ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-THAG004

HASMI: A CONFIGURABLE PYTHON FRAMEWORK FOR AUTOMATED HARMONIC ANALYSIS AND SCAN ORCHESTRATION AT EMIL

A. Balzer, A. Efimenko, P. Sreelatha Devi, K. Holldack
Helmholtz-Zentrum Berlin (HZB), Berlin, Germany

Abstract

We present HASMI (Harmonic Analyzer State Machine Interface), a highly configurable Python framework designed to automate data acquisition and analytical tasks for optimizing coordinated undulator and monochromator movements at synchrotron beamlines. Built upon EPICS, HASMI features a comprehensive scan library and a command-line interface for managing multi-actuator scans. The framework offers extensive customization through a user-configurable database for defining scan types, actual scan parameters, and user preferences. Integrated analytical tools enable precise identification of undulator harmonic positions using advanced techniques such as convolution and customized cross-correlation. Beamline operations are automated and parameterized through a generalized state machine, which provides both an intuitive command-line interface and a dedicated EPICS interface. The state machine operates efficiently as a continuous background service alongside the monochromator IOC. By integrating multiple alignment scans, visualization, and automated data analysis into a streamlined 'one-button' procedure, HASMI significantly enhances the speed, performance, and reliability of beamline operations for users of the four hard X-ray CPMU17-DCM branches at EMIL.

INTRODUCTION

At BESSY II, efforts are underway to establish a more holistic and unified software environment [1] based on Bluesky [2]. However, the current controls landscape is still heterogeneous: software such as SPEC, SpecsLab Prodigy and Igor Pro® as well as many other frameworks are deployed in day-to-day operations. With few exceptions, the Experimental Physics and Industrial Controls System (EPICS) [3] and its Channel Access (CA) protocol [4] provide the common denominator.

In this context, it is natural to seek reusable solutions for core tasks that operate independently of a particular experimental framework. An interoperable and configurable approach offers the greatest benefit, as it can be adapted to diverse requirements across beamlines. HASMI addresses this need by exposing its functionality through multiple access layers: an EPICS CA interface, a command-line interface (CLI), and a Python API. This flexibility allows operators to use HASMI directly through standard operator interfaces (OPIs), while integrators can embed it seamlessly into existing automation frameworks.

Undulator Lookup Table (ULT) Correction

A central task in undulator–monochromator synchronization is the maintenance of a reliable lookup table (ULT) that maps undulator gap values to monochromator energies. In practice, the energy scale of the ULT often requires fine-tuning, which can be achieved by introducing a linear correction with slope and offset. The gap is then related to the setpoint energy E by

$$\text{gap} = f(\text{slope} \cdot E + \text{offset}), \tag{1}$$

and the corrected photon energy follows from the inverse mapping as

$$E = \frac{f^{-1}(gap) - \text{offset}}{\text{slope}}. \tag{2}$$

Two reference scans are acquired at the beginning and end of a selected energy range. The harmonic analyzer evaluates both scans, identifies the respective harmonic peak positions with sub-sample precision, and derives an updated calibration in terms of slope and offset. These values are a proposed *correction*, which is stored in a manifest file together with metadata and the previously active calibration values.

The correction can then be applied to the corresponding EPICS process variables. This mechanism ensures that the undulator–monochromator relation remains consistent over time, compensating for drifts or mechanical changes in the beamline environment.

TECHNICAL BACKGROUND AND DESIGN

Software Foundations

HASMI builds on two modular Python frameworks under active development at HZB: a multi-actuator *scan* framework and an *asyncio*-based device orchestration framework, the *scheduler*. Both are lightweight, designed for beamline production tests and automation.

Scan Library The scan library provides flexible multi-actuator scanning with the capablilty of

- parallel scans over independent axes,
- concurrent execution of move sequences or other devices, and
- continuous-mode scans where one or more axes run at constant velocity while others are synchronized (continuous mode performance depends on the capabilities of the motion controller and IOC.)

A plugin architecture exposes hooks before, during, and after scan steps, enabling integration of custom controllers,

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-THAG004

devices, and loggers. Scans are configured by typed data models and can be launched via a lightweight CLI layer or as callable Python objects.

