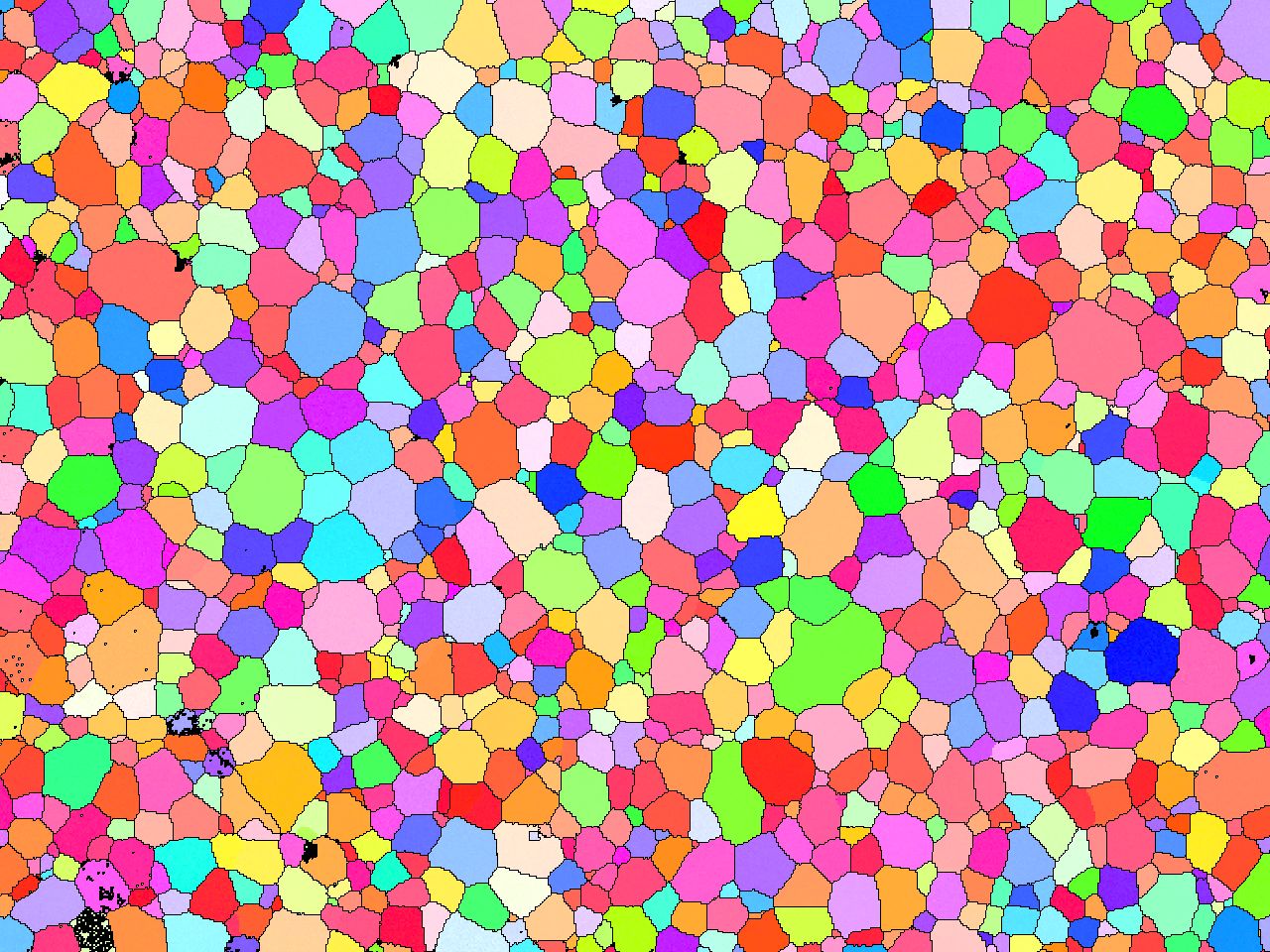
# Userbook

## Introduction

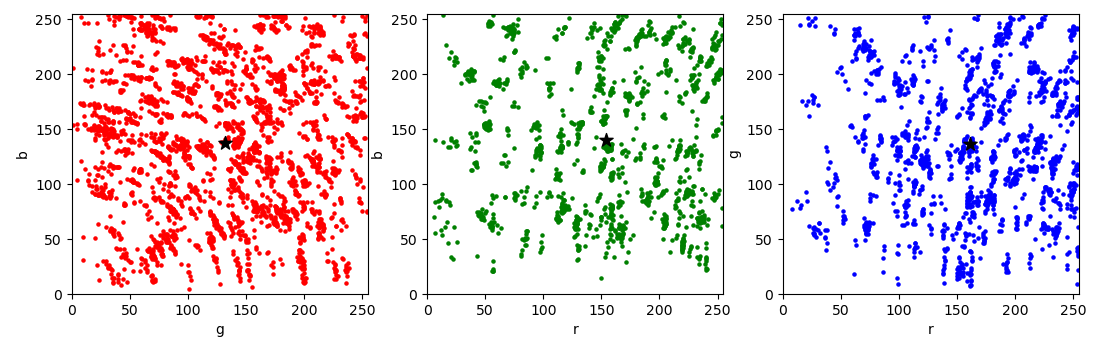
This program is to extract the information in an EBSD ipf map and represent a complete picture with a small number of parameters

## Usage

1. Installation After installing the libraries mentioned in requirements, you can use this program.
2. Load data At present, it mainly supports pictures in formats such as jgp, png, and bmp, and we will upload programs that support pictures in formats such as tiff, hdf5 in the future.

