# Documentation: Calibration and Recalibration Modules

Generated Documentation

February 4, 2026

## Contents

# 1 Overview

This documentation covers two complementary modules for geometric calibration in electron diffraction:

- **calibration.py**: Initial geometric calibration setup using reference standards (CIF files)
- **recalibration.py**: Iterative center refinement using diffraction patterns

## 1.1 calibration.py Overview

`calibration.py` provides tools for setting up geometric calibration from scratch. It generates reference diffraction peaks from crystallographic structures and launches the interactive pyFAI-calib2 calibration tool.

## 1.2 recalibration.py Overview

`recalibration.py` provides tools for precise diffraction center refinement in TEM (Transmission Electron Microscopy) images. It implements two strategies:

- **With beamstop**: Fit ellipse to visible diffraction ring
- **Without beamstop**: Locate beam position from maximum intensity

Core functionality includes pixel-to-PONI coordinate conversion, detector orientation handling, and interactive visualization.

# Part I
# Module: calibration.py

## 2 Initial Geometric Calibration

### 2.1 Overview

The calibration module provides a complete workflow for initial geometric calibration:

1. Load reference crystallographic structure (CIF file)
2. Compute expected diffraction peaks
3. Load experimental diffraction image (DM4 format)
4. Generate reference distance file for pyFAI
5. Launch interactive calibration tool (pyFAI-calib2)

### 2.2 build_calibration_data_from_cif()

```
build_calibration_data_from_cif(cif_file, wavelength,
                                n_peaks=15, two_theta_range=(0,
                                   2))
```

**Purpose:** Generate expected diffraction peaks from crystal structure using XRD simulation.

**Physical Basis:**

The function computes theoretical diffraction patterns using Bragg's law and structure factor calculations. For each reflection (hkl):

**Step 1: Structure Loading**

Load crystal structure from CIF file:

- Lattice parameters
- Atomic positions
- Occupancy and temperature factors

**Step 2: XRD Calculator Initialization**

Initialize XRDCalculator with electron wavelength $\lambda$:

$$d = \frac{\lambda}{2\sin\theta}$$

(Bragg's law)

where $d$ is d-spacing and $2\theta$ is scattering angle.

**Step 3: Pattern Computation**

For each allowed reflection:

$$F_{hkl} = \sum_j f_j e^{2\pi i(hx_j + ky_j + lz_j)}$$

where $F_{hkl}$ is structure factor, $f_j$ are atomic scattering factors.
Intensity: $I_{hkl} \propto |F_{hkl}|^2$

**Step 4: Peak Selection**

Extract top $n_{\text{peaks}}$ reflections by intensity within $2\theta$ range.

**Output File: distances.txt**

Saves d-spacings (one per line) for use in pyFAI-calib2:

```
3.1234
2.2156
1.8934
...
```

**Parameters:**

- `cif_file`: Path to CIF crystallographic file
- `wavelength`: Electron wavelength in Ångströms
- `n_peaks`: Number of top peaks to extract (default: 15)
- `two_theta_range`: $2\theta$ range in degrees (default: 0-2°)

**Returns:**

- List of dictionaries, each containing:
  - `hkl`: Miller indices as tuple
  - `d`: d-spacing in Ångströms
  - `two_theta`: Bragg angle in degrees
  - `intensity`: Relative intensity

### 2.2.1 Example Output

```
[
    {'hkl': (1,1,1), 'd': 2.355, 'two_theta': 0.85, 'intensity':
       1.000},
    {'hkl': (2,0,0), 'd': 2.039, 'two_theta': 0.98, 'intensity':
       0.856},
    ...
]
```

## 2.3 perform_geometric_calibration()

```
perform_geometric_calibration(cif_file: str, dm4_file: str)
```

**Purpose:** Complete calibration workflow: prepare reference data and launch interactive pyFAI-calib2 tool.

**Workflow Steps:**

**Step 1: Load Experimental Data**

Load DM4 file using custom reader:

