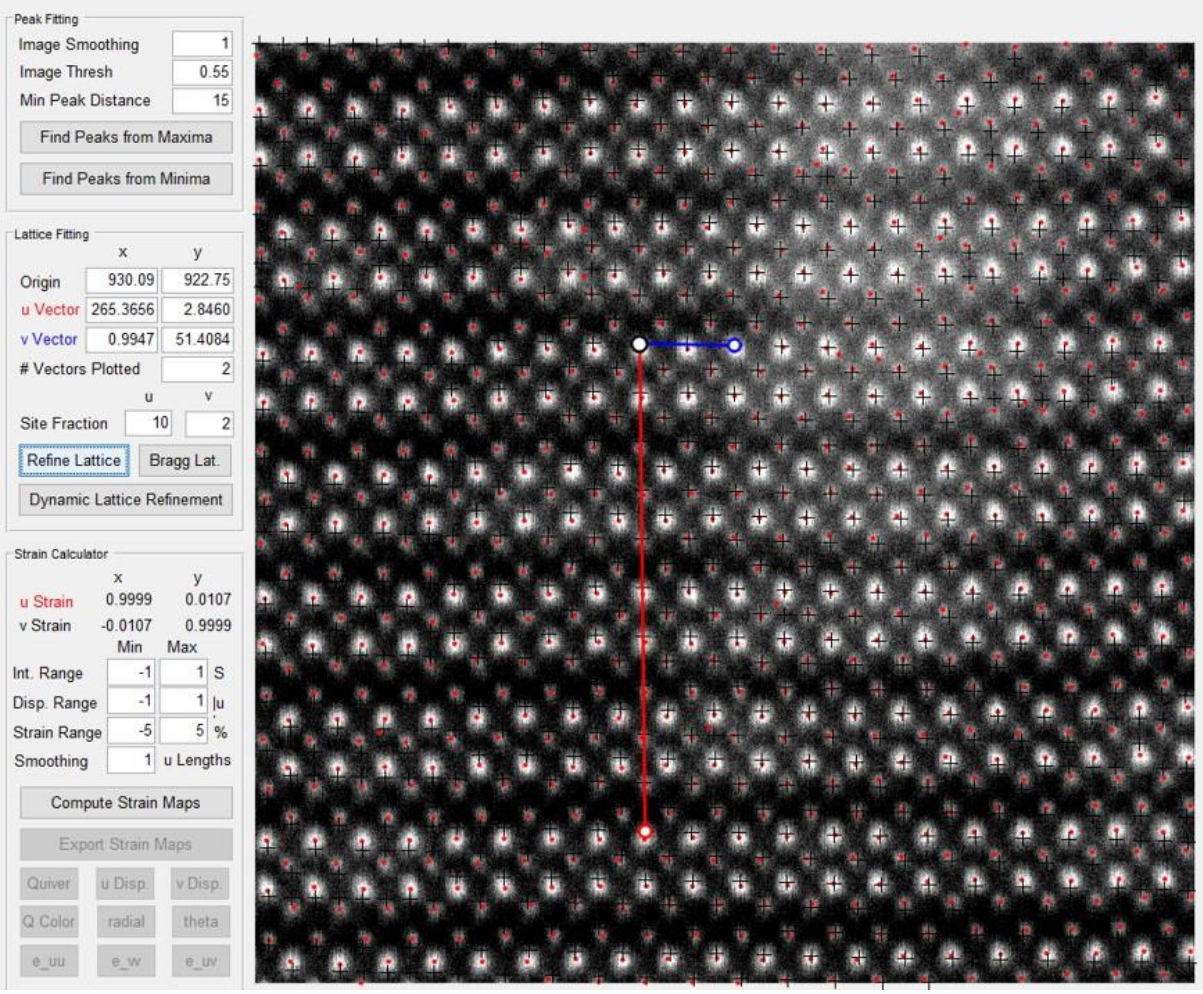

Figure 3-2: The result of the unit cell identification and the result of the atom position approximation accomplished by "Realspacelattice01" code

image and select the entire image. Click on the ‘Find Peaks from Maxima’ button and then the position of atoms will be shown with red dots.

The next step is to assign each atom the x and y value according to u and v vectors. In the case of $Ca_3Ru_{1.9}Ti_{0.1}O_7$, The # vector plotted is 2, which means the vectors contain two unit cells; and the site fraction for u and v are 10 and 2 respectively, which means the number of atoms contain in each unit vector. The value also depends on the needs of subsequent data processing, for example, to measure the average displacement of oxygen atoms in the unit cell of $LiNbO_3$, the u and v vectors are defined as twice the length of a unit cell. Afterwards, drag the tip of the vector to the approximate length and angle of the unit cell. Then click on Refine Lattice. The x shown on atoms is the position fitted with respect to the vectors. If the results is satisfying, you can close the window of this code. The new data will be saved as a data sets in the workspace of MATLAB. The result of the initial peak fitting and the assignment of the unit-cell vector is shown in Figure 3-2.

3.3.2 Fixing Errors and finding precise location of the atoms

The atom positions acquired from the rough peak fitting contains errors and the position is not often accurate. The errors were generated during the process of the peak fitting by the code due to the noise in the image and the miss fitting of the peaks outside the atoms. The example of the errors occurred in the image after the approximate peak fitting is shown in Figure 3-2. By

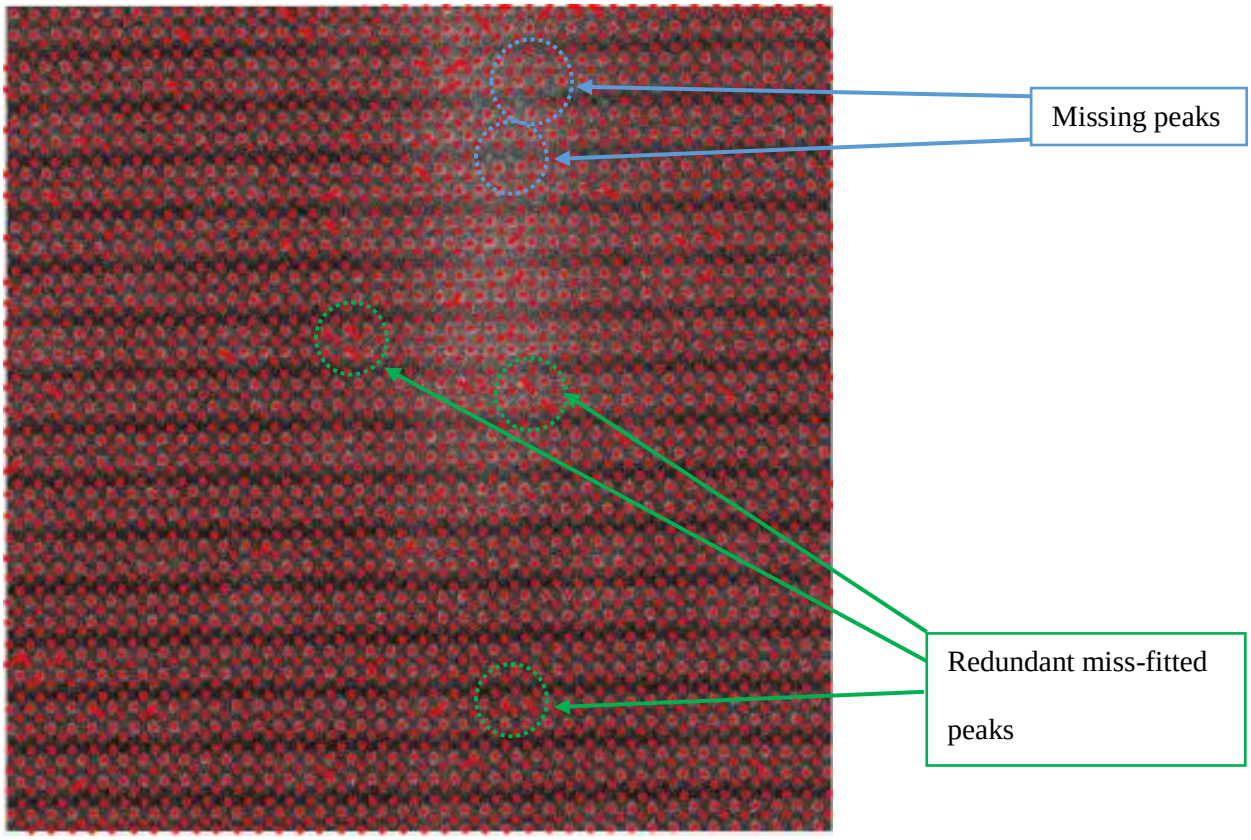

Figure 3-3: Demonstration of errors produced by approximate peak fitting. Missing points are shown using navy arrows and circles, while the redundant peaks are shown using green arrows and circles

manually adding missing peaks and deleting redundant ones, the errors can be mostly eliminated.

Use the mouse to select the errors in the image one by one by clicking on the redundant miss-fitted peaks and the missing peak positions in the image. The coordinates of the errors will show on the image and after selecting all the errors, export the coordinates data into the workspace. Run the self-built code ‘missing_pts_v2’, and the errors will be removed

automatically. The process is shown in Figure 3-4. A comparison of an initial peak fitting image and its processed error-free image is shown in Figure 3-5.

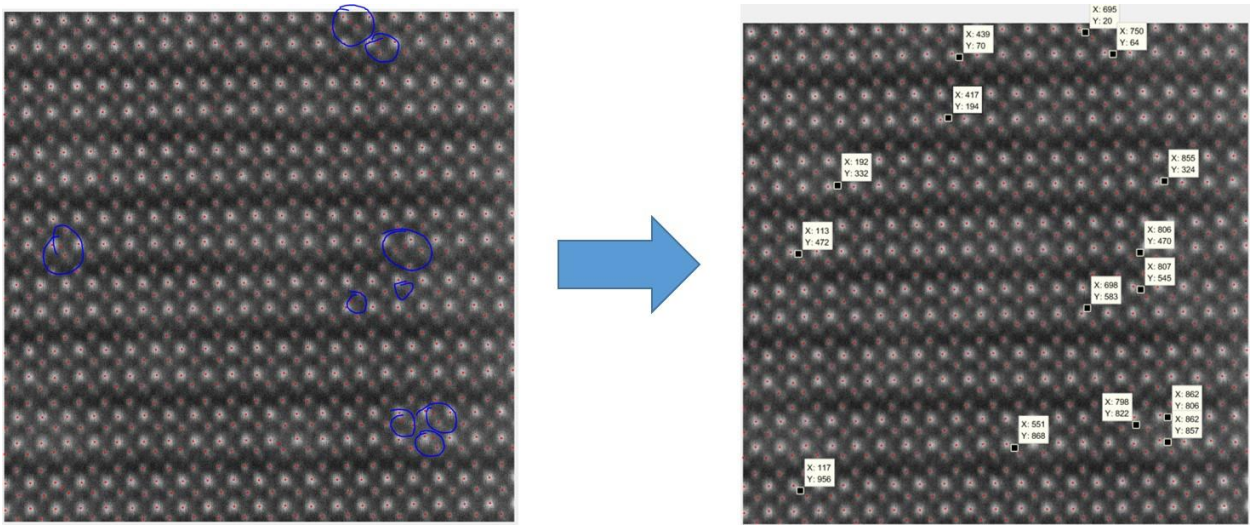

Figure 3-4: The process of finding and selecting the error. The coordinates shown in the right figure will be exported and processed with the self-built code 'missing_pts_v2'.

After obtaining the error-free image, the Gaussian fitting of the intensity inside the atoms in the image was performed. The fitting code was named "STEMlat01_gs", written by Dr. Greg Stone. In short, this code selects the region that contains the intensity from an atom, collect the intensity inside that atom and perform Gaussian fitting with respect to the intensity and the location of the pixel. The equation of the Gaussian fitting is as shown: $= \sum_{i=1}^{n} a_i e^{[-\left(\frac{x-b_i}{c_i}\right)^2]}$. In this equation, a is the amplitude, b is the centroid, c is related to the peak width, and n is the number of peaks to fit. [24] By applying the Gaussian fitting to the intensity of atoms in the STEM image, the most precise atom positions were achieved. The Gaussian fitting process and the image with precise atom positions are shown in Figure 3-6. Since the precise positions of all atoms in the STEM images were obtained, statistical analysis can be performed with high accuracy.

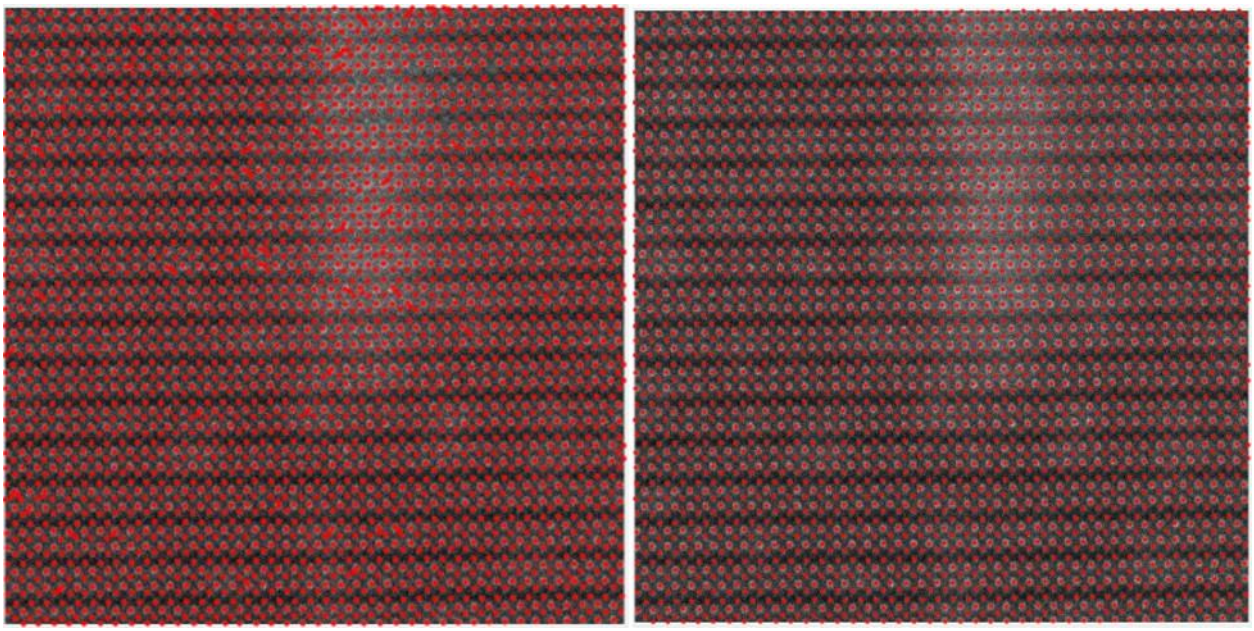

Figure 3-5: A comparison of an initial peak fitting image (left) and its processed error-free image (right)

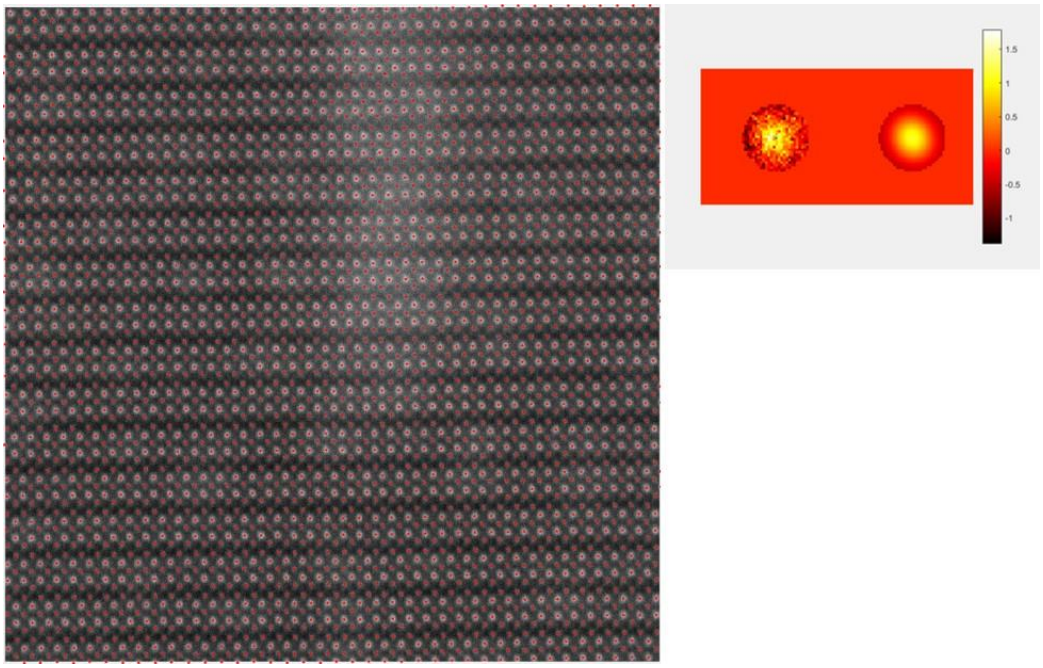

Figure 3-6: The Gaussian fitting process (right) and the resulting image with precise atom positions (left). The code "STEMlat01_gs", selects the region that contains the intensity from an atom, collect the intensity inside that atom and perform Gaussian fitting with respect to the intensity and the location of the pixel.