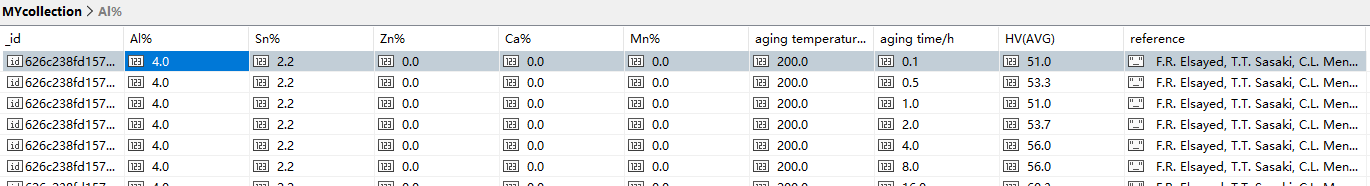


1. Analysis To minimize the number of parameters that need to be saved, we default the number of clusters to 3. Of course, you can also simply change the code to increase the number of clusters as needed.



Finally, the information we get through an EBSD ipf map is: 1. The number of grains contained in the image and their average size; 2. The 3 most representative RGB values (representing orientation information) and their concentrations (expressed with a Gaussian distribution).

1. Database We use 'mongoDB' to efficiently store data, including the above analysis data and various data related to the performance obtained from papers or experiments.



1. In this way, we can build a database of EBSD images associated with material properties. It should be noted that each piece of data will only take up very little storage space.

## Modules

**① Data\_analysis.py**

In this program, you can find several functions that analyze the EBSD IPF map.

* Sep\_color

The function of this function is to divide the RGB value in the image into three parts, namely the red part, the green part and the blue part. These data will be used for subsequent clustering algorithms.

Input: one image matrix

Output: lists of three parts

* Cluster\_3parts

This function is used to cluster the three parts separately, and use the three centers and their respective Gaussian distributions to represent the color information of the entire image.

Input: one image matrix

Output: cluster center; parameters of a Gaussian distribution

**② Grain\_parameters.py**

This program separates each grain, rather than calculating on a per-pixel basis. By default, we consider two adjacent pixels belong to the same grain when their RGB values are in the range of 0.9~1.1 times.

* Color\_list\_bygrain

This function firstly scales the image to a reasonable size, which is to speed up the calculation. According to our test, the changed multiple is within a certain range, which has little effect on the accuracy of the calculation.

Then, the program classifies the RGB pixels within a certain range as one same grain, so we can get the number and size of grains contained in the picture. These data will be used in subsequent analysis.

Input: the path of an image; width and height of your desired size

Output: Size and RGB value of each grain

* Color\_list\_new

Recreate an RGB list based on the cluster center points and Gaussian distribution parameters.

Input: cluster center; parameters of a Gaussian distribution; the number of grains

Output: a new RGB list

**③ Ipf\_map.py**

This program is used to visualize and calculate inverse pole figures.

* OrientationMatrix2Euler

Compute the Euler angles from the orientation matrix.

Input: the 3\*3 orientation matrix

Output: the 3 Euler angles in degrees

* OrientationMatrix2Rodrigues

Compute the rodrigues vector from the orientation matrix.

Input: the 3\*3 orientation matrix

Output: the Rodrigues vector as a 3 components array

* OrientationMatrix2Quaternion

Compute the quaternion from the orientation matrix.

Input: the 3\*3 orientation matrix

Output: the quaternion as a 4 components array

* Rodrigues2Axis

Compute the axis/angle representation from the Rodrigues vector.

Input: the Rodrigues vector as a 3 components array

Output: A tuple in the (axis, angle) form

* Axis2OrientationMatrix
* Euler2Axis
* Euler2Rodrigues
* Euler2OrientaionMatrix
* Hexagonal\_622\_Sym

This is a utility function that returns the rotation matrices corresponding to the 12 Crystal Symmetry elements O[622].

Input: None

Output: a rotation matrix

* Hexagonal\_622\_Sym1

This is a utility function that returns the rotation matrices corresponding to the 12 Crystal Symmetry elements O[622].

Input: None

Output: a rotation matrix

* Calc\_IPF\_position

This function projects the sample direction (ND) into the inverse pole figure (IPF) reference frame, applies crystal rotations and antipodal symmetry and returns the position(x,y) of ND in the fundamental zone of hexagonal symmetry

Packages needed: import numpy as np

Dependent on Functions: euler\_to\_g() and Hexagonal\_622\_Sym()

Input: Bunge Euler angles

Output: an array of position

* Euler2rgb

Convert Euler angles to RGB values, the default is OIM rules.

Input: Bunge Euler angles

Output: an array of RGB

* Calc\_mis

Calculate the misorientation angle between two Euler angles

Input: two array with 3 Euler angles

Output: misorientation value

* Draw\_background

Draw inverse pole figure background

Input: 3 axis

Output: a picture of ipf background

**④ Mongo\_dataDB**

Save pictures and numbers into the database with one click.

**⑤ Mongo\_csv2dataDB**

Save the data in the csv file into the database with one click.

**⑥ Mongo\_excel2dataDB**

Save the data in the excel file into the database with one click.