Actuators are abstracted via a factory pattern (single/multi actuators, simulated or EPICS-backed variants) and can be monitored by pluggable monitors (e.g. a queue-based plotter). Data handling is performed by a minimal data manager that can handle scalar and array data from EPICS Process Variables (PVs) and user devices. YAML + Jinja2 [5, 6] based configuration files and templates define scan types and parameter presets.

Scheduler The scheduler is a lightweight, fully asynchronous framework for orchestrating and automating sequences of device operations. An abstract device framework is central to the scheduler: devices can represent actuators, scans, or logic units.

Sequences are expressed declarative in YAML, where each step specifies the devices and their target parameters. The scheduler engine interprets this structure and determines the execution pattern (parallel or serial).

Based on a flexible abstraction, devices used for HASMI include actuators, scans, sequence and toggle used to configure EPICS PVs and automation. Ophyd [7] can be integrated as needed. These device sequences run in parallel and/or in series according to a declarative YAML + Jinja2 description. This enables script-free automation needed for HASMI, such as:

- toggle devices like photon shutters,
- configure beamline PVs for measurements,
- scans: performing an approach sequence, parallel scan of mono energy and piezo stage

EPICS and Ecosystem Integration Both libraries avoids the problem of stringly typed code. The program logic does not depends on string fragments (like from naming conventions or identifiers). This is achieved by using configuration files, data classs and structured types instead of hard coding string fragments. This makes it very flexible and easily to integrate in different cite specific environments via the CLI or by importing the Python modules.

Limitations and Scope The frameworks are targeted at developers, engineers, and system integrators, providing modular low-level control for production testing, integration, and debugging. They are developed in an agile manner, continuously refined to meet evolving requirements while keeping dependencies minimal. The software inherit limitations from the controllers/IOCs (e.g. continuous scanning depends on controller performance). Detailed descriptions of both frameworks will be published separately.

Related Work

At BESSY II, monochromator IOCs expose EPICS records that compute slope/offset correction parameters from user input and automatically apply them to the active undulator lookup table at the database level. While it works for the specific case where the monochromator's spectral window is scanned through the undulator harmonics, this approach is limited: it offers no higher-level automation and cannot support the inverse scenario of scanning the undulator against the monochromator. HASMI addresses these limitations automates both directions of coordination and integrates seamlessly with beamline operations. Several facilities have reported automation frameworks and methods related to HASMI's goals [2, 8, 9] that motivate HASMI's design choices: a configurable state-machine pattern with EPICS integration and a flexible automation/analyzer stack.

Software Design

HASMI adds an application level abstraction layer above the devices and integrates with the EPICS control system. It automates the coordinated workflow of undulator–monochromator alignment. The architecture consists of several cooperating components.

PvObserver The observer monitors process variables (PVs) and exchanges data asynchronously with the state machine via a message queue. The data class HASMIConfig instance actually describes a usable scan setup and enforces several rules so that the state machine doesn't try to run with nonsense values.

State Machine Pattern The state machine implements a generalized, parameterized workflow for the alignment procedure. It exposes an *EPICS* interface and is intended to run continuously next to the monochromator IOC. The state machine hierarchy consists of an abstract base class (AbstractStateMachineHarmonicAnalyzer), a common intermediate class that extends the base with reusable features, and concrete implementations that derive from the common class to provide beamline-specific behavior. It tracks current and previous states, supports forward and backward navigation through a predefined loop, and includes dedicated error and confirm states for robust recovery and operator validation.

Control Layer Scheduler and Scan Library interpret declarative YAML configs and execute the scans.

Harmonic Analyzer Strategy Pattern The analyzer evaluates scan data, locates harmonic peaks needed for the calculation of slope/offset corrections. It can also be used offline for post-mortem evaluation. It is built on NumPy [10] and consists of a strategy-based signal analysis core that detects harmonic peaks from transient intensity vs. energy/gap. It finds the maximum of the normalized intensity series and maps that row to the associated gap with the maximum of the corrosponding undulator harmonics. It is based on argmax, center-of-gravity, and convolution-based methods with configurable kernels (Mexican Hat, Pearson types, Gaussian, boxcar, file-defined). Peak finding is deligated via a clean strategy interface returning a `MaxResult` (peak

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-THAG004

index/coordinates, score, diagnostics). Each kernel is represented as an object whose primary role is to produce a numerical array on the upsampled analysis scale, suitable for convolution or filtering operations. Convolution strategies optionally normalize kernels and support integer upsampling with either repeat or linear interpolation. This decoupling allows method-specific diagnostics without changing the state machine.