- Extract raw image data

- Extract metadata: wavelength, detector parameters

**Step 2: Generate Reference Peaks**
Call `build_calibration_data_from_cif()` to create `distances.txt`
**Step 3: Convert to EDF Format**
Convert DM4 → EDF (ESRF Data Format):

- Store image data
- Embed metadata as headers

`fabio` library handles format conversion.
**Step 4: Print Experimental Settings**
Display calibration parameters to console:

```
======================================================================

EXPERIMENT SETTINGS TO INPUT IN PYFAI-CALIB2:
======================================================================

Camera description=Gatan K2 IS
pixel_size_x=5.0X5.0 ( m )
image dimension=3840X3712 (pixels)
Electron wavelength=0.0251
======================================================================

Launching pyFAI-calib2
```

Operator should input these values in the pyFAI-calib2 GUI manually.
**Step 5: Launch Interactive Calibration**
Execute pyFAI-calib2 via subprocess:

conda run -n epdfpy pyFAI-calib2 -c distances.txt image.edf

User performs interactive refinement:

1. Manually click on diffraction rings in image
2. pyFAI fits geometry to user picks
3. Refine until residual minimized
4. Save PONI file

**Step 6: Cleanup**
Delete temporary EDF file after calibration completes.
**Parameters:**

- `cif_file`: Path to reference crystal structure
- `dm4_file`: Path to calibration diffraction image

**Output:**

- `distances.txt`: Reference d-spacings
- `*.poni`: Calibration parameters (saved by pyFAI-calib2)

## 2.4 get_calibration_parameters()

```
get_calibration_parameters(poni_file: str)
```

**Purpose:** Parse PONI file and extract geometric calibration parameters in structured format.

**Algorithm:**

1. Open PONI text file
2. Parse key-value lines
3. Extract numeric parameters
4. Parse JSON detector configuration if present
5. Return dictionary with all parameters

**PONI File Format:**
Example PONI file:

```
# Calibration file for pyFAI
# Using format for pyFAI detector configuration

Distance: 0.2344
Poni1: 0.001234
Poni2: 0.001456
Rot1: 0.00234
Rot2: 0.00134
Rot3: 0.00512
Wavelength: 2.51e-12
Detector_config: {"pixel1": 5e-6, "pixel2": 5e-6, "max_shape":
   [3712, 3840]}
```

**Extracted Parameters:**

| Parameter | Meaning | Unit |
|---|---|---|
| `distance` | Sample-detector distance | m |
| `poni1` | Vertical beam center | m |
| `poni2` | Horizontal beam center | m |
| `rot1` | Tilt around x-axis | rad |
| `rot2` | Tilt around y-axis | rad |
| `rot3` | In-plane rotation | rad |
| `wavelength` | Electron wavelength | m |
| `pixel1`, `pixel2` | Pixel dimensions | m |
| `max_shape` | Image dimensions | pixels |

**Parameters:**

- `poni_file`: Path to PONI calibration file

**Returns:**

- Dictionary with all extracted parameters

**Example Usage:**

```
params = get_calibration_parameters('calibration.poni')
print(f"Distance: {params['distance']*1000:.2f} mm")
print(f"Center: ({params['poni1']*1e6:.2f}, {params['poni2']*1e6
    :.2f}) m ")
print(f"Wavelength: {params['wavelength']*1e12:.4f} pm")
```

**Raises:**

- `FileNotFoundError`: If PONI file does not exist

# Part II
# Module: recalibration.py

## 2.5 center_to_poni()

```
center_to_poni(cx, cy, px_size, py_size, dist,
               rot1=0.0, rot2=0.0, rot3=0.0)
```

**Purpose:** Convert pixel coordinates to pyFAI PONI (Point Of Normal Incidence) format, accounting for detector rotations.