Chapter 4

RESULTS AND DISCUSSION

4.1 $LiNbO_3$

4.1.1 BF-STEM Image of $LiNbO_3$

STEM images of $LiNbO_3$ were obtained at 300 keV in a probe spherical aberration corrected FEI Titan^3 G2*. One of those images is shown in Figure 4-1(A). To achieve the oxygen imaging, the $LiNbO_3$ samples were previously prepared using FIB with the thickness of less than 30 nm. The samples were imaged using aberration-corrected STEM with four Nb atoms and a central oxygen atom form each unit

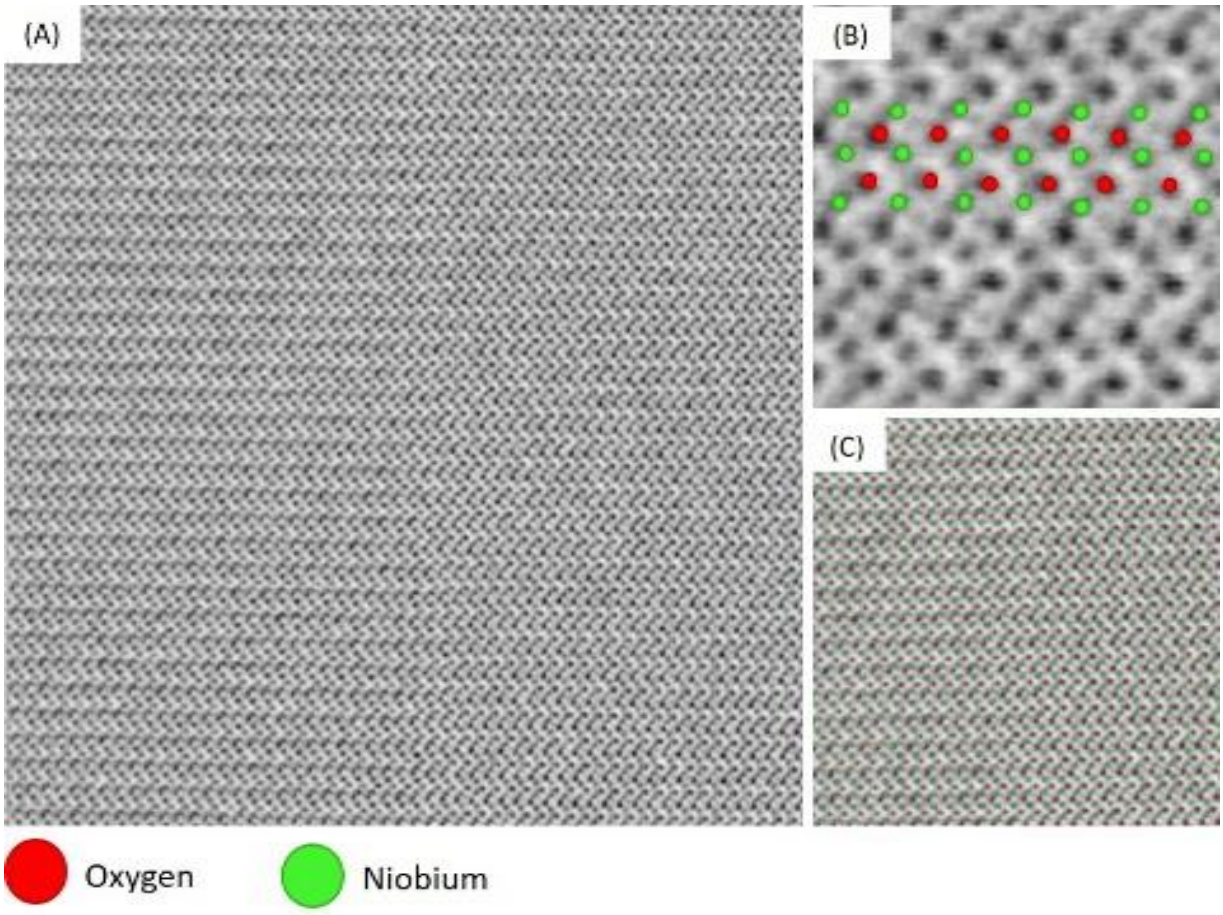

Figure 4-1: (A) a BF-STEM image of $LiNbO_3$ viewed from the [$1\bar{1}00$] zone axis. (B) The top-left corner of the image of (A), which is magnified to show the crystal structure of $LiNbO_3$. O and Nb atoms are labeled using red and green respectively. (C) the image after the atom identification. All O and Nb atoms are labeled in red and green respectively

cell. Lithium is not possible to be imaged since it is a very low scattering center and is also slightly overlapped with the Nb atoms along the column. From magnified image in Figure 4-1 (B), it can be found that the Oxygen atoms is displaced off the center of the four nearby Niobium atoms. This displacement results in non-centrosymmetry, which is the origin of the ferroelectricity in $LiNbO_3$.

4.1.2 The Visualization of the Polarization Vectors

The BF-STEM image of $LiNbO_3$ viewed from the $[1\bar{1}00]$ zone axis was taken on a domain wall region, but the domains are not clear to visualize without further image processing. We firstly implemented the code to identify all oxygen atoms and niobium atoms in the image. In short, this code sorts out all atom positions with respect to the unit cell vectors assigned in the previous step. The atom identification process result is shown in Figure 4-1 (C). As shown in the schematic of the $LiNbO_3$ unit cell in Figure 4-2, it is visible that the Oxygen atoms displaces in two different directions in two adjacent rows. To obtain the displacement of the oxygen atoms with respect to the Nb atoms, we need to calculate an average value of the oxygen displacement regardless of the displacement direction. We first select the unit cells in $LiNbO_3$ to include two adjacent rows of Oxygen atoms and three rows of Niobium atoms, as shown in Figure 4-2. The average positions of the two Oxygen atoms and their surrounding Nb atoms in

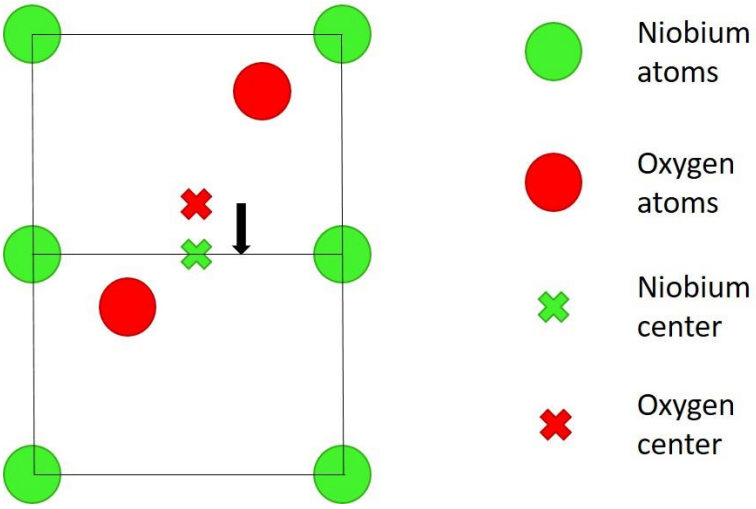

Figure 4-2: The schematics of the unit cell used for analyzing the atom displacement

each unit cell were then obtained. The displacement is calculated by comparing the oxygen center and the niobium center.

To visualize the polarization caused by the off-center oxygen displacement, we use self-built code to draw a vector from the average position of the Nb atoms to the average position of the O atoms for each unit cell. This vector map can help visualize the change of direction and magnitude of the polarization in two domains and across the domain wall. The vector map of polarization of $LiNbO_3$ is shown in Figure 4-3 (a). From the map, the two domains of opposite polarization direction and the domain wall in the middle is clearly visualized. Since the polarization directions in two adjacent domains are opposite to each other, the domain wall is identified as the 180-degree domain wall. The width of the wall is equivalent to that of six unit cells in the sample. In Figure 4-3 (b), a closer observation was made at the domain wall region where the switching of polarization directions occurred. There is the in-plane rotation of the polarization vectors perpendicular to the domain wall (in this case from left to right), which is the characteristic of the Néel-wall. Additionally, there is a change in magnitude as well, which is the Ising-wall feature. The schematic of this change across the domain wall is depicted in Figure 4-3 (c). In 2009, Donghwa Lee et al. predicted the mixed Ising-Néel-Bloch characteristic in the domain wall of $LiNbO_3$. [18] This vector map obtained from STEM imaging and data processing is the first direct experimental evidence of such mixed characteristic in the $LiNbO_3$.

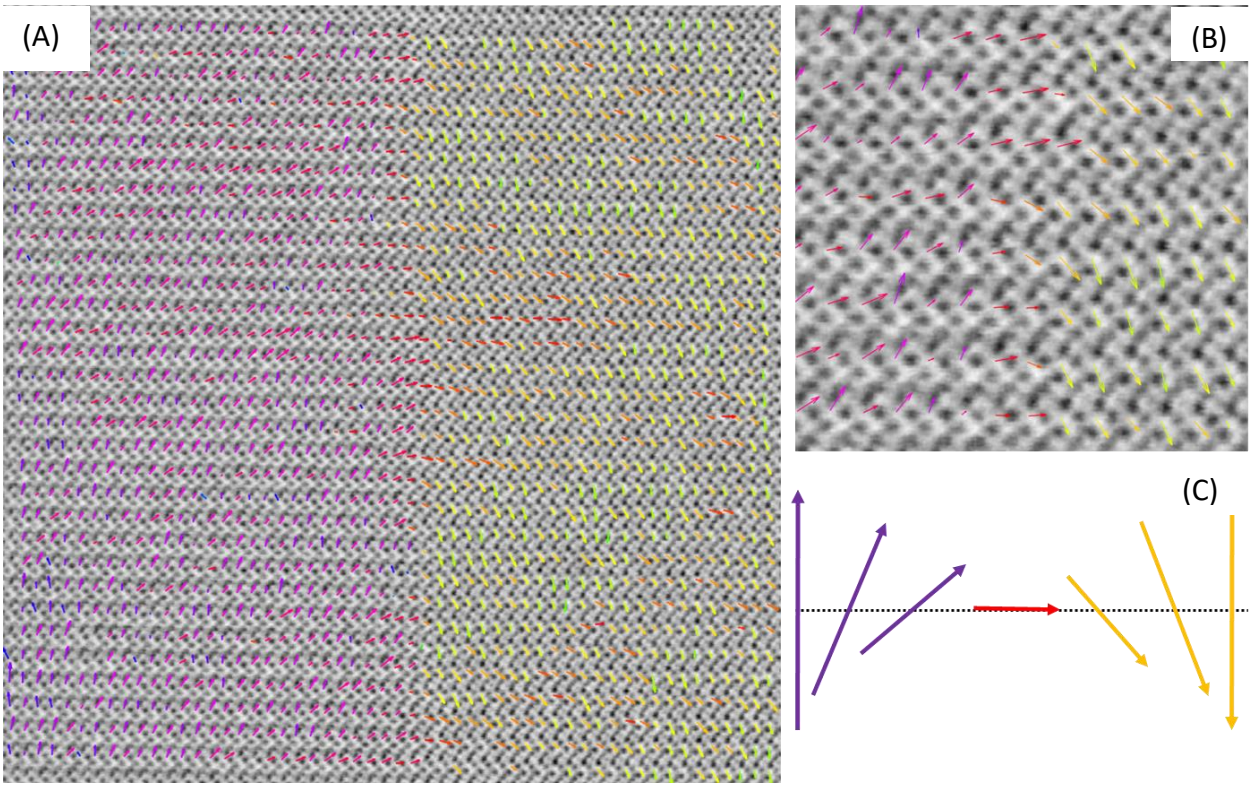

Figure 4-3: (A) the vector map generated on the BF-STEM image of $LiNbO_3$. With the help of this vector map, two domains of opposite polarization direction and the 180° domain wall in the middle are clearly visualized; (B) a magnified image from (a) at the domain wall region. The change in the polarization vectors across the domain wall can be observed directly; (C) the schematic of the change in polarization across the domain wall.

4.1.3 Statistical Analysis

We further conducted statistical analysis near the wall to quantitatively measure the mixed Ising and Néel characteristic of the domain wall. For this analysis, the perpendicular and parallel to wall movement of the Oxygen atoms are analyzed statistically in the two in-plane directions (x and y) with respect to the center of the defined cell. As shown in Figure 4-2, the movement of the oxygen atoms are measured by comparing to the center of the two oxygen atoms and the center of the six niobium atoms in one unit cell. The Ising-like domain wall exhibits a gradual magnitude change without rotation of the polarization. [18] As a result, to measure the Ising-type walls, we calculated the change in magnitude parallel to the domain wall, as shown in Figure 4-4. In (A), we measured the parallel-to-wall displacement of the O and Nb atoms with respect to the center of the unit cell; in (B) the plot shows the relative

displacement of O atoms to Nb atoms. The result shows that the magnitude of the polarization parallel to the domain wall changes gradually. To be exact, the magnitude in the first domain decreases gradually at the domain wall and increases in the other direction on the other side of the wall, which matches the definition of the Ising-like domain walls. This quantitative measurement proves the Ising-like behavior of the domain wall, the displacement of the atoms across the domain wall is approximately 30 pm.

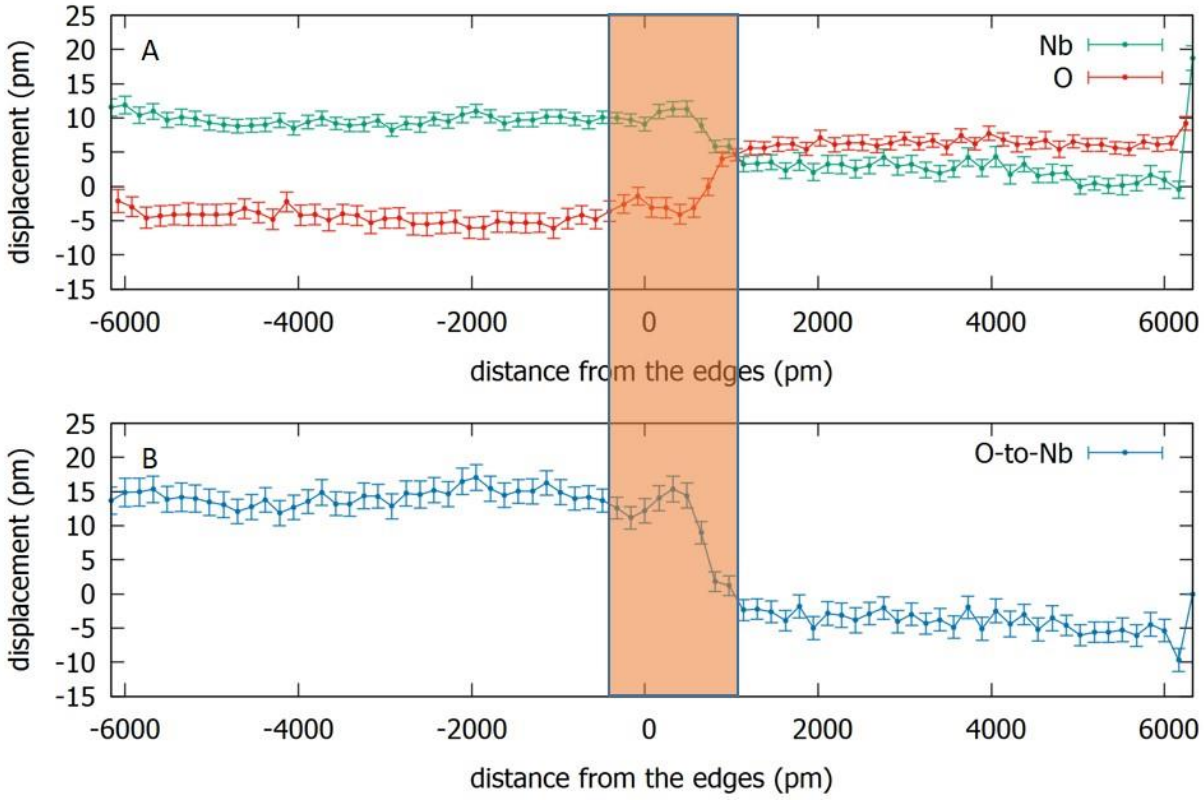

Figure 4-4: (A) the measurement of the parallel-to-wall displacement of the O and Nb atoms with respect to the center of the unit cell. (B) The plot showing the relative displacement of O atoms to Nb atoms

On the other hand, the Néel type walls show in-plane rotation without the change in magnitude. It is common in ferromagnetic materials since the magnetic field is quantized. [18] To examine this, we measured the displacement of atoms in the direction perpendicular to the wall with respect to the defined cell. If there is a mixed Ising and Néel wall components, the magnitude of displacement in perpendicular to wall direction will change gradually, as the component in this direction changes due to the mixed wall components. As a result, to measure the Néel type walls, we calculated the change in magnitude parallel to the domain wall, as shown in Figure 4-5. In (A), we measured the atom position for Nb and O in the

direction perpendicular to the wall with respect to the center of the unit cell; in (B) the plot shows the relative displacement of O atoms to Nb atoms. The result shows that the magnitude in the direction perpendicular to the domain wall decreases at the domain wall region; to be exact, the magnitude in the first domain decreases gradually at the domain wall and increases in the same direction on the other side of the wall, which matches the behavior of the Néel type domain walls. The displacement of the atoms across the wall is approximately 10 pm. This quantitative measurement proves the Néel type characteristic of the domain wall.

