Instructions for pyTSXRD

1. Introduction

pyTSXRD is a package for grainmapping of polycrystalline surfaces using grazing incidence Surface X-ray Diffraction (SXRD). The notebook is an example of how it can be implemented. For use for beamlines other than P21.2 at PETRA III at DESY in Hamburg, the tools might have to be adjusted.

The package is based on py3DXRD (https://github.com/agshabalin/py3DXRD/tree/main) in collaboration with the author.

2. Steps

Cell 1

Loading relevant packages, defining variables and preparing for the rest of the analysis. The set_SweepProcessor is adapted to sweeps at P21 and will have to be adapted for data from other beamlines.

Cell 2

```
"""TEST FOR CENTER ROTATION. LOADING RAW DATA. PEAKSERCHING, MERGING, APPLYING INSTRUMENT CONFIGURATION."""
# Test for the middle point to see if parameters are good.
# Here we print and plot and if we have a good sample we should see at least one grain
slow_mid = len(slow_translations)//2
fast_mid = len(fast_translations)//2
for i_load in load_states[0:1]: #
    for i_slow in slow_translations[slow_mid:slow_mid+1]:     # (e.g. idty1).
        for i_fast in fast_translations[fast_mid:fast_mid+1]: # (e.g. idtz2).
            print(double_separator + f'\nLOAD = {i_load}, TRANSLATIONS: slow = {i_slow}, fast = {i_fast}')
            DATA = set_DATA(path_gen, i_load, i_slow, i_fast, detectors, material)
            DATA.process_images(frames = 'all', save_tifs = False, q0_pos = 'auto',
                                rad_ranges = 'auto', thr = 'auto') #load and process images
            DATA.peaksearch(peaksearch_thrs = 'auto', peakmerge_thrs = 'auto', min_peak_dist = 10) #search for peaks
            DATA.index(thr = thr) #index peaks
            DATA.gvectorEvaluator.remove_not_ranges(ds_ranges = [], tth_ranges=tth_ranges,
                                                    omega_ranges=[omega_ranges],
                                                    eta_ranges=eta_ranges) #only concider good rings
            ds_tol = DATA.gvectorEvaluator.merged[0].peakIndexer.geometry.ds_from_tth(0.12)
            # these params only for calculating tth and eta ranges
            DATA.evaluateGvectors(ds_tol=ds_tol, tth_gap=tth_gap,
                                  ds_gap=ds_gap, eta_gap=eta_gap)

            DATA.searchGrains(grainSpotter = set_GrainSpotter(path_gen, material,tth_ranges,
                                                              eta_ranges,omega_ranges)) #find grains
            DATA.runPolyXSim(polyxsim = set_PolySim(path_gen, DATA.grainSpotter, material),
                             GE_SIM_list=[DATA.gvectorEvaluator.merged[0]],also_plot=True)
            pickle.dump(DATA, open(DATA.directory+DATA.name+"_DATA.p","wb") )
```

To test the parameters, it is a good idea to first load and search for grains in the center rotation since loading all rotations are time-consuming. Here we plot and print more information than when we load all images to see any possible issues. The rotation at the center will be a full measurement of an area at the center of the sample so at least one grain can usually be detected from this one scan.