Presets and Preset Manager A preset in HASMI is a named collection of validated configuration parameters (e.g., scan ranges, harmonics, analysis methods). The Preset Manager loads a database of built-in presets shipped with the framework and allows users to create and persist their own presets in a separate file. To ensure flexibility without losing reproducibility, user-defined presets take precedence. If a user preset has the same name as a built-in one, the user version overrides the default. This separation preserves factory defaults while enabling beamline staff and users to adapt parameters. Scheduler presets and state-specific configuration refine the automation workflow for specific needs.

Interfaces HASMI integrates operator and analysis frameworks through EPICS Channel Access (CA), command-line tools, and Python APIs. It provides two main interfaces: CLI tools for scan data analysis and optional plots; EPICS PVs for operation through the operator interface or from other EPICS clients. Both rely on the analyzer core and manifest schema.

Configuration and Data HASMI specific configuration consists of yaml files for each state. A manifest YAML records each run (parameters, file paths, derived slope/offset), enabling reproducible evaluation and offline analysis. Data frames produced by scans are consumed by the analyzer.

Safety, Logging, and Extensibility All motion limits are enforced at the controller/IOC level. HASMI validates configurations, clamps parameters, and fails safely on inconsistent metadata. Structured logging and optional debug plots support incident analysis. New beamlines can be integrated by adapting configuration files and presets without modifying the core logic. For fundamental changes or additional feature mix-ins, a new state machine can be derived from the abstract state machine pattern.

METHODS AND EVALUATION

Data Acquisition and automation is orchestrated by the scheduler, with scans executed through the scan library. Input channels comprise intensity detector, undulator gap, monochromator energy, and ring-current (for normalization). Trajectories may be stepped or continuous, while scan parameters such as dwell times are defined in YAML beamline templates.

Pre-Processing

The following optional pre-processing of the scan data has been implemented:

- *Ring current normalization*: Pure version of intensity normalization. Divides intensity by ring current (zero-safe; returns NaN where ring current = 0).
- *Detrend (linear)*: Removes the best-fit line $ai + b$ from y using least squares.
- *Peak window*: Returns a copy of the signal masked outside the interval $[\,\text{center} - w/2,\ \text{center} + w/2\,]$. If the `center` position is not defined, the global arg max is used.
- *Normalization*: Vector normalization using either L^1 or L^2 norm.
- *Resample*: It can be upsampled by integer repeat (step/hold) or by integer linear interpolation between samples.

Signal Processing

Given measured samples (x, y) (e.g. energy or gap versus normalized intensity), the analyzer determines the peak position using one of the following methods:

- DEFAULT: discrete maximum (argmax),
- COG: center of gravity within a local window,
- CUSTOM: convolution with a user-selected kernel k (e.g. Mexican Hat, Pearson VI/VII, Gaussian, rectangular/boxcar, or a file-defined kernel).

The convolution methods allow interpolation of fractional peak indices, which are then mapped back onto the original axis.

Choosing the Right Parameter Values for Smoothing and Peak-Detection Kernels: For different kernels, the key parameters can be chosen based on the expected peak width or smoothing scale W (in samples):

- **Mexican Hat kernel:** The parameter σ determines the scale of the kernel, i.e. how wide its central peak extends. For peak detection, a practical choice is

  $$\sigma \approx \tfrac{W}{3},$$

  where W denotes the full width of the harmonic peak. The kernel length is then set to $L \approx 6\sigma$, rounded up to the next odd integer to ensure symmetry.

- **Sinc low-pass kernel:** The cutoff frequency controls how the filter suppresses high-frequency components. For a desired smoothing width W,

  $$\text{cutoff} \approx \frac{1}{2W}, \qquad L \approx 6W,$$

  again with L rounded up to the next odd integer.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-THAG004

- **Gaussian kernel:** For a Gaussian smoothing kernel, the scale parameter σ sets the degree of smoothing. If a full width at half maximum (FWHM) W_{FWHM} is specified,

$$\sigma = \frac{W_{\mathrm{FWHM}}}{2\sqrt{2\ln 2}} \approx \frac{W_{\mathrm{FWHM}}}{2.355}.$$

  For a smoothing width W (in samples), a practical rule is

$$\sigma \approx \tfrac{W}{3}, \qquad L \approx 6\sigma,$$

  with L rounded up to the next odd integer to yield a symmetric, centered kernel that spans $\pm 3\sigma$.