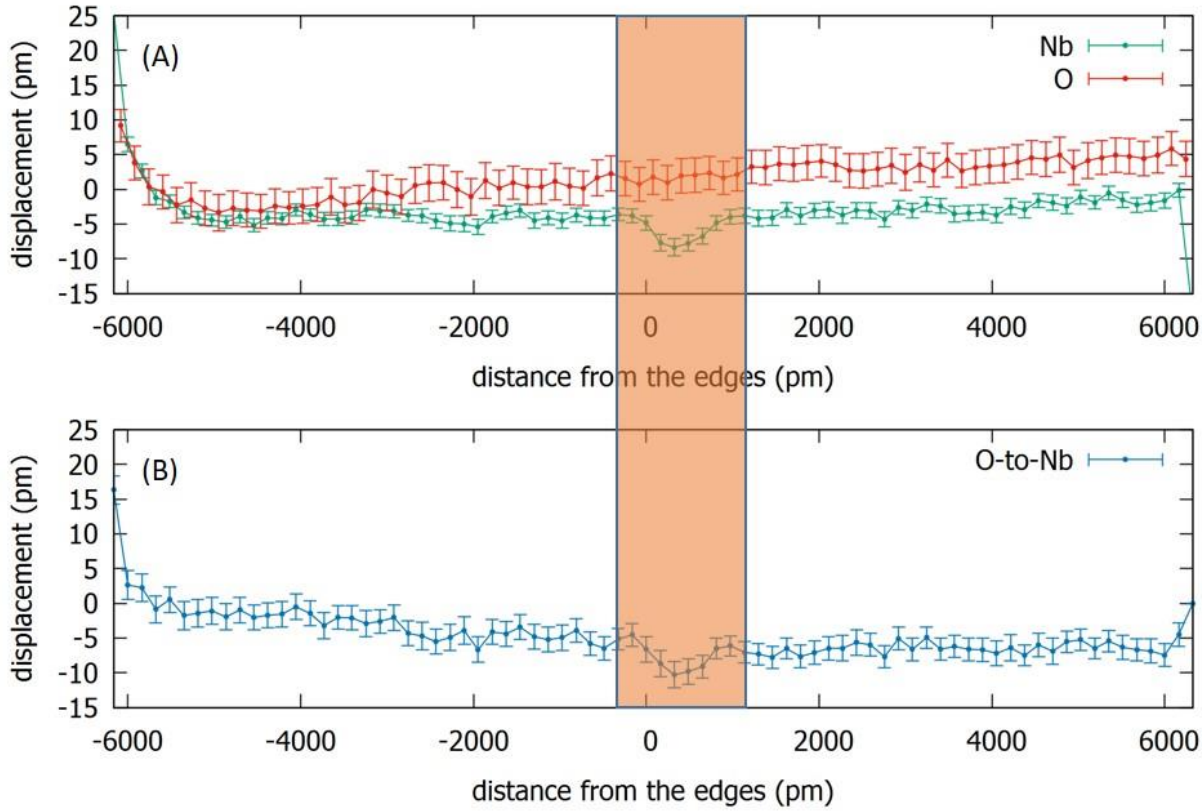

Figure 4-5: (A) the measurement of the perpendicular-to-wall displacement of the O and Nb atoms with respect to the center of the unit cell. (B) The plot showing the relative displacement of O atoms to Nb atoms

Chapter 5 CONCLUSION

5.1 Summary of Results

An investigation of the domain wall type in $LiNbO_3$ was completed. The BF-STEM imaging was previously performed on the $LiNbO_3$. This thesis explore statistical analysis methods that enabled generations of polarization vector maps and displacements to visualize the polarization in the structure. The statistical analysis was conducted to quantitatively measure the displacement. The polarization vector map visualized the polarization in the whole image, showing both gradual magnitude and the in-plane rotation across the 180° domain wall, which exhibit both Ising-type and Néel-type feature of the 180° domain wall in the $LiNbO_3$.

The statistical analysis quantitatively measured the perpendicular and parallel to wall movement of the Oxygen atoms along the perpendicular to wall directions. In parallel to wall direction, the displacement shift of the atoms across the domain wall is approximately 30 pm. The magnitude in the first domain decreases gradually at the domain wall and increases in the other direction on the other side of the wall. On the other hand, in perpendicular to wall direction, the result shows that the displacement magnitude in the perpendicular to the domain wall direction slightly changes at the domain wall region. To be exact, the magnitude in the first domain decreases gradually at the domain wall and increases in the same direction on the other side of the wall. The displacement of the atoms across the wall is approximately 10 pm. The quantitative measurement supports the observation of the Ising-like and the Néel-like behavior across the domain wall in the polarization vector map. The comparison of the results from the polarization vector map and the statistical analysis is shown in Figure 5-1. Consequently, both the polarization map and the statistical analysis on the 180° domain wall in $LiNbO_3$ experimentally

observed the mixed Ising-Néel feature. This result strongly supports the prediction by Donghwa Lee et al. [18]

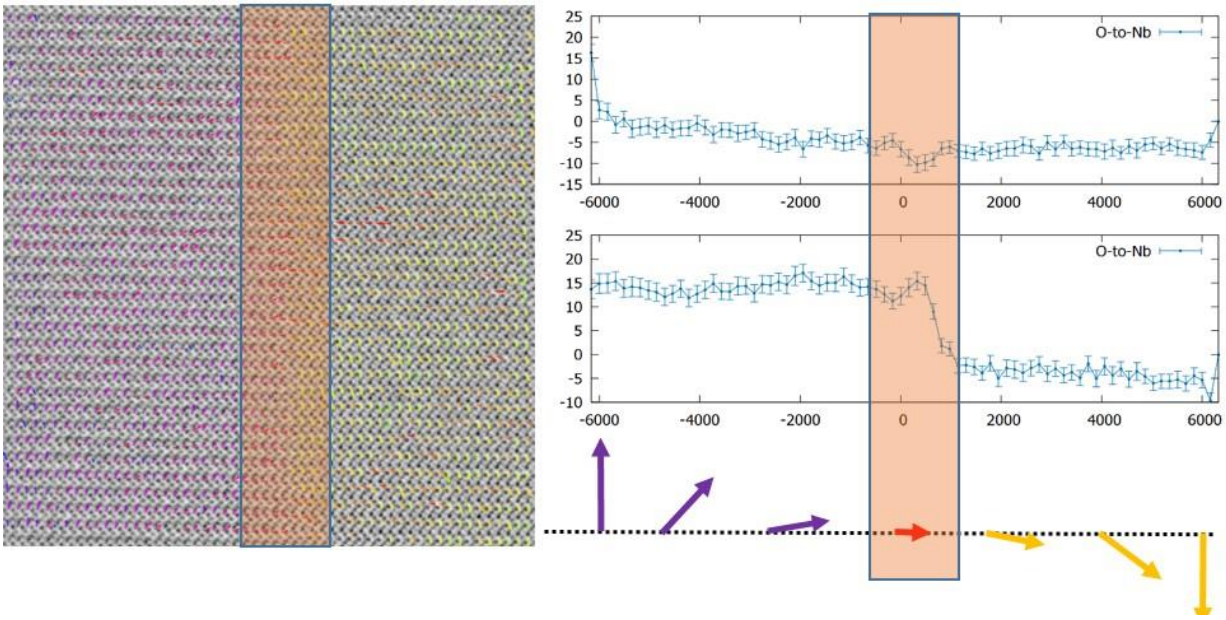

Figure 5-1: The comparison of the results from the polarization vector map and the statistical analysis.

Chapter 6

FUTURE WORK

The results presented in this work are a beginning for further study on domain walls and interfaces not only in $LiNbO_3$ but also in other complex oxide systems. Several future possibilities of research are listed in this chapter.

There is a recent report indicating that a ferroelectric material can undergo phase transformation under the external voltage, instead of exhibiting atomic displacements in its structure. If a material undergoes phase transformation under voltage, it cannot be considered as a ferroelectric material. This finding may put doubt on many materials now considered as ferroelectric materials because very few were examined if undergoes phase transition under voltage. [25]

Consequently, one future study on this topic is conducting in-situ TEM experiment on some known ferroelectric materials, and see if they undergo phase transition under voltage instead of having atom displacement in the structure. $LiNbO_3$ and other materials can be examined using this method, to test if it is truly a ferroelectric material according to the strict definition.

Another direction would be to further explore the structure of the domain walls in non-congruent $LiNbO_3$ sample with a stoichiometric ratio for Nb, O, and Li, which has a different density of defects than the studied congruent $LiNbO_3$ sample. It is believed that defects or substitutional atoms can introduce strain and affect the atomic structure of the domain walls.

