

# **Quantitative Analysis of EBSD Data for Cubic Systems**

Submitted in partial fulfillment of the requirements for the degrees of  
Bachelor of Technology and Master of Technology

by

**Anil Kumar**

(100110057)

Supervisors:

Prof. M. P. Gururajan

Prof. Prita Pant



Department of Metallurgical Engineering and Materials Science  
INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

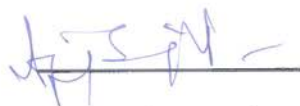
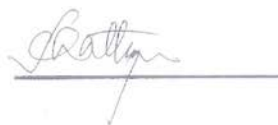
June 30, 2015



## APPROVAL CERTIFICATE

The dissertation entitled "Quantitative Analysis of EBSD Data for Cubic Systems" by Anil Kumar, 100110057 is approved for the degree of Bachelor of Technology and Master of Technology in Metallurgical Engineering and Materials Science with specialization in Ceramics and Composites.

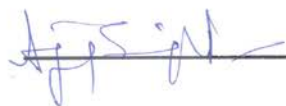
### Examiners



Supervisor



Chairperson



Date: 29.06.2015

Place: MEMS, IIT BOMBAY

## Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Anil Kumar

(Signature)

Anil Kumar

(Name of the student)

100110057

(Roll No.)

Date: July 1<sup>st</sup>, 2015

# Contents

<b>List of Figures .....</b>	<b>iv</b>
<b>List of Code Snippets .....</b>	<b>v</b>
<b>List of Tables .....</b>	<b>vi</b>
<b>Abstract .....</b>	<b>vii</b>
<b>Introduction .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 Objective .....	2
1.3 Organization of the Report .....	2
<b>Literature Survey .....</b>	<b>3</b>
2.1 Crystal Orientations .....	3
2.2 Crystal Symmetry .....	8
2.3 Misorientation and Disorientation calculations .....	8
2.4 Inverse Pole Figures .....	10
2.5 Grain Boundaries .....	13
2.6 Kernel Average Misorientation (KAM) .....	14
2.7 Grain Average Misorientation (GAM) .....	15
2.8 Grain Orientation Spread .....	16
2.9 Grain Size .....	17
<b>Calculations .....</b>	<b>18</b>
3.1 The Primitives .....	18
3.2 Grain Boundary Plot .....	24
3.3 Grain Map .....	25
3.4 Kernel Average Misorientation (KAM) .....	26
3.5 Grain Average Misorientation (GAM) .....	27
3.6 Grain Orientation Spread (GOS) .....	29
3.7 Grain Size Distribution (GSD) .....	30
<b>Results .....</b>	<b>31</b>
4.1 Grain Maps .....	32
4.2 Grain Boundary Plots .....	35
4.3 Grain Average Misorientation (GAM) .....	38
4.4 Grain Size Distribution (GSD) .....	43
4.5 Kernel Average Misorientation (KAM) .....	48
4.6 Grain Orientation Spread (GOS) .....	53
<b>Conclusion .....</b>	<b>57</b>
<b>Future Work .....</b>	<b>59</b>
<b>References .....</b>	<b>60</b>
<b>Appendix .....</b>	<b>61</b>
<b>User Manual .....</b>	<b>63</b>

# List of Figures

## Figures:

Fig. 2.1 Euler Angles .....	5
Fig. 2.2 Axis-Angle representation .....	6
Fig. 2.3 Stereographic projections .....	10
Fig. 2.4 Standard triangles .....	11
Fig. 2.5 Assigning colors .....	12
Fig. 2.6 Grain boundaries .....	13
Fig. 2.7 Kernel average misorientation .....	14
Fig. 2.8 Grain average misorientation .....	15
Fig. 2.9 Grain orientation spread .....	16
Fig. 2.7 Grain size calculations .....	17
Fig 3.1 Disorientation array representation .....	19
Fig. 4.1 Grain Map: 2.5% deformed Cu & annealed at 800 °C (EDAX) .....	32
Fig. 4.1 Grain Map: 2.5% deformed Cu & annealed at 800 °C (designed software) .....	32
Fig. 4.2 Grain Map: 40% Deformed Cu Annealed at 180°C for 105 Min (EDAX) .....	33
Fig. 4.2 Grain Map: 40% Deformed Cu Annealed at 180°C for 105 Min (designed software) .....	33
Fig. 4.3 Grain Map: 40% Deformed Cu Annealed at 180°C for 205 Min (EDAX) .....	34
Fig. 4.3 Grain Map: 40% Deformed Cu Annealed at 180°C for 205 Min (designed software) .....	34
Fig. 4.4 Grain Boundaries: 2.5% deformed Cu & annealed at 800 °C (EDAX) .....	35
Fig. 4.4 Grain Boundaries: 2.5% deformed Cu & annealed at 800 °C (designed software) .....	35
Fig. 4.5 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 90 Min (EDAX) .....	36
Fig. 4.5 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 90 Min (designed software)..	36
Fig. 4.6 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 205 Min (EDAX) .....	37
Fig. 4.6 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 205 Min(designed software).	37

## List of Code Snippets

Code snippet 3.1: Storing Euler angles and x-y coordinates .....	20
Code snippet 3.2: Calculating disorientation .....	21
Code snippet 3.3: Calculation of quaternions from Euler angles .....	21
Code snippet 3.4: Finding the minimum misorientation angle .....	22
Code snippet 3.5: Calculation of Grain ID .....	23
Code snippet 3.6: Storing grain boundary coordinates .....	24
Code snippet 3.7: Plotting grain boundary .....	24
Code snippet 3.8: Grain map calculations .....	25
Code snippet 3.9: KAM Calculation .....	26
Code snippet 3.10: KAM_data_calc function .....	26
Code snippet 3.11: GAM calculation .....	25
Code snippet 3.12: GAM_Final calculation .....	28
Code snippet 3.13: Distribution in bins .....	28
Code snippet 3.14 GOS calculations .....	29
Code snippet 3.15 GSD calculations .....	30

## List of Tables

Table 4.1 KAM of stainless steel .....	38
Table 4.2 KAM of 2.5% deformed Cu & annealed at 800 °C .....	39
Table 4.3 KAM of 40% Deformed Cu Annealed at 180°C for 90 Min .....	40
Table 4.4 KAM of 40% Deformed Cu Annealed at 180°C for 105 Min .....	41
Table 4.5 KAM of 40% Deformed Cu Annealed at 180°C for 205 Min .....	42
Table 4.6 GAM of stainless steel .....	43
Table 4.7 GAM of 2.5% deformed Cu & annealed at 800 °C .....	44
Table 4.8 GAM of 40% Deformed Cu Annealed at 180°C for 90 Min .....	45
Table 4.9 GAM of 40% Deformed Cu Annealed at 180°C for 105 Min .....	46
Table 4.10 GAM of 40% Deformed Cu Annealed at 180°C for 205 Min .....	47
Table 4.11 GOS of 40% Deformed Cu Annealed at 180°C for 105 Min .....	48
Table 4.12 GOS of 2.5% deformed Cu & annealed at 800 °C .....	49
Table 4.13 GOS of 40% Deformed Cu Annealed at 180°C for 205 Min .....	50
Table 4.14 GSD of stainless steel .....	51
Table 4.15 GSD of 2.5% deformed Cu & annealed at 800 °C .....	52
Table 4.16 GSD of 40% Deformed Cu Annealed at 180°C for 90 Min .....	53
Table 4.17 GSD of 40% Deformed Cu Annealed at 180°C for 105 Min .....	54
Table 4.18 GSD of 40% Deformed Cu Annealed at 180°C for 205 Min .....	55



# **Abstract**

A comprehensive software has been developed to analyze crystalline orientations through the analysis of Electron Backscatter Diffraction (EBSD) data from a Scanning Electron Microscope. The software uses Euler angles, x-y coordinates and confidence index of scan points from the data file to determine and plot Grain Map and Grain boundaries. The software also determines Kernel average misorientation, Grain average misorientation, Grain Orientation Spread and Grain sizes, distributes them in bins and uses bar plots to display the data. This software is designed to work on EBSD scans of materials having FCC and BCC crystalline structure in hexagonal grids.



# Chapter 1

## Introduction

This chapter introduces the motivation behind this project and the objective. The chapter also describes the organization of the report which briefly presents the content of this report.

### 1.1 Motivation

Most of engineering materials are crystalline. Hence, the crystal and microstructure of materials is of great interest. In particular, it is of interest to determine the grain boundaries, the grain sizes and the misorientation across grain boundaries.

Electron Backscatter Diffraction is a technique which examines the crystallographic orientation of the materials. Using any indexing software, the microstructural characteristics can be determined. There are several other programs which do the same (or even more) calculations. However as we looked about, we noticed these programs have one or more of the following properties:

- 1. These commercial programs are like Black boxes:** In commercial software, the user inputs the data and gets the result without having much idea of how the data is getting processed. But along with these computations, this project also explains each algorithm and the steps of calculations involved.
- 2. The commercial programs are too expensive:** Most of the users don't even use all the features but they do pay for these features which makes them even more expensive. The high

price limits its availability. However, the designed software will be shared under GNU General Public License and we are delighted to welcome you to use this software for free and/or update it!

3. **Limited customizability:** The commercial software offers some customization but the customizability is limited. The designed software with source code in hand is way more customizable.
4. **Require extensive installation:** The commercial software require extensive installation and may depend on other software/packages. But the designed software requires only *gnuplot* and updated *.NET Framework (4.5 or later)*.

## 1.2 Objective

The objective of this project is to:

- 1) Design a user-friendly comprehensive software which can analyze crystal orientations from an EBSD data file.
- 2) Determine and plot grain map and grain boundaries.
- 3) Calculate KAM, GAM, GOS and GSD.
- 4) Use plots/bar plots to display the data.

## 1.3 Organization of report

The report comprises of 4 chapters and an appendix. Chapter 1 gives the introduction and objective of the report. Chapter 2 consists of literature survey. Chapter 3 describes the algorithms used for calculation of values with the help of C++ code snippets and chapter 4 shows the results. Chapter 5 consists of conclusion and Appendix consists of the manual for using the software and its source code.

## Chapter 2

### Literature Survey

This chapter introduces the representation of crystal orientations, describes the ways of calculating misorientations and disorientations and defines Grain boundary, Grain map, Kernel average misorientation (KAM), Grain average misorientation (GAM) and Grain size distribution (GSD). This chapter also illustrates the calculation of GAM, KAM and GSD.

#### 2.1 Crystal Orientations

Most engineering materials are crystalline in nature. To specify the crystal orientation of a particular crystal, we need to define two (preferably right handed Cartesian) coordinate systems, one relating to the whole sample ( $G_s$ ) and the other relating to the crystal ( $G_c$ ). The term crystal orientation of the crystal describes the orientation of the principle axes of the crystal ( $G_c$ ) relative to the principal axes of the sample ( $G_s$ ).

The choice of directions (for principal axes), is in principle arbitrary [1], although it is convenient to adapt it to the crystal symmetry. Hence for crystals of orthogonal branch (e.g. cubic), the axes [001], [010] and [100] already form an orthogonal frame and can conveniently be adapted as the crystal coordinate system.

The crystal orientations can be represented in various forms, the most commonly used are Transformation matrix using direction cosines, Euler angles, Angle-Axis representation and Quaternions. A brief introduction on all these representation is given below:

### 2.1.1 Transformation Matrix

Consider a set of two right handed Cartesian coordinate systems  $Oxyz$  ( $G_{\text{sample}}$ ) and  $Ox'y'z'$  ( $G_{\text{crystal}}$ ). It is easily seen that by a suitable continuous rotation about  $O$ , the set of axes  $Oxyz$  may be brought into coincidence with the set  $Ox'y'z'$ . (Note that if one set is right-handed and other set is left handed then it is impossible to bring them into coincidence by rotation alone).

Let the direction cosines of  $Ox'$  relative to the axes  $Oxyz$  be  $l_{11}$ ,  $l_{12}$  and  $l_{13}$ . Further, denote the direction cosines of  $Oy'$  and  $Oz'$  by  $l_{21}$ ,  $l_{22}$ ,  $l_{23}$  and  $l_{31}$ ,  $l_{32}$ ,  $l_{33}$ . We may conveniently summarize this by the following matrix:

$$\begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

This matrix is known as **transformation matrix**.

Let a point  $P$  has coordinates  $(x', y', z')$  relative to coordinate system  $Ox'y'z'$  and  $(x, y, z)$  relative to the coordinate system  $Oxyz$ . Then the  $x'$ ,  $y'$  and  $z'$  coordinates of  $P$  are orthogonal projections of  $OP$  on  $Ox'$ ,  $Oy'$  and  $Oz'$ . Hence, using the transformation matrix, we can calculate  $(x', y', z')$  as following [2]:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

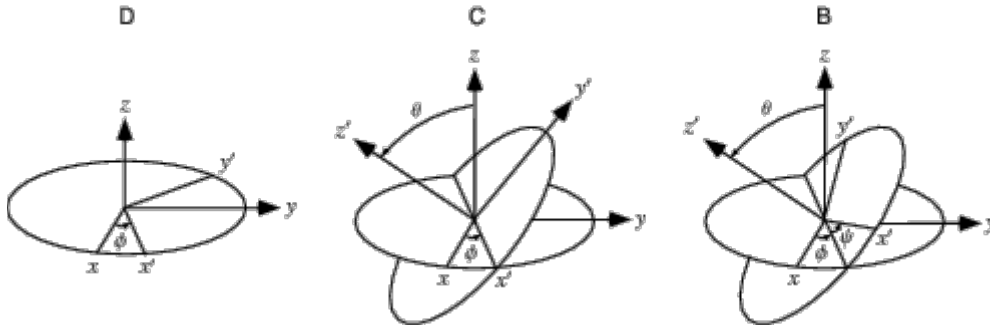
Since  $Ox'$ ,  $Oy'$  and  $Oz'$  are mutually perpendicular and  $l_{ij}$  are direction cosines, so they must satisfy orthonormality conditions and the determinant of transformation matrix must be +1.

## 2.1.2 Euler Angles

Consider a set of two right handed Cartesian coordinate systems  $Oxyz$  ( $G_s$ ) and  $Ox'y'z'$  ( $G_c$ ). According to Euler's rotation theorem, any rotation of  $Oxyz$  to coincide with  $Ox'y'z'$  can be carried out using three Euler angles  $\phi$ ,  $\theta$  and  $\psi$  in the following sequence:

- 1) The coordinate axes are rotated about  $z$  axis in the counterclockwise (as viewed from positive  $z$ ) through an angle  $\phi$  in the range  $[0, 2\pi]$ .
- 2) Then rotate the axes about the former  $x$  axis (now  $x'$ ) in the counterclockwise (as viewed from positive  $x'$ ) through an angle  $\theta$  in the range  $[0, \pi]$ .
- 3) Then rotate the axes about the former  $z$  axis (now  $z'$ ) in the counterclockwise (as viewed from positive  $z'$ ) through an angle  $\psi$  in the range  $[0, 2\pi]$ .