In practice the kernels are tweaked to a slightly smaller width for the best results.

Translational Displacement: When signals are shifted, their relative displacement affects the position of the resulting peak in convolution and cross–correlation. In the case of convolution, the translational offsets add up, while in cross–correlation they appear with opposite sign:

$$f_1(\alpha - a) * f_2(\alpha - b) = g(x - (a + b)), \tag{3}$$

$$f_1(\alpha - a) \circ f_2(\alpha - b) = g(x - (a - b)). \tag{4}$$

Thus, translational displacement has to be corrected carefully, as it may arise from both the upsampling process and from the kernel shape itself (e.g. while using a user defined kernel)

Calibration and ULT Update

We map the measured peak position meas_N through the old ULT and then correct for slope and offset. The update procedure depends on which subsystem—undulator or monochromator—is scanned. When one is varied while the other is held fixed, different slices of the overall composition are sampled:

Case 1: Scan Monochromator, Undulator Fixed (Default). The observed peak positions vary with the monochromator setting, mapping the measured peak to photon energy (E), i.e. $\mathrm{meas} \mapsto E$. The correction is determined from two samples (meas_1, E_1) and (meas_2, E_2), scaling the old slope accordingly:

$$\mathrm{slope}_{\mathrm{new}} = \frac{E_2 - E_1}{\mathrm{meas}_2 - \mathrm{meas}_1} \cdot \mathrm{slope}_{\mathrm{old}} \tag{5}$$

$$\mathrm{offset}_{\mathrm{new}} = \mathrm{slope}_{\mathrm{old}} \cdot E_1 + \mathrm{offset}_{\mathrm{old}} - \mathrm{slope}_{\mathrm{new}} \cdot \mathrm{meas}_1 \tag{6}$$

Case 2: Scan Undulator, Monochromator Fixed (Operands Swapped). Observed peaks vary with the undulator setting, mapping the energy (E) to the measured peak position, i.e. $E \mapsto \mathrm{meas}$. Here, the roles of energy and measured positions are interchanged. The correction is determined from the two points (E_1, meas_1) and (E_2, meas_2):

$$\mathrm{slope}_{\mathrm{new}} = \frac{\mathrm{meas}_2 - \mathrm{meas}_1}{E_2 - E_1} \cdot \mathrm{slope}_{\mathrm{old}} \tag{7}$$

$$\mathrm{offset}_{\mathrm{new}} = \mathrm{slope}_{\mathrm{old}} \cdot \mathrm{meas}_1 + \mathrm{offset}_{\mathrm{old}} - \mathrm{slope}_{\mathrm{new}} \cdot E_1 \tag{8}$$

Slope uncertainty depends on the spacing $E_2 - E_1 = \Delta E$ of the peaks. A small difference amplifies errors. The offset is sensitive to absolute point positions, where small slope changes cause large shifts.

Symmetry of the Scan Modes The measured intensity depends only on the relative shift between the undulator emission profile and the monochromator's spectral bandwidth. Whether we scan the undulator while keeping the monochromator fixed, or vice versa, we are probing the same overlap function but with opposite sign of the shift. This is exactly the property of the cross-correlation [11]:

$$(f_1 \circ f_2)(x) = (f_2 \circ f_1)(-x) \tag{9}$$

Equation 9 shows that exchanging which of the two signals is moved (undulator or monochromator) flips the sign of the position variable x. Thus, the two scan modes yield spectra that are mirror images with respect to a common matching position. If either subsystem is miscalibrated, the peak positions and the center of symmetry is displaced accordingly. Figure 1 shows this effect on experimental data.