**Coordinate Systems:**

**Input (pixel coordinates):**

- $(c_x, c_y)$: Center position in pixel indices
- Horizontal axis: columns (width direction)
- Vertical axis: rows (height direction)

**Output (PONI coordinates):**

- `poni1`: Vertical beam position (meters from image top)
- `poni2`: Horizontal beam position (meters from image left)

### 2.5.1 Algorithm

The conversion involves three steps:

**Step 1: Pixel to meter conversion**

$$\mathbf{v} = \begin{pmatrix} c_y \cdot p_y \\ c_x \cdot p_x \\ d \end{pmatrix}$$

where $p_x, p_y$ are pixel sizes (m/pixel) and $d$ is sample-detector distance.

**Step 2: Rotation matrices**

Three rotation matrices represent detector tilts:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 \\ 0 & \sin\theta_1 & \cos\theta_1 \end{pmatrix}$$

(rotation around x-axis)

$$R_y = \begin{pmatrix} \cos\theta_2 & 0 & \sin\theta_2 \\ 0 & 1 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 \end{pmatrix}$$

(rotation around y-axis)

$$R_z = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(in-plane rotation)

Combined rotation: $\mathbf{R} = R_z R_y R_x$

**Step 3: Transform to detector frame**

$$\begin{pmatrix} \text{poni1} \\ \text{poni2} \\ z \end{pmatrix} = \mathbf{R}^{-1}\mathbf{v}$$

**Parameters:**

- `cx, cy`: Center coordinates (pixels)
- `px_size, py_size`: Pixel dimensions (m/pixel)
- `dist`: Sample-detector distance (meters)
- `rot1, rot2, rot3`: Rotation angles (radians)

**Returns:**

- `poni1`: Vertical PONI coordinate (meters)
- `poni2`: Horizontal PONI coordinate (meters)

# 3 Ellipse Fitting for Beamstop Images

## 3.1 fit_ellipse_to_ring()

```
fit_ellipse_to_ring(image, threshold_rel=0.5, min_size=50)
```

**Purpose:** Estimate diffraction ring center from binary image via covariance ellipse fitting.

**Algorithm Overview**

The function implements a 3-step process:

### 3.1.1 Step 1: Automatic Thresholding

Compute Otsu threshold, then apply relative scaling:

$$T = \text{threshold\_otsu}(I) \times r_{\text{threshold}}$$

$$B(x, y) = \begin{cases} 1 & \text{if } I(x, y) > T \\ 0 & \text{otherwise} \end{cases}$$

Then remove isolated pixels (morphological opening):

$$B' = \text{remove\_small\_objects}(B, \text{min\_size})$$

**Effect:** Extracts bright ring pixels while suppressing noise.

### 3.1.2 Step 2: Connected Component Labeling

Label connected regions and extract pixel coordinates:

$$\text{regions} = \text{label}(B')$$

Select largest region (assumed to be the diffraction ring):

$$\text{region}_{\max} = \arg\max_r |\text{region}_r|$$

Extract pixel coordinates: $\mathbf{P} = \{(x_i, y_i)\}_{i=1}^{N}$

### 3.1.3 Step 3: Ellipse Fitting via Covariance

Compute center (centroid):

$$\bar{x} = \frac{1}{N}\sum_i x_i, \quad \bar{y} = \frac{1}{N}\sum_i y_i$$

Compute covariance matrix:

$$\mathbf{C} = \text{cov}(x - \bar{x}, y - \bar{y})$$

Eigendecomposition: $\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$
Semi-axes from eigenvalues: $a = \sqrt{\lambda_1}$, $b = \sqrt{\lambda_2}$
Orientation angle: $\theta = \arctan 2(v_{1y}, v_{1x})$

**Parameters:**

- `image`: 2D array of TEM diffraction data
- `threshold_rel`: Relative threshold (0-1, default 0.5)
- `min_size`: Minimum object size (pixels, default 50)

**Returns:**

- `cx, cy`: Center coordinates (pixels)
- `a, b`: Semi-major and semi-minor axes (pixels)
- `angle`: Ellipse orientation (radians)

**Raises:**

- `ValueError`: If no ring detected (adjust `threshold_rel`)