The following Fig. illustrates the rotation:



**Fig 2.1** Euler Angles (source: Weisstein, Eric W. "Euler Angles." From MathWorld-A Wolfram Web Resource. <http://mathworld.wolfram.com/EulerAngles.html>)

As represented in the attached Fig. 1<sup>st</sup>, 2<sup>nd</sup> and 3<sup>rd</sup> rotations can be described by transformation matrices D, C and B respectively as follows [4]:

$$D = \begin{bmatrix} \cos \phi & \sin \phi & 0 \\ -\sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix}, B = \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So the combined transformation matrix  $A = BCD$  can be represented as following:

$$A = \begin{bmatrix} \cos \psi \cos \phi - \cos \theta \sin \phi \sin \psi & \cos \psi \sin \phi + \cos \theta \cos \phi \sin \psi & \sin \psi \sin \theta \\ -\sin \psi \cos \phi - \cos \theta \sin \phi \cos \psi & -\sin \psi \sin \phi + \cos \theta \cos \phi \cos \psi & \cos \psi \sin \theta \\ \sin \theta \sin \phi & -\sin \theta \cos \phi & \cos \theta \end{bmatrix}$$

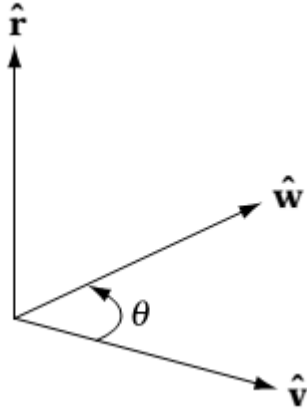
### 2.1.3 Axis-Angle representation

A general rotation in  $R^3$  can be represented in two parts: a vector  $\hat{\mathbf{r}}$  that lies along the axis of rotation, and a scalar  $\theta$  that corresponds to a counter clockwise rotation around the axis.

Generating axis-angle rotation [5] can be quite straight forward in some cases, for example:

$$\theta = \cos^{-1}(\hat{\mathbf{v}} \cdot \hat{\mathbf{w}})$$

$$\mathbf{r} = (\hat{\mathbf{v}} \times \hat{\mathbf{w}})$$



**Fig. 2.2** Axis-Angle representation. Rotation along  $\mathbf{r}$  by angle  $\theta$ , rotates  $\mathbf{v}$  into  $\mathbf{w}$ .

However, transformation matrix from  $\mathbf{r}(\mathbf{x}, \mathbf{y}, \mathbf{z}) / \theta$  pair similar to the transformation matrix from direction cosines is of the following form:

$$g = \begin{bmatrix} (1 - x^2) \cos \theta + x^2 & xy(1 - \cos \theta) + z \sin \theta & xz(1 - \cos \theta) - y \sin \theta \\ xy(1 - \cos \theta) - z \sin \theta & (1 - y^2) \cos \theta + y^2 & yz(1 - \cos \theta) + x \sin \theta \\ xz(1 - \cos \theta) + y \sin \theta & yz(1 - \cos \theta) - x \sin \theta & (1 - z^2) \cos \theta + z^2 \end{bmatrix}$$

So from a given rotation matrix, we can calculate  $\mathbf{r}(\mathbf{x}, \mathbf{y}, \mathbf{z})$  and  $\theta$ [6]:

$$\theta = \cos^{-1} \frac{(g_{11} + g_{22} + g_{33} - 1)}{2}$$

$$x = \frac{(g_{23} - g_{32})}{2 \sin \theta} \quad \text{or} \quad x = \left( \frac{g_{11} + 1}{2} \right)^{\frac{1}{2}} \quad \text{if } \sin \theta = 0$$

$$y = \frac{(g_{31} - g_{13})}{2 \sin \theta} \quad \text{or} \quad y = \left( \frac{g_{22} + 1}{2} \right)^{\frac{1}{2}} \quad \text{if } \sin \theta = 0$$

$$z = \frac{(g_{12} - g_{21})}{2 \sin \theta} \quad \text{or} \quad z = \left( \frac{g_{33} + 1}{2} \right)^{\frac{1}{2}} \quad \text{if } \sin \theta = 0$$



### 2.1.4 Quaternions

It can be considered as a variant of Angle-Axis representation. Quaternions only need four values (still effectively just 3 degrees of freedom, after normalization). The key advantages of using Quaternions are, but not limited to, efficient concatenations, symmetry operations and easy manipulations to find disorientation between two systems.

A general representation for a quaternion  $\mathbf{q}$  is of the following form:

$$\mathbf{q} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k} + w$$

where  $x$ ,  $y$ ,  $z$  and  $w$  are real numbers.

However, this representation can be simplified as following:

$$\mathbf{q} = (x, y, z, w)$$

A rotation about the unit vector  $\mathbf{n}$  by an angle  $\theta$  can be computed using the quaternion:

$$\mathbf{q} = (s, \mathbf{v}) = (\cos(\frac{1}{2}\theta), \mathbf{n}\sin(\frac{1}{2}\theta))$$

The components of this quaternion are called Euler parameters. After rotation, a point  $\mathbf{p} = (0, \mathbf{p})$  is then given by:

$$\mathbf{p}^{new} = \mathbf{q} \cdot \mathbf{p} \cdot \mathbf{q}^{-1}$$

where, dot (.) represents quaternion multiplication [11].

Input file to the designed software contains Euler angles for each pixels which are then converted into quaternions for further manipulations. Following is the relation between quaternions and Euler angles [12]

$$q1 = \cos(\frac{1}{2}(\phi - \psi)) \times \sin(\frac{1}{2}\theta)$$

$$q2 = \sin(\frac{1}{2}(\phi - \psi)) \times \sin(\frac{1}{2}\theta)$$

$$q3 = \sin(\frac{1}{2}(\phi + \psi)) \times \cos(\frac{1}{2}\theta)$$

$$q4 = \cos(\frac{1}{2}(\phi + \psi)) \times \cos(\frac{1}{2}\theta)$$

Here  $q4$  represents the real/angular part of quaternion, the very same notation has been followed in this report and the code throughout.

### From Angle-Axis to Quaternions

Once Angle-Axis pair has been calculated (§2.1.3), it is one step process to transform the same orientation in quaternions as following:

$$\mathbf{q} = (x.\sin(\theta/2), y.\sin(\theta/2), z.\sin(\theta/2), \cos(\theta/2)) \quad \text{-----}(1)$$

In this report transformation matrix using Euler angles, Angle-Axis representation and Quaternions have been used together to find the minimum misorientation between two grains.

## 2.2 Crystal Symmetry

Construction of transformation matrices using direction cosines of two coordinate system and Euler angles has been described in §2.1.1 and §2.1.2. However, there are many transformation matrices, when applied on any orientation, they result in identical orientation as following:

$$g_e = g_{tm} g_1$$

Here  $g_{tm}$  is a transformation matrix, but if this represents a symmetry operation then,  $g_1$  and  $g_e$  will be equivalent.

There are 24 symmetrically equivalent orientations [7] for a cubic symmetry. They are shown in detail in Appendix §A1.

## 2.3 Misorientation and Disorientation calculation

In this section there will be a brief introduction to misorientation and disorientation and then a systematic procedure will be described for efficient calculation of disorientation for each scanned pixel.

### 2.3.1 Misorientation

Misorientation between two crystals is a measure of the difference between their orientations. Following is the procedure to calculate misorientation:

- 1) As we have the Euler angles of each crystals, convert Euler angles to quaternions using §2.1.4, say these quaternions are p and q.
- 2) The quaternion  $q_0 = p^{-1} * q$  represents the quaternion corresponding to misorientation.
- 3) Due to cubic symmetry, there are 24 quaternions like  $q_0$  which represent the same misorientation. These symmetry operations in terms of quaternions has been mentioned in Appendix §A1. These 24 different quaternions encourage us to use the concept of disorientation.
- 4) Every single quaternion from these 24 quaternions, has one angular/real component (like e4 in  $q = \{e1, e2, e3, e4\}$  as mentioned in §2.1.4).

Then, misorientation/rotation angle is  $\emptyset(\text{in radian}) = 2 \times \cos^{-1}(\text{angular component})^{***}$

### 2.3.2 Disorientation

As mentioned above, there are 24 possible quaternions representing the same misorientation and hence there are 24 corresponding misorientation/rotation angles. **Among all these misorientations, the misorientation having minimum rotation angle is called Disorientation.**

Throughout this report, it can be seen that disorientation is the only value that matters, misorientation has been used to calculate disorientation, but misorientation is never directly used for any further calculations.

As  $\cos^{-1}$  is a monotonously decreasing function in  $[-1, 1]$ , both inclusive, we need to find the maximum of angular components among all the 24 quaternions. Then disorientation,

$$\emptyset(\text{in radian}) = 2 \times \cos^{-1}(\text{Maximum (angular component)}_{\text{among all 24 quaternions}})^{***}$$

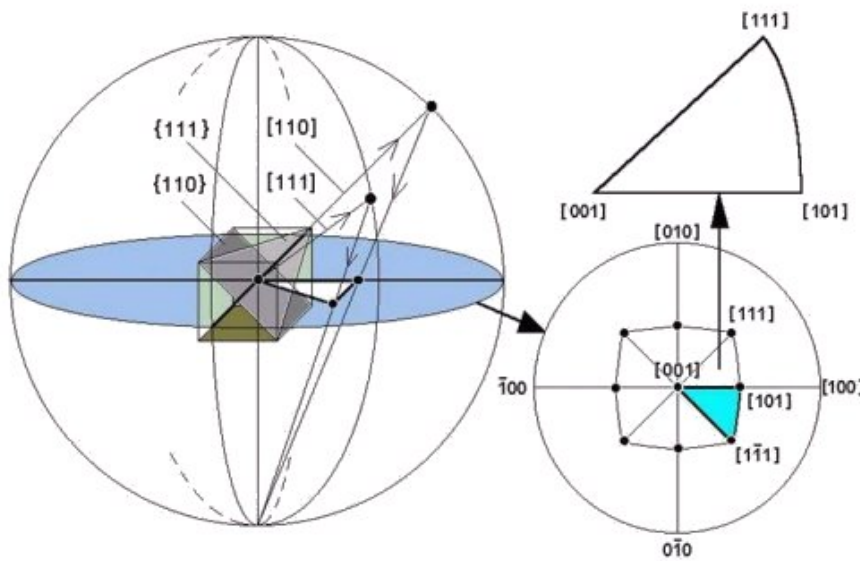
**\*\*\*** This formula is a result of the fact that every quaternion  $q$  is of the form

$$q = (s, \mathbf{v}) = (\mathbf{n} \sin(\frac{1}{2}\theta), \cos(\frac{1}{2}\theta)) \quad \text{from §2.1.4}$$

Where,  $\mathbf{n}$  is the unit vector and  $\theta$  is the angle of rotation around  $\mathbf{n}$

## 2.4 Inverse Pole Figure (IPF)

A pole figure shows sample directions aligned with a particular crystallographic pole (e.g. [001]) but an inverse pole figure does the opposite, indicating the crystallographic poles aligned with a specified sample direction (e.g. [001]). Thus, a [001] inverse pole figure shows which crystal direction in the crystal lattice is aligned with the normal of the sample reference frame. This is often of interest for samples in which the processing history strongly prefers a single direction.



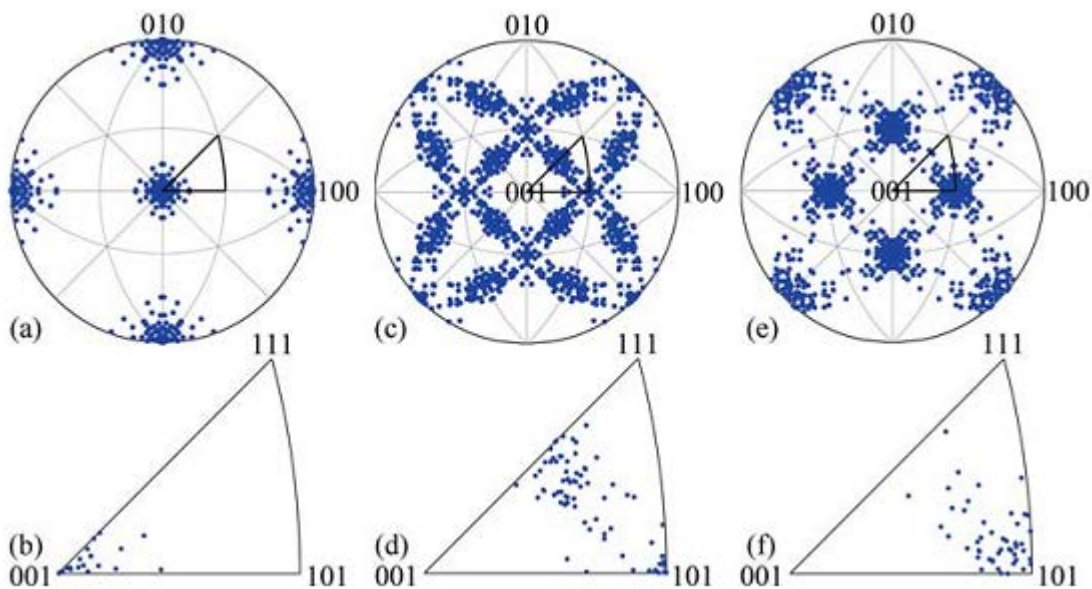
**Fig. 2.3** Stereographic projections and Construction of standard triangle using three directions: [001], [101] and [111] (source: <http://www.ebsd.com/popup/inversepolefigure.htm>)

If we have the Euler angles corresponding to the rotation of sample reference frame to crystal reference frame, the same can be used to calculate the corresponding rotation matrix (from §2.1.2). And the transpose of this matrix will represent the rotation in the opposite direction (from crystal reference frame to sample reference frame). Let's call it *CtoS matrix*.

Following is the procedure to get the [001] Inverse Pole Figure map:

1. Consider crystal reference frame as the working reference frame. Use *CtoS matrix* to rotate [001] axis and determine its final position, say *rotated axis*.

2. Head of the normalized *rotated axis* gives the coordinates of its intersection with unit sphere, as there has been no translation at all, just rotation. Let's say that  $[hkl]$  is the intersection point.
3. There is twenty-four-fold redundancy in the stereographic projection of these  $[hkl]$  values due to cubic symmetry. This redundancy encourages presentation by a single stereographic triangle. The standard stereographic triangle of the cubic system is that containing the crystallographic directions  $[hkl]$  for which  $l \geq h \geq k \geq 0$  \*\*, which have been outlined using bold lines in following Figure [13]:



**Fig. 2.4** (a) A cube texture, and (b) the standard stereographic triangle of the cube texture. (c) A copper texture, and (d) the standard stereographic triangle of the copper texture. (e) A brass texture, and (f) the standard stereographic triangle of the brass texture (source: Adam Schwartz Mukul Kumar Brent L. Adams· David P. Field, “Electron Backscatter Diffraction in Materials Science”, §3.3 Pole Figures pp. 40)

\*\* Suppose that  $[hkl]$  is represented by a combination three non-orthogonal axes as:

$$[hkl] = u[001] + v[101] + w[111]$$

$$\rightarrow u = l-h; v = h-k; \text{ and } w = k;$$

If stereographic projection of  $[hkl]$  falls in the standard triangle constructed by the stereographic projection of  $[001]$ ,  $[101]$  and  $[111]$ , then  $u$ ,  $v$  and  $w$  must be between 0 and 1, both inclusive.

$$\rightarrow l-h \geq 0 \text{ and } h-k \geq 0 \text{ and } k \geq 0 \quad (\text{lower limits of } u, v \text{ and } w)$$

$$\rightarrow l \geq h \text{ and } h \geq k \text{ and } k \geq 0$$

$$\rightarrow l \geq h \geq k \geq 0$$

The  $u$ ,  $v$  and  $w$  values calculated above can be used to composite a color and hence represent the considered crystal using that color in the Inverse Pole Figure plot.