Appendix A

Codes Employed

A.1 Maximum Peak Fitting

The image processing code Realspacelattice01, written by Collin Ophus from NCEM, is used to find the approximate position. In short, this code helps identify the pixel with the highest intensity value within the radius of an atom, and it prompts the user to select the unit cell of the crystal and assign each atom in the unit cell a unique u and v unit-cell vector value. The u and v unit-cell vector values are crucial for later processing.

```
function varargout = RealspaceLattice01(varargin)
% varargout.calibration = varargin.calibration;
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @RealspaceLattice01_OpeningFcn, ...
                   'gui_OutputFcn',  @RealspaceLattice01_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before RealspaceLattice01 is made visible.
function RealspaceLattice01_OpeningFcn(hObject, eventdata, handles, varargin)

% initialize main struct "s" if not provided
if isstruct(varargin{1})
    handles.s = varargin{1};
    if nargin == 5
        I = varargin{2};
        handles.s.imageIntMean = mean(I(:));
        I = I - handles.s.imageIntMean;
        handles.s.imageIntSD = sqrt(mean(I(:).^2));
```

```
        I = I / handles.s.imageIntSD;
        handles.s.image = I;
    end
else
    handles.s = struct();
    % get image data, normalize
    I = varargin{1};
    handles.s.imageIntMean = mean(I(:));
    I = I - handles.s.imageIntMean;
    handles.s.imageIntSD = sqrt(mean(I(:).^2));
    I = I / handles.s.imageIntSD;
    handles.s.image = I;
    % inialize all needed variables
    handles.s.imageSize = size(handles.s.image);
    xr = round(handles.s.imageSize(1)*[.4 .6]) + [1 0];
    yr = round(handles.s.imageSize(2)*[.4 .6]) + [1 0];
    handles.s.p = [xr(1) yr(1);xr(2) yr(1);xr(2) yr(2);xr(1) yr(2)];
    handles.s.pos = [-1 -1 0 0 0 0];
    % Control variables
    handles.s.peakSmoothing = 1;
    handles.s.peakThresh = 0;
    handles.s.peakMinDist = 0;
    % lattice variables
    handles.s.lat = ...
        [handles.s.imageSize/2;
        [handles.s.imageSize(1)/50 0];
        [0 handles.s.imageSize(2)/50]];
    handles.s.latNumPlot = 4;
    handles.s.latSiteFrac = [1 1];
    % Strain variables
    handles.s.strainIntRange = [-1 1];
    handles.s.strainDispRange = [-1 1];
    handles.s.strainRange = [-5 5];
    handles.s.strainSmooth = 1;
    % Mean UC
    handles.s.UCsize = [32 32];
    handles.s.UCrep = [2 2];
    handles.s.UCsigma = 1;
    handles.s.UCmean = zeros(handles.s.UCsize);
    handles.s.UCsig = zeros(handles.s.UCsize);
    handles.s.UCcount = zeros(handles.s.UCsize);
end

if ~isfield(handles.s,'imagePlotScale')
    handles.s.imagePlotScale = [-1 3];
end

% GUI initialization
tic
handles.flagROI = 0;
handles.flagLat = 0;

% Plot x,y coordinate vectors in lower right
set(handles.figure1,'currentaxes',handles.axesXYcoord);
cla
```

```
% hold on
line([1 0 0],[0 0 1],'linewidth',2,'color','k')
line([-.2 0 -.2]+1.025,[-.2 0 .2],'linewidth',2,'color','k')
line([-.2 0 .2],[-.2 0 -.2]+1.025,'linewidth',2,'color','k')
text(1,0.35,'y','fontsize',12,'horizontalalign','center')
text(0.3,1,'x','fontsize',12)
% hold off
axis equal off
xlim([-.3 1.3])
ylim([-.3 1.3])
set(gca,'ydir','reverse')


% Image coordinate system
[handles.ya,handles.xa] = meshgrid(1:handles.s.imageSize(2),...
    1:handles.s.imageSize(1));

% Initial plotting for main axes
set(handles.figure1,'currentaxes',handles.axesMain);
imagesc(handles.s.image);
caxis(handles.s.imagePlotScale)
hold on
% Image and lattice peak positions
handles.posImage = scatter(handles.s.pos(:,2),handles.s.pos(:,1),...
    'marker','.','markeredgecolor',[1 0 0],'sizedata',80);
handles.posFit = scatter(handles.s.pos(:,6),handles.s.pos(:,5),...
    'marker','+','sizedata',80,...
    'markeredgecolor',[0 0 0],'markerfacecolor','none');
% Draw ROI selection region
handles = drawROI(handles);
% Draw lattice
handles.u = plot(handles.s.lat(1,2)...
    + [0 handles.s.lat(2,2)*handles.s.latNumPlot],...
    handles.s.lat(1,1)...
    + [0 handles.s.lat(2,1)*handles.s.latNumPlot],...
    'linewidth',2,'color','r',...
    'marker','o','markerfacecolor','w','markersize',8);
handles.v = plot(handles.s.lat(1,2)...
    + [0 handles.s.lat(3,2)*handles.s.latNumPlot],...
    handles.s.lat(1,1)...
    + [0 handles.s.lat(3,1)*handles.s.latNumPlot],...
    'linewidth',2,'color','b',...
    'marker','o','markerfacecolor','w','markersize',8);
handles.or = plot(handles.s.lat(1,2),...
    handles.s.lat(1,1),...
    'linewidth',2,'color','k',...
    'marker','o','markerfacecolor','w','markersize',10);



hold off
axis equal off
colormap(gray(256))
% handles.s.imageSize
```

```
ylim([1 handles.s.imageSize(1)])
xlim([1 handles.s.imageSize(2)])



% Set up GUI text values and drawing things
updatePeakValues(handles)
updateLatValues(handles);
updateStrainValues(handles);


% If strain maps exist, activate buttons
if isfield(handles.s,'strainEuu')
    set(handles.pushbuttonStrainExport,'enable','on');
    set(handles.pushbuttonStrainU,'enable','on');
    set(handles.pushbuttonStrainV,'enable','on');
    set(handles.pushbuttonStrainRadial,'enable','on');
    set(handles.pushbuttonStrainTheta,'enable','on');
    set(handles.pushbuttonStrainEuu,'enable','on');
    set(handles.pushbuttonStrainEvv,'enable','on');
    set(handles.pushbuttonStrainEuv,'enable','on');
    set(handles.pushbuttonQuiverPlot,'enable','on');
    set(handles.pushbuttonQuiverColour,'enable','on');
end


% intialize mean UC to black
drawMeanUC(handles)
% Update UC size values
updateUCValues(handles)

% Choose default command line output for RealspaceLattice01
handles.output = handles.s;
% Update handles structure
guidata(hObject, handles);
% UIWAIT makes RealspaceLattice01 wait for user response (see UIRESUME)
uiwait(handles.figure1);


function drawMeanUC(handles)
% mean UC plotting
set(handles.figure1,'currentaxes',handles.axesMeanUC);
imagesc(repmat(handles.s.UCmean,handles.s.UCrep))
axis equal off
colormap(gray(256))
% caxis([0 1])


% --- Outputs from this function are returned to the command line.
function varargout = RealspaceLattice01_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.s;
delete(handles.figure1);
```

```
function drawpeakStrain(handles)
% mean UC plotting
set(handles.figure1,'currentaxes',handles.axespeakStrain);
imagesc(repmat(handles.s.UCmean,handles.s.UCrep))
axis equal off
colormap(gray(256))
% caxis([0 1])


function updatePeakValues(handles)
set(handles.editPeakImageSmoothing,'string',handles.s.peakSmoothing);
set(handles.editPeakThresh,'string',handles.s.peakThresh);
set(handles.editPeakMinDist,'string',handles.s.peakMinDist);

function updateLatValues(handles)
set(handles.editLatOrX,'string',sprintf('%.02f',handles.s.lat(1,1)));
set(handles.editLatOrY,'string',sprintf('%.02f',handles.s.lat(1,2)));
set(handles.editLatUX,'string',sprintf('%.04f',handles.s.lat(2,1)));
set(handles.editLatUY,'string',sprintf('%.04f',handles.s.lat(2,2)));
set(handles.editLatVX,'string',sprintf('%.04f',handles.s.lat(3,1)));
set(handles.editLatVY,'string',sprintf('%.04f',handles.s.lat(3,2)));
% Lower
set(handles.editLatNumPlot,'string',handles.s.latNumPlot);
set(handles.editLatSiteFracU,'string',handles.s.latSiteFrac(1));
set(handles.editLatSiteFracV,'string',handles.s.latSiteFrac(2));
% Drawing
set(handles.or,'xdata',handles.s.lat(1,2));
set(handles.or,'ydata',handles.s.lat(1,1));
set(handles.u,'xdata',handles.s.lat(1,2) ...
    +[0 handles.s.lat(2,2)]*handles.s.latNumPlot);
set(handles.u,'ydata',handles.s.lat(1,1) ...
    +[0 handles.s.lat(2,1)]*handles.s.latNumPlot);
set(handles.v,'xdata',handles.s.lat(1,2) ...
    +[0 handles.s.lat(3,2)]*handles.s.latNumPlot);
set(handles.v,'ydata',handles.s.lat(1,1) ...
    +[0 handles.s.lat(3,1)]*handles.s.latNumPlot);
% strain lattice
uS = handles.s.lat(2,:);
uS = uS / norm(uS);
set(handles.textStrainLatUX,'string',sprintf('%.04f',uS(1)));
set(handles.textStrainLatUY,'string',sprintf('%.04f',uS(2)));
set(handles.textStrainLatVX,'string',sprintf('%.04f',-uS(2)));
set(handles.textStrainLatVY,'string',sprintf('%.04f',uS(1)));


function updateStrainValues(handles)
set(handles.editStrainIntMin,'string',handles.s.strainIntRange(1));
set(handles.editStrainIntMax,'string',handles.s.strainIntRange(2));
set(handles.editStrainDispMin,'string',handles.s.strainDispRange(1));
set(handles.editStrainDispMax,'string',handles.s.strainDispRange(2));
set(handles.editStrainMin,'string',handles.s.strainRange(1));
```

```
set(handles.editStrainMax,'string',handles.s.strainRange(2));
set(handles.editStrainSmooth,'string',handles.s.strainSmooth);




function handles = drawROI(handles)
if isfield(handles,'roi')
    delete(handles.roi);
end
handles.roi = patch(handles.s.p(:,2),handles.s.p(:,1),'r',...
    'linewidth',2,'edgecolor','g','facecolor','none',...
    'marker','s','markerfacecolor','w','markersize',8);


% function handles = drawLat(handles)
% if isfield(handles,'or')
%     delete(handles.or);
% end
% if isfield(handles,'u')
%     delete(handles.u);
% end
% if isfield(handles,'v')
%     delete(handles.v);
% end



% set(handles.roi,'vertices',[handles.s.p(:,2) handles.s.p(:,1)]);
% Update handles structure
% guidata(hObject, handles);




function editPeakMinDist_Callback(hObject, eventdata, handles)
% Set peak image smoothing value
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='.') == 1
    t = str2double(t);
    handles.s.peakMinDist = t;
end
updatePeakValues(handles);
% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function editPeakMinDist_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editPeakThresh_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.peakThresh = t;
end
updatePeakValues(handles);
% Update handles structure
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function editPeakThresh_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function editPeakImageSmoothing_Callback(hObject, eventdata, handles)
% Set peak image smoothing value
t = get(hObject,'String');
% if min(isstrprop(t,'digit')) == 1
if min(isstrprop(t,'digit') | t=='.') == 1
    t = min(str2double(t),16);  % Force max to 16
    handles.s.peakSmoothing = t;
end
updatePeakValues(handles);
% Update handles structure
guidata(hObject, handles);



% --- Executes during object creation, after setting all properties.
function editPeakImageSmoothing_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editPeakImageSmoothing (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


% --- Executes on button press in pushbuttonPeakMax.
function pushbuttonPeakMax_Callback(hObject, eventdata, handles)
% Use LP filtering
[qx,qy] = makeFourierCoords(handles.s.imageSize,1/norm(handles.s.lat(2,:)));
[qya,qxa] = meshgrid(qy,qx);
q2 = qxa.^2 + qya.^2;
wFilt = 1-1./sqrt(1+q2.^8/(.5^16));  % Order 8 Butterworth filter
I = handles.s.image;
I = I - mean(I(:));
I = real(ifft2(fft2(I).*wFilt));
% Scale image by SDs in local area
in = inpolygon(handles.xa,handles.ya,handles.s.p(:,1),handles.s.p(:,2));
I = I - mean(I(in));
I = I / sqrt(mean(I(in).^2));
% find peaks from image maxima
if handles.s.peakSmoothing > 0
    sm = fspecial('gaussian',2*ceil(3*handles.s.peakSmoothing),...
        handles.s.peakSmoothing);
    Ism = conv2(I,sm,'same');
else
    Ism = I;
end
p =   Ism > circshift(Ism,[-1 -1]) ...
    & Ism > circshift(Ism,[ 0 -1]) ...
    & Ism > circshift(Ism,[ 1 -1]) ...
    & Ism > circshift(Ism,[-1  0]) ...
    & Ism > circshift(Ism,[ 1  0]) ...
    & Ism > circshift(Ism,[-1  1]) ...
    & Ism > circshift(Ism,[ 0  1]) ...
    & Ism > circshift(Ism,[ 1  1]) ...
    & Ism > handles.s.peakThresh;
[xp,yp,Ip] = find(p.*Ism);
% If min distance is > 0, filter out peaks
if handles.s.peakMinDist > 0
    data = sortrows([xp yp Ip],3);
    del = false(size(data,1),1);
    r2 = handles.s.peakMinDist^2;
    for a0 = 1:(size(data,1)-1)
        d2 = (data(a0,1)-data((a0+1):end,1)).^2 ...
            + (data(a0,2)-data((a0+1):end,2)).^2;
        if min(d2) < r2
            del(a0) = true;
        end
    end
    data(del,:) = [];
    xp = data(:,1);
    yp = data(:,2);
end
% Filter via ROI
in = inpolygon(xp,yp,handles.s.p(:,1),handles.s.p(:,2));
xp(~in) = [];
yp(~in) = [];
```

```
% Output values
handles.s.pos = [xp yp zeros(length(xp),4)];
set(handles.posImage,'xdata',handles.s.pos(:,2),...
    'ydata',handles.s.pos(:,1));
set(handles.posFit,'xdata',-1,'ydata',-1);
% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in pushbuttonPeakMin.
function pushbuttonPeakMin_Callback(hObject, ~, handles)
% Use LP filtering
[qx,qy] = makeFourierCoords(handles.s.imageSize,1/norm(handles.s.lat(2,:)));
[qya,qxa] = meshgrid(qy,qx);
q2 = qxa.^2 + qya.^2;
wFilt = 1-1./sqrt(1+q2.^8/(.5^16));  % Order 8 Butterworth filter
I = handles.s.image;
I = I - mean(I(:));
I = real(ifft2(fft2(I).*wFilt));
% % Scale image by SDs in local area
in = inpolygon(handles.xa,handles.ya,handles.s.p(:,1),handles.s.p(:,2));
I = I - mean(I(in));
I = I / sqrt(mean(I(in).^2));
% find peaks from image minima
if handles.s.peakSmoothing > 0
    sm = fspecial('gaussian',2*ceil(3*handles.s.peakSmoothing),...
        handles.s.peakSmoothing);
    Ism = -conv2(I,sm,'same');
else
    Ism = -I;
end
% figure(1)
% clf
% imagesc(Ism)
% axis equal off
% colormap(violetFire)
p =   Ism > circshift(Ism,[-1 -1]) ...
    & Ism > circshift(Ism,[ 0 -1]) ...
    & Ism > circshift(Ism,[ 1 -1]) ...
    & Ism > circshift(Ism,[-1  0]) ...
    & Ism > circshift(Ism,[ 1  0]) ...
    & Ism > circshift(Ism,[-1  1]) ...
    & Ism > circshift(Ism,[ 0  1]) ...
    & Ism > circshift(Ism,[ 1  1]) ...
    & Ism > handles.s.peakThresh;
[xp,yp,Ip] = find(p.*Ism);
% If min distance is > 0, filter out peaks
if handles.s.peakMinDist > 0
    data = sortrows([xp yp Ip],3);
    del = false(size(data,1),1);
    r2 = handles.s.peakMinDist^2;
    for a0 = 1:(size(data,1)-1)
        d2 = (data(a0,1)-data((a0+1):end,1)).^2 ...
            + (data(a0,2)-data((a0+1):end,2)).^2;
        if min(d2) < r2
```

```
            del(a0) = true;
        end
    end
    data(del,:) = [];
    xp = data(:,1);
    yp = data(:,2);
end
% Filter via ROI
in = inpolygon(xp,yp,handles.s.p(:,1),handles.s.p(:,2));
xp(~in) = [];
yp(~in) = [];
% Output values
handles.s.pos = [xp yp zeros(length(xp),4)];
set(handles.posImage,'xdata',handles.s.pos(:,2),...
    'ydata',handles.s.pos(:,1));
set(handles.posFit,'xdata',-1,'ydata',-1);
% Update handles structure
guidata(hObject, handles);
```

```
function editLatOrX_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatOrX_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    xR = [max(2-[0 handles.s.lat(2,1) handles.s.lat(3,1)] ...
        *handles.s.latNumPlot) ...
        min(handles.s.imageSize(1)-1 ...
        -[0 handles.s.lat(2,1) handles.s.lat(3,1)] ...
        *handles.s.latNumPlot)];
    t = max(min(t,xR(2)),xR(1));
    handles.s.lat(1,1) = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);
```

```
function editLatOrY_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatOrY_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
```

```
    yR = [max(2-[0 handles.s.lat(2,2) handles.s.lat(3,2)] ...
        *handles.s.latNumPlot) ...
        min(handles.s.imageSize(2)-1 ...
        -[0 handles.s.lat(2,2) handles.s.lat(3,2)] ...
        *handles.s.latNumPlot)];
    t = max(min(t,yR(2)),yR(1));
    handles.s.lat(1,2) = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);


function editLatUX_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatUX_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.lat(2,1) = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);

function editLatUY_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatUY_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.lat(2,2) = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);

function editLatVX_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatVX_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.lat(3,1) = t;
end
updateLatValues(handles);
```

```
% Update handles structure
guidata(hObject, handles);

function editLatVY_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatVY_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.lat(3,2) = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);


function editLatNumPlot_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatNumPlot_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit')) == 1
    t = max(round(str2double(t)),1);
    handles.s.latNumPlot = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);


function editLatSiteFracU_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editLatSiteFracU_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit')) == 1
    t = max(round(str2double(t)),1);
    handles.s.latSiteFrac(1) = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);


function editLatSiteFracV_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end
function editLatSiteFracV_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit')) == 1
    t = max(round(str2double(t)),1);
    handles.s.latSiteFrac(2) = t;
end
updateLatValues(handles);
% Update handles structure
guidata(hObject, handles);


function pushbuttonLatRefine_Callback(hObject, eventdata, handles)
% Need at least 3 points to fit linear 2D lattice
if size(handles.s.pos,1) >= 3
    f = handles.s.latSiteFrac;
    or = handles.s.lat(1,:);
    u = handles.s.lat(2,:);
    v = handles.s.lat(3,:);

    dMax = sqrt(max((handles.s.pos(:,1)-or(1)).^2 + (handles.s.pos(:,2)-
or(2)).^2));
    dMin = sqrt(min((handles.s.pos(:,1)-or(1)).^2 ...
        + (handles.s.pos(:,2)-or(2)).^2)) + 5*mean([norm(u) norm(v)]);

    for r = dMin:20:dMax
        sub = (handles.s.pos(:,1) - or(1)).^2 ...
            + (handles.s.pos(:,2) - or(2)).^2 < r^2;

        % Compute a,b values
        a = round(f(1)*((handles.s.pos(sub,2)-or(2))*v(1) ...
            - (handles.s.pos(sub,1)-or(1))*v(2))/(v(1)*u(2)-v(2)*u(1)))/f(1);
        b = round(f(2)*((handles.s.pos(sub,2)-or(2))*u(1) ...
            - (handles.s.pos(sub,1)-or(1))*u(2))/(v(2)*u(1)-v(1)*u(2)))/f(2);
        % Refine lattice
        A = [ones(sum(sub),1) a b];
        xbeta = A \ handles.s.pos(sub,1);
        ybeta = A \ handles.s.pos(sub,2);
        or = [xbeta(1) ybeta(1)];
        u = [xbeta(2) ybeta(2)];
        v = [xbeta(3) ybeta(3)];
    end

    % Compute a,b values
    a = round(f(1)*((handles.s.pos(:,2)-or(2))*v(1) ...
        - (handles.s.pos(:,1)-or(1))*v(2))/(v(1)*u(2)-v(2)*u(1)))/f(1);
    b = round(f(2)*((handles.s.pos(:,2)-or(2))*u(1) ...
        - (handles.s.pos(:,1)-or(1))*u(2))/(v(2)*u(1)-v(1)*u(2)))/f(2);
    % Refine lattice
    A = [ones(size(handles.s.pos,1),1) a b];
    xbeta = A \ handles.s.pos(:,1);
    ybeta = A \ handles.s.pos(:,2);
    or = [xbeta(1) ybeta(1)];
```

```
    u = [xbeta(2) ybeta(2)];
    v = [xbeta(3) ybeta(3)];

    handles.s.lat(1,:) = or;
    handles.s.lat(2,:) = u;
    handles.s.lat(3,:) = v;
    % generated fitted positions
    xf = or(1) + a*u(1) + b*v(1);
    yf = or(2) + a*u(2) + b*v(2);
    % Export and plot fitted positions
    handles.s.pos(:,3:6) = [a b xf yf];
    handles.s.pos = sortrows(sortrows(handles.s.pos,4),3);
    set(handles.posFit,'xdata',yf,'ydata',xf);
    updateLatValues(handles);
    % Update handles structure
    guidata(hObject, handles);
end




% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
if isequal(get(hObject,'waitstatus'),'waiting')
    uiresume(hObject);
else
    delete(hObject);
end
% handles.output = handles.s;
% delete(hObject);




% ******** GUI CONTROL FUNCTIONS **********
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% Get cursor position
xyMain = get(handles.axesMain,'Currentpoint');
xy = [xyMain(1,2) xyMain(1,1)];
% Check to see if xy is on top of a current roi point
minDist = (.015*handles.s.imageSize(1));
[val,ind] = min((handles.s.p(:,1)-xy(1)).^2 ...
    + (handles.s.p(:,2)-xy(2)).^2);
% Determine whether click is single or double click
if toc < 1/3
    % double click detected
    if val < minDist^2
        % point deletion if possible (more than 3 points)