Cell 3

```
"""LOADING RAW DATA. PEAKSERCHING, MERGING, APPLYING INSTRUMENT CONFIGURATION."""
# For all points
for i_load in load_states[0:1]: #
    for i_slow in slow_translations[:]:     # (e.g. idty1).
        for i_fast in fast_translations[:]: # (e.g. idtz2).
            print(double_separator + f'\nLOAD = {i_load}, TRANSLATIONS: slow = {i_slow}, fast = {i_fast}')
            DATA = set_DATA(path_gen, i_load, i_slow, i_fast, detectors, material)
            DATA.process_images(frames = 'all', save_tifs = False, q0_pos = 'auto',
                                rad_ranges = 'auto', thr = 'auto') #load and process images
            DATA.peaksearch(peaksearch_thrs = 'auto', peakmerge_thrs = 'auto', min_peak_dist = 10) #search for peaks
            DATA.index(thr = thr) #index peaks
            #only concider good rings
            DATA.gvectorEvaluator.remove_not_ranges(ds_ranges = [], tth_ranges=DATA.grainspotter['tth_ranges'],
                                                    omega_ranges=DATA.grainspotter['omega_ranges'],
                                                    eta_ranges=DATA.grainspotter['eta_ranges'])
            ds_tol = DATA.gvectorEvaluator.merged[0].peakIndexer.geometry.ds_from_tth(0.12)
            # these params only for calculating tth and eta ranges
            DATA.evaluateGvectors(ds_tol=ds_tol, tth_gap=tth_gap, ds_gap=ds_gap, eta_gap=eta_gap)
            #DATA.searchGrains(grainSpotter = set_GrainSpotter(path_gen, material)) #find grains
            #DATA.runPolyXSim(polyxsim = set_PolySim(path_gen, DATA.grainSpotter, material),GE_SIM_list=[DATA.gvecto
            pickle.dump(DATA, open(DATA.directory+DATA.name+"_DATA.p","wb") )
```

This cell loads the full dataset, peak-searches and performs some data clean-up. This is the most time-consuming part depending on the size of your dataset. Here we don't plot and print as much information as for cell 2, but it can be added if that is of interest.

Cell 4

```
list_DATApaths, GE_list = set_paths(path_gen, load_states, slow_translations, fast_translations,
                                    detectors, material)
ind_mid = int(len(list_DATApaths)/2)
DATA  = copy.deepcopy(pickle.load(open(list_DATApaths[ind_mid], "rb")))
DATA.set_attr('directory', DATA.directory.replace(DATA.directory.split('/')[-2]+'/', '') )
DATA.set_attr('name', 's000_all_filtered')
DATA.set_attr('sample_pix_x',np.linspace(-3.5, 3.5, 701)) #sample size and pixels used when creating map
DATA.set_attr('sample_pix_y',np.linspace(-3.5, 3.5, 701))
DATA.set_attr('beamsize', 0.1)
DATA.set_attr('plot_range' ,[50,150,250,350,450,550,650])
DATA.set_attr('label_range',[-3,-2,-1,0,1,2,3])
#Merge data and remove peaks outside of ranges
DATA_ALL = merge_DATA_list(DATA, list_DATApaths, spot3d_id_reg = 5000000)
DATA_ALL.print()
DATA_ALL.gvectorEvaluator.calc_histo(omega_pixsize=0.5,eta_pixsize=0.5,ds_eta_omega_file = None,plot=False,
                                     save_arrays = False)
```

Here we merge the datasets from all rotations into one. We also set some parameters that we need later that will only depend on the size of your sample. If plot=True, more information will be shown. This will also load the result from cell 3 so that this slow step only has to be run once.