### ***Another approach to color the crystals:***

Since the stereographic projection of  $[001]$ ,  $[101]$  and  $[111]$  are known, assign these coordinates perfectly Red, Green and Blue colors as following:

$$\text{RED} \equiv \{ 0, 0 \}$$

$$\text{GREEN} \equiv \{ (\sqrt{2} - 1), 0 \}$$

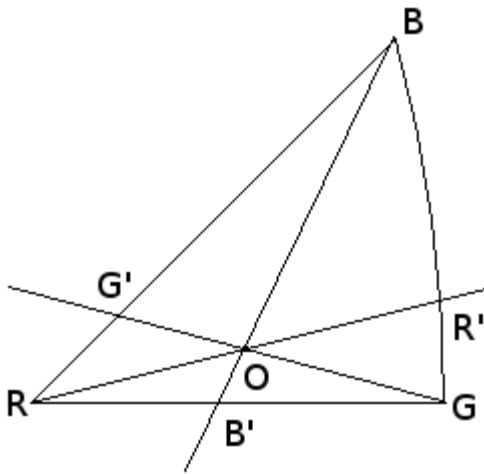
$$\text{BLUE} \equiv \{ (\sqrt{3} / 2 - 1.0 / 2.0), (\sqrt{3} / 2 - 1.0 / 2.0) \}$$

Everything else falling in this standard triangle is a combination of these three colors. For example, if we need to find the Red, Green and Blue color components for a point O then,

$$\text{red} = ||OR'|| / ||RR'||$$

$$\text{green} = ||OG'|| / ||GG'||$$

$$\text{blue} = ||OB'|| / ||BB'||$$



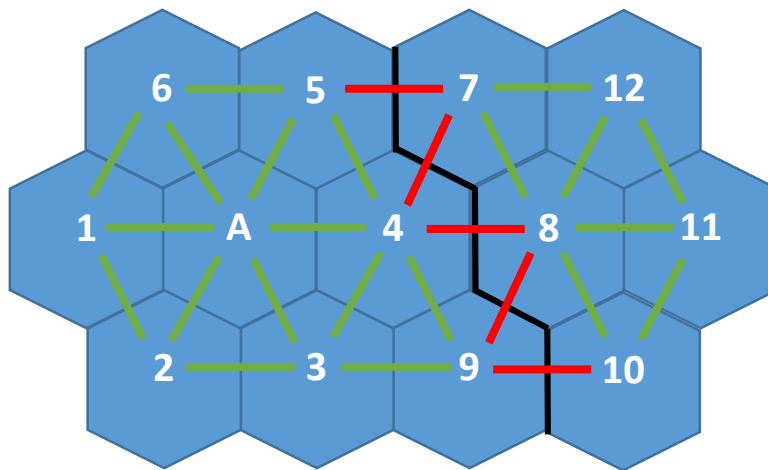
**Fig. 2.5** Assigning Red, Green and Blue color values for a crystal in Inverse Pole Figure plot

One of these two approaches are followed for each crystal to generate the colored Inverse Pole Figure plot for an EBSD scan. Both these approaches are expected to generate similar results.

## 2.5 Grain Boundaries

Before determining the grain boundary, the tolerance for the grain boundary needs to be defined. This tolerance is either a user defined value or the default value ( $5^\circ$ ). Once this tolerance is defined, it is treated as a constant for the entire calculation.

To determine the grain boundary, disorientation between every two neighbor pixels is compared with the tolerance. If the disorientation between these pixels is less than the tolerance, then both pixels belong to the same grain but if disorientation exceeds the tolerance, then there is a grain boundary between these two pixels.



**Fig. 2.6** Grain Boundary is shown with the black line

In Fig. 2.6, thirteen different pixels (A, 1, 2, 3..... 12) have been considered. Then, as shown in §2.3.2 disorientation between every neighbor pixel has been calculated. **Green line between two neighbor pixels represents that the disorientation between these two pixels is less than the tolerance value, while red line between two pixels represents that the disorientation between these two pixels is greater than the tolerance value.** Hence across every red line there exists a grain boundary, which is represented by the black line.

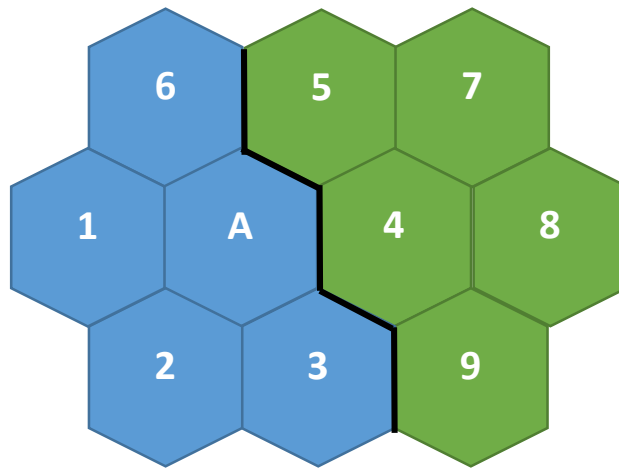
## 2.6 Kernel Average Misorientation (KAM)

Kernel Average Misorientation is defined for every pixel of the sample and not for any grain. So total number of KAM values of sample must be equal to the total number of scanned pixels. KAM of a pixel is average disorientation of that pixel with its neighbors (with the condition that disorientation exceeding the tolerance value is excluded). Following lines[8] describe the usefulness of KAM:

*The usefulness of KAM really depends upon the question being asked. Is the question, “Are there areas of the sample in which the very local misorientation is high?” (KAM is useful.) Or is the question, “Does this grain deform from one end with respect to the other?” (KAM is less useful.)*

-- Electron Backscatter Diffraction in Materials Science, 2<sup>nd</sup> Edition

Following example illustrates the calculation of Kernel Average Misorientation



**Fig. 2.7** Two grains separated by a grain boundary (black line)

In Fig. 2.8, there are two grains (in two colors), separated by a black colored grain boundary. For the sake simplicity, it is assumed that these are the only pixels, both grains have.

Pixel A has 4 neighbors (excluding pixel A and pixel 3 across the grain boundary)

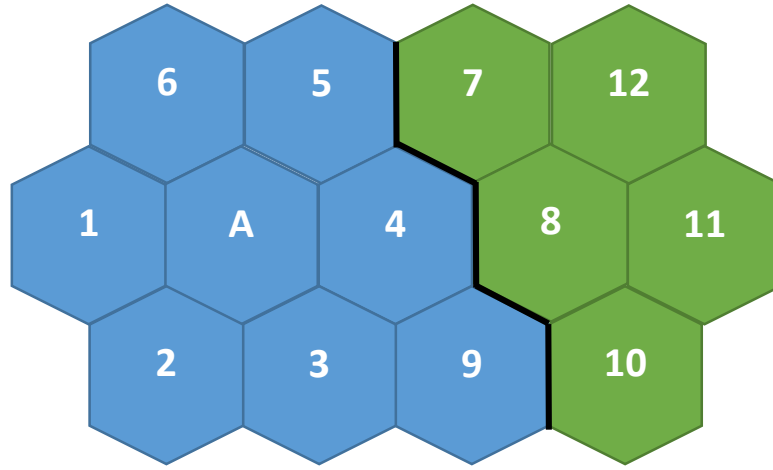
$$\Rightarrow KAM_A = \frac{\Delta g_1 + \Delta g_2 + \Delta g_3 + \Delta g_4}{4}$$



## 2.7 Grain Average Misorientation (GAM)

Grain average misorientation [9] is defined for an entire grain and not for any particular pixel. So for a given sample, the total number of grain average misorientation values should be same as the number of grains in the sample.

To calculate GAM of a grain, total disorientation of every pixel with their neighbors is calculated (with the condition that disorientation exceeding the tolerance value is excluded), also keeping track of the number of neighbors of that pixel. Once finished with the calculation for all these pixels, weighted average of these (total) disorientations is calculated to get Grain Average Misorientation of that grain.



**Fig. 2.8** Grain Average Misorientation calculation

In Fig. 2.9, there are two grains (in two colors), separated by a black colored grain boundary. Larger grain consists of 8 pixels while the smaller grain consists 5 pixels. For the sake simplicity, it is assumed that these are the only pixels, both grains have.

Pixel A has 6 neighbors  $\rightarrow \Delta g_A = \sum_{i=1}^6 \Delta g_{iA}$  and  $N_A = 6$  (excluding pixels across GB)

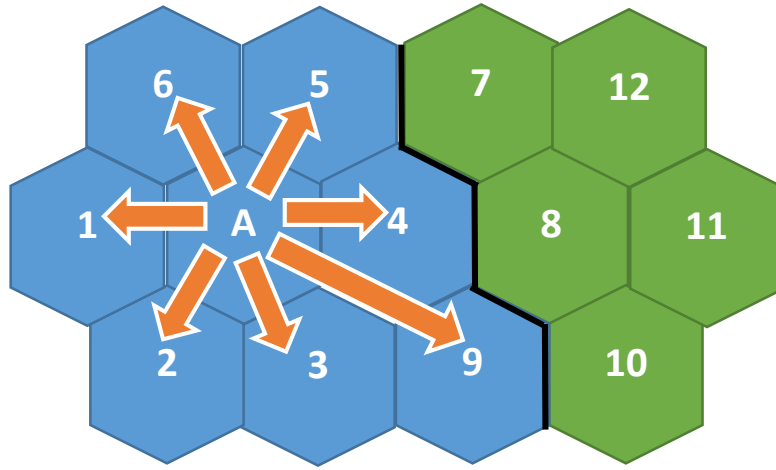
Pixel 9 has just two neighbors  $\rightarrow \Delta g_9 = \Delta g_{49} + \Delta g_{39}$  and  $N_9 = 2$  (excluding pixels across GB)

$$\text{So GAM}_{\text{blue}} = \frac{(\Delta g_A + \Delta g_1 + \Delta g_2 + \Delta g_3 + \Delta g_4 + \Delta g_5 + \Delta g_6 + \Delta g_9)}{(N_A + N_1 + N_2 + N_3 + N_4 + N_5 + N_6 + N_9)}$$

## 2.8 Grain Orientation Spread

To calculate Grain Orientation Spread, we need to calculate average orientation of each grain. The spread is then, the average deviation between the orientation of each point in the grain and the average orientation for the grain.

Neither the software gives any hint for the calculation for average orientation of a grain nor is there any standard method\*\*. So, for the sake of simplicity and unambiguity, this report treats the orientation of the pixel at geometrical center of a grain as the average orientation of the grain.



**Fig. 2.9** Here A is geometric center of the (blue colored) grain. Taking pixel A as a reference, misorientation between pixel A and all other pixels of the same grain is calculated and average of all these misorientations is the grain orientation spread of this grain.

If A is the geometric center of the grain and  $\Delta g_{A1}$  is the misorientation between pixels A and 1:

$$\text{GOS}_{\text{blue grain}} = \frac{\Delta g_{A1} + \Delta g_{A2} + \Delta g_{A3} + \Delta g_{A4} + \Delta g_{A5} + \Delta g_{A6} + \Delta g_{A9}}{7}$$

\*\*[14] showed that  $\frac{\sum_i q_i}{|\sum_i q_i|}$  (i.e. barycentric mean with renormalization) is the optimal mean quaternion. When this definition of mean quaternion was implemented in this software, the output was very different from the expected result.

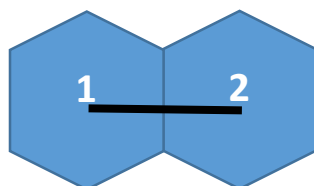
## 2.9 Grain Size

Grain size represents the equivalent diameter of a grain. So the total number of grain size values for a given sample is exactly same as the number of grains in that sample.

There are two different ways to calculate the Grain Size.

### Method 1:

This is an easy and intuitive way to calculate the grain size of a particular grain:



**Fig. 2.10** Grain size calculation (black line represents the distance between two pixels)

Consider the couple of pixels shown above. The black line represents the difference in x coordinates of two consecutive scanned pixels. Now, considering each pixel as a regular hexagon, area of each pixel is calculated. This area is multiplied by the number of pixels in that grain which gives total area of the grain.

Now considering this grain as a circular grain of diameter  $d$ , calculate the area of the circular grain and equate it with the area calculated earlier to get the equivalent diameter  $d$ .

Source of error: Hexagon is not actually a regular hexagon.

### Method 2:

This method differs from the previous one by a constant factor:

*The area is calculated by summing up the number of points in a grain multiplied by the product of the square of the step size and a factor depending on the type of scanning grid. For square grids the factor is one and for hexagonal grids the factor is the one half times the square root of 3. The grain size may also be specified by diameter. The diameter of a particular grain is calculated by determining the area of a grain and then assuming the grain is a circle. The diameter is then equal to 2 times the square root of the area divided by  $\pi$ .*

-- EDAX® OIM Analysis help

**To match the result of this project with that of the software, Method 2 has been used.**

## **Chapter 3**

### **Calculations**

This chapter describes step by step calculation of grain characteristics with the help of pieces of C++ program using just crystalline orientations and confidence index.

#### **3.1 The Primitives**

In this section, the calculation of some primitive data has been shown which will be used in every further calculations/plots. In the beginning, there will be brief introduction to the text file which is the input for this software, having the crystalline orientation of the sample (at a particular x, y coordinates), confidence index and other derived information. However the programmed software only uses the information about crystalline orientations and confidence index, and everything else is computed by the software itself.

### 3.1.1 The Source File

The source text file also known as the grain file, typically looks like following **Fig. 3.1**:

5.58729	0.32385	1.11176	14.00000	0.00000	1720.6	0.800	0.67	2	1	Iron - Gamma
3.57342	1.42657	5.99927	14.50000	0.00000	1848.1	0.657	1.13	2	1	Iron - Gamma
3.57343	1.42658	5.99928	15.00000	0.00000	1848.1	0.657	1.13	2	1	Iron - Gamma
2.44998	2.82632	0.47038	15.50000	0.00000	1778.2	0.600	0.98	2	1	Iron - Gamma
5.60537	0.32181	1.09106	16.00000	0.00000	1877.2	0.543	1.09	2	1	Iron - Gamma
5.58184	0.32112	1.11729	16.50000	0.00000	1937.7	0.486	1.09	2	1	Iron - Gamma
5.56588	0.32103	1.13598	17.00000	0.00000	1717.3	0.857	1.01	2	1	Iron - Gamma
5.57271	0.31424	1.12357	17.50000	0.00000	1852.5	0.314	1.56	2	1	Iron - Gamma
3.57869	1.42279	6.00119	18.00000	0.00000	1736.5	0.314	1.16	2	1	Iron - Gamma
2.47262	2.81783	0.48100	18.50000	0.00000	1250.5	0.543	1.22	2	1	Iron - Gamma
1.86837	2.31987	6.02192	19.00000	0.00000	1239.4	0.057	1.88	0	0	Iron - Gamma
0.50146	1.35725	0.77549	19.50000	0.00000	1354.1	0.314	1.83	3	1	Iron - Gamma
0.50146	1.35725	0.77549	20.00000	0.00000	1548.7	0.314	1.25	3	1	Iron - Gamma
2.27794	0.82348	2.84312	20.50000	0.00000	1401.4	0.029	1.66	0	0	Iron - Gamma
0.51125	1.32157	0.76196	21.00000	0.00000	1530.4	0.229	1.18	3	1	Iron - Gamma
5.44591	2.30344	0.31623	21.50000	0.00000	1512.2	0.971	0.92	3	1	Iron - Gamma
2.28597	0.81773	2.83509	22.00000	0.00000	1394.7	0.229	1.66	3	1	Iron - Gamma
2.28597	0.81773	2.83509	22.50000	0.00000	1394.7	0.229	1.66	3	1	Iron - Gamma
0.49881	1.35523	0.77475	23.00000	0.00000	1696.7	0.400	1.48	3	1	Iron - Gamma

Three Euler angles  $\phi$ ,  $\theta$   
and  $\psi$  respectively

x and y coordinates  
of pixels

Image  
quality

Confidence  
index

Grain  
ID

Phase  
name

Fit

Edge

As it has been described above, the designed software uses first 5 columns and 7<sup>th</sup> column for all its calculation.