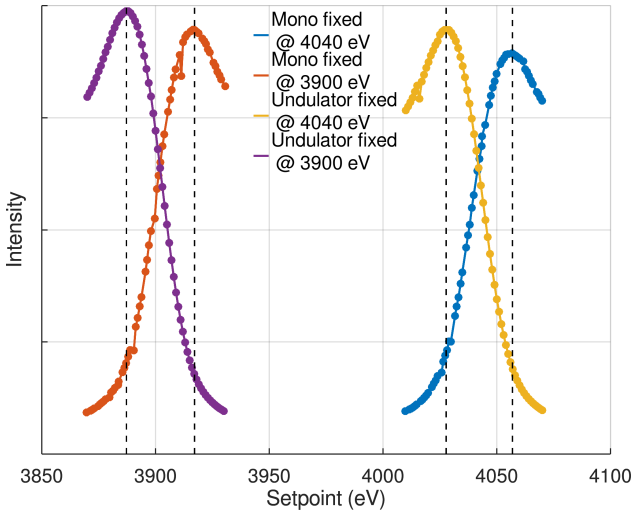

Figure 1: Comparison of two scan modes: (i) scanning the undulator while keeping the monochromator fixed, and (ii) scanning the monochromator while keeping the undulator fixed. Both cases are shown around 3900 eV and 4040 eV. With optimal alignment, the maxima should align at the two energies, and the resulting scans should appear as mirror images around the common energy point.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-THAG004

One-Button Alignment Workflow

The state machine implements a one button workflow (Fig. 2): initiating data acquisition, evaluating spectral response, applying correction parameters to lookup tables. A special confirm state can be used to wait for user input. It can go backward (fail) forward (confirm) and do sanity checks. For unattended runs it can let the state machine auto-advance to the next state after a configurable timeout if no explicit confirm/stop event arrives.

1. **Scans**: refined scans around the peaks (step scan or continuous mode).
2. **Analysis**: peak detection via the configured strategy.
3. **Calibration**: computation of *slope* and *offset* to update the undulator lookup table parameters.
4. **Verification**: optional scan test. That can include additional parallel axes like a piezo stage for the DCM.

Each run produces a manifest YAML capturing parameters and meta data.

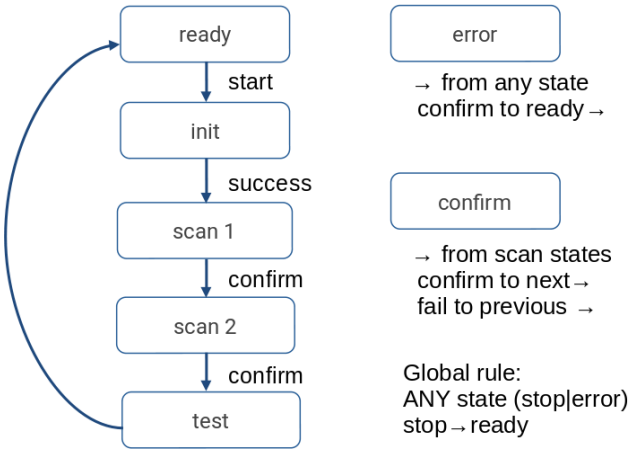

Figure 2: HASMI flow chart. Any state can transition to ready state or error state. In the confirm state, user interaction can move backward or forward within the defined state loop.

Post-Mortem Evaluation of Scan Data

Sometimes it is necessary to perform a *post-mortem* evaluation of scan data. This can be required, for example, to improve the undulator lookup tables themselves or to compare different peak-finding strategies for robustness. Such offline analysis also ensures reproducibility: the same data can be re-processed with identical parameters or new methods to test their reliability and performance.

Workflow Overview An example workflow for offline evaluation is file-based and independent of the online control stack:

1. **Run the analyzer** with `hasmi-max`, selecting a suitable max-finder method and kernel parameters.
2. **Use presets** via `hasmi-max-preset` for consistent and convinient re-analysis across many files.
3. **Save results** in machine-readable YAML files and optional plots (PDF/PNG).