Cell 5

```
"""SEACHING FOR GRAINS"""
#Runs commands several times, removes used vectors. Can find more grains but slow
g_list = []
list_g_lists = []
gve_list = copy.copy(DATA_ALL.gvectorEvaluator.gvectors)
spot3d_id_to_remove = []
# tolerances for grouping gvectors
group_y_tol = 0.5
group_tth_tol = 0.5
group_eta_tol = 0.18
group_omega_tol = 1.5

for itr in range(16):
    print(double_separator)
    print('Iteration: ',itr)
    print(double_separator)
    GE_SIM_list = []
    for GE in DATA_ALL.gvectorEvaluator.merged:
        GE_SIM_list += GE.merged
    DATA_ALL.set_attr('grains', [])
    DATA_ALL.gvectorEvaluator.set_attr('gvectors', [gve for gve in gve_list if
                                                    gve['spot3d_id'] not in spot3d_id_to_remove])
    DATA_ALL.gvectorEvaluator.group_gvectors(group_y_tol=group_y_tol, group_tth_tol=group_tth_tol,
                                             group_eta_tol=group_eta_tol, group_omega_tol=group_omega_tol)
    DATA_ALL.evaluateGvectors(ds_tol='auto', tth_gap=tth_gap, ds_gap=ds_gap, eta_gap=eta_gap)
    DATA_ALL.searchGrains(grainSpotter = set_GrainSpotter(path_gen, material,tth_ranges,eta_ranges,omega_ranges))
    DATA_ALL.runPolyXSim(polyxsim = set_PolySim(path_gen, DATA_ALL.grainSpotter, material),
                         GE_SIM_list = [DATA_ALL.gvectorEvaluator.merged[ind_mid].merged[0]])
    for g in DATA_ALL.grains:
        g.compute_density_map(GE_SIM_list, DATA_ALL.sample_pix_x, DATA_ALL.sample_pix_y, DATA_ALL.beamsize,
                              dens_omg_tol, dens_eta_tol, dens_tth_tol, support_thr=12,final_map=False,
                              bigbeam=False)
```

```
    for g in DATA_ALL.grains:
        for gvm in g.gvectors:
            for gve in gve_list:
                if gvm['stage_y'] == gve['stage_y']:
                    if gvm['omega'] == gve['omega']:
                        GE, p = DATA_ALL.gvectorEvaluator.trace_peak_by_spot3d_id(gve['spot3d_id'])
                        if p['spot3d_id'] == gvm['spot3d_id']:
                            spot3d_id_to_remove.append(gve['spot3d_id'])
    g_list += [g for g in DATA_ALL.grains if len(g.gvectors) > 0]
    list_g_lists += [DATA_ALL.grains]

DATA_ALL.set_attr('grains', g_list)
n_grains = len(DATA_ALL.grains)
print('Number of grains:', n_grains)
pickle.dump(DATA_ALL, open(DATA_ALL.directory+DATA_ALL.name+"_DATA_ALL.p","wb") )
```

Here we search for grains in the dataset. The thresholds for grouping **G** vectors (peaks that will be considered "the same **G** vector but at different sample positions) is set with the group_..._tol. The loop will group the vectors, assign them to a powder ring, search for grains, remove all vectors that fit to the grains found from the full dataset and then repeat for the dataset of vectors that are left. The number of iterations should be such that there are no or almost no new grains found in the last iteration.

Cell 6 and 9

This cell can be used if you want to load the result so that no slow steps have to be repeated.

Cell 7

```
"""REMOVE DUPLICATES"""
n_grains = len(DATA_ALL.grains)
print('Number of grains:', n_grains )
DATA_ALL.remove_duplicates(ang_tol=.5, pos_tol=0.1) #If set too high it can reove too many grains
```

Here we remove grains that might have been found more than once in cell 5. This is done via thresholds on difference in orientation and center of mass-position. It is good to keep the values small here and only remove grains that we are certain are duplicates. Duplicates can rather be combined in cell 14.

Cell 8

```
"""CREATE DENSITY MAP FOR ALL GRAINS FOUND IN THE PREVIOUS STEP"""
GE_SIM_list = []
n_grains = len(DATA_ALL.grains)
for GE in DATA_ALL.gvectorEvaluator.merged:
    GE_SIM_list += GE.merged
for i,g in enumerate(DATA_ALL.grains):
    g.compute_density_map(GE_SIM_list, DATA_ALL.sample_pix_x, DATA_ALL.sample_pix_y, DATA_ALL.beamsize,
                          dens_omg_tol, dens_eta_tol, dens_tth_tol, support_thr=8,sample_rot=0,
                          final_map=True,bigbeam=False)
    print('Density maps %.2f%% done   ' %((i+1)/n_grains*100), end="\r")

pickle.dump(DATA_ALL, open(DATA_ALL.directory+DATA_ALL.name+"_DATA_ALL.p","wb") )
```

This cell takes all grains found in cell 5 and assigns all **G** vectors within the thresholds (dens_XXX_tol) and that has a reasonable overlap (support_thr) with the other vectors, to the grain. The sample (the density map) can be rotated with any angle. final_beam solves issues with gradients in the intensity, that are not related to the variation in number of overlapping vectors.