        if size(handles.s.p,1) > 3
            handles.s.p(ind,:) = [];
        end
```

```
        handles = drawROI(handles);
    else
        % point creation
        testRTuv = zeros(size(handles.s.p,1),6);
        for a0 = 1:size(handles.s.p,1)
            u = handles.s.p(mod(a0,size(handles.s.p,1))+1,:) ...
                - handles.s.p(a0,:);
            v = [u(2) u(1)] / norm(u);
            % Solve click location as superpositions of u and v
            T = ((xy(2)-handles.s.p(a0,2))*v(1) ...
                - (xy(1)-handles.s.p(a0,1))*v(2))/(v(1)*u(2)-v(2)*u(1));
            R = ((xy(2)-handles.s.p(a0,2))*u(1) ...
                - (xy(1)-handles.s.p(a0,1))*u(2))/(v(2)*u(1)-v(1)*u(2));
            testRTuv(a0,:) = [T R u v];
        end
        [minR,ind] = min(abs(testRTuv(:,2)));
        if minR < minDist && testRTuv(ind,1) > 0 && testRTuv(ind,1) < 1
            % Generate new point
            pNew = handles.s.p(ind,:) ...
                + testRTuv(ind,1)*testRTuv(ind,3:4);
            if ind == size(handles.s.p,1)
                pp = [handles.s.p; pNew];
            else
                pp = [handles.s.p(1:ind,:);
                    pNew; handles.s.p((ind+1):end,:)];
            end
            handles.s.p = pp;
            handles = drawROI(handles);
        end
    end
else
    % single click case --> move points or lattice vectors
    % Check distance to  ROI points
    if val < minDist^2
        handles.flagROI = ind;
    else
        % else check for distance to lattice points
        lat = handles.s.lat;
        lat(2,:) = lat(1,:) + lat(2,:)*handles.s.latNumPlot;
        lat(3,:) = lat(1,:) + lat(3,:)*handles.s.latNumPlot;
        [val,ind] = min((lat(:,1)-xy(1)).^2 ...
            + (lat(:,2)-xy(2)).^2);
        if val < minDist^2
            handles.flagLat = ind;
        end
    end

end
tic
% Update handles structure
guidata(hObject, handles);



% --- Executes on mouse motion over figure - except title and menu.
```

```
function figure1_WindowButtonMotionFcn(hObject, eventdata, handles)
if handles.flagROI > 0
    xyMain = get(handles.axesMain,'Currentpoint');
    xy = [xyMain(1,2) xyMain(1,1)];
    if xy(1) < 1
        xy(1) = 1;
    end
    if xy(2) < 1
        xy(2) = 1;
    end
    if xy(1) > handles.s.imageSize(1)
        xy(1) = handles.s.imageSize(1);
    end
    if xy(2) > handles.s.imageSize(2)
        xy(2) = handles.s.imageSize(2);
    end
    handles.s.p(handles.flagROI,:) = xy;
    handles = drawROI(handles);
end
if handles.flagLat > 0
    xyMain = get(handles.axesMain,'Currentpoint');
    xy = [xyMain(1,2) xyMain(1,1)];
    if handles.flagLat == 1
        xR = [max(2-[0 handles.s.lat(2,1) handles.s.lat(3,1)] ...
            *handles.s.latNumPlot) ...
            min(handles.s.imageSize(1)-1 ...
            -[0 handles.s.lat(2,1) handles.s.lat(3,1)] ...
            *handles.s.latNumPlot)];
        yR = [max(2-[0 handles.s.lat(2,2) handles.s.lat(3,2)] ...
            *handles.s.latNumPlot) ...
            min(handles.s.imageSize(2)-1 ...
            -[0 handles.s.lat(2,2) handles.s.lat(3,2)] ...
            *handles.s.latNumPlot)];
    else
        xR = [1 handles.s.imageSize(1)];
        yR = [1 handles.s.imageSize(2)];
    end
    if xy(1) < xR(1)
        xy(1) = xR(1);
    end
    if xy(2) < yR(1)
        xy(2) = yR(1);
    end
    if xy(1) > xR(2)
        xy(1) = xR(2);
    end
    if xy(2) > yR(2)
        xy(2) = yR(2);
    end
    if handles.flagLat == 1
        handles.s.lat(1,:) = xy;

    elseif handles.flagLat == 2
        handles.s.lat(2,:) = (xy - handles.s.lat(1,:)) ...
            / handles.s.latNumPlot;
```

```
    else
        handles.s.lat(3,:) = (xy - handles.s.lat(1,:)) ...
            / handles.s.latNumPlot;
    end
    updateLatValues(handles);
end
% Update handles structure
guidata(hObject, handles);



% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonUpFcn(hObject, eventdata, handles)
if handles.flagROI > 0
    handles.flagROI = 0;
end
if handles.flagLat > 0
    handles.flagLat = 0;
end
% Update handles structure
guidata(hObject, handles);



% --- Executes on button press in pushbuttonImageZoom.
function pushbuttonImageZoom_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonImageZoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of pushbuttonImageZoom
if get(hObject,'Value') == true
    set(handles.figure1,'currentaxes',handles.axesMain);
    zoom on
else
    set(handles.figure1,'currentaxes',handles.axesMain);
    zoom off
end

function pushbuttonImageReset_Callback(hObject, eventdata, handles)
set(handles.figure1,'currentaxes',handles.axesMain);
ylim([1 handles.s.imageSize(1)])
xlim([1 handles.s.imageSize(2)])


function pushbuttonLatDynamic_Callback(hObject, eventdata, handles)


if size(handles.s.pos,1) >= 3
    f = handles.s.latSiteFrac;
    or = handles.s.lat(1,:);
    u = handles.s.lat(2,:);
    v = handles.s.lat(3,:);
```

```
pos = handles.s.pos;

radiusThresh2 = (min([norm(u) norm(v)]./f)*.25) ^ 2;
abVals = [ones(size(pos,1),2)*-1e8 zeros(size(pos,1),1)];
% Get origin
[~,ind] = min((pos(:,1)-or(1)).^2 + (pos(:,2)-or(2)).^2);
abVals(ind,:) = [0 0 1];
% List of points not yet assigned - [ind,x,y]
xyList = [(1:size(pos,1))' pos];
xyList(ind,:) = [];
% List of trial points - [a,b,x,y]
dab = [-1 0;0 -1;1 0;0 1];
dxy = dab(:,1)*u + dab(:,2)*v;
xyTrial = [dab repmat(pos(ind,1:2),[4 1])+dxy];

% While there are still trial points, keep checking!
while size(xyTrial,1) > 0

    xyTrialNew = [];

    for a0 = 1:size(xyTrial,1)
        [val,ind] = min((xyList(:,2)-xyTrial(a0,3)).^2 ...
            + (xyList(:,3)-xyTrial(a0,4)).^2);

        if val < radiusThresh2
            % Write out (a,b) value
            indSelect = xyList(ind,1);
            abVals(indSelect,:) = [xyTrial(a0,1:2) 1];

            % Add trial points
            if isempty(xyTrialNew)
                xyTrialNew = [repmat(xyTrial(a0,1:2),[4 1]) + dab ...
                    repmat(xyList(ind,2:3),[4 1]) + dxy];
            else
                xyTrialNew = [xyTrialNew;
                    repmat(xyTrial(a0,1:2),[4 1]) + dab ...
                    repmat(xyList(ind,2:3),[4 1]) + dxy];
            end

            % Delete point from list
            xyList(ind,:) = [];
        end
    end

    % Remove points already found
    if ~isempty(xyTrialNew)
        [~,~,inds] = intersect(abVals(:,1:2),xyTrialNew(:,1:2),'rows');
        xyTrialNew(inds,:) = [];
    end

    % Merge trial points
    if ~isempty(xyTrialNew)
        [abUnique,~,inds] = unique(xyTrialNew(:,1:2),'rows');
        xyTrialNew2 = zeros(size(abUnique,1),4);
```

```
            for a0 = 1:size(abUnique,1)
                xyTrialNew2(a0,:) = [abUnique(a0,:) ...
                    mean(xyTrialNew(inds==a0,3:4),1)];
            end
        else
            xyTrialNew2 = [];
        end

        % Remove any a,b values that have already been detected from
        % xyTrial - protection against double-identities!


        xyTrial = xyTrialNew2;
    end

    % Delete positions not found
    keep = abVals(:,3) == 1;
    handles.s.pos = handles.s.pos(keep,:);
    a = abVals(keep,1);
    b = abVals(keep,2);

    % Compute a,b values
    %     a = round(f(1)*((handles.s.pos(:,2)-or(2))*v(1) ...
    %         - (handles.s.pos(:,1)-or(1))*v(2))/(v(1)*u(2)-v(2)*u(1)))/f(1);
    %     b = round(f(2)*((handles.s.pos(:,2)-or(2))*u(1) ...
    %         - (handles.s.pos(:,1)-or(1))*u(2))/(v(2)*u(1)-v(1)*u(2)))/f(2);
    % Refine lattice
    A = [ones(size(handles.s.pos,1),1) a b];
    xbeta = A \ handles.s.pos(:,1);
    ybeta = A \ handles.s.pos(:,2);
    or = [xbeta(1) ybeta(1)];
    u = [xbeta(2) ybeta(2)];
    v = [xbeta(3) ybeta(3)];

    % generated fitted positions
    xf = or(1) + a*u(1) + b*v(1);
    yf = or(2) + a*u(2) + b*v(2);
    % Export and plot fitted positions
    handles.s.pos(:,3:6) = [a b xf yf];
    set(handles.posFit,'xdata',yf,'ydata',xf);
    updateLatValues(handles);

    % Update lattice
    handles.s.lat(1,:) = or;
    handles.s.lat(2,:) = u;
    handles.s.lat(3,:) = v;

    % Update handles structure
    guidata(hObject, handles);
end
```

```
function editStrainIntMin_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editStrainIntMin_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.strainIntRange(1) = t;
end
updateStrainValues(handles);
% Update handles structure
guidata(hObject, handles);


function editStrainIntMax_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editStrainIntMax_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.strainIntRange(2) = t;
end
updateStrainValues(handles);
% Update handles structure
guidata(hObject, handles);


function editStrainDispMin_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editStrainDispMin_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.strainDispRange(1) = t;
end
updateStrainValues(handles);
% Update handles structure
guidata(hObject, handles);


function editStrainDispMax_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editStrainDispMax_Callback(hObject, eventdata, handles)
```

```
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.strainDispRange(2) = t;
end
updateStrainValues(handles);
% Update handles structure
guidata(hObject, handles);


function editStrainMin_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editStrainMin_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.strainRange(1) = t;
end
updateStrainValues(handles);
% Update handles structure
guidata(hObject, handles);


function editStrainMax_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editStrainMax_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='-' | t=='.') == 1
    t = str2double(t);
    handles.s.strainRange(2) = t;
end
updateStrainValues(handles);
% Update handles structure
guidata(hObject, handles);


function editStrainSmooth_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editStrainSmooth_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='.') == 1
    t = str2double(t);
    handles.s.strainSmooth = t;
end
updateStrainValues(handles);
```

```
% Update handles structure
guidata(hObject, handles);



function pushbuttonStrainCompute_Callback(hObject, eventdata, handles)
% Compute displacement and then strain maps
% Strain coordinate system vectors
uS = handles.s.lat(2,:);
uS = uS / norm(uS);
vS = [-uS(2) uS(1)];
%  Diplacement values
dx = handles.s.pos(:,1) - handles.s.pos(:,5);
dy = handles.s.pos(:,2) - handles.s.pos(:,6);
du = dx*uS(1) + dy*uS(2);
dv = dx*vS(1) + dy*vS(2);
%  Diplacement maps
inds = sub2ind(handles.s.imageSize,...
    round(handles.s.pos(:,1)),round(handles.s.pos(:,2)));
handles.s.strainDispU = zeros(handles.s.imageSize);
handles.s.strainDispV = zeros(handles.s.imageSize);
count = zeros(handles.s.imageSize);
handles.s.strainDispU(inds) = du;
handles.s.strainDispV(inds) = dv;
count(inds) = 1;
% apply smoothing
sigma = handles.s.strainSmooth * norm(handles.s.lat(2,:));
sm = fspecial('gaussian',2*ceil(3*sigma)+1,sigma);
handles.s.strainDispU = convolve2(handles.s.strainDispU,sm,'wrap');
handles.s.strainDispV = convolve2(handles.s.strainDispV,sm,'wrap');
count = convolve2(count,sm,'wrap');
sub = count > 0;
handles.s.strainDispU(sub) = handles.s.strainDispU(sub)./count(sub);
handles.s.strainDispV(sub) = handles.s.strainDispV(sub)./count(sub);
% Compute strain field
dUdx = (circshift(handles.s.strainDispU,[1 0]) ...
    - circshift(handles.s.strainDispU,[-1 0])) / 2;
dUdy = (circshift(handles.s.strainDispU,[0 1]) ...
    - circshift(handles.s.strainDispU,[0 -1])) / 2;
dVdx = (circshift(handles.s.strainDispV,[1 0]) ...
    - circshift(handles.s.strainDispV,[-1 0])) / 2;
dVdy = (circshift(handles.s.strainDispV,[0 1]) ...
    - circshift(handles.s.strainDispV,[0 -1])) / 2;
handles.s.strainEuu = dUdx*uS(1) + dUdy*uS(2);
handles.s.strainEvv = dVdx*vS(1) + dVdy*vS(2);
handles.s.strainEuv = (dUdx*vS(1) + dUdy*vS(2) ...
    + dVdx*uS(1) + dVdy*uS(2))/2;
% Scale strain displacement maps to be in unit cell units
handles.s.strainDispU = handles.s.strainDispU / norm(handles.s.lat(2,:));
handles.s.strainDispV = handles.s.strainDispV / norm(handles.s.lat(2,:));
% Make intensity map for scaling
handles.s.strainIval = (handles.s.image - handles.s.strainIntRange(1)) ...
    / (handles.s.strainIntRange(2) - handles.s.strainIntRange(1));
handles.s.strainIval(handles.s.strainIval<0) = 0;
handles.s.strainIval(handles.s.strainIval>1) = 1;
```

```
% Make mask for colouring
m = inpolygon(handles.xa,handles.ya,handles.s.p(:,1),handles.s.p(:,2));
handles.s.mask = bwdist(~m) / sigma;
handles.s.mask(handles.s.mask<0) = 0;
handles.s.mask(handles.s.mask>1) = 1;
handles.s.mask = 1 - handles.s.mask;
% Enable plotting buttons
set(handles.pushbuttonStrainExport,'enable','on');
set(handles.pushbuttonStrainU,'enable','on');
set(handles.pushbuttonStrainV,'enable','on');
set(handles.pushbuttonStrainRadial,'enable','on');
set(handles.pushbuttonStrainTheta,'enable','on');
set(handles.pushbuttonStrainEuu,'enable','on');
set(handles.pushbuttonStrainEvv,'enable','on');
set(handles.pushbuttonStrainEuv,'enable','on');
set(handles.pushbuttonQuiverPlot,'enable','on');
set(handles.pushbuttonQuiverColour,'enable','on');
% Update handles structure
guidata(hObject, handles);



function pushbuttonStrainExport_Callback(hObject, eventdata, handles)
% Create and save images
fbase = inputdlg('File name stem for strain images?',...
    'Export strain map figures');
% Raw image
fname = [fbase{1} '_intensity_' ...
    num2str(handles.s.strainIntRange(1)) '_to_' ...
    num2str(handles.s.strainIntRange(2)) '_SD.png'];
imwrite(uint8(255*handles.s.strainIval),fname,'png');
% U disp
Imap = (handles.s.strainDispU - handles.s.strainDispRange(1)) ...
    / (handles.s.strainDispRange(2) - handles.s.strainDispRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
fname = [fbase{1} '_u_disp_' ...
    num2str(handles.s.strainDispRange(1)) '_to_' ...
    num2str(handles.s.strainDispRange(2)) '_UCs.png'];
imwrite(uint8(255*Istrain),fname,'png');
% V disp
Imap = (handles.s.strainDispV - handles.s.strainDispRange(1)) ...
    / (handles.s.strainDispRange(2) - handles.s.strainDispRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
fname = [fbase{1} '_v_disp_' ...
    num2str(handles.s.strainDispRange(1)) '_to_' ...
    num2str(handles.s.strainDispRange(2)) '_UCs.png'];
imwrite(uint8(255*Istrain),fname,'png');
% Radial disp
strainDispRadial = sqrt(handles.s.strainDispU.^2 ...
```

```
    +handles.s.strainDispV.^2);
Imap = (strainDispRadial) ...
    / (handles.s.strainDispRange(2));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
fname = [fbase{1} '_radial_disp_' ...
    '0_to_' num2str(handles.s.strainDispRange(2)) '_UCs.png'];
imwrite(uint8(255*Istrain),fname,'png');
% theta disp
strainTheta = atan2(handles.s.strainDispV,handles.s.strainDispU);
Imap = mod((strainTheta + 0)/(2*pi),1);
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
Istrain = zeros(size(Imap,1),size(Imap,2),3);
Istrain(:,:,1) = (1-Imap)*(2/3);
Istrain(:,:,2) = 1-handles.s.mask;
Istrain(:,:,3) = handles.s.strainIval;
Istrain = hsv2rgb(Istrain);
Istrain(Istrain<0) = 0;
Istrain(Istrain>1) = 1;
fname = [fbase{1} '_theta_disp_2_Pi_rad.png'];
imwrite(uint8(255*Istrain),fname,'png');
% Color radial + angle map
strainTheta = atan2(handles.s.strainDispV,handles.s.strainDispU);
Imap = mod((strainTheta + 0)/(2*pi),1);
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
strainDispRadial = sqrt(handles.s.strainDispU.^2 ...
    +handles.s.strainDispV.^2);
Imap2 = (strainDispRadial) ...
    / (handles.s.strainDispRange(2));
Imap2(Imap<0) = 0;
Imap2(Imap2>1) = 1;
Istrain = zeros(size(Imap,1),size(Imap,2),3);
Istrain(:,:,1) = Imap;
Istrain(:,:,2) = (1-handles.s.mask);
Istrain(:,:,3) = Imap2.*(1-handles.s.mask);
Istrain = hsv2rgb(Istrain);
Istrain(Istrain<0) = 0;
Istrain(Istrain>1) = 1;
fname = [fbase{1} '_polar_disp_' ...
    '0_to_' num2str(handles.s.strainDispRange(2)) ...
    '_UCs.png'];imwrite(uint8(255*Istrain),fname,'png');
imwrite(uint8(255*Istrain),fname,'png');


% Euu
Imap = (handles.s.strainEuu*100 - handles.s.strainRange(1)) ...
    / (handles.s.strainRange(2) - handles.s.strainRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
```

```
fname = [fbase{1} '_e_uu_' ...
    num2str(handles.s.strainRange(1)) '_to_' ...
    num2str(handles.s.strainRange(2)) '_percent.png'];
imwrite(uint8(255*Istrain),fname,'png');
% Evv
Imap = (handles.s.strainEvv*100 - handles.s.strainRange(1)) ...
    / (handles.s.strainRange(2) - handles.s.strainRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
fname = [fbase{1} '_e_vv_' ...
    num2str(handles.s.strainRange(1)) '_to_' ...
    num2str(handles.s.strainRange(2)) '_percent.png'];
imwrite(uint8(255*Istrain),fname,'png');
% Euv
Imap = (handles.s.strainEuv*100 - handles.s.strainRange(1)) ...
    / (handles.s.strainRange(2) - handles.s.strainRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
fname = [fbase{1} '_e_uv_' ...
    num2str(handles.s.strainRange(1)) '_to_' ...
    num2str(handles.s.strainRange(2)) '_percent.png'];
imwrite(uint8(255*Istrain),fname,'png');
% Dilation
Idilate = ((1+handles.s.strainEuu).*(1+handles.s.strainEvv))-1;
Imap = (Idilate*100 - handles.s.strainRange(1)/2) ...
    / (handles.s.strainRange(2)/2 - handles.s.strainRange(1)/2);
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
fname = [fbase{1} '_scaling_' ...
    num2str(handles.s.strainRange(1)/2) '_to_' ...
    num2str(handles.s.strainRange(2)/2) '_percent.png'];
imwrite(uint8(255*Istrain),fname,'png');



function pushbuttonStrainU_Callback(hObject, eventdata, handles)
Imap = (handles.s.strainDispU - handles.s.strainDispRange(1)) ...
    / (handles.s.strainDispRange(2) - handles.s.strainDispRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
plotImage(Istrain)


function pushbuttonStrainV_Callback(hObject, eventdata, handles)
Imap = (handles.s.strainDispV - handles.s.strainDispRange(1)) ...
    / (handles.s.strainDispRange(2) - handles.s.strainDispRange(1));
Imap(Imap<0) = 0;
```

```
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
plotImage(Istrain)


function pushbuttonStrainRadial_Callback(hObject, eventdata, handles)
strainRadial = sqrt(handles.s.strainDispU.^2 ...
    + handles.s.strainDispV.^2);
Imap = (strainRadial) ...
    / (handles.s.strainDispRange(2));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
plotImage(Istrain)
function pushbuttonStrainTheta_Callback(hObject, eventdata, handles)
strainTheta = atan2(handles.s.strainDispV,handles.s.strainDispU);
Imap = mod((strainTheta + 0)/(2*pi),1);
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
% Make hue map for theta
Istrain = zeros(size(Imap,1),size(Imap,2),3);
Istrain(:,:,1) = (1-Imap)*(2/3);
Istrain(:,:,2) = 1-handles.s.mask;
Istrain(:,:,3) = handles.s.strainIval;
Istrain = hsv2rgb(Istrain);
Istrain(Istrain<0) = 0;
Istrain(Istrain>1) = 1;
plotImage(Istrain)


function pushbuttonStrainEuu_Callback(hObject, eventdata, handles)
Imap = (handles.s.strainEuu*100 - handles.s.strainRange(1)) ...
    / (handles.s.strainRange(2) - handles.s.strainRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
plotImage(Istrain)

function pushbuttonStrainEvv_Callback(hObject, eventdata, handles)
Imap = (handles.s.strainEvv*100 - handles.s.strainRange(1)) ...
    / (handles.s.strainRange(2) - handles.s.strainRange(1));
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
plotImage(Istrain)

function pushbuttonStrainEuv_Callback(hObject, eventdata, handles)
Imap = (handles.s.strainEuv*100 - handles.s.strainRange(1)) ...
    / (handles.s.strainRange(2) - handles.s.strainRange(1));
```

```
Imap(Imap<0) = 0;
Imap(Imap>1) = 1;
% Imap = round(255*Imap) + 1;
Istrain = makeStrainImage(handles,Imap);
plotImage(Istrain)


function Istrain = makeStrainImage(handles,Imap)
Istrain = zeros(handles.s.imageSize(1),handles.s.imageSize(2),3);
% Istrain = ind2rgb(Imap,flipud(jet(256)));
Istrain(:,:,1) = (1-Imap)*(2/3);
Istrain(:,:,2) = 1;
Istrain(:,:,3) = handles.s.strainIval;
% Istrain(Istrain<0) = 0;