# 4 Recalibration Methods

## 4.1 recalibrate_no_beamstop()

```
recalibrate_no_beamstop(dm4file, ponifile, output_ponifile=None)
```

**Purpose:** Refine beam center using maximum intensity position (for images without beamstop).

**Workflow:**

1. Load DM4 image and extract metadata
2. Find beam position: $(\hat{y}, \hat{x}) = \arg\max I(y, x)$
3. Apply detector orientation correction
4. Convert pixel coordinates to PONI via `center_to_poni()`
5. Update and save PONI file

**Detector Orientation Handling:**
The function corrects for detector frame conventions (4 possible orientations):

| Orientation | Description | `c_y` correction | `c_x` correction |
|---|---|---|---|
| 1 | Top-left | $H - y$ | $W - x$ |
| 2 | Top-right | $H - y$ | $x$ |
| 3 | Bottom-right | $y$ | $x$ |
| 4 | Bottom-left | $y$ | $W - x$ |

Table 1: Detector orientation transformations. $H$, $W$ = image height, width

**Parameters:**

- `dm4file`: Path to DM4 image file
- `ponifile`: Path to reference PONI calibration
- `output_ponifile`: Output PONI path (optional)

**Returns:**

- AzimuthalIntegrator object with updated geometry

## 4.2 recalibrate_with_beamstop()

```
recalibrate_with_beamstop(dm4file, ponifile, threshold_rel=0.5,
                          min_size=50, output_ponifile=None, plot
                             =False)
```

**Purpose:** Precise center refinement using ellipse fitting to visible diffraction ring (for images with beamstop).

**Workflow:**

1. Load DM4 image
2. Fit ellipse to brightest ring via `fit_ellipse_to_ring()`
3. Extract center coordinates $(x_c, y_c)$
4. Apply detector orientation correction (4 possible configurations)
5. Convert to PONI coordinates
6. Update AzimuthalIntegrator object
7. Optional: Display diagnostic plot

**Advantage over no_beamstop:** Uses circular symmetry of diffraction ring rather than single intensity maximum, providing more robust center estimation.

**Parameters:**

- `dm4file`: DM4 image path
- `ponifile`: Reference PONI file
- `threshold_rel`: Relative threshold for ring extraction (0-1)
- `min_size`: Minimum connected component size
- `output_ponifile`: Optional output PONI path
- `plot`: Display visualization (default: False)

**Returns:**

- AzimuthalIntegrator with refined geometry

**Visualization (when plot=True):**
A figure displays:

- Image in grayscale with dynamic contrast (1st-99th percentile)
- Fitted ellipse overlay (red, with semi-axes dimensions)
- Center position marker (green plus sign)
- Detector orientation in title
- Legend with coordinate values

# 5 Utility Functions

## 5.1 update_ai()

```
update_ai(initial_ponifile, poni1_new, poni2_new,
          output_ponifile=None)
```

**Purpose:** Load PONI file, update beam center coordinates, optionally save.
**Parameters:**

- `initial_ponifile`: Reference PONI file
- `poni1_new, poni2_new`: New PONI coordinates (meters)
- `output_ponifile`: Output file (optional)

**Returns:**

- Updated AzimuthalIntegrator object

# 6 Detector Orientations

PyFAI supports 4 detector orientations via the orientation attribute:

| Value | Name | Origin | X-axis | Y-axis |
|---|---|---|---|---|
| 1 | Top-left | Top-left corner | → (right) | ↓ (down) |
| 2 | Top-right | Top-right corner | ← (left) | ↓ (down) |
| 3 | Bottom-right | Bottom-right corner | ← (left) | ↑ (up) |
| 4 | Bottom-left | Bottom-left corner | → (right) | ↑ (up) |

Table 2: PyFAI detector orientation conventions

The recalibration functions automatically apply the correct transformation based on the input PONI file's orientation.

