PYTHON EXPERT APPLICATIONS FOR LARGE BEAM INSTRUMENTATION SYSTEMS AT CERN

J. Martínez Samblas[*], E. Calvo Giraldo, M. Gonzalez-Berges, M. Krupa
European Organization for Nuclear Research (CERN), Geneva, Switzerland

Abstract

In recent years, beam diagnostics systems with increasingly large numbers of monitors, and systems handling vast amounts of data have been deployed at CERN. Their regular operation and maintenance poses a significant challenge. These systems have to run 24/7 when the accelerators are operating and the quality of the data they produce has to be guaranteed. This paper presents our experience developing applications in Python which are used to assure the readiness and availability of these large systems. The paper will first give a brief introduction to the different functionalities required, before presenting the chosen architectural