% Istrain(Istrain>1) = 1;
Istrain = hsv2rgb(Istrain);
% Istrain = rgb2hsv(Istrain);
Istrain = repmat(handles.s.mask.*handles.s.strainIval,[1 1 3]) ...
    + repmat(1-handles.s.mask,[1 1 3]) .* Istrain;

% Istrain = Imap;



function plotImage(Istrain)
figure(11)
clf
imagesc(Istrain)
axis equal off
% colormap(jet(256))
set(gca,'position',[0 0 1 1])


function pushbuttonImagePan_Callback(hObject, eventdata, handles)
if get(hObject,'Value') == true
    set(handles.figure1,'currentaxes',handles.axesMain);
    pan on
else
    set(handles.figure1,'currentaxes',handles.axesMain);
    pan off
end


% --- Executes on button press in pushbuttonQuiverPlot.
function pushbuttonQuiverPlot_Callback(hObject, eventdata, handles)
% Make a quicver plot
scale = str2double(inputdlg('Scaling of vectors',...
    '[Set to "1" for unscaled]'));
figure(11)
clf
imagesc(handles.s.strainIval)
hold on
```

```
quiver(handles.s.pos(:,6),handles.s.pos(:,5),...
    (handles.s.pos(:,2)-handles.s.pos(:,6))*scale,...
    (handles.s.pos(:,1)-handles.s.pos(:,5))*scale,...
    'autoscale','off','color','r','linewidth',1)
hold off
axis equal off
colormap(gray(256))
set(gca,'position',[0 0 1 1])


function pushbuttonQuiverColour_Callback(hObject, eventdata, handles)
% Make a color quicver plot
scale = str2double(inputdlg('Scaling of vectors',...
    '[Set to "1" for unscaled]'));
figure(11)
clf
imagesc(handles.s.strainIval)
hold on
for a0 = 1:size(handles.s.pos,1)
    xy = handles.s.pos(a0,5:6);
    dxy = handles.s.pos(a0,1:2)-handles.s.pos(a0,5:6);
    c = hsv2rgb(reshape(...
        [mod(atan2(-dxy(2),-dxy(1))/(2*pi),1) 1 1],[1 1 3]));
    line([0 dxy(2)]*scale+xy(2),[0 dxy(1)]*scale+xy(1),...
        'linewidth',1,'color',c)
end
% quiver(handles.s.pos(:,6),handles.s.pos(:,5),...
%     (handles.s.pos(:,2)-handles.s.pos(:,6))*scale,...
%     (handles.s.pos(:,1)-handles.s.pos(:,5))*scale,...
%     'autoscale','off','color','r','linewidth',1)
hold off
axis equal off
colormap(gray(256))
set(gcf,'renderer','zbuffer')
set(gca,'position',[0 0 1 1])


function [qx,qy] = makeFourierCoords(N,pSize)
% This function generates image Fourier coordinates (2D)
if length(pSize) == 1
    L = N(1:2)*pSize;
elseif length(pSize) == 2
    L = N(1:2).*pSize;
end
if mod(N(1),2) == 0
    qx = circshift(((-N(1)/2):(N(1)/2-1))/L(1),[1 -N(1)/2]);
else
    qx = circshift(((-N(1)/2+.5):(N(1)/2-.5))/L(1),[1 -N(1)/2+.5]);
end
if mod(N(2),2) == 0
    qy = circshift(((-N(2)/2):(N(2)/2-1))/L(2),[1 -N(2)/2]);
else
    qy = circshift(((-N(2)/2+.5):(N(2)/2-.5))/L(2),[1 -N(2)/2+.5]);
end
```

```
function editUCu_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editUCu_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit')) == 1
    t = max(round(str2double(t)),2);
    handles.s.UCsize(1) = t;
end
updateUCValues(handles);
% Update handles structure
guidata(hObject, handles);


function editUCv_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editUCv_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit')) == 1
    t = max(round(str2double(t)),2);
    handles.s.UCsize(2) = t;
end
updateUCValues(handles);
% Update handles structure
guidata(hObject, handles);


function editUCrepu_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editUCrepu_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit')) == 1
    t = max(round(str2double(t)),1);
    handles.s.UCrep(1) = t;
end
updateUCValues(handles);
drawMeanUC(handles);
% Update handles structure
guidata(hObject, handles);


function editUCrepv_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end
function editUCrepv_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit')) == 1
    t = max(round(str2double(t)),1);
    handles.s.UCrep(2) = t;
end
updateUCValues(handles);
drawMeanUC(handles);
% Update handles structure
guidata(hObject, handles);



function editUCsigma_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function editUCsigma_Callback(hObject, eventdata, handles)
t = get(hObject,'String');
if min(isstrprop(t,'digit') | t=='.') == 1
    t = max(str2double(t),0);
    handles.s.UCsigma = t;
end
updateUCValues(handles);
[handles] = computeMeanUC(handles);
drawMeanUC(handles);
% Update handles structure
guidata(hObject, handles);



function pushbuttonUCsize_Callback(hObject, eventdata, handles)
% Query user about length of longer direction
% or = handles.s.lat(1,:);
u = handles.s.lat(2,:);
v = handles.s.lat(3,:);
L = inputdlg(' Length of longest UC vector in pixels? ');
if min(isstrprop(L{1},'digit')) == 1
    L = str2double(L{1});
    if norm(u) > norm(v)
        leng = round([L L*norm(v)/norm(u)]);
        leng = max(leng,[1 1]);
    else
        leng = round([L*norm(u)/norm(v) L]);
        leng = max(leng,[1 1]);
    end
    handles.s.UCsize = leng;
    %     handles = computeMeanUCsig(handles);
    handles = computeMeanUC(handles);
    drawMeanUC(handles);
    updateUCValues(handles);
```

```
    % Update handles structure
    guidata(hObject, handles);
end


function pushbuttonUCcompute_Callback(hObject, eventdata, handles)
% compute the mean UC!
% handles = computeMeanUCsig(handles);
handles = computeMeanUC(handles);
drawMeanUC(handles);
% Update handles structure
guidata(hObject, handles);



% This piece may be reused
% function [handles] = computeMeanUCsig(handles,I)
% % ROI subset
% or = handles.s.lat(1,:);
% u = handles.s.lat(2,:);
% v = handles.s.lat(3,:);
% in = inpolygon(handles.xa,handles.ya,handles.s.p(:,1),handles.s.p(:,2));
% aInd = mod(round(handles.s.UCsize(1)*((handles.ya(in)-or(2))*v(1) ...
%     - (handles.xa(in)-or(1))*v(2))/(v(1)*u(2)-v(2)*u(1)))-1,...
%     handles.s.UCsize(1))+1;
% bInd = mod(round(handles.s.UCsize(2)*((handles.ya(in)-or(2))*u(1) ...
%     - (handles.xa(in)-or(1))*u(2))/(v(2)*u(1)-v(1)*u(2)))-1,...
%     handles.s.UCsize(2))+1;
% if nargin == 1
%     handles.s.UCsig = accumarray([aInd bInd],...
%         handles.s.image(in),handles.s.UCsize);
% else
%     handles.s.UCsig = accumarray([aInd bInd],...
%         I(in),handles.s.UCsize);
% end
% handles.s.UCcount = accumarray([aInd bInd],...
%     sum(in(:)),handles.s.UCsize);
% sub = handles.s.UCcount > 0;
% handles.s.UCmean = handles.s.UCsig ./ handles.s.UCcount



% This piece may be reused
function [handles] = computeMeanUC(handles,I)
% compute signal of unit cell
% ROI subset
or = handles.s.lat(1,:);
u = handles.s.lat(2,:);
v = handles.s.lat(3,:);
in = inpolygon(handles.xa,handles.ya,handles.s.p(:,1),handles.s.p(:,2));
aInd = mod(round(handles.s.UCsize(1)*((handles.ya(in)-or(2))*v(1) ...
    - (handles.xa(in)-or(1))*v(2))/(v(1)*u(2)-v(2)*u(1)))-1,...
    handles.s.UCsize(1))+1;
bInd = mod(round(handles.s.UCsize(2)*((handles.ya(in)-or(2))*u(1) ...
```

```
    - (handles.xa(in)-or(1))*u(2))/(v(2)*u(1)-v(1)*u(2)))-1,...
    handles.s.UCsize(2))+1;
if nargin == 1
    handles.s.UCsig = accumarray([aInd bInd],...
        handles.s.image(in),handles.s.UCsize);
else
    handles.s.UCsig = accumarray([aInd bInd],...
        I(in),handles.s.UCsize);
end
handles.s.UCcount = accumarray([aInd bInd],...
    ones(length(aInd),1),handles.s.UCsize);
% Compute mean UC as UCsignal / UCcount
sm = fspecial('gaussian',2*ceil(3*handles.s.UCsigma)+1,handles.s.UCsigma);
sig = convolve2(handles.s.UCsig,sm,'wrap');
count = convolve2(handles.s.UCcount,sm,'wrap');
sub = count > 0;
handles.s.UCmean = zeros(handles.s.UCsize);
handles.s.UCmean(sub) = sig(sub) ./ count(sub);
% Compute RMS deviation of UC
imageROImeanUC = handles.s.UCsig(sub2ind(handles.s.UCsize,aInd,bInd)) ...
    ./ handles.s.UCcount(sub2ind(handles.s.UCsize,aInd,bInd));
% UCmeanSDsig = accumarray([aInd bInd],...
%     imageROImeanUC,handles.s.UCsize);
if nargin == 1
    UCmeanSDsig = accumarray([aInd bInd],...
        (handles.s.image(in)-imageROImeanUC).^2,handles.s.UCsize);
else
    UCmeanSDsig = accumarray([aInd bInd],...
        (I(in)-imageROImeanUC).^2,handles.s.UCsize);
end
UCmeanSDsig = convolve2(UCmeanSDsig,sm,'wrap');
UCmeanSDsig(sub) = UCmeanSDsig(sub) ./ count(sub);
handles.s.UCmeanSD = sqrt(UCmeanSDsig);
% handles.s.temp = UCmeanSDsig;
% temp = zeros(handles.s.imageSize);
% temp(sub2ind(handles.s.imageSize,handles.xa(in),handles.ya(in))) =
imageROImeanUC;
% handles.s.temp = temp;
% SD = accumarray([aInd bInd],...
%     sum(in(:)),handles.s.UCsize);



function updateUCValues(handles)
set(handles.editUCu,'string',handles.s.UCsize(1));
set(handles.editUCv,'string',handles.s.UCsize(2));
set(handles.editUCrepu,'string',handles.s.UCrep(1));
set(handles.editUCrepv,'string',handles.s.UCrep(2));
set(handles.editUCsigma,'string',handles.s.UCsigma);


function pushbuttonUCcomputeLP_Callback(hObject, eventdata, handles)
% Use LP filtering
[qx,qy] = makeFourierCoords(handles.s.imageSize,1/norm(handles.s.lat(2,:)));
[qya,qxa] = meshgrid(qy,qx);
```

```
q2 = qxa.^2 + qya.^2;
wFilt = 1-1./sqrt(1+q2.^8/(.5^16));  % Order 8 Butterworth filter
I = handles.s.image;
I = I - mean(I(:));
I = real(ifft2(fft2(I).*wFilt));
% Scale image by SDs in local area
% in = inpolygon(handles.xa,handles.ya,handles.s.p(:,1),handles.s.p(:,2));
% I = I - mean(I(in));
% I = I / sqrt(mean(I(in).^2));
% compute the mean UC!
% handles = computeMeanUCsig(handles,I);
handles = computeMeanUC(handles);
drawMeanUC(handles);
% Update handles structure
guidata(hObject, handles);


function pushbuttonBraggLattice_Callback(hObject, eventdata, handles)
% Search for lattice in FFT space
I = handles.s.image;
% Generate filtering mask with bwdist
in = inpolygon(handles.xa,handles.ya,handles.s.p(:,1),handles.s.p(:,2));
mask = bwdist(~in);
mask = sin(mask*(pi/2/max(mask(:)))).^2;
% Create FFT image
Ifft2 = fftshift(abs(fft2(I.*mask)));
Ifft = Ifft2.^2;
[qx,qy] = makeFourierCoords(size(Ifft),1);
qx = fftshift(qx);
qy = fftshift(qy);
% Determine plot colour range
sm = fspecial('gaussian',5,1);
Ifft2 = conv2(Ifft2,sm,'same');
Ifft2 = sqrt(Ifft2);
Ivals = sort(Ifft2(:));
inds = round([.93 .999]*length(Ivals))+[1 0];
Ir = Ivals(inds);
% Get input from user
figure(11)
clf
imagesc(Ifft2)
axis equal off
colormap(hot(256))
set(gca,'position',[0 0 1 1])
caxis(Ir)
[ym,xm] = ginput(2);
% Find largest peak in Ifft close by
r2 = 8^2;
p =   Ifft > circshift(Ifft,[-1 -1]) ...
    & Ifft > circshift(Ifft,[ 0 -1]) ...
    & Ifft > circshift(Ifft,[ 1 -1]) ...
    & Ifft > circshift(Ifft,[-1  0]) ...
    & Ifft > circshift(Ifft,[ 1  0]) ...
    & Ifft > circshift(Ifft,[-1  1]) ...
    & Ifft > circshift(Ifft,[ 0  1]) ...
```

```
    & Ifft > circshift(Ifft,[ 1  1]);
[xp,yp,Ip] = find(p.*Ifft);
sub = (xp-xm(1)).^2 + (yp-ym(1)).^2 < r2;
data = sortrows([xp(sub) yp(sub) Ip(sub)],-3);
xy = data(1,1:2);
Icut = Ifft(xy(1)+(-1:1),xy(2)+(-1:1));
dx = (Icut(3,2)-Icut(1,2))/(2*Icut(2,2)-Icut(3,2)-Icut(1,2))/2;
dy = (Icut(2,3)-Icut(2,1))/(2*Icut(2,2)-Icut(2,3)-Icut(2,1))/2;
if dx > 0
    pqx = qx(xy(1))*(1-dx) + qx(xy(1)+1)*dx;
else
    pqx = qx(xy(1))*(1+dx) + qx(xy(1)-1)*-dx;
end
if dy > 0
    pqy = qy(xy(2))*(1-dy) + qy(xy(2)+1)*dy;
else
    pqy = qy(xy(2))*(1+dy) + qy(xy(2)-1)*-dy;
end
qvec = [pqx pqy];
handles.s.lat(2,:) = qvec./norm(qvec)^2;
% uNew = qvec./norm(qvec)^2;
sub = (xp-xm(2)).^2 + (yp-ym(2)).^2 < r2;
data = sortrows([xp(sub) yp(sub) Ip(sub)],-3);
xy = data(1,1:2);
Icut = Ifft(xy(1)+(-1:1),xy(2)+(-1:1));
dx = (Icut(3,2)-Icut(1,2))/(2*Icut(2,2)-Icut(3,2)-Icut(1,2))/2;
dy = (Icut(2,3)-Icut(2,1))/(2*Icut(2,2)-Icut(2,3)-Icut(2,1))/2;
if dx > 0
    pqx = qx(xy(1))*(1-dx) + qx(xy(1)+1)*dx;
else
    pqx = qx(xy(1))*(1+dx) + qx(xy(1)-1)*-dx;
end
if dy > 0
    pqy = qy(xy(2))*(1-dy) + qy(xy(2)+1)*dy;
else
    pqy = qy(xy(2))*(1+dy) + qy(xy(2)-1)*-dy;
end
qvec = [pqx pqy];
handles.s.lat(3,:) = qvec./norm(qvec)^2;
% generated fitted positions
or = handles.s.lat(1,:);
u = handles.s.lat(2,:);
v = handles.s.lat(3,:);
% Compute a,b values
f = handles.s.latSiteFrac;
a = round(f(1)*((handles.s.pos(:,2)-or(2))*v(1) ...
    - (handles.s.pos(:,1)-or(1))*v(2))/(v(1)*u(2)-v(2)*u(1)))/f(1);
b = round(f(2)*((handles.s.pos(:,2)-or(2))*u(1) ...
    - (handles.s.pos(:,1)-or(1))*u(2))/(v(2)*u(1)-v(1)*u(2)))/f(2);
xf = or(1) + a*u(1) + b*v(1);
yf = or(2) + a*u(2) + b*v(2);
% Export and plot fitted positions
handles.s.pos(:,3:6) = [a b xf yf];
set(handles.posFit,'xdata',yf,'ydata',xf);
updateLatValues(handles);
% Update handles structure
```

```
guidata(hObject, handles);
% Bring main window to front
figure(handles.figure1)

% Fix stupid ginput
function [out1,out2,out3] = ginput(arg1)
out1 = []; out2 = []; out3 = []; y = [];
c = computer;
if ~strcmp(c(1:2),'PC')
    tp = get(0,'TerminalProtocol');
else
    tp = 'micro';
end

if ~strcmp(tp,'none') && ~strcmp(tp,'x') && ~strcmp(tp,'micro'),
    if nargout == 1,
        if nargin == 1,
            out1 = trmginput(arg1);
        else
            out1 = trmginput;
        end
    elseif nargout == 2 || nargout == 0,
        if nargin == 1,
            [out1,out2] = trmginput(arg1);
        else
            [out1,out2] = trmginput;
        end
        if  nargout == 0
            out1 = [ out1 out2 ];
        end
    elseif nargout == 3,
        if nargin == 1,
            [out1,out2,out3] = trmginput(arg1);
        else
            [out1,out2,out3] = trmginput;
        end
    end
else

    fig = gcf;
    figure(gcf);

    if nargin == 0
        how_many = -1;
        b = [];
    else
        how_many = arg1;
        b = [];
        if  ischar(how_many) ...
                || size(how_many,1) ~= 1 || size(how_many,2) ~= 1 ...
                || ~(fix(how_many) == how_many) ...
                || how_many < 0
            error(message('MATLAB:ginput:NeedPositiveInt'))
        end
        if how_many == 0
```

```
            % If input argument is equal to zero points,
            % give a warning and return empty for the outputs.

            warning (message('MATLAB:ginput:InputArgumentZero'));
        end
    end

    % Setup the figure to disable interactive modes and activate pointers.
    initialState = setupFcn(fig);

    % onCleanup object to restore everything to original state in event of
    % completion, closing of figure errors or ctrl+c.
    c = onCleanup(@() restoreFcn(initialState));


    % We need to pump the event queue on unix
    % before calling WAITFORBUTTONPRESS
    drawnow
    char = 0;

    while how_many ~= 0
        % Use no-side effect WAITFORBUTTONPRESS
        waserr = 0;
        try
            keydown = wfbp;
        catch %#ok<CTCH>
            waserr = 1;
        end
        if(waserr == 1)
            if(ishghandle(fig))
                cleanup(c);
                error(message('MATLAB:ginput:Interrupted'));
            else
                cleanup(c);
                error(message('MATLAB:ginput:FigureDeletionPause'));
            end
        end
        % g467403 - ginput failed to discern clicks/keypresses on the figure
it was
        % registered to operate on and any other open figures whose handle
        % visibility were set to off
        figchildren = allchild(0);
        if ~isempty(figchildren)
            ptr_fig = figchildren(1);
        else
            error(message('MATLAB:ginput:FigureUnavailable'));
        end
        %         old code -> ptr_fig = get(0,'CurrentFigure'); Fails when
the
        %         clicked figure has handlevisibility set to callback
        if(ptr_fig == fig)
            if keydown
                char = get(fig, 'CurrentCharacter');
                button = abs(get(fig, 'CurrentCharacter'));
```

```
            else
                button = get(fig, 'SelectionType');
                if strcmp(button,'open')
                    button = 1;
                elseif strcmp(button,'normal')
                    button = 1;
                elseif strcmp(button,'extend')
                    button = 2;
                elseif strcmp(button,'alt')
                    button = 3;
                else
                    error(message('MATLAB:ginput:InvalidSelection'))
                end
            end
            axes_handle = gca;
            drawnow;
            pt = get(axes_handle, 'CurrentPoint');

            how_many = how_many - 1;

            if(char == 13) % & how_many ~= 0)
                break;
            end

            out1 = [out1;pt(1,1)]; %#ok<AGROW>
            y = [y;pt(1,2)]; %#ok<AGROW>
            b = [b;button]; %#ok<AGROW>
        end
    end

    % Cleanup and Restore
    cleanup(c);

    if nargout > 1
        out2 = y;
        if nargout > 2
            out3 = b;
        end
    else
        out1 = [out1 y];
    end
end
    function key = wfbp
fig = gcf;
current_char = []; %#ok<NASGU>
waserr = 0;
try
    h=findall(fig,'Type','uimenu','Accelerator','C');   % Disabling ^C for
edit menu so the only ^C is for
    set(h,'Accelerator','');                            % interrupting the
function.
    keydown = waitforbuttonpress;
    current_char = double(get(fig,'CurrentCharacter')); % Capturing the
character.
```

```
    if~isempty(current_char) && (keydown == 1)          % If the character
was generated by the
        if(current_char == 3)                           % current keypress
AND is ^C, set 'waserr'to 1
            waserr = 1;                                 % so that it errors
out.
        end
    end

    set(h,'Accelerator','C');                           % Set back the
accelerator for edit menu.
catch %#ok<CTCH>
    waserr = 1;
end
drawnow;
if(waserr == 1)
    set(h,'Accelerator','C');                          % Set back the
accelerator if it errored out.
    error(message('MATLAB:ginput:Interrupted'));
end
if nargout>0, key = keydown; end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function initialState = setupFcn(fig)
% Store Figure Handle.
initialState.figureHandle = fig;
% Suspend figure functions
initialState.uisuspendState = uisuspend(fig);
% Disable Plottools Buttons
initialState.toolbar = findobj(allchild(fig),'flat','Type','uitoolbar');
if ~isempty(initialState.toolbar)
    initialState.ptButtons =
[uigettool(initialState.toolbar,'Plottools.PlottoolsOff'), ...
        uigettool(initialState.toolbar,'Plottools.PlottoolsOn')];
    initialState.ptState = get (initialState.ptButtons,'Enable');
    set (initialState.ptButtons,'Enable','off');
end
% Setup FullCrosshair Pointer without warning.
oldwarnstate = warning('off', 'MATLAB:hg:Figure:Pointer');
% set(fig,'Pointer','fullcrosshair');
warning(oldwarnstate);
% Adding this to enable automatic updating of currentpoint on the figure
set(fig,'WindowButtonMotionFcn',@(o,e) dummy());
% Get the initial Figure Units
initialState.fig_units = get(fig,'Units');
function restoreFcn(initialState)
if ishghandle(initialState.figureHandle)
    % Figure Units
    set(initialState.figureHandle,'Units',initialState.fig_units);
    set(initialState.figureHandle,'WindowButtonMotionFcn','');
    % Plottools Icons
    if ~isempty(initialState.toolbar) && ~isempty(initialState.ptButtons)
        set (initialState.ptButtons(1),'Enable',initialState.ptState{1});
        set (initialState.ptButtons(2),'Enable',initialState.ptState{2});
    end
    % UISUSPEND
```

```
    uirestore(initialState.uisuspendState);
end
function dummy()
% do nothing, this is there to update the GINPUT WindowButtonMotionFcn.
function cleanup(c)
if isvalid(c)
    delete(c);
end


function pushbuttonAbout_Callback(hObject, eventdata, handles)
msgbox(['These scripts were created by Colin Ophus [clophus@lbl.gov] ' ...
    'at the National Center for Electron Microscopy, ' ...
    'Lawrence Berkeley National Laboratory, ' ...
    'Berkeley, CA, USA.  Use at your own risk, ' ...
    'as accuracy cannot be guaranteed!']);


% --- Executes when figure1 is resized.
function figure1_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

A.2 Error Fixing

Use the mouse to select the errors in the image one by one by clicking on the redundant miss-fitted peaks and the missing peak positions in the image. The coordinates of the errors will show on the image and after selecting all the errors, export the coordinates data into the workspace. Run the self-built code ‘missing_pts_v2’, and the errors will be removed automatically.