# 7 Usage Examples

## 7.1 Recalibrate from Beamstop Image

```
from recalibration import recalibrate_with_beamstop

ai_new = recalibrate_with_beamstop(
    dm4file='sample_001.dm4',
    ponifile='calibration.poni',
    threshold_rel=0.5,
    min_size=80,
    output_ponifile='sample_001_recal.poni',
    plot=True
)

# Perform integration with refined geometry
_, img = load_data('sample_001.dm4', verbose=False)
q, I = ai_new.integrate1d(img, 1000, unit="q_A^-1")
```

## 7.2 Recalibrate from Non-Beamstop Image

```
from recalibration import recalibrate_no_beamstop

ai_new = recalibrate_no_beamstop(
    dm4file='sample_002.dm4',
    ponifile='calibration.poni',
    output_ponifile='sample_002_recal.poni'
)
```

## 7.3 Batch Recalibration

```
import glob

dm4_files = glob.glob('samples/*.dm4')

for dm4_file in dm4_files:
    ai = recalibrate_with_beamstop(
        dm4file=dm4_file,
        ponifile='calibration.poni',
        threshold_rel=0.5,
        plot=False  # disable for batch processing
    )
    # Use ai for integration...
```

# 8 Parameter Selection Guide

## 8.1 threshold_rel

Controls intensity threshold for ring extraction:

- **Low (0.3-0.4)**: Include weaker outer rings, more noise
- **Medium (0.5)**: Recommended default, good balance
- **High (0.7-0.9)**: Extract only brightest pixels, risk of missing ring

**Optimization**: Adjust based on signal-to-noise ratio. Use `plot=True` to verify ring detection.

## 8.2 min_size

Minimum connected component size in pixels:

- **Small (20-30)**: Include noise blobs, computational overhead
- **Medium (50-100)**: Recommended, balances noise rejection and ring detection
- **Large (200+)**: Very strict filtering, risk of losing ring

**Rule of thumb**: min_size $\approx 0.1 \times$ ring_area

# 9 Performance Considerations

## 9.1 Computational Complexity

- Thresholding and morphology: $O(N)$ where $N$ = image size
- Connected component labeling: $O(N)$ with Union-Find
- Ellipse fitting: $O(M)$ where $M$ = number of ring pixels
- Typical image (2048×2048): $< 100$ ms

## 9.2 Memory Usage

- Image storage: 8 MB for 2048×2048 float64
- Binary mask: 0.5 MB
- Label map: 8 MB
- Total: $< 20$ MB per image

## 9.3 Accuracy

- Center detection precision: ±1-2 pixels typical
- In PONI space: $\sim 10 - 50$ µm (depending on detector distance)
- Ellipse fitting provides sub-pixel accuracy

# 10 Error Handling and Troubleshooting

## 10.1 Common Issues

| Problem | Cause | Solution |
|---|---|---|
| No ring detected | `threshold_rel` too high | Lower to 0.3-0.4 |
| Noise instead of ring | `threshold_rel` too low | Increase to 0.6-0.8 |
| Multiple rings detected | `min_size` too small | Increase `min_size` |
| Incorrect center | Beamstop covers center | Use `recalibrate_no_beamstop` |
| PONI file error | Orientation mismatch | Check detector orientation attribute |

## 10.2 Debugging

Enable visualization to diagnose:

```
# Visualize ring detection
ai = recalibrate_with_beamstop(
    dm4file='sample.dm4',
    ponifile='calibration.poni',
    threshold_rel=0.5,
    min_size=50,
    plot=True  # Shows ellipse fitting result
)
```

If center appears wrong:

1. Verify PONI file orientation: `ai.detector.orientation.value`
2. Check image is not rotated/flipped compared to calibration
3. Try manually specifying detector orientation in PONI file

# 11 Dependencies

- `numpy`: Array operations
- `scikit-image`: Image processing (thresholding, morphology, labeling)
- `pyFAI`: Azimuthal integration and PONI file handling
- `matplotlib`: Visualization (optional, only if `plot=True`)
- `filereader`: Custom module for DM4 file I/O

# 12 References

- pyFAI documentation: `https://pyfai.readthedocs.io/`
- scikit-image documentation: `https://scikit-image.org/`
- numpy: `https://numpy.org/`