Following points should be noted about this text file:

- Every row has a unique combination of x and y coordinates, it represents a unique pixel.
- 9<sup>th</sup> column is the grain ID column which shows the information about the pixel belonging to a particular grain (calculated by EDAX® OIM Analysis software) at a tolerance of 5° (default value). When the designed software needs grain ID, it calculates for the user defined tolerance value and doesn't uses this calculated value (specifically for 5°).

- The only reason behind including confidence index for calculation is that EDAX® OIM Analysis software ignores every pixel having confidence index less than a particular value.

### 3.1.2 Processing the Source File

The designed software first counts the number of lines in the text file. The number of lines directly represents the no of pixels scanned. This number is stored in the variable DATA\_SIZE and treated as a constant throughout the complete execution.

Complete data of file is stored in vectors (or vector of vectors). Following code shows the declaration and storage of Euler angles and x-y coordinates in corresponding vectors:

```
vector <vector <double>> euler(DATA_SIZE, vector <double>(3));
vector <double> x_coord(DATA_SIZE);
vector <double> y_coord(DATA_SIZE);

while (infile2 && (i < DATA_SIZE))
{
    infile2 >> euler[i][0] >> euler[i][1] >> euler[i][2] >> x_coord[i] >> y_coord[i] >> iq[i]\
        >> ci[i] >> fit[i] >> grain_id[i] >> edge[i] >> phase_name1 >> phase_name2\
        >> phase_name3;

    ++i;
}
```

**Code snippet 3.1:** Storing Euler angles and x-y coordinates

### 3.1.3 Calculation of Disorientation array

The calculation of disorientation array is always the 1<sup>st</sup> step. It is first step because disorientation array is used to determine the grain IDs and hence grain boundaries. It is called disorientation “array” just because for every scanned pixel there are 6 neighbors (except on the edges), so every pixels carries 6 disorientation values.

The calculation of disorientation array has already been described in §2.3.2. The Euler angles are first converted into Quaternions and then it is just quaternion manipulations as shown in the following code snippet:

```
double misorientation(const double ci1, const double ci2, const vector <double> &euler1,
const vector <double> &euler2)
{
    if (ci1 < min_ci || ci2 < min_ci) //incorporating low ci pixels!
        return 99.99;

    if (euler1 == euler2)
        return 0.0;

    vector <double> quat1(4), quat2(4), quatF(4);
    eulerToQuat(euler1, quat1);
    eulerToQuat(euler2, quat2);
    quatInverse(quat1); //got it inverted
    quatMultQuat(quat1, quat2, quatF); //effectively inverse(q1)*q2

    double e1 = quatF[0], e2 = quatF[1], e3 = quatF[2], e4 = quatF[3];
    //e4 is the real/angular part
```

**Code snippet 3.2:** Calculating disorientation

In the above snippet, `eulerToQuat` is a function shown below:

```
void eulerToQuat(const vector <double> &euler, vector <double> &q)
{
    //from http://mathworld.wolfram.com/EulerParameters.html follows x -convention as
by Goldstein (1980, p. 176)
    q[0] = cos(0.5 * (euler[0] - euler[2]))*sin(0.5 * euler[1]);
    q[1] = sin(0.5 * (euler[0] - euler[2]))*sin(0.5 * euler[1]);
    q[2] = sin(0.5 * (euler[0] + euler[2]))*cos(0.5 * euler[1]);
    q[3] = cos(0.5 * (euler[0] + euler[2]))*cos(0.5 * euler[1]); //actually q[3]
represents the real/angular part

    makeItUnitQuatOrVector(q);
    //finding the minima of real/angular part makes no sense here, so not going for
the symmetry opertaions
}
```

**Code snippet 3.3:** calculation of quaternions from Euler angles

Now, as quaternions  $q(e_1, e_2, e_3, e_4)$  has been calculated, just symmetry operations are remaining. As it has already been said that the target is to find the minimum rotation angle and in quaternions, 4<sup>th</sup> component represents the rotation angle. Hence symmetry operations are done only at the 4<sup>th</sup> component (increases efficiency).

From the above symmetry operations, the minimum rotation angle is needed. Since here, the 4<sup>th</sup> component is in the form of  $\cos(\text{angle}/2)$  and  $\cos$  is a decreasing function. So maximum 4<sup>th</sup> component needs to be found whose  $\cos^{-1}$  will give the minimum rotation angle. Following is the code snippet:

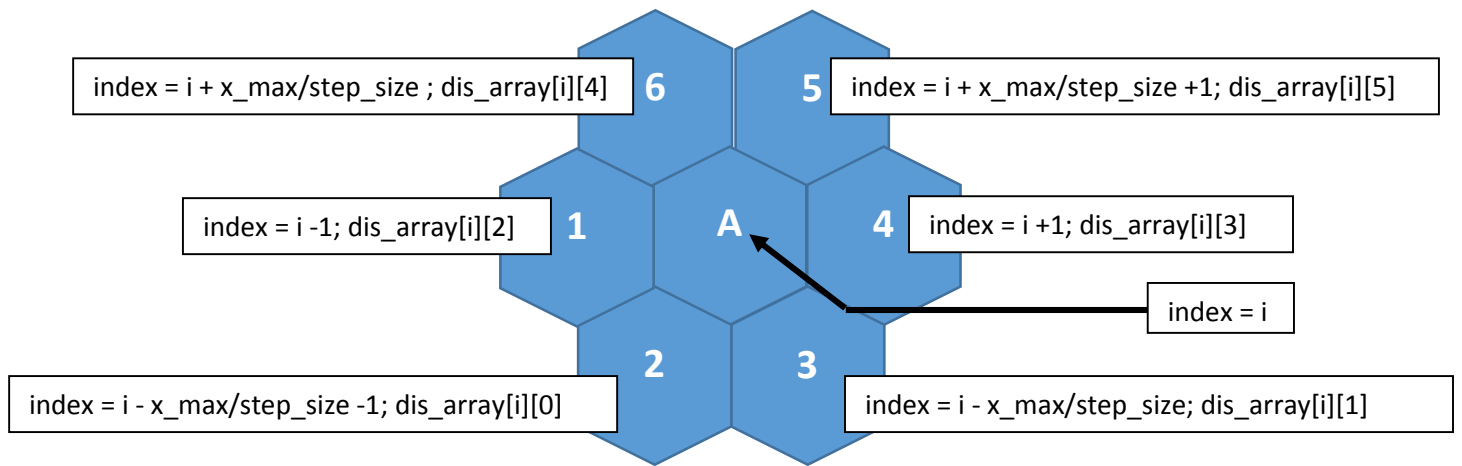
```
double largest = abs(e4);

for (int i = 0; i < 24; i++)
    if (largest < abs(eq[i]))
        largest = abs(eq[i]);

return 2 * acos(largest) * 180 / pi;
```

**Code snippet 3.4:** Finding the minimum misorientation angle

So the return value is the disorientation between both pixels. This completes the disorientation calculation! Following Fig. show the way all scanned pixels are stored in disorientation array:



**Fig 3.1** Disorientation array representation



### 3.1.4 Grain ID

As the name suggests, the Grain ID of a pixel tells the about the grain it belongs to. Initially, all pixels are assigned grain IDs equal to their row numbers and then calculation begins. In the end grain ID of a grain is the ID of one of its pixels and the whole grain shares the same common ID.

To calculate grain ID, disorientation array is used to get the disorientation between two neighbor pixels. If their disorientation is less than the user defined tolerance, the serial number of both pixels are passed in `union_pix` function. The `union_pix` function first checks their IDs. If they are already connected (sharing same ID) it does nothing, but if they are not connected it assigns them a common ID and this common ID is the ID of one of the pixels. Hence now onwards both these pixels belong to the same grain. Following code snippet describes the process using weighted union find algorithm:

```
void union_pix(int index1, int index2)
{
    int l = find_root(index1);
    int m = find_root(index2);
    int size_l = size[l];
    int size_m = size[m];

    if (l != m)
    {
        if (size_l < size_m)    //weighted! efficient!
        {
            id[l] = m;
            size[m] += size[l];
        }
        else
        {
            id[m] = l;
            size[l] += size[m];
        }
    }
}
```

**Code snippet 3.5:** Calculation of Grain ID

Calculation of Grain ID finishes this primitive section. Following sections will illustrate further calculations based on these primitives.

## 3.2 Grain Boundary Plot

As described in §2.4, grain boundaries are determined by comparing disorientation between two pixels with the user defined tolerance value. If the disorientation exceeds the tolerance, the x-y coordinates of the center of these pixels is stored, which represents the grain boundary between these two pixels. Following code snippet describes the procedure:

```
if ((x_coord[i] < (x_max - (step_size - 0.05) / 2)) && (x_coord[i] > (step_size - 0.05))) //every column
//except 1st, 2nd and last
{
    if (i > static_cast<int>(x_max / step_size)) //every row except 1st row
    {
        if (t > orient_angle)
        {
            xb.push_back((x_coord[i] + x_coord[i - 1]) / 2);
            yb.push_back((y_coord[i] + y_coord[i - 1]) / 2);
        }
        if (u > orient_angle)
        {
            xb.push_back((x_coord[i] + x_coord[i - static_cast<int>(x_max / step_size)]) / 2);
            yb.push_back((y_coord[i] + y_coord[i - static_cast<int>(x_max / step_size)]) / 2);
        }
        if (v > orient_angle)
        {
            xb.push_back((x_coord[i] + x_coord[i - static_cast<int>(x_max / step_size + 1)]) / 2);
            yb.push_back((y_coord[i] + y_coord[i - static_cast<int>(x_max / step_size + 1)]) / 2);
        }
    }
}
```

**Code snippet 3.6:** Storing grain boundary coordinates

Grain boundary coordinates are stored in xb and yb vectors following code snippet plots these boundaries using EasyBMP library[10]:

```
for (int i = 0; i < xb_size; ++i)
{
    l = static_cast<int>(xb[i]);
    m = static_cast<int>(yb[i]);
    bound(l, m)->Red = 0;
    bound(l, m)->Green = 0;
    bound(l, m)->Blue = 255; //coloring the boundaries blue
}
```

**Code snippet 3.7:** Plotting grain boundary

### 3.3 Grain Map:

As described in §2.4, in IPF, the vector pointing along a given sample direction, 001 here, is projected onto planes determined by the local crystallographic orientation; another equivalent idea is to get this vector is following:

- 1) Assume both reference orientation axes are coinciding at origin.
- 2) Calculate a rotation matrix = transpose of (Euler to matrix transformation).
- 3) Use this rotation matrix to rotate 001 of crystal; that would be 001 of sample, bingo!
- 4) We have just rotated, no translation; so just normalize it to find its intersection with unit sphere
- 5) Take its projection on the plane determined by local crystallographic orientation, in fact, that's the plane we are dealing with.

Following code snippet shows this procedure:

```
eulerToMatrix(euler[i], matrix); //creating rotation matrix;
squareMatrixTranspose(matrix); //transposed to get the IPF here

matrix_mult_3331(matrix, vect, result_); //rotating the vector vect to get result_
for (int j = 0; j < 24; ++j)
{
    tempMatrix = syms[j];
    matrix_mult_3331(tempMatrix, result_, ans_);

    vec_mod = vectorMod_(ans_); //its mod is always one!

    ans_[0] /= vec_mod;
    ans_[1] /= vec_mod;
    ans_[2] /= vec_mod;

    u = ans_[2] - ans_[0];
    v = ans_[0] - ans_[1];
    w = ans_[1];

    if ((u >= tol) && (v >= tol) && (w >= tol))
    {
        //stereographic projection has been found!
    }
}
```

**Code snippet 3.8:** Grain map calculations

### 3.4 Kernel Average Misorientation (KAM)

As it has been explained in §2.6, KAM is calculated for every pixel by averaging the disorientation of that pixel with their neighbors (excluding the case when disorientation exceeds the tolerance). Following code snippet explains the calculation.

```
for (int i = 0; i < DATA_SIZE; ++i)
{
    {
        if (x_coord[i] < (x_max - step_size + 0.05) && x_coord[i] > (step_size - 0.05))
            //every point on the grid except 1st two and last two columns
        {
            if (i > static_cast<int>(x_max / step_size) && (i < DATA_SIZE - static_cast<int>(x_max / step_size + 1)))
                // every row except bottom and top
                kam[i] = KAM_data_calc(dis_array, max_dis_angle, i);
        }
    }
}
```

**Code snippet 3.9:** KAM Calculation

`KAM_data_calc` is the function which takes care of the pixels on the edges and compares the disorientation between two pixels with user defined tolerance. Following code snippet explains it:

```
double KAM_data_calc(const vector <vector <double>> &dis_array, const double &misorient, const int &i,
const int &p, const int &q, const int &r, const int &s) //function used inside kam_calculation
{
    double count = 0;
    double sum = 0;

    if (p != 0 && q != 0 && r != 0 && s != 0)
    {
        //if (id[i] == id[i - (num + 1)])// &&
        if (dis_array[i][0] <= misorient)
        {
            sum += dis_array[i][0];
            ++count;
        }
    }
}
```

**Code snippet 3.10** `KAM_data_calc` function

So  $(\text{sum} / \text{count})$  is returned and stored in `kam[i]`, which is Kernel Average Misorientation of that pixel. Use of the function `KAM_data_calc()` shortens the code.

### 3.5 Grain Average Misorientation (GAM)

As described in §2.7 GAM represents the misorientation of an entire grain. In GAM, total disorientation of each pixel is calculated, keeping track of the number of neighbors too. So for GAM, two separate variables are needed to serve the purpose. In the following code snippet `gam[i]` is the total disorientation of a pixel and `m[i]` is the number of neighbors:

```
for (int i = 0; i < DATA_SIZE; ++i)
{
    if (x_coord[i] < (x_max - step_size + 0.05) && x_coord[i] > (step_size - 0.05))
        //every point on the grid except 1st two and last two columns
    {
        if (i > static_cast<int>(x_max/step_size)&&(i < DATA_SIZE - static_cast<int>(x_max / step_size + 1)))
            // every row except bottom and top
            tie(gam[i], m[i]) = GAM_data_calc(dis_array, num, i);

        else if (i > DATA_SIZE - static_cast<int>(x_max / step_size + 1))
            //top row, only 4 neighbours
            tie(gam[i], m[i]) = GAM_data_calc(dis_array, num, i, 4, 5);

        else if (i < static_cast<int>(x_max / step_size))
            //bottom row, 4 neighbours
            tie(gam[i], m[i]) = GAM_data_calc(dis_array, num, i, 0, 1);
    }
}
```

**Code snippet 3.11:** GAM calculation

`GAM_data_calc()` is quite similar to `KAM_data_calc()` and takes care of the total disorientation of a pixel and its neighbors. Now, to calculate the GAM of a grain, sum of all these disorientation of all pixels is needed.