```
function [ r] = missing_pts_v2( a,r)
p=r.pos(:,1:2);
p1 = r.pos(:,1:3);
c=zeros(length([a.Position])/2,2);
b = [a.Position];

%uses for loop to rearrange the order of the matrix
for k = 1:length([a.Position])
    d = (k+1)/2;
    e = k/2;
    if mod(k+2,2)==0

        c(e,1) = b(k);
```

```
    else
        if mod(k+2,2)==1
         c(d,2)=b(k);
        end
    end
end
c1 = c;
%use for loop to delete repeated rows and get rid of extra points
for k = 1:length(c)
    for x = 1:length(p)
        if p(x,:) == c(k,:)
            c1(k,:) = NaN;
            p(x,1) = NaN;
            r.pos(x,1) = NaN;
        end
    end
end

c1(isnan(c(:,1)),:)=[];
p(isnan(p(:,1)),:)=[];
r.pos(isnan(r.pos(:,1)),:)=[];
% here the extra pts are gone
s = size(r.pos);
p = [p;c1];
%here p is the x y value we need
d = length(p) - s(1);
z = zeros(d,s(2));
r.pos = [r.pos;z];

r.pos(:,1:2) = p;
r.pos(isnan(r.pos(:,1)),:)=[];
end
```

A.3 Statistical Analysis on $LiNbO_3$

To obtain the displacement of the oxygen atoms with respect to the Nb atoms, we need to calculate an average value of the oxygen displacement regardless of the displacement direction. We first select the unit cells in $LiNbO_3$ to include two adjacent rows of Oxygen atoms and three rows of Niobium atoms, as shown in Figure 4-2. The average positions of the two Oxygen atoms and their surrounding Nb atoms in each unit cell were then obtained. The displacement is calculated by comparing the oxygen center and the niobium center. The displacements are collected in both perpendicular and parallel to wall directions, and the statistical analysis was done by calculating the mean value and standard deviation for each atom rows.

```
function s = LNO_BFv5(s)
%Takes the multi-peak fitted atom positions and assigns the atoms into
%A-sites and B-sites.
AverageAxis = 2; %is is for y, 2 is for x. Use 2 for a vertical DW and 1 for
a horizontal DW
posRefineM = s.posRefineM; %get the refined peak positions
image = s.image; %get the image
calibration = 7;
u = norm(s.lat(2,:));
v = norm(s.lat(3,:));
image = 65535 - image; %my atoms are white, while the background is black. I
flip the contrast here.
abPos = s.abPos; %abPos after multipeak fitting, the STEMlat01m_gs.m file
contains both the atom position data, and the unit cell coordinates.
Np = length(posRefineM); %no. of atoms present
aSite = zeros(Np,6);
bSite = zeros(Np,6);
bSite(:,1:2) = abPos(:,1:2); %RealspaceLattice01 lattice positions
bSite(:,3:4) = abPos(:,9:10); % Multi-peak refined coordinates
bSite(:,5:6) = abPos(:,3:4); %Refined lattice positions.
%Remove lattice mismarkings, all coordinates should be (I, I) or (I.5,I.5)
%where I is an integer.
for counter = 1:Np;
    if(mod((bSite(counter,1) + bSite(counter,2)),1) ~= 0)%mod returns
remainder
        bSite(counter,:) = NaN;%if this atom's coordinate is all 0.5,
                               %then make this row all NAN
    end;
end;
clearvars counter;
bSite=bSite(~isnan(bSite(:,2)),:);% isnan marks NAN to be 1,
                                  %this code line select all a sites.
aSite = bSite;
%Next we divide the RealspaceLattice peak positions, and if they are
%integers they are A-sites, else they are B-sites
NumberAtoms = length(bSite);
for counter = 1:NumberAtoms;
    if(mod(bSite(counter,1),1) ~= 0)
    aSite(counter,:) = NaN;
    else bSite(counter,:) = NaN;
    end;
end;
clearvars counter;
bSite=bSite(~isnan(bSite(:,2)),:);
aSite=aSite(~isnan(aSite(:,2)),:);

if AverageAxis == 1
    OtherAxis = 2;
elseif AverageAxis == 2
    OtherAxis = 1;
end;
```

```
a_parallel = unique(aSite(:,AverageAxis)); %take the unique A-site
coordinates parallel to DW
b_parallel = unique(bSite(:,AverageAxis)); %take the unique B-site
coordinates parallel to DW
a_perpendicular = unique(aSite(:,OtherAxis)); %take the unique A-site
coordinates perpendicular to DW
b_perpendicular = unique(bSite(:,OtherAxis)); %take the unique B-site
coordinates perpendicular to DW
d_aSite_parallel = zeros(length(aSite),7); %
d_bSite_parallel = zeros(length(bSite),7);
d_aSite_perpendicular = zeros(length(aSite),7);
d_bSite_perpendicular = zeros(length(bSite),7);

for sorter = 1:length(a_parallel);
    id = aSite(:,AverageAxis) == a_parallel(sorter);%if asite = a_parallel,
but they have different length, i don't get it.
    dX = aSite(id,6) - aSite(id,2);%difference in realLattice and Refined
    dY = aSite(id,5) - aSite(id,1);
    d_aSite_parallel(sorter,:) = [a_parallel(sorter) mean(dY) mean(dX)
std(dY) std(dX) std(dY) std(dX)];
%     std = sqrt(std(dY)^2+std(dX)^2-2*length(a_parallel)*std(dY)*std(dX));
    d_aSite_parallel(sorter,4) =
d_aSite_parallel(sorter,4)/(sqrt(length(dY)));% this is used to calculate
    d_aSite_parallel(sorter,5) =
d_aSite_parallel(sorter,5)/(sqrt(length(dX)));
end;
clearvars sorter id dX dY;

for sorter2 = 1:length(b_parallel);
    id2 = bSite(:,AverageAxis) == b_parallel(sorter2);
    dX = bSite(id2,6) - bSite(id2,2);
    dY = bSite(id2,5) - bSite(id2,1);
    d_bSite_parallel(sorter2,:) = [b_parallel(sorter2) mean(dY) mean(dX)
std(dY) std(dX) std(dY) std(dX)];
    d_bSite_parallel(sorter2,4) =
d_bSite_parallel(sorter2,4)/(sqrt(length(dY)));
    d_bSite_parallel(sorter2,5) =
d_bSite_parallel(sorter2,5)/(sqrt(length(dX)));
end;
clearvars sorter2 id dX dY;

for sorter3 = 1:length(a_perpendicular);
    id = aSite(:,OtherAxis) == a_perpendicular(sorter3);
    dX = aSite(id,6) - aSite(id,2);
    dY = aSite(id,5) - aSite(id,1);
    d_aSite_perpendicular(sorter3,:) = [a_perpendicular(sorter3) mean(dY)
mean(dX) std(dY) std(dX) std(dY) std(dX)];
    d_aSite_perpendicular(sorter3,4) =
d_aSite_perpendicular(sorter3,4)/(sqrt(length(dY)));
    d_aSite_perpendicular(sorter3,5) =
d_aSite_perpendicular(sorter3,5)/(sqrt(length(dX)));
end;
clearvars sorter3 id dX dY;

for sorter4 = 1:length(b_perpendicular);
```

```
    id = bSite(:,OtherAxis) == b_perpendicular(sorter4);
    dX = abs(bSite(id,6) - bSite(id,2));
    dY = abs(bSite(id,5) - bSite(id,1));
    d_bSite_perpendicular(sorter4,:) = [b_perpendicular(sorter4) mean(dY)
mean(dX) std(dY) std(dX) std(dY) std(dX)];
    d_bSite_perpendicular(sorter4,4) =
d_bSite_perpendicular(sorter4,4)/(sqrt(length(dY)));
    d_bSite_perpendicular(sorter4,5) =
d_bSite_perpendicular(sorter4,5)/(sqrt(length(dX)));
end;
clearvars sorter4 id dX dY;


d_aSite_parallel(d_aSite_parallel==0)=NaN;%if d_asite_parallel = 0, mark as
NAN
d_bSite_parallel(d_bSite_parallel==0)=NaN;
d_aSite_perpendicular(d_aSite_perpendicular==0)=NaN;
d_bSite_perpendicular(d_bSite_perpendicular==0)=NaN;
d_aSite_parallel=d_aSite_parallel(~isnan(d_aSite_parallel(:,2)),:);%check if
2nd column of d_asite_parallel is NAN, if yes, replace with zeros. it
assigned a-para with non-zero values.
d_bSite_parallel=d_bSite_parallel(~isnan(d_bSite_parallel(:,2)),:);
d_aSite_perpendicular=d_aSite_perpendicular(~isnan(d_aSite_perpendicular(:,2)
),:);
d_bSite_perpendicular=d_bSite_perpendicular(~isnan(d_bSite_perpendicular(:,2)
),:);
d_aSite_parallel(isnan(d_aSite_parallel))=0;%marks back zeros to NAN marked
rows
d_bSite_parallel(isnan(d_bSite_parallel))=0;
d_aSite_perpendicular(isnan(d_aSite_perpendicular))=0;
d_bSite_perpendicular(isnan(d_bSite_perpendicular))=0;

sub_parallel = zeros(length(d_aSite_parallel),4);
if length(d_aSite_parallel)>length(d_bSite_parallel)
    for  i = 1:length(d_bSite_parallel)
    sub_parallel(i,1) = d_aSite_parallel(i,2)- d_bSite_parallel(i,2);%Y
    sub_parallel(i,2) = d_aSite_parallel(i,3)- d_bSite_parallel(i,3);%X
    sub_parallel(i,3) = sqrt(d_aSite_parallel(i,6)^2 +
d_bSite_parallel(i,6)^2 -
2*length(d_aSite_parallel)*d_aSite_parallel(i,6)*d_bSite_parallel(i,6));%stdY
    %sub_parallel(i,4) = sqrt(d_aSite_parallel(i,7)^2 +
d_bSite_parallel(i,7)^2 -
2*length(d_aSite_parallel)*d_aSite_parallel(i,7*d_bSite_parallel(i,7)));%stdX
    sub_parallel(i,3) = sub_parallel(i,3)/length((d_aSite_parallel));
    sub_parallel(i,4) = sub_parallel(i,4)/length((d_aSite_parallel));
    % sub_parallel(i,3) = sub_perpendicular(i,3) = abs(d_bSite_parallel(i,4)-
d_aSite_parallel(i,4));
    end
else length(d_aSite_parallel)< length(d_bSite_parallel);
    for  i = 1:length(d_aSite_parallel)
    sub_parallel(i,1) = d_aSite_parallel(i,2)- d_bSite_parallel(i,2);%Y
    sub_parallel(i,2) = d_aSite_parallel(i,3)- d_bSite_parallel(i,3);%X
    sub_parallel(i,3) = sqrt(d_aSite_parallel(i,6)^2 +
d_bSite_parallel(i,6)^2 -
2*length(d_aSite_parallel)*d_aSite_parallel(i,6)*d_bSite_parallel(i,6));%stdY
```

```
    %sub_parallel(i,4) = sqrt(d_aSite_parallel(i,7)^2 +
d_bSite_parallel(i,7)^2 -
2*length(d_aSite_parallel)*d_aSite_parallel(i,7*d_bSite_parallel(i,7)));%stdX
    sub_parallel(i,3) = sub_parallel(i,3)/length((d_aSite_parallel));
    sub_parallel(i,4) = sub_parallel(i,4)/length((d_aSite_parallel));
    % sub_parallel(i,3) = sub_perpendicular(i,3) = abs(d_bSite_parallel(i,4)-
d_aSite_parallel(i,4));
    end
end
sub_perpendicular = zeros(length(d_aSite_perpendicular),3);
for  i = 1:length(d_bSite_perpendicular)
sub_perpendicular(i,1) = d_aSite_perpendicular(i,2)-
d_bSite_perpendicular(i,2);
sub_perpendicular(i,2) = d_aSite_perpendicular(i,3)-
d_bSite_perpendicular(i,3);
% sub_perpendicular(i,3) = abs(d_bSite_parallel(i,5)- d_aSite_parallel(i,5));
end

FittingCurve1 =
fit(d_aSite_parallel(:,1),sub_parallel(:,1),'smoothingspline');
FittingCurve2 =
fit(d_aSite_parallel(:,1),sub_parallel(:,2),'smoothingspline');

y2=FittingCurve2(d_aSite_parallel(:,1));
[fy1,fy2]= differentiate(FittingCurve2,d_aSite_parallel(:,1));

% figure;
% plot(d_aSite_parallel(:,1),y2,'g');
% hold on;
% plot(d_aSite_parallel(:,1),fy1,'r');
% % plot(d_aSite_parallel(:,1),fy2,'b');
% hold off;

%Calibration of data diagram
d_aSite_parallel(:,1:5) = calibration*d_aSite_parallel(:,1:5);
d_bSite_parallel(:,1:5) = calibration*d_bSite_parallel(:,1:5);
sub_parallel(:,1:4) = calibration*sub_parallel(:,1:4);
d_aSite_parallel(:,1) = u * d_aSite_parallel(:,1);
d_aSite_parallel(:,3) = u * d_aSite_parallel(:,3);
d_aSite_parallel(:,5) = u * d_aSite_parallel(:,5);
d_aSite_parallel(:,2) = v * d_aSite_parallel(:,2);
d_aSite_parallel(:,4) = v * d_aSite_parallel(:,4);
d_bSite_parallel(:,1) = u * d_bSite_parallel(:,1);
d_bSite_parallel(:,3) = u * d_bSite_parallel(:,3);
d_bSite_parallel(:,5) = u * d_bSite_parallel(:,5);
d_bSite_parallel(:,2) = v * d_bSite_parallel(:,2);
d_bSite_parallel(:,4) = v * d_bSite_parallel(:,4);
sub_parallel(:,1) = v*sub_parallel(:,1);
sub_parallel(:,3) = v*sub_parallel(:,3);
sub_parallel(:,2) = u*sub_parallel(:,2);

figure;
%
errorbar(d_aSite_parallel(:,1),d_aSite_parallel(:,2),d_aSite_parallel(:,4),'g
');
```