Peak Finding With the data file, the harmonic analyzer is run via `hasmi-max`:

```
hasmi-max --files _emil_2025_440_id.dat \
          --intensity kth09 \
          --gap U17:gap \
          --energy idenergy \
          --ring-current ringc \
          --method CUSTOM \
          --kernel mexican_hat \
          --sigma 2000 --length 12001 \
          --scale-factor 200 \
          --yaml \  # output to yaml
          --plot    # plot result
```

Input parameters: input scan file, intensity, gap, energy, and ring-current columns; peak-finding method `CUSTOM` with a Mexican Hat kernel (σ = 2000, L = 12001) as discussed in subsection "Signal Processing"; upsampling factor 200 (see subsection "Pre-Processing"). The wrapper `hasmi-max-preset` provides tailored defaults for a beamline, e.g.:

```
hasmi-max-preset --preset emil-mexhat \
  --files _emil_2025_440_id.dat
```

This guarantees reproducible results and allows convenient batch-processing. Integrated bash completion further improves usability. The result is demonstrated in Fig. 3.

RESULTS

The EMIL undulator CPMU17 uses a triangulator-based encoder system for gap position feedback, which has a relatively low resolution of 1 µm. This limits the minimum practical step size for undulator scans. At the same time, simply scanning the monochromator and increasing the number of scan points is not attractive, since it lengthens data acquisition.

To enhance reproducibility, we use HASMI's custom convolution-based max finder, which returns a fractional peak index between measured data points. For example, Gaussian smoothing (Fig. 4) enables sub-sample localization of the maximum position, reducing sensitivity to encoder step size.

Figure 1 illustrates the effect of misalignment. Scans can be performed in either direction: by keeping the monochromator fixed and varying the undulator, or vice versa. The resulting spectra appear mirrored and shifted away from the center, but both approaches allow HASMI to extract consistent slope and offset values.

A particularly powerful method is the Mexican Hat kernel (Ricker wavelet). This filter has the property that, when

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-THAG004

tuned to the approximate width of the harmonic peak, it amplifies the principle maximum while suppressing spurious or broader side peaks originating from off-axis radiation. Unlike Gaussian smoothing, which primarily reduces noise, the Mexican Hat emphasizes peak-like structures by its zero-mean shape, characterized by a positive central maximum and negative flanking regions. This makes it especially effective when scanning over a large energy range where the true harmonic peak may appear only as a local maximum (Fig. 3). In such cases, the Mexican Hat filtering increases robustness: it guides the analyzer toward the correct harmonic, improving both reliability and automation of the ULT update procedure as shown in Fig. 1.

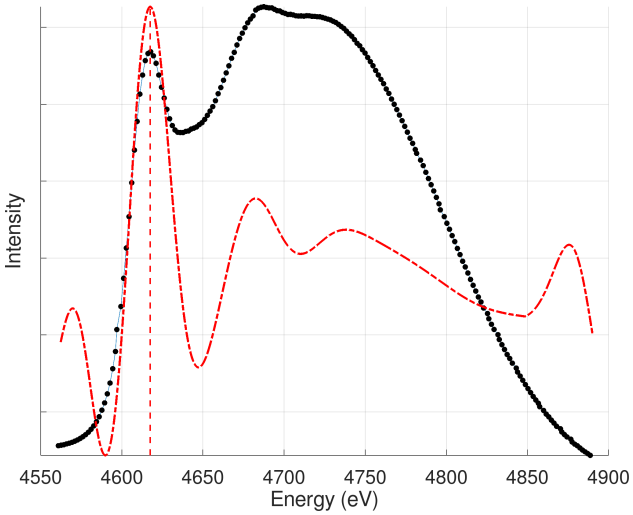

Figure 3: Application of a Mexican Hat custom kernel in the max finder, finding the local maximum of the main peak while suppressing broader neighboring peaks.

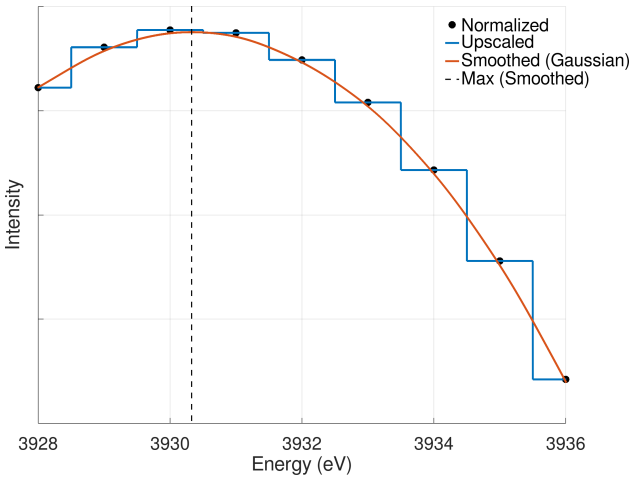

Figure 4: Application of Gaussian smoothing, enabling subsample localization of the maximum.