Cell 10

Cell for plotting different steps in the density-map-process.

Cell 11

```
"""TAKES DENSITY MAP AND RETURNS A MATRIX FOR EACH GRAIN"""
completeness_th=0.45 #th for masking of the density map
DATA_ALL.make_grainmatrix(completeness_th=completeness_th,also_plot=False,save_matrix=False)
```

This cell takes the density-map and returns a matrix per grain with the completeness. This is done by first applying a Gaussian filter to the density map. The completeness is defined $C_{map} = \frac{density}{expected\ \boldsymbol{G}\ vectors}$. The map of grains where the maximum completeness <completeness_th-0.15 is set to zero. The grain is first

defined as where $C_{map} > completeness\_th \times \max(C_{map})$. We then apply the following to minimize that larger grains tend to have a higher completeness, $C_{final} = C_{map}\sqrt{(\max(C_{\mathrm{map}}))\sqrt{r}}$. A good completeness_th can vary depending on the size of grains and general quality of the measurement. If also_plot=True, each completeness matrix will be shown.

Cell 12

```
"""CORRECT FOR TILT OF SAMPLE"""
mu = -0.09
DATA_ALL.tilt_correction(ang_inc=mu)
```

It has up until this point been assumed that the surface is parallel with the beam. Here we can correct for the grazing incidence angle. This will be a very small correction, depending on sample and beam energy.

Cell 13

```
"""Define values we need convert plot from pixel values to mm"""
DATA_ALL.set_attr('plot_range' ,[50,150,250,350,450,550,650])
DATA_ALL.set_attr('label_range',[-3,-2,-1,0,1,2,3])

#REMOVE PARTS OF GRAINS OUTSIDE OF GRAINBOUNDARIES
radius = 701/2
DATA_ALL.apply_samplemask(radius,also_plot=False,plot_invers=False,save_invers=False)

#TAKE THE MOST COMPLETE GRAIN IN EACH POINT WHERE GRAINS OVERLAPP. OPTION TO PLOT & SAVE
grain_th = 100 #If a grain has a total number of pixels below the th, the grain will be remove
DATA_ALL.map_sample(also_plot=False,grain_th=grain_th)

#PLOT AS AN INVERSE POLEFIGURE MAP
name_file = 'pd_1' #extension to file name of map
DATA_ALL.plot_colors(name_file,plot_type='full',also_save=False,mark_grainnumber=False,mark_millers=False)

pickle.dump(DATA_ALL, open(DATA_ALL.directory+DATA_ALL.name+"_DATA_MAP.p","wb") )
```

The radius is the sample radius in pixels. The apply_samplemask is removing any pixels outside the sample boundaries.

The grain_th is the area limit of a grain. This approach should be limited by the beam width so here the (beam width)^2 has been used. This can be skipped by setting grain_th to 0.

The result can be plotted as an inverse pole figure map (either for one grain separately or the full sample). The grains can be marked with numbers or miller indices.

Cell 14

```
"""SOME GRAINS MIGHT BE SPLIT INTO TWO OR MORE PARTS, HERE THEY CAN BE COMBINED"""
ang_tol = 1.73 #threshold to still be concidered the same grain
pos_tol = 15 #distance in pixel index of any part of the grain to still be concidered the same grain
DATA_ALL.combine_duplicates(ang_tol, pos_tol)

#PLOT AS AN INVERSE POLEFIGURE MAP
name_file = 'pd_2' #extension to file name of map
DATA_ALL.plot_colors(name_file,plot_type='full',also_save=True,mark_grainnumber=False,mark_millers=False)
pickle.dump(DATA_ALL, open(DATA_ALL.directory+DATA_ALL.name+"_DATA_MAP.p","wb") )
```

There is a risk that a single grain has been split into more than one in previous steps. Here they can be combined. If the thresholds are too high, it will combine too many grains so this should be done with caution. The final map can then be plotted as in cell 13.