So another vector `gam_total` will be used to store the sum of all disorientation all pixels of a grain. But this `gam_total` vector will also contain some unwanted zeros, (only) corresponding to the rows which are not grain IDs. To eliminate these zeros and to discard those grains whose grain size is less than the user defined value of minimum grain size, another vector `gam_final` is needed which keeps the fresh and filtered values of `gam[i]` to get the GAM of all grains. Following code snippet describes the above mentioned stepwise calculation:

```

void GAM_grain(const vector<double> &gam, const vector<double> &m, const int &DATA_SIZE,\
const int &min_grain_size, vector<double> &gam_final)
{
    vector<double> gam_total(DATA_SIZE), gam_n(DATA_SIZE);
    for (int i = 0; i < DATA_SIZE; ++i)
    {
        {
            int root_i = id[i]; //basically id[] stores the root values
            gam_total[root_i] += gam[i];
            gam_n[root_i] += m[i];
        }
    }

    for (unsigned int i = 0; i < DATA_SIZE; ++i) //final step of GAM_grain calculation!
        if (gam_n[i] != 0 && size[i] >= min_grain_size)
        {
            gam_final.push_back(gam_total[i] / (gam_n[i]));
        }
    }
}

```

**Code snippet 3.12** GAM\_Final calculation

Till now, final calculation is finished, but distribution of these calculated values in bins is still not done yet. Following code snippet contains a function which does this job (gam\_final ->values):

```

void reArrange(const vector<double> &values, const double min_, vector<double> &y_axis,\
vector<double> &x_axis, const int &bins)
{
    double max_ = *max_element(values.begin(), values.end());
    double x_range = (max_ - min_) / bins; //usually this means (max_-0)/bins;
    double l = x_range / 2.0;
    x_axis[0] = min_ + l;
    for (int i = 1; i < bins; ++i)
        x_axis[i] = x_axis[i - 1] + x_range;

    double values_size = values.size();
    x_axis[bins - 1] += 0.1; //to incorporate the last value.
    for (int i = 0; i < values_size; ++i)
        for (int j = 0; j < bins; ++j)
            if (values[i] >= (x_axis[j] - 1) && values[i] < (x_axis[j] + 1))
                y_axis[j] += 1;

    x_axis[bins - 1] -= 0.1; //to restore to the original value.
}

```

**Code snippet 3.13:** Distribution in bins

### 3.6 Grain Orientation Spread

As described in §2.8, we have used the orientation of geometric center of a grain as the average orientation of the grain. The average of scan point number of scan points of a grain represents the geometric center of the grain and hence its orientation represents the average orientation of the grain. Following snippet will show the calculation of geometric center:

```
for (int i = 0; i < DATA_SIZE; ++i)
{
    if (ci[i] >= min_ci)
    {
        sumOfIndices[id[i]] += i; //just summing the indices at their respective
root;
        count[id[i]] += 1;
    }
}

vector<int> avgOfIndices(DATA_SIZE);
int tempIndex = 0;
for (int i = 0; i < DATA_SIZE; ++i)
{
    if ((ci[i] >= min_ci) && (count[i] >= 1) && (id[i] == i)) //that means if i goes
through, i is root here; i = id[i]
    {
        tempIndex = static_cast<int>(sumOfIndices[i] / count[i]);
        avgOfIndices[i] = ((id[tempIndex] == i) ? tempIndex : i); //checking if i ==
id[i]
    }
}
//The ith scan point is the reference for that grain
```

**Code snippet 3.14:** GOS calculations

Once the reference point for each grain is determined, the average of misorientation between the reference point and every scan point of the grain is assigned as the orientation spread of that grain.

### 3.7 Grain Size Distribution

As described in §2.8, there are two methods to calculate equivalent diameter of a grain. This project uses method 2 for the calculations.

From method 2, incorporating all constants for a hexagonal scan, the equivalent diameter for a grain of size `size_[i]` comes out to be:

$$\text{Diameter} = 1.05 * \text{step\_size} * \text{sqrt}(\text{size\_}[i])$$

Following code snippet describes the calculation of grain size in a vector `size_[i]` and to eliminate the zeros and to discard the grains of size less than the minimum grain size, another vector `diameter` is needed which stores the fresh and filtered values:

```
void GSD_Calculation(const int &DATA_SIZE, const int &bins, const vector<double> &ci,\nconst int &min_grain_size, const double &step_size)\n{\n    vector<int> size_(DATA_SIZE);\n    vector<double> diameter;\n    for (int i = 0; i < DATA_SIZE; ++i)\n        if (ci[i] >= 0.1)\n            size_[i] += 1;\n\n    for (int i = 0; i < DATA_SIZE; ++i)\n        if (size_[i] >= min_grain_size && (ci[i] >= 0.1))\n            diameter.push_back(1.05*step_size*sqrt(size_[i]));\n\n    vector<double> x_axis(bins), y_axis(bins);\n    reArrange(diameter, *min_element(diameter.begin(), diameter.end()), y_axis, x_axis, bins);
```

**Code snippet 3.15:** Grain size distribution

Last line of the code snippet calls the `reArrange()` function which distributes the values in the respective bins. This is the same function which was used earlier in GAM calculations. This finishes the grain size distribution calculation.



## **Chapter 4**

### **Results**

In this chapter, the results of the designed software will be compared with the results of the EDAX OIM® Data Analysis software. The results will be in form of figures, tables and bar plots. In this comparison every page contains two plot/data, out of which 1<sup>st</sup> plot/data is obtained from EDAX OIM® Data Analysis software while the 2<sup>nd</sup> plot/data is obtained from the designed software.

## 4.1 Grain Maps

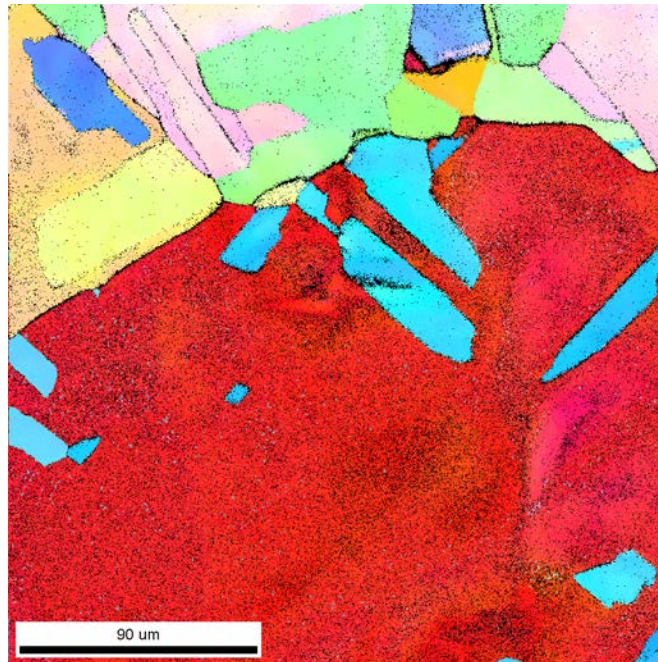


Fig. 4.1 Grain Map: 2.5% deformed Cu & annealed at 800 °C (EDAX)

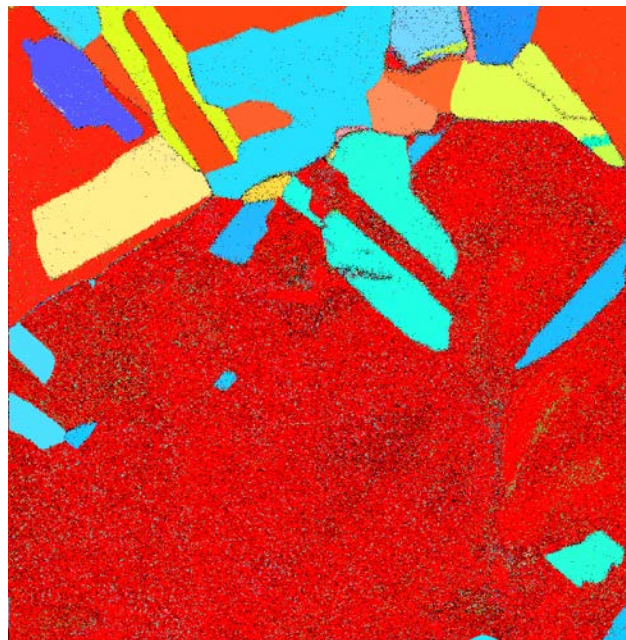


Fig. 4.1 Grain Map: 2.5% deformed Cu & annealed at 800 °C (designed software)

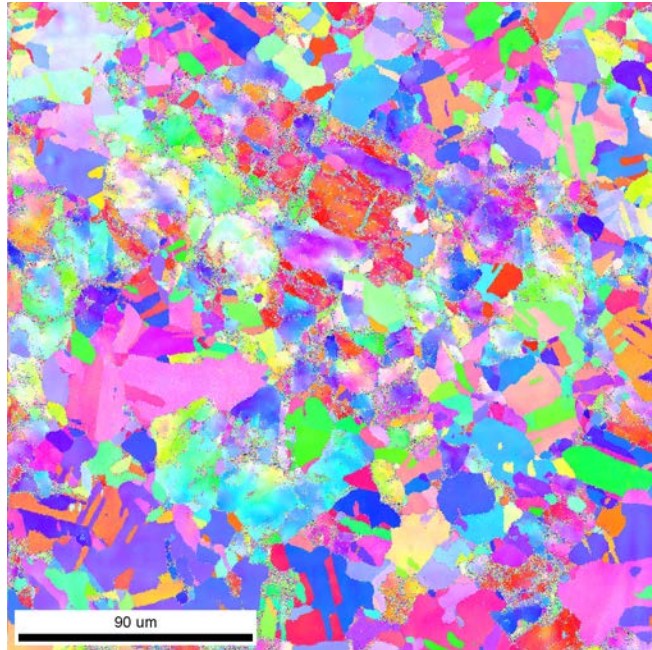


Fig. 4.2 Grain Map: 40% Deformed Cu Annealed at 180°C for 105 Min (EDAX)

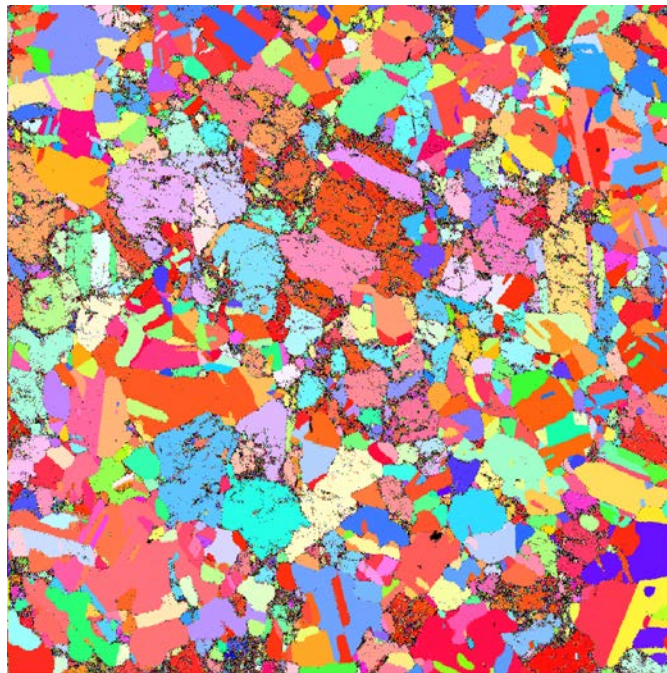


Fig. 4.2 Grain Map: 40% Deformed Cu Annealed at 180°C for 105 Min (designed software)



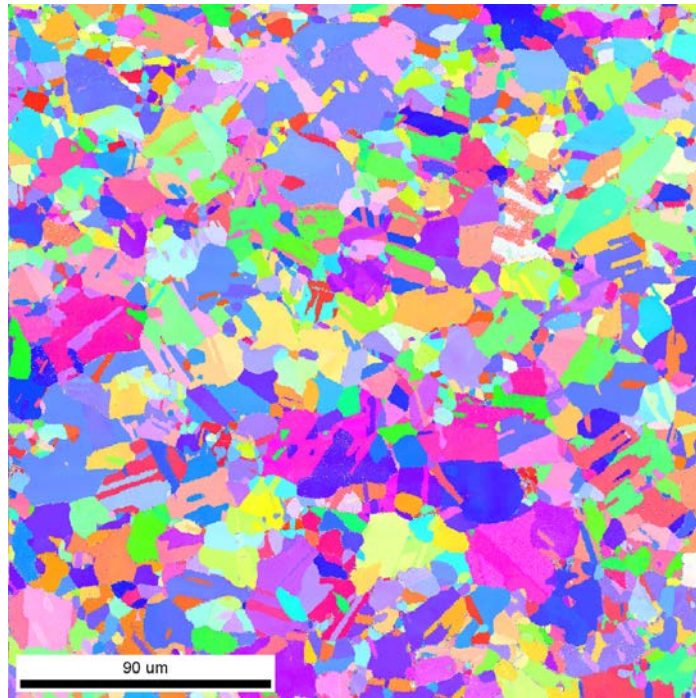


Fig. 4.3 Grain Map: 40% Deformed Cu Annealed at 180°C for 205 Min (EDAX)

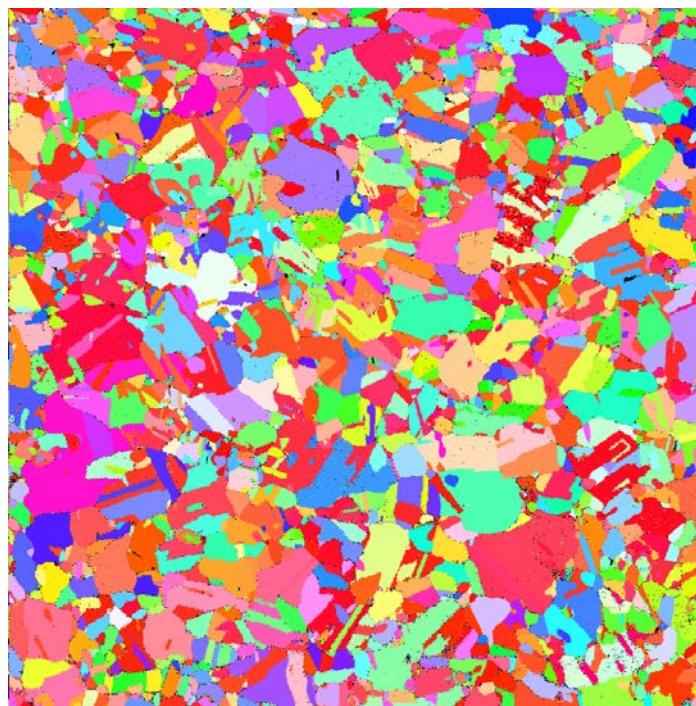


Fig. 4.3 Grain Map: 40% Deformed Cu Annealed at 180°C for 205 Min (designed software)

## 4.2 Grain Boundary Plots

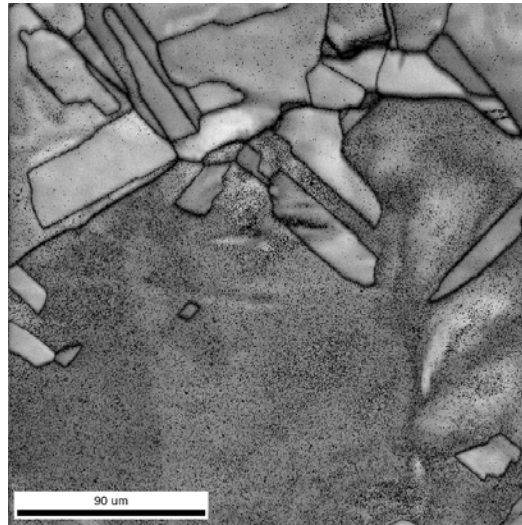


Fig. 4.4 Grain Boundaries: 2.5% deformed Cu & annealed at 800 °C (EDAX)

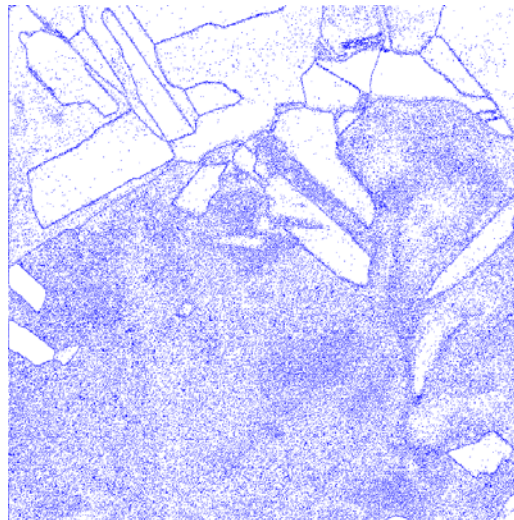


Fig. 4.4 Grain Boundaries: 2.5% deformed Cu & annealed at 800 °C (designed software)

Black pixels in EDAX data analysis software and Blue pixels in designed software represents grain boundary for tolerance = 5°

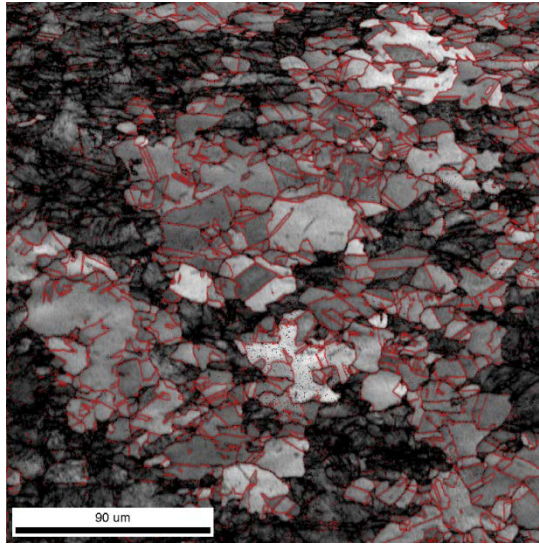


Fig. 4.5 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 90 Min (EDAX)

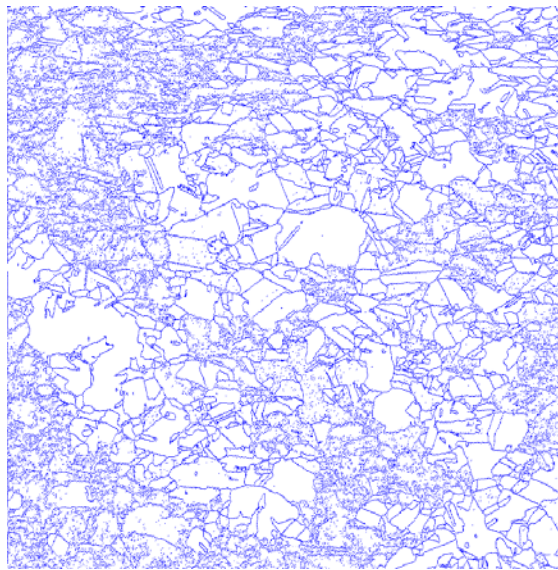


Fig. 4.5 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 90 Min (designed software)

Red pixels in EDAX data analysis software and Blue pixels in designed software represents grain boundary for tolerance = 5°



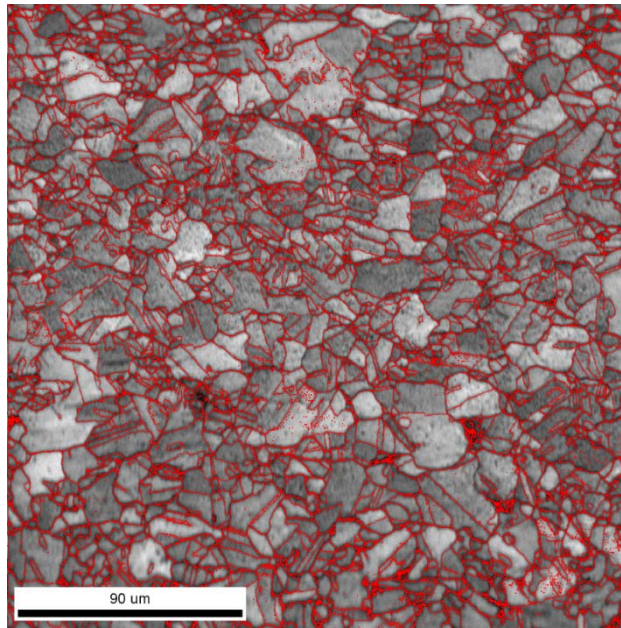


Fig. 4.6 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 205 Min (EDAX)

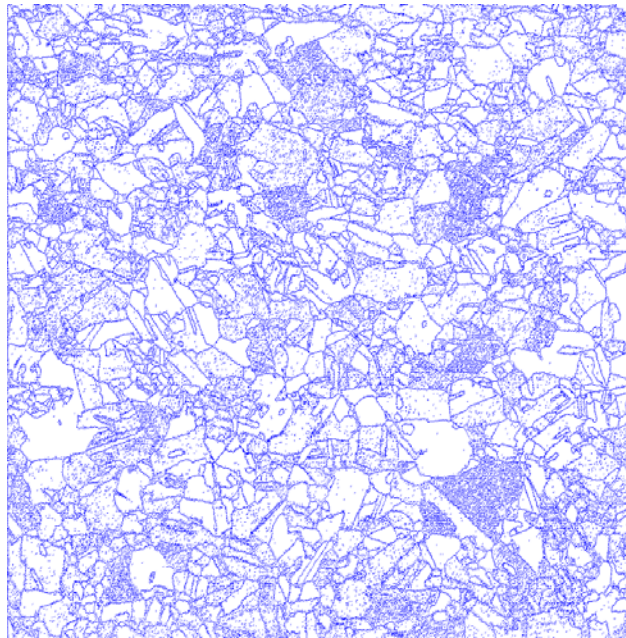


Fig. 4.6 Grain Boundaries: 40% Deformed Cu Annealed at 180°C for 205 Min (Designed software)

Red pixels in EDAX data analysis software and Blue pixels in designed software represents grain boundary for tolerance = 5°

### 4.3 Kernel Average Misorientation (KAM)

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)	$\Delta n * 100 / \text{EDAX}$
1	0.125	39281	58722	-19441	-49.49
2	0.375	348177	350532	-2355	-0.676
3	0.625	621969	620975	+994	+0.16
4	0.875	330460	329276	+1184	+0.36
5	1.125	126868	126388	+480	+0.38
6	1.375	48101	47942	+159	+0.33
7	1.625	19906	19846	+60	+0.3
8	1.875	9418	9398	+20	+0.21
9	2.125	4810	4801	+9	+0.18
10	2.375	2971	2965	+6	+0.2
11	2.625	1681	1682	-1	-0.05
12	2.875	1095	1092	+3	+0.27
13	3.125	779	779	0	0
14	3.375	566	569	-3	-0.5
15	3.625	369	369	0	0
16	3.875	291	291	0	0
17	4.125	213	214	-1	+0.46
18	4.375	183	182	+1	+0.54
19	4.625	152	152	+0	0
20	4.875	18169	114	+18055	+99.37

Table 4.1 KAM of stainless steel

Details:

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.1 (configured)



Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)	$\Delta n * 100 / \text{EDAX}$
1	0.125	2626	69197	-66571	-2535
2	0.375	45209	45494	-285	-0.63
3	0.625	106928	107043	-115	-0.1
4	0.875	112503	112435	+68	+0.06
5	1.125	106416	106243	+173	+0.16
6	1.375	95581	95429	+152	+0.16
7	1.625	78619	78459	+160	+0.2
8	1.875	59273	59058	+215	+0.36
9	2.125	42189	42092	+97	+0.23
10	2.375	29406	29282	+124	+0.42
11	2.625	19962	19915	+47	+0.23
12	2.875	13663	13591	+72	+0.52
13	3.125	9045	9001	+44	+0.48
14	3.375	5766	5745	+21	+0.36
15	3.625	3663	3639	+24	+0.66
16	3.875	2294	2284	+10	+0.43
17	4.125	1354	1337	+17	1.25
18	4.375	833	838	-5	-0.6
19	4.625	472	466	+6	+1.27
20	4.875	66742	279	+66463	+99.58

Table 4.2 KAM of 2.5% deformed Cu and annealed at 800 °C

In this case, the EDAX data analysis software included all pixels (even those having confidence index less than 0.1).

Details:

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)	$\Delta n * 100 / \text{EDAX}$
1	0.125	3247	107429	-104182	-3208
2	0.375	122318	123022	-4	-0.003
3	0.625	187955	187742	213	+0.11
4	0.875	131433	131086	+347	+0.26
5	1.125	84332	84087	+245	+0.29
6	1.375	54572	54381	+191	+0.35
7	1.625	36247	36091	+156	+0.43
8	1.875	24491	24467	+24	+0.09
9	2.125	16286	16190	+96	+0.59
10	2.375	11218	11173	+45	+0.4
11	2.625	7790	7747	+43	+0.55
12	2.875	5223	5205	+18	+0.34
13	3.125	3748	3741	+7	+0.18
14	3.375	2643	2628	+15	+0.57
15	3.625	2033	2029	+4	+0.19
16	3.875	1437	1430	+7	+0.48
17	4.125	1166	1168	-2	-0.17
18	4.375	899	899	0	0
19	4.625	740	737	+3	+0.4
20	4.875	103626	575	+103051	+99.44

Table 4.3 KAM of 40% Deformed Cu Annealed at 180°C for 90 Min

In this case, the EDAX data analysis software included all pixels (even those having confidence index less than 0.1).

Details:

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)	$\Delta n * 100 / \text{EDAX}$
1	0.125	1840	112994	-111154	-6040.97
2	0.375	75404	75904	-500	-0.66
3	0.625	155572	155568	+4	+0.03
4	0.875	133486	133214	+272	+0.2
5	1.125	95982	95763	+289	+0.3
6	1.375	67207	67036	+171	+0.25
7	1.625	47383	47258	+125	+0.26
8	1.875	33095	32991	+104	+0.31
9	2.125	23159	23084	+75	+0.32
10	2.375	16330	16293	+37	+0.22
11	2.625	11937	11877	+60	+0.5
12	2.875	8612	8895	-283	-3.2
13	3.125	6092	6085	+7	+0.11
14	3.375	4516	4487	+29	+0.64
15	3.625	3255	3232	+23	+0.7
16	3.875	2329	2344	-15	-0.64
17	4.125	1819	1808	+11	+0.6
18	4.375	1389	1389	0	0
19	4.625	1131	1114	+17	+1.5
20	4.875	111060	791	+110269	+988

Table 4.4 KAM of 40% Deformed Cu Annealed at 180°C for 105 Min

In this case, the EDAX data analysis software included all pixels (even those having confidence index less than 0.1).

Details:

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)	$\Delta n * 100 / \text{EDAX}$
1	0.125	6848	27252	-20404	-297.96
2	0.375	188933	189669	-736	-0.38
3	0.625	264249	263605	+644	+0.24
4	0.875	152978	152467	+511	+0.33
5	1.125	76081	75774	-307	-0.4
6	1.375	39646	39526	+120	+0.3
7	1.625	21342	21293	+49	+0.23
8	1.875	11811	11783	+28	+0.23
9	2.125	7365	7339	+26	+0.35
10	2.375	4384	4374	+10	+0.23
11	2.625	2814	2805	+9	+0.32
12	2.875	1726	1724	+2	+0.12
13	3.125	1156	1148	+8	+0.69
14	3.375	818	819	-1	-0.12
15	3.625	666	660	+6	+0.9
16	3.875	488	491	-2	-0.4
17	4.125	366	363	+3	+0.81
18	4.375	296	298	+1	+0.34
19	4.625	279	279	0	0
20	4.875	20366	158	+20208	+99.24

Table 4.5: KAM of 40% Deformed Cu Annealed at 180°C for 205 Min

In this case, the EDAX data analysis software included all pixels (even those having confidence index less than 0.1).