```
set(gca,'fontsize',18)
ylabel('displacement in y direction (pm)');
xlabel('x axis coordinates (pm)');
%1st:a-para, 2nd: mean(dy),4th: std(dy)
hold on;
%
errorbar(d_bSite_parallel(:,1),d_bSite_parallel(:,2),d_bSite_parallel(:,4),'r
');
errorbar(d_aSite_parallel(:,1),sub_parallel(:,1),sub_parallel(:,3),'b');
hold off;

figure;
%
errorbar(d_aSite_parallel(:,1),d_aSite_parallel(:,3),d_aSite_parallel(:,5),'g
'); %X axis means
set(gca,'fontsize',18)
ylabel('displacement in y direction (pm)');
xlabel('x axis coordinates (pm)');
hold on;
%
errorbar(d_bSite_parallel(:,1),d_bSite_parallel(:,3),d_bSite_parallel(:,5),'r
'); %Y axis means
errorbar(d_aSite_parallel(:,1),sub_parallel(:,2),sub_parallel(:,3),'b');
hold off;

figure;
errorbar(d_aSite_parallel(:,1),d_aSite_parallel(:,2),d_aSite_parallel(:,4),'g
');
set(gca,'fontsize',18)
ylabel('displacement in y direction (pm)');
xlabel('x axis coordinates (pm)');%1st:a-para, 2nd: mean(dy),4th: std(dy)
hold on;
errorbar(d_bSite_parallel(:,1),d_bSite_parallel(:,2),d_bSite_parallel(:,4),'r
');
% errorbar(d_aSite_parallel(:,1),sub_parallel(:,1),sub_parallel(:,3),'b');
hold off;

figure;
errorbar(d_aSite_parallel(:,1),d_aSite_parallel(:,3),d_aSite_parallel(:,5),'g
'); %X axis means
set(gca,'fontsize',18)
ylabel('displacement in y direction (pm)');
xlabel('x axis coordinates (pm)');
hold on;
%
errorbar(d_bSite_parallel(:,1),d_bSite_parallel(:,3),d_bSite_parallel(:,5),'r
'); %Y axis means
% errorbar(d_aSite_parallel(:,1),sub_parallel(:,2),sub_parallel(:,3),'b');
hold off;

% figure;
%
errorbar(d_aSite_perpendicular(:,1),d_aSite_perpendicular(:,2),d_aSite_perpen
dicular(:,4),'g');
% hold on;
```

```
%
errorbar(d_bSite_perpendicular(:,1),d_bSite_perpendicular(:,2),d_bSite_perpen
dicular(:,4),'r');
% hold off;
%
% figure;
%
errorbar(d_aSite_perpendicular(:,1),d_aSite_perpendicular(:,3),d_aSite_perpen
dicular(:,5),'g'); %X axis means
% hold on;
%
errorbar(d_bSite_perpendicular(:,1),d_bSite_perpendicular(:,3),d_bSite_perpen
dicular(:,5),'r'); %Y axis means
% hold off;

s.bSite = bSite;
s.aSite = aSite;
s.d_aSite_parallel = d_aSite_parallel;
s.d_bSite_parallel = d_bSite_parallel;
s.subpara = sub_parallel;
figure;
imagesc(image); colormap(gray(65535)); axis image off;
hold on;
scatter(aSite(:,4),aSite(:,3),'g.');
scatter(bSite(:,4),bSite(:,3),'r.');
hold off;
end
```

A.4 Statistical Analysis on $Ca_3Ru_2O_7$

The displacements of Ca atoms with respect to two nearest Ca atoms in two adjacent rows in the $Ca_3Ru_2O_7$ are collected in both perpendicular and parallel to wall directions, and the statistical analysis was done by calculating the mean value and standard deviation for each atom rows.

```
function s = Polar_Ruthenate(s)
%Function written by Debaangshu Mukherjee to geberate polar displacement
%maps of 327 Ruthenates of the general formula Ca3Ru2O7 from ADF-STEM
%images.
%This assumes that a long c-axis was chosen for the Ruthenate
%RealspaceLattice01 calculations rather than a generic perovskite lattice.
%Also, the codes assume that a general STEmlat01_gs.m was run rather than a
%multi-peak fit which is overkill.

WorkPos(:,1:2) = s.pos(:,3:4); %value according to u3 v4 vectors
WorkPos(:,3) = s.posRefine(:,3) - s.lat(1,1);
WorkPos(:,4) = s.posRefine(:,4) - s.lat(1,2);
```

```
WorkPos(:,3:4) = WorkPos(:,3:4)/s.lat(2:3,:); %Refined Lattice Positions,
find ratio w/respect to u and v
WorkPos(:,5:6) = s.posRefine(:,3:4); %Refined Pixel Positions
WorkPos(:,7) = WorkPos(:,1) - floor(WorkPos(:,1));%decimals u
WorkPos(:,8) = WorkPos(:,2) - floor(WorkPos(:,2));% decimals v

Np = length(WorkPos);
PolarLattice = zeros(Np,10);

% Get the Polar displace atoms
PolarLattice(:,3:4) = WorkPos(:,5:6);%refined pixel position
PolarLattice(:,5) = WorkPos(:,7) - WorkPos(:,8);%decimal of u (0.1
increments) minus decimal of v(0.5 increment)
for i=1:Np
    if  (PolarLattice(i,5) >= 0.199) && (PolarLattice(i,5) <= 0.201) %this
roundaboutway is to circumvent MATLAB's precision issues, if u-v = 0.2?
        PolarLattice(i,5) = WorkPos(i,1);% u vector value
        PolarLattice(i,6) = WorkPos(i,2);% v vector value
    else PolarLattice(i,1) = NaN;
    end;
end;
PolarLattice(isnan(PolarLattice(:,1)),:)=[]; %delete NaN values

PolarLattice(:,2) = PolarLattice(:,6); %v value
PolarLattice(:,1) = PolarLattice(:,5) - 0.2; % u values -0.2 = same decimal
as v
PolarLattice(:,5) = PolarLattice(:,5) + 0.2; % u values + 0.2, what is this?
scratch(:,1:2) = WorkPos(:,1:2);
Np2 = length(PolarLattice);
minx = min(WorkPos(:,2));% min v
miny = min(WorkPos(:,1));% min u
maxx = max(WorkPos(:,2));% max v
maxy = max(WorkPos(:,1));% max u

% this for loop filters values smaller or larger than original values
for k = 1:Np2
    if PolarLattice(k,1) < miny
        PolarLattice(k,1) = NaN;
    elseif PolarLattice(k,2) < minx
        PolarLattice(k,2) = NaN;
    elseif PolarLattice(k,5) > maxy
        PolarLattice(k,5) = NaN;
    elseif PolarLattice(k,6) > maxx
        PolarLattice(k,6) = NaN;
    end;
end;
PolarLattice(isnan(PolarLattice(:,1)),:)=[]; %delete NaN values

Np3 = length(PolarLattice);
for j=1:Np3
    %knnsearch returns Idx, which is a column vector of the indices in Mdl.X
representing the nearest neighbors.
    previous = knnsearch(scratch,[PolarLattice(j,1) PolarLattice(j,2)]);%
previews is a column vector, list the # of closest point in scratch
```

```
    after = knnsearch(scratch,[PolarLattice(j,5) PolarLattice(j,6)],'k',2);%
after is a vector, list the # of closest point in scratch

    PolarLattice(j,1) = WorkPos(previous(1,1),5);
    PolarLattice(j,2) = WorkPos(previous(1,1),6);
    PolarLattice(j,5) = WorkPos(after(1,1),5);% result 1:after(1,1); result
2: after(1,2)
    PolarLattice(j,6) = WorkPos(after(1,1),6);

%     PolarLattice(j,1) = PolarLattice(j,WorkPos(previous(1,1),5));
%     PolarLattice(j,2) = PolarLattice(j,WorkPos(previous(1,1),6));
%     PolarLattice(j,5) = PolarLattice(j,WorkPos(after(1,1),5));
%     PolarLattice(j,6) = PolarLattice(j,WorkPos(after(1,1),5));

end;
s.WorkPos = WorkPos;
s.PolarLattice = PolarLattice;
s.PolarLattice(:,7) = ((s.PolarLattice(:,5)+s.PolarLattice(:,1))/2) -
s.PolarLattice(:,3);
s.PolarLattice(:,8) = ((s.PolarLattice(:,6)+s.PolarLattice(:,2))/2) -
s.PolarLattice(:,4);
[s.PolarLattice(:,9),s.PolarLattice(:,10)] =
cart2pol(s.PolarLattice(:,8),s.PolarLattice(:,7));
s.PolarLattice(:,9) = (s.PolarLattice(:,9)/(2*pi)) + 0.5;
mediandisp = median(s.PolarLattice(:,10));
for iii = 1:Np3
    if s.PolarLattice(iii,10) > 4*mediandisp
        s.PolarLattice(iii,10) = NaN;
    end;
end;
s.PolarLattice(isnan(s.PolarLattice(:,10)),:)=[]; %delete NaN values
Np = length(s.PolarLattice);
QuiverColor = zeros(Np,3);
for ArrowColor = 1:Np;
ans22 = hsv2rgb([s.PolarLattice(ArrowColor,9),1,1]);
QuiverColor(ArrowColor,:) = ans22;
end;
figure;
imagesc(s.image); colormap(gray(65535)); axis image off;
hold on;
scatter(s.PolarLattice(:,2),s.PolarLattice(:,1),'y.');
scatter(s.PolarLattice(:,4),s.PolarLattice(:,3),'b.');
scatter(s.PolarLattice(:,6),s.PolarLattice(:,5),'y.')
%scatter(s.PolarLattice(:,8),s.PolarLattice(:,7),'g.');
hold off;

figure;
imagesc(s.image); colormap(gray(65535)); axis image off;
hold on;
for QuiverPlotting = 1:Np;
quiver(s.PolarLattice(QuiverPlotting,4),s.PolarLattice(QuiverPlotting,3),7.5*
(s.PolarLattice(QuiverPlotting,8)),7.5*(s.PolarLattice(QuiverPlotting,7)),'Co
lor',[QuiverColor(QuiverPlotting,1),QuiverColor(QuiverPlotting,2),QuiverColor
(QuiverPlotting,3)],'AutoScale','off','LineWidth',1.25,'MaxHeadSize',1);
end;
```

```
hold off;
end
```

BIBLIOGRAPHY

[1] J. C. Colin Ophus, "Correcting nonlinear drift distortion of scanning probe and scanning transmission electron microscopies from image pairs with orthogonal scan directions," *Ultramicroscopy,* vol. 1, no. 9, p. 162, 2016.

[2] William D. Callister, Jr. and David G. Rethwisch, Fundamentals of Materials Science and Engineering, New York City: John Wiley & Sons, Inc. , 2012.

[3] P. Ghosez, Microscopic Properties of Ferroelectric Oxides from First-Prinsciples, Lausanne: Troisieme Cycle de la Physique en Suisse Romande, 2002.

[4] Karin M. Rabe, Charles H. Ahn, Jean-Marc Triscone, Physics of Ferroelectrics: A Modern Perspective, Heidelberg: Springer, 2007.

[5] C. Menoret, J. M. Kiat, B. Dkhil, M. Dunlop, H. Dammak, O. Hernandez, "Structural evolution and polar order in Sr1-xBaxTiO3," *Physical Review,* vol. 12, no. 14, p. 65, 2002.

[6] W. Zhong, D. Vanderbilt, K. M. Rabe:, "Phase transitions in BaTiO3 from first principles," *Physical Review Letter,* vol. 3540, p. 73, 1994.

[7] M. E. Lines, A. M. Glass, Principles and Applications of Ferroelectrics and Relate Materials, Clarendon: Oxford, 1977.

[8] C. J. Fennie, K. M. Rabe, "Structural and dielectric properties of Sr2TiO4 from first principles," *Physical Review B,* no. 184111, p. 68, 2003.

[9] C. J. Fennie, K. M. Rabe, "First-principles investigation of ferroelectricity in epitaxially strained Pb2TiO4," *Physical Review B*, no. 100102, p. 71, 2005.

[10] Herbert, A. J. Moulson and J. M., Electroceramics, Chichester: John Wiley & Sons Ltd, 2003.

[11] Blount, P. W. Anderson and E. I., "SYMMETRY CONSIDERATIONS ON MARTENSITIC TRANS FORMATIONS: "Ferroelectric" Metals?," *Physical Review Letters*, vol. 14, no. 7, pp. 217 - 219, 1965.

[12] Shi Y, Guo Y, Wang X, Princep A, Khalyavin D, Boothroyd A, et al., "A ferroelectric-like structural transition in a metal," *Nature Materials*, vol. 12, no. 11, pp. 1024-1027, 2013.

[13] M. Ghita, M. Fornari, D. J. Singh & S. V. Halilov, "Interplay between A-site and B-site driven instabilities in perovskites," *Physical Review B*, vol. 054114, p. 72, 2005.

[14] Bousquet, E., Dawber, M., Stucki, N., Lichtensteiger, C., Hermet, P., Gariglio, S., . . . Ghosez, P. , "Improper ferroelectricity in perovskite oxide artificial superlattices," *Nature*, vol. 732, no. 6, p. 452, 2008.

[15] Nicole A. Benedek and Craig J. Fennie, "Hybrid Improper Ferroelectricity: A Mechanism for Controllable Polarization-Magnetization Coupling," *Physical Review Letters*, vol. 107204, p. 106, 2011.

[16] G. Catalan, J. Seidel, R. Ramesh,J. F. Scott, "Domain wall nanoelectronics," *Review of Modern Physics*, vol. 84, no. 1, pp. 119-150, 2012.

[17] Chun-Lin Jia, Shao-Bo Mi, Knut Urban, Ionela Vrejoiu, Marin Alexe & Dietrich Hesse, "Atomic-scale study of electric dipoles near charged and uncharged domain walls in ferroelectric films," *Nature Materials*, vol. 7, pp. 57-61, 2008.

[18] Donghwa Lee, Rakesh K. Behera, Pingping Wu, Haixuan Xu, Y. L. Li, Susan B. Sinnott, Simon R. Phillpot, L. Q. Chen, and Venkatraman Gopalan, "Mixed Bloch-Néel-Ising character of 180 ferroelectric domain walls," *Physical Review B,* vol. 80, no. 6, 2009.

[19] Greg Stone, Colin Ophus, Turan Birol, Jim Ciston, Che-Hui Lee, Ke Wang, Craig J. Fennie, Darrell G. Schlom, Nasim Alem & Venkatraman Gopalan, "Atomic scale imaging of competing polar states in a Ruddlesden–Popper layered oxide," *Nature Communications,* vol. 7, p. 12572, 2016.

[20] C. B. C. David B. Williams, Transmission Electron: A Textbook for Materials Science, New York: Springer Science+Business Media, 2009.

[21] Konstantin Iakoubovskii, Kazutaka Mitsuishi, Yoshiko Nakayama and Kazuo Furuya, "Mean free path of inelastic electron scattering in elemental solids and oxides using transmission," *Physical Review B,* vol. 77, no. 10, p. 104102, 2008.

[22] N.A., "Smoothing an Image," Dartmouth Univeristy, [Online]. Available: http://northstar-www.dartmouth.edu/doc/idl/html_6.2/Smoothing_an_Image.html. [Accessed 6 April 2017].

[23] N.A., "Image Thresholding," Mathworks, 2017. [Online]. Available: https://www.mathworks.com/discovery/image-thresholding.html. [Accessed 6 April 2017].

[24] "Gaussian Models," The MathWorks, Inc., 2017. [Online]. Available: https://www.mathworks.com/help/curvefit/gaussian.html. [Accessed 2 March 2017].

[25] Lide Yao, Sampo Inkinen and Sebastiaan van Dijken, "Direct observation of oxygen vacancy-driven structural and resistive phase transitions in La2/3Sr1/3MnO3," *Nature Communications,* vol. 8, 2017.

[26] Naoya Shibata, Scott D. Findlay, Yuji Kohno, Hidetaka Sawada, Yukihito Kondo& Yuichi Ikuhara, "Differential phase-contrast microscopy at atomic resolution," *Nature Physics Letter*, no. 8, pp. 611-615, 2012.

ACADEMIC VITA

Academic Vita of Leixin Miao
Email: miaoleixin1994@gmail.com

Education

The Pennsylvania State University *Expected Graduation: May 2017*

- Schreyer Honors College, Bachelor of Science in Materials Science and Engineering

Research Experience

Undergraduate Research Assistant *September 2015 – January 2017*

***Computational Materials Science Group**, Penn State Materials Science and Engineering Dept.*

- Calculated elastic constant and defect formation energy of Co-based superalloys with DFT
- Analyzed data from calculation and explained impact of different alloying elements
- Constructed unit cells of superalloy in different configurations

Undergraduate Research Assistant *May 2016 - Present*

***Materials Characterization Group**, Penn State Materials Science and Engineering Dept.*

- Processed data and images obtained from TEM
- Wrote code to improve image quality and accelerate image processing
- Performed Atom position refining on TEM image to calibrate precise atomic positions
- Constructed displacement maps to determine atom displacement in ferroelectric materials
- Observed samples under optical microscope and TEM

Industry Experience

Research & Development Internship *May 2014 - July 2014*

Huaguang Advanced Welding Materials Co., Ltd, Hangzhou, China

- Conducted multiple measurement experiment to determine the quality of welding products
- Prepared samples of various alloy materials for metallography observation
- Translated the English version of specifications in welding industry into Chinese

Awards and Honors

- Honors Certification at Penn State Harrisburg *Spring 2015*
- George W. Brindley Award for Nonmetallic Chemistry *Fall 2015*
- Undergraduate Research Fellowship
- The Third Place in Undergraduate Research Poster Competition *Spring 2016*
- Member of Phi Kappa Phi Honor Society
- Donald W. Hamer Scholarship in Electronic and Photonic Materials *Fall 2016*
- Richard M. Wardrop, Jr. Honor Scholarship
- Dorothy Pate Enright Endowed Scholarship *Spring 2017*

Skills

Lab Skills:

- **Experimental:** TEM Sample Preparation, Atom position refining of TEM Image, Polarization Vector Mapping, Electrical Resistivity Measurement, Metallography Sample Preparation, Optical Microscopy
- **Computational:** Density Function Theory Calculation, Data Analysis

Software: MATLAB, Vienna Ab initio Simulation Package, Mathematica

Language: Chinese (Native Proficiency), English (Bilingual Proficiency)