CONCLUSIONS

We have presented the design and implementation of HASMI, a configurable Python framework for the automated optimization of undulator–monochromator coordination. The system has been deployed at the EMIL CPMU17–DCM beamlines and integrated into user operation; an additional variant is available for the UE48/PGM branch.

HASMI integrates tightly with EPICS, providing both a command-line interface and a process-variable interface. This enables either direct "one-button" user operation or seamless embedding into higher-level environments such as Bluesky, SPEC, or other frameworks. Declarative programming through YAML configuration files and reusable templates allows experimenters to adapt and extend automation scenarios to different devices, actuators, or scan modes with minimal effort. Preset databases, which combine built-in defaults and user-defined entries, provide an additional layer of flexibility and reproducibility.

The underlying `scan` and `scheduler` packages contribute essential functionality: flexible multi-actuator scanning in both stepwise and continuous modes, pluggable monitoring and logging, and asynchronous orchestration of device sequences. These capabilities enable HASMI to execute coupled alignment tasks on undulators and monochromators efficiently and reproducibly.

On the analytical side, the harmonic analyzer achieves precise peak detection even in noisy data. Strategies based on convolution with tailored kernels, such as the Mexican Hat, allow robust identification of local maxima in complex datasets. The modular software architecture permits selective reuse of components, ranging from scanning and scheduling to data analysis and plotting, across different beamline contexts.

Overall, HASMI demonstrates that a modular, declarative, and EPICS-integrated approach can significantly improve the speed, robustness, and user-friendliness of beamline operations, while remaining extensible for future requirements.

ACKNOWLEDGEMENTS

We thank Stefan Gottschlich for clarifying CPMU17 limitations and Mihaela Gorgoi for UE48 requirements. We are grateful to Ruslan Ovsyannykov for IT support, to Eva Lott for her work with BlueSky integration, and to Peter Baumgärtel and Jens Viefhaus for their encouragement and support. Many other colleagues at BESSY II contributed with feedback and discussions, which we gratefully acknowledge.

REFERENCES

[1] R. Müller *et al.*, "Experimental Data Taking and Management: The Upgrade Process at BESSY II and HZB", in *Proc. ICALEPCS'23*, Cape Town, South Africa, Oct. 2023, pp. 84–89. doi:10.18429/JACoW-ICALEPCS2023-MO2A004

[2] Bluesky Project, https://blueskyproject.io/

[3] Experimental Physics and Industrial Control System, https://epics.anl.gov/

[4] J. O. Hill. "Channel access: A software bus for the LAACS,"*Nucl. Instrum. Methods Phys. Res., Sect. A*, vol.

ISBN: 978-3-95450-255-4 ISSN: 2226-0358 doi: 10.18429/JACoW-ICALEPCS2025-THAG004

293, no. 1–2, pp. 352–355, Aug. 1990. doi:10.1016/0168-9002(90)91459-o

[5] YAML Ain't Markup Language (YAML®), YAML Language Development Team, https://yaml.org/spec/1.2.2/

[6] Jinja, Pallets Team, https://jinja.palletsprojects.com/

[7] Ophyd, Bluesky Project, https://blueskyproject.io/ophyd/

[8] M. Heesen *et al.*, "Pysmlib: A Python Finite State Machine Library for EPICS", in *Proc. ICALEPCS'21*, Shanghai, China, Feb. 2022, pp, 330-336. doi:10.18429/JACoW-ICALEPCS2021-TUBL05

[9] A. Björling *et al.* "Contrast – a lightweight Python framework for beamline orchestration and data acquisition", J. Synchrotron Radiat., vol. 28, no. 4, pp. 1253–1260, Jun. 2021. doi:10.1107/s1600577521005269

[10] C. R. Harris *et al.*, "Array programming with NumPy,"*Nature*, vol. 585, pp. 357–362, Jun. 2020. doi:10.1038/s41586-020-2649-2

[11] R. N. Bracewell, *The Fourier Transform and Its Applications*, 3rd ed., New York, NY, USA, McGraw-Hill, 1999.