Details:

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

#### 4.4 Grain Average Misorientation (GAM)

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.10336	651	650	+1
2	0.310081	180	179	+1
3	0.516802	1063	1070	-7
4	0.723522	1535	1525	+8
5	0.930243	326	321	+5
6	1.13696	84	85	-1
7	1.34368	52	49	+3
8	1.5504	18	18	0
9	1.75713	8	9	-1
10	1.96385	5	4	+1
11	2.17057	3	3	0
12	2.37729	1	1	0
13	2.58401	1	1	0
14	2.79073	0	0	0
15	2.99745	0	0	0
16	3.20417	0	0	0
17	3.41089	0	0	0
18	3.61761	0	0	0
19	3.82433	0	0	0
20	4.03105	2	2	0

Table 4.6 GAM of stainless steel

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.1

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.146142	776	778	-2
2	0.438426	1874	1876	-2
3	0.73071	2563	2561	+2
4	1.02299	2508	2510	-2
5	1.31528	1656	1657	-1
6	1.60756	1140	1139	+1
7	1.89985	801	802	-1
8	2.19213	534	533	+1
9	2.48441	402	401	+1
10	2.7767	292	292	0
11	3.06898	213	213	0
12	3.36126	123	123	0
13	3.65355	88	89	0
14	3.94583	73	73	0
15	4.23812	54	54	0
16	4.5304	22	22	0
17	4.82268	29	29	0
18	5.11497	6	6	0
19	5.40725	0	0	0
20	5.69954	1	1	0

Table 4.7 GAM of 2.5% deformed Cu & annealed at 800 °C

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.14024	39	39	0
2	0.420719	255	257	-2
3	0.701198	867	866	+1
4	0.981677	864	868	-4
5	1.26216	838	830	+8
6	1.54264	751	749	+2
7	1.82311	626	625	+1
8	2.10359	515	512	+3
9	2.38407	406	404	+2
10	2.66455	285	284	+1
11	2.94503	277	274	+3
12	3.22551	249	248	+1
13	3.50599	181	184	-3
14	3.78647	178	174	+4
15	4.06695	145	147	-2
16	4.34743	141	139	+2
17	4.62791	122	121	+1
18	4.90838	87	88	-1
19	5.18886	1	1	0
20	5.46934	4	4	0

Table 4.8 GAM of 40% Deformed Cu Annealed at 180°C for 90 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.138417	57	57	0
2	0.415251	190	192	-2
3	0.692085	672	668	+4
4	0.968919	896	892	+4
5	1.24575	802	806	-4
6	1.52259	844	845	-1
7	1.79942	804	799	+5
8	2.07626	630	632	-2
9	2.35309	512	511	+1
10	2.62992	449	447	+2
11	2.90676	345	344	+1
12	3.18359	308	310	-2
13	3.46043	283	280	+3
14	3.73726	249	249	0
15	4.01409	184	186	-2
16	4.29093	179	180	-1
17	4.56776	185	185	-0
18	4.8446	132	130	-2
19	5.12143	11	12	-1
20	5.39826	3	3	0

Table 4.9 GAM of 40% Deformed Cu Annealed at 180°C for 105 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0



Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.123836	47	48	-1
2	0.371509	203	204	-1
3	0.619182	1132	1130	+2
4	0.866854	1244	1240	+4
5	1.11453	727	725	+2
6	1.3622	381	381	0
7	1.60987	227	227	0
8	1.85755	120	119	+1
9	2.10522	96	96	0
10	2.35289	54	54	0
11	2.60056	48	47	+1
12	2.84824	37	37	0
13	3.09591	23	22	+1
14	3.34358	30	30	0
15	3.59125	18	18	0
16	3.83893	15	14	+1
17	4.0866	12	13	-1
18	4.33427	10	10	0
19	4.58194	20	20	0
20	4.82962	6	6	0

Table 4.10 GAM of 40% Deformed Cu Annealed at 180°C for 205 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

## 4.5 Grain Orientation Spread

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.285692	1123	1038	+85
2	0.857076	2797	2253	+544
3	1.42846	1804	1888	-84
4	1.99984	1244	1385	-141
5	2.57123	465	558	-93
6	3.14261	103	159	-56
7	3.714	56	109	-53
8	4.28538	32	84	-52
9	4.85677	35	43	-8
10	5.42815	28	50	-22
11	5.99953	15	26	-11
12	6.57092	10	21	-11
13	7.1423	5	25	-20
14	7.71369	4	22	-18
15	8.28507	3	17	-14
16	8.85646	4	10	-6
17	9.42784	2	9	-7
18	9.99922	1	4	-3
19	10.5706	2	7	-5
20	11.142	2	40	-38

Table 4.11 GOS of 40% Deformed Cu Annealed at 180°C for 105 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

**Note:** The binning provided here is not the same as in the output file of the designed application. Here, the data has been manually put into same bins to compare the results.

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.08041	34	33	+1
2	0.24123	166	158	+8
3	0.40205	284	235	+49
4	0.56287	264	252	+12
5	0.72369	187	165	+22
6	0.88451	142	158	-16
7	1.04533	104	92	+12
8	1.20615	63	70	-7
9	1.36697	39	55	-16
10	1.52779	25	26	-1
11	1.68861	12	37	-25
12	1.84943	10	13	-3
13	2.01025	4	15	-11
14	2.17107	2	9	-7
15	2.33189	4	6	-2
16	2.49271	1	1	0
17	2.65353	1	5	-4
18	2.81435	1	2	-1
19	2.97517	1	1	-0
20	3.13599	1	20	-19

Table 4.12 GOS of 2.5% deformed Cu & annealed at 800 °C

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

**Note:** The binning provided here is not the same as in the output file of the designed application. Here, the data has been manually put into same bins to compare the results.

Bin no	Average Angle [Degree]	Number (EDAX)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	0.0874962	86	87	-1
2	0.262489	304	282	+22
3	0.437481	854	429	+425
4	0.612473	1313	672	+641
5	0.787466	907	723	+184
6	0.962458	415	635	-220
7	1.13745	206	484	-278
8	1.31244	125	323	-198
9	1.48744	61	253	-192
10	1.66243	53	152	-99
11	1.83742	33	98	-65
12	2.01241	35	100	-65
13	2.18741	24	53	-29
14	2.3624	20	39	-19
15	2.53739	2	22	-20
16	2.71238	5	20	-15
17	2.88737	1	15	-14
18	3.06237	0	13	-13
19	3.23736	5	10	-5
20	3.41235	1	57	-56

Table 4.13 GOS of 40% Deformed Cu Annealed at 180°C for 205 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

**Note:** The binning provided here is not the same as in the output file of the designed application. Here, the data has been manually put into same bins to compare the results.

## 4.6 Grain Size Distribution (GSD)

Bin no	Average Angle	Number (EDAX)	Number (designed soft using grain id)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	1.96051	1802	1802	1790	+18
2	4.3965	497	497	497	0
3	6.83249	363	363	363	0
4	9.26848	335	335	335	0
5	11.7045	263	263	263	0
6	14.1405	195	195	195	0
7	16.5765	128	128	128	0
8	19.0124	109	109	109	0
9	21.4484	59	59	59	0
10	23.8844	61	61	61	0
11	26.3204	44	44	44	0
12	28.7564	23	23	23	0
13	31.1924	18	18	18	0
14	33.6284	12	12	12	0
15	36.0644	6	6	6	0
16	38.5004	3	3	3	0
17	40.9364	4	4	4	0
18	43.3723	4	4	4	0
19	45.8083	1	1	1	0
20	48.2443	2	2	2	0

Table 4.14 GSD of stainless steel

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.1

Bin no	Diameter [microns]	Number (EDAX)	Number (designed soft using grain id)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	5.58145	13129	13114	13116	+13
2	15.8533	9	9	9	0
3	26.1252	7	7	7	0
4	36.3971	5	5	5	0
5	46.6689	2	2	2	0
6	56.9408	1	1	1	0
7	67.2127	1	1	1	0
8	77.4846	0	0	0	0
9	87.7564	0	0	0	0
10	98.0283	0	0	0	0
11	108.3	0	0	0	0
12	118.572	0	0	0	0
13	128.844	0	0	0	0
14	139.116	0	0	0	0
15	149.388	0	0	0	0
16	159.66	0	0	0	0
17	169.931	0	0	0	0
18	180.203	0	0	0	0
19	190.475	0	0	0	0
20	200.747	1	1	1	0

Table 4.15 GSD of 2.5% deformed Cu & annealed at 800 °C

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

Bin no	Diameter [microns]	Number (EDAX)	Number (designed soft using grain id)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	1.53965	5752	5736	5737	+15
2	3.72792	509	508	508	+1
3	5.91619	239	239	239	0
4	8.10447	140	139	139	+1
5	10.2927	83	83	83	0
6	12.481	36	37	37	-1
7	14.6693	26	28	25	+1
8	16.8576	16	16	16	0
9	19.0458	11	11	11	0
10	21.2341	9	9	9	0
11	23.4224	6	6	6	0
12	25.6107	0	0	0	0
13	27.7989	0	0	0	0
14	29.9872	1	1	1	0
15	32.1755	0	0	0	0
16	34.3638	2	2	2	0
17	36.552	0	0	0	0
18	38.7403	0	0	0	0
19	40.9286	0	0	0	0
20	43.1169	0	1	1	-1

Table 4.16 GSD of 40% Deformed Cu Annealed at 180°C for 90 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0

<b>Bin no</b>	<b>Diameter [microns]</b>	<b>Number (EDAX)</b>	<b>Number (designed soft using grain id)</b>	<b>Number (designed Soft)</b>	<b><math>\Delta n</math> (EDAX-Designed software)</b>
<b>1</b>	1.28018	6614	6601	6607	+7
<b>2</b>	2.94952	413	414	414	-1
<b>3</b>	4.61886	254	253	253	+1
<b>4</b>	6.2882	144	145	145	-1
<b>5</b>	7.95754	112	111	111	+1
<b>6</b>	9.62688	59	59	59	0
<b>7</b>	11.2962	40	40	40	0
<b>8</b>	12.9656	25	25	25	0
<b>9</b>	14.6349	21	21	21	0
<b>10</b>	16.3042	14	14	14	0
<b>11</b>	17.9736	9	9	9	0
<b>12</b>	19.6429	7	7	7	0
<b>13</b>	21.3123	3	3	3	0
<b>14</b>	22.9816	7	7	7	0
<b>15</b>	24.6509	1	1	1	0
<b>16</b>	26.3203	4	4	4	0
<b>17</b>	27.9896	5	5	5	0
<b>18</b>	29.6589	0	0	0	0
<b>19</b>	31.3283	1	1	1	0
<b>20</b>	32.9976	1	2	2	-1

Table 4.17 GSD of 40% Deformed Cu Annealed at 180°C for 105 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0



Bin no	Diameter [microns]	Number (EDAX)	Number (designed soft using grain id)	Number (designed Soft)	$\Delta n$ (EDAX-Designed software)
1	1.1771	2613	2610	2602	+11
2	2.64029	707	707	708	-1
3	4.10347	409	410	410	-1
4	5.56666	243	243	243	0
5	7.02984	158	158	158	0
6	8.49302	105	104	106	-1
7	9.95621	57	58	58	-1
8	11.4194	50	49	48	+2
9	12.8826	36	36	36	0
10	14.3458	25	25	25	0
11	15.8089	11	11	11	0
12	17.2721	9	9	9	0
13	18.7353	11	11	11	0
14	20.1985	5	5	5	0
15	21.6617	3	3	3	0
16	23.1249	2	2	2	0
17	24.5881	0	0	0	0
18	26.0512	2	2	2	0
19	27.5144	2	2	2	0
20	28.9776	2	2	2	0

Table 4.18 GSD of 40% Deformed Cu Annealed at 180°C for 205 Min

Details:

Minimum grain size considered = 2

Misorientation for grain boundary = 5°

Minimum Confidence index = 0.0



## Chapter 5

### Conclusion

We conclude here the performance of the designed software with respect to that of EDAX OIM<sup>®</sup> Data Analysis software. Results of both software programs have been compared in the previous section. On the basis of the calculations and results, following are the concluding points:

1. We have designed a software which can be used on Windows based computers to analyze the EBSD data for cubic systems.
2. Our software can identify grain boundaries, plot grain maps, and calculate GOS, GSD, GAM and KAM.
3. The results generated from the software are generally in agreement with what is calculated by the EBSD software. The differences and the possible reasons are as follows:

**Grain Map:** Colors assigned to some of the grains are different from those by EDAX OIM<sup>®</sup> Data Analysis software. There are different ways of assigning colors to the scan points. The commercial software doesn't document its method of assigning color.

**Grain Orientation Spread (GOS):** As mentioned in §3.6, EDAX OIM<sup>®</sup> Data Analysis software's documentation chooses *average orientation* of a grain as the reference point, but it has never

said anything about the calculation of *average orientation*. So, instead of average orientation, we have chosen the orientation of scan point at geometric center as the reference.

**Kernel Average Misorientation (KAM):** The results of the designed software and EDAX OIM® Data Analysis software are in excellent match (within a range of 1%) except at the two extremes (minima and maxima) of the average angle but how EDAX OIM® Data Analysis software handles the points at these extremes is not documented.

As shown in the results section, the results of Grain Boundary, GAM and GSD are in excellent match with that of the EDAX OIM® Data Analysis software.

## **Chapter 6**

### **Future Work**

Although we have tried to make the software as comprehensive as possible, it has certain limitations. Based on the results and limitations, following are the possible areas of improvement:

1. As mentioned earlier, the designed software works only for the materials having cubic crystalline structure because in grain boundary calculations we have used cubic symmetry. Similarly, it can be programmed to do computations for HCP crystalline structures too.
2. The results of Grain map, GOS and KAM are not in perfect match with that of the EDAX OIM® Data Analysis software due to lack of documentation. It requires further work.
3. The input file for this software is an output of EDAX OIM® Data Analysis software, we are still dependent on a commercial software. It will be a great improvement if the raw data of EBSD scan is decoded which will make this designed open source software completely independent of the commercial software.

## References

1. Olaf Engler and Valerie Randle, "Introduction to Texture Analysis", 2<sup>nd</sup> edition, 2009, CRC Press, §2.3.1 Coordinate systems pp. 20-22.
2. D.E. Bourne and P.C. Kendall, "Vector Analysis and Cartesian Coordinates", 3<sup>rd</sup> edition, 1992, CRC Press, §1.5 Rotation of Axes, pp. 9-14
3. Weisstein, Eric W. "Euler Angles." From MathWorld--A Wolfram Web Resource. <http://mathworld.wolfram.com/EulerAngles.html>
4. Herbert Goldstein, "Classical Mechanics", 2<sup>nd</sup> edition, 1980, Addison-Wesley, §4.4 The Euler Angles, pp. 143-147
5. James Verth and Larsh Bishop, "Essential mathematics for games and interactive applications", 2<sup>nd</sup> edition, 2008, CRC Press, §5.4 Angle-Axis representation, pp. 181-185
6. Olaf Engler and Valerie Randle, "Introduction to Texture Analysis", 2<sup>nd</sup> edition, 2009, CRC Press, §2.7.1 Angle/Axis of rotation, pp. 39-42
7. Katja Jöchen, "Homogenization of the Linear and Non-linear Mechanical Behavior of Polycrystals", 2013, KIT Scientific Publishing, §2.3.2 Crystallographically equivalent orientations
8. Adam J. Schwartz, Mukul Kumar, Brent L. Adams, David P. Field "Electron Backscatter Diffraction in Materials Science", 2<sup>nd</sup> Edition, 2013, Springer Science & Business Media, §18.2.1 Mapping Orientations and Misorientations pp. 253-255
9. Stuart I. Wright, Matthew M. Nowell, and David P. Field, "A Review of Strain Analysis Using Electron Backscatter Diffraction", Microscopy Society of America 2011, Microscopy and Microanalysis / Volume 17 / Issue 03 / May 2011, pp. 316-329,
10. Paul Macklin, "EasyBMP library", version 1.06, 2006, <http://easybmp.sourceforge.net/>
11. Arvo 1994, Hearn and Baker 1996
12. Herbert Goldstein, "Classical Mechanics", 2<sup>nd</sup> edition, 1980, Addison-Wesley, §4.4 The Euler Angles, pp. 155; and Weisstein, Eric W. "Euler Parameters." From MathWorld--A Wolfram Web Resource. <http://mathworld.wolfram.com/EulerParameters.html>
13. Adam J. Schwartz, Mukul Kumar, Brent L. Adams, David P. Field "Electron Backscatter Diffraction in Materials Science", 2<sup>nd</sup> Edition, 2013, Springer, §3.3 Pole Figures pp. 38-40
14. Claus Gramkov, "On Averaging Rotations", TriVision Ltd., Billedskaerervej 19, DK-5230 Odense M, Denmark

# Appendix

## A1: Matrix and Quaternion representation of Symmetry operations

There are 24 symmetrically equivalent orientations for a cubic symmetry, following are the matrix representations.

$$\begin{array}{ccccc}
 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} & \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \\
 \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} & \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \\
 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix} \\
 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} & 
 \end{array}$$

Similarly, corresponding to 24 symmetrically equivalent orientations for a cubic systems, following are the 24 Quaternions:

$$\begin{array}{l}
 (e_1, e_2, e_3, e_4) \\
 (e_4, e_3, -e_2, -e_1) \\
 (-e_3, e_4, e_1, -e_2) \\
 (e_2, -e_1, e_4, -e_3) \\
 ((e_1 + e_2 + e_4 - e_3), (e_3 + e_2 + e_4 - e_1), (e_1 + e_3 + e_4 - e_2), (e_4 - e_1 - e_2 - e_3)) / 2 \\
 ((e_1 - e_2 - e_4 + e_3), (e_2 - e_3 - e_4 + e_1), (e_3 - e_1 - e_4 + e_2), (e_4 + e_2 + e_1 + e_3)) / 2 \\
 ((e_1 + e_2 - e_4 - e_3), (e_2 - e_3 + e_4 - e_1), (e_3 + e_1 + e_4 + e_2), (e_4 - e_2 + e_1 - e_3)) / 2 \\
 ((e_1 - e_2 + e_4 + e_3), (e_2 + e_3 - e_4 + e_1), (e_3 - e_1 - e_4 - e_2), (e_4 + e_2 - e_1 + e_3)) / 2 \\
 ((e_1 + e_2 + e_4 + e_3), (e_2 + e_3 - e_4 - e_1), (e_3 - e_1 + e_4 - e_2), (e_4 + e_2 - e_1 - e_3)) / 2
 \end{array}$$

$$\begin{aligned}
& ((e_1 - e_2 - e_4 - e_3), (e_2 - e_3 + e_4 + e_1), (e_3 + e_1 - e_4 + e_2), (e_4 - e_2 + e_1 + e_3)) / 2 \\
& ((e_1 - e_2 + e_4 - e_3), (e_2 + e_3 + e_4 + e_1), (e_3 + e_1 - e_4 - e_2), (e_4 - e_2 - e_1 + e_3)) / 2 \\
& ((e_1 + e_2 - e_4 + e_3), (e_2 - e_3 - e_4 - e_1), (e_3 - e_1 + e_4 + e_2), (e_4 + e_2 + e_1 - e_3)) / 2 \\
& ((e_2 - e_3), (e_4 - e_1), (e_4 + e_1), (-e_2 - e_3)) / \sqrt{2} \\
& ((e_4 + e_2), (e_3 - e_1), (e_4 - e_2), (-e_1 - e_3)) / \sqrt{2} \\
& ((e_4 - e_3), (e_4 + e_3), (e_1 - e_2), (-e_2 - e_1)) / \sqrt{2} \\
& ((e_2 - e_3), (e_4 + e_1), (-e_4 + e_1), (-e_2 + e_3)) / \sqrt{2} \\
& ((e_4 - e_2), (e_3 + e_1), (-e_4 - e_2), (-e_1 + e_3)) / \sqrt{2} \\
& ((e_4 + e_3), (-e_4 + e_3), (-e_1 - e_2), (-e_1 + e_2)) \sqrt{2} \\
& ((e_1 + e_4), (e_2 + e_3), (e_3 - e_2), (e_4 - e_1)) / \sqrt{2} \\
& ((e_1 - e_3), (e_4 + e_2), (e_3 + e_1), (e_4 - e_2)) / \sqrt{2} \\
& ((e_1 + e_2), (e_2 - e_1), (e_4 + e_3), (e_4 - e_3)) / \sqrt{2} \\
& ((e_1 - e_4), (e_2 - e_3), (e_3 + e_2), (e_4 + e_1)) / \sqrt{2} \\
& ((e_1 + e_3), (e_2 - e_4), (e_3 - e_1), (e_2 + e_4)) / \sqrt{2} \\
& ((e_1 - e_2), (e_1 + e_2), (e_3 - e_4), (e_4 + e_3)) / \sqrt{2}
\end{aligned}$$



# **User Manual (Version 1.00)**

## Abstract

We define and document a simple, efficient and comprehensive software to analyze crystalline orientations through the analysis of Electron Backscatter Diffraction (EBSD) data from a Scanning Electron Microscope. The input file for this software is an output file by *EDAX OIM™ Data Analysis software* although, the designed software uses very elementary information like crystallographic orientations of scan points and their position coordinates for its analysis. From these information, this software determines and plot Grain Map and Grain boundaries and also calculates Kernel average misorientation, Grain average misorientation, Grain Orientation Spread and Grain sizes, distributes them in bins and uses bar plots to display the data. This software is designed to work on EBSD scans of materials having FCC and BCC crystalline structure in hexagonal grids.

## **1. Introduction to the EBSD Data Processing Software**

The application of Scanning Electron Microscopy in materials science is increasing day by day, so is the need of indexing programs which process these data. There are several other programs which do the same (or even more) calculations. However as we looked about, we noticed these programs have one or more of the following properties:

1. Too expensive and hence limited availability (and we don't even use their all features)
2. Some sort of Black box (no one knows how the raw data is processed)
3. Limited customizability
4. Require extensive installation

This designed software is intended to solve the aforementioned problems. This software is written in C++11 for windows (7 and 8) operating systems and is assisted by another light weight user interface written in Visual Basic. This program will be shared under GNU General Public License, so we are delighted to welcome you to use it for free and/or update it!

### **Following are the requirements of this software:**

Operating systems: Windows 7 and Windows 8 (and 8.1)

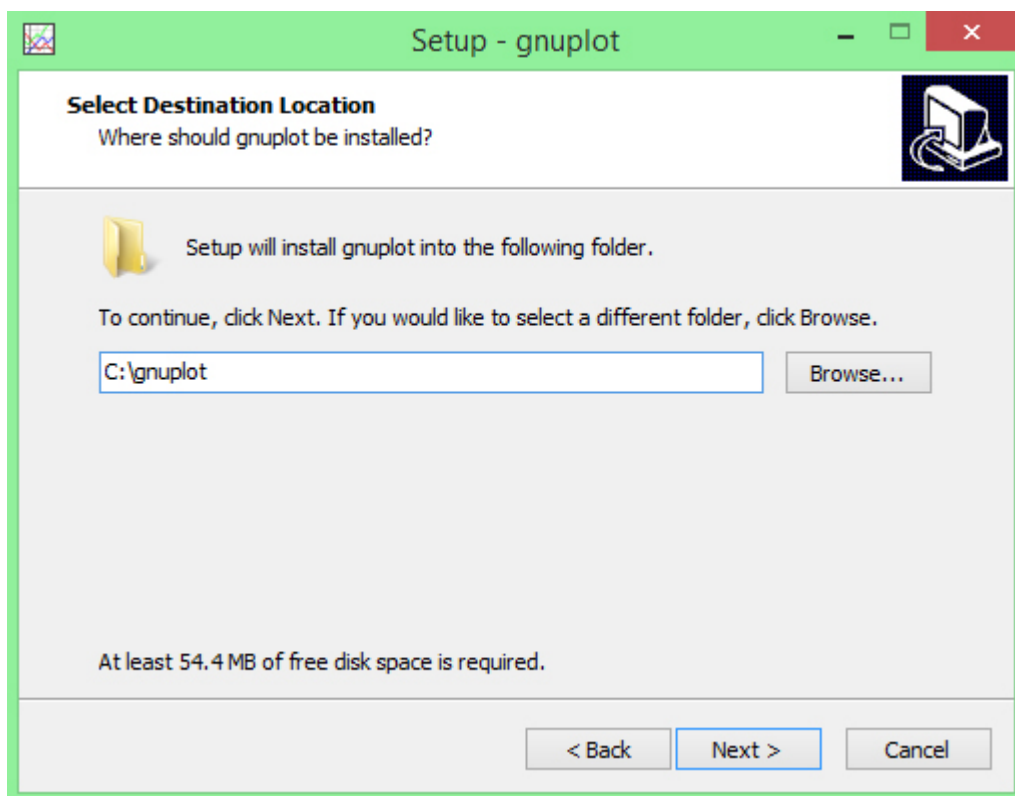
RAM: 512 MB (recommended 2 GB)

.NET framework version: 4.5 or later

## 2. Installation of the software program

The application doesn't need installation! It just works from the directory it is kept in. This application uses *gnuplot* for plotting. Following are the instructions to install gnuplot:

1. Go to the URL: <http://www.gnuplot.info/download.html> or google "gnuplot download" if the link is dead.
2. Look for the download links and then download the latest version (while writing this, "version 5.0 patchlevel 1" is the latest one).
3. There is just one modification: While choosing installation directory, chose C:\gnuplot like the following figure shows:



4. Then onwards, go on with the default options and finish installation of *gnuplot*.

**Note:** If the designed software asks for updated .NET Framework (4.5 or later), google "latest .net framework" and follow the very first link (on Microsoft download center) to download and install the latest version.

### 3. Processing EBSD Data using software

#### Step 1: Edit the input file

The input to this software is an output of the *EDAX OIM™ Data Analysis software* although, the designed software uses very elementary information. Following is an example of what the input file of this software looks like:

```
1 # Header: Project1::40%-180°C-105Min::All data::Grain Size 6/10/2013
2 #
3 # Column 1-3: phi1, PHI, phi2 (orientation of point in radians)
4 # Column 4-5: x, y (coordinates of point in microns)
5 # Column 6: IQ (image quality)
6 # Column 7: CI (confidence index)
7 # Column 8: Fit (degrees)
8 # Column 9: Grain ID (integer)
9 # Column 10: edge (1 for grains at edges of scan and 0 for interior grains)
10 # Column 11: phase name
11 2.96175 1.06281 2.83889 0.00000 0.00000 857.4 0.029 1.72 1 1 Copper
12 6.04100 2.11220 3.38375 0.30000 0.00000 843.2 0.029 2.31 1 1 Copper
13 5.31542 0.73758 4.84646 0.60000 0.00000 382.7 0.000 2.20 2 1 Copper
14 0.62504 2.69853 5.58822 0.90000 0.00000 361.8 0.000 2.01 3 1 Copper
15 5.31845 2.90026 2.67719 1.20000 0.00000 401.1 0.086 2.18 4 1 Copper
16 5.81741 1.38232 1.45742 1.50000 0.00000 356.4 0.029 1.87 4 1 Copper
17 1.96017 1.25529 5.03742 1.80000 0.00000 321.3 0.029 2.20 5 1 Copper
18 2.39901 1.91242 5.16047 2.10000 0.00000 372.8 0.000 2.61 6 1 Copper
19 5.40580 2.87909 5.10552 2.40000 0.00000 363.7 0.000 2.17 7 1 Copper
20 1.47507 1.95012 0.02884 2.70000 0.00000 379.0 0.000 1.93 8 1 Copper
21 5.40499 1.43006 4.71126 3.00000 0.00000 333.0 0.029 1.99 9 1 Copper
22 2.89655 2.38393 0.98948 3.30000 0.00000 303.4 0.000 2.24 10 1 Copper
23 4.18886 1.31854 2.00068 3.60000 0.00000 356.2 0.086 2.17 11 1 Copper
```

All we need is to remove the first few lines (precisely 10 here) of file which gives the details of the project. So the start of the file should be from the lines where data starts like following:

```
1 2.96175 1.06281 2.83889 0.00000 0.00000 857.4 0.029 1.72 1 1 Copper
2 6.04100 2.11220 3.38375 0.30000 0.00000 843.2 0.029 2.31 1 1 Copper
3 5.31542 0.73758 4.84646 0.60000 0.00000 382.7 0.000 2.20 2 1 Copper
4 0.62504 2.69853 5.58822 0.90000 0.00000 361.8 0.000 2.01 3 1 Copper
5 5.31845 2.90026 2.67719 1.20000 0.00000 401.1 0.086 2.18 4 1 Copper
6 5.81741 1.38232 1.45742 1.50000 0.00000 356.4 0.029 1.87 4 1 Copper
7 1.96017 1.25529 5.03742 1.80000 0.00000 321.3 0.029 2.20 5 1 Copper
8 2.39901 1.91242 5.16047 2.10000 0.00000 372.8 0.000 2.61 6 1 Copper
9 5.40580 2.87909 5.10552 2.40000 0.00000 363.7 0.000 2.17 7 1 Copper
10 1.47507 1.95012 0.02884 2.70000 0.00000 379.0 0.000 1.93 8 1 Copper
11 5.40499 1.43006 4.71126 3.00000 0.00000 333.0 0.029 1.99 9 1 Copper
12 2.89655 2.38393 0.98948 3.30000 0.00000 303.4 0.000 2.24 10 1 Copper
13 4.18886 1.31854 2.00068 3.60000 0.00000 356.2 0.086 2.17 11 1 Copper
```

The left most column here is the line number and is not a part of the data file. These line numbers also indicate that the start of the file is now different from that of the previous file.

After editing the input file, all steps are pretty straight forward:

### Step 2: Run “RunMe.exe” file

“RunMe.exe” file provides the interface to access the source file, take user response and run the main application file.

### Step 3: Select the source file

This program doesn’t ask user to put the source file in any specific folder. Click on the browse button to browse the file. Following image shows the browse button:



### Step 4: Select the options

From the next page, select the options you want to calculate/plot. Of course, you can select multiple options. Following image shows that user wants to calculate Kernel Average Misorientation:

- ☐ 1. Grain Map
- ☐ 2. Grain Boundaries
- ☒ 3. Kernel Average Misorientation
- ☐ 4. Grain Average Misorientation
- ☐ 5. Grain Orientation spread
- ☐ 6. Grain Size Distribution

### Step 5: Provide parameters

The next page asks you to provide parameters for calculations/plots. Notice that the boxes already contain some default/standard values. If some of these boxes are non-editable, it’s because they are not needed. Following image shows the page:

Minimum confidence index: .....	<input type="text" value="0.1"/>
Misorientation for grain boundary: .....	<input type="text" value="5"/>
For GAM, GOS or GSD, the minimum grains size that should be considered: .....	<input type="text" value="2"/>
Number of bins: .....	<input type="text" value="20"/>

☐ Check to go for advance binning options

Notice that box corresponding to *minimum grain size* is non-editable because there is no need of *minimum grain size* for calculation of Kernel Average Misorientation.

#### Step 6 (optional step): Advance binning option

If this option is not selected, the software divides the range of values linearly in number of bins. But if user wants to divide the bins on logarithmic scale, advance binning option must be checked. Following image shows the advance binning option for Kernel average misorientation:

##### For KAM

- ☐ Linear division
- ☐ Logarithmic division
- ☐ Linear division of  % data in  bins and rest of the data in remaining bins

Click finish and this is all that needs to be done by user, this software does the rest of the job!

## 4. Error Messages

When main application file is executed, it generates few extra text files to store the information provided by user. So execution of main application depends on these extra files. If one of these files (or source file) is not found, application throws error messages to let the user know about the problem. Following are the user codes and messages:

S.N.	Error codes	Description of error
1.	001	Error in opening/reading "filepath.txt" while reading the address of source file
2.	002	Error in opening/reading "source file" while counting number of lines in source file
3.	003	Error in opening/reading "source file" while working on phase name
4.	004	Error in opening/reading "source file" while correcting data size
5.	005	Error in opening/reading "source file" while reading data
6.	006	Error in opening/reading "response.txt" while reading the response of User
7.	007	Error in creating "KAM.txt" file for storing KAM data
8.	008	Error in creating "GAM.txt" file for storing GAM data
9.	009	Error in creating "GOS.txt" file for storing GOS data
10.	010	Error in creating "GSD.txt" file for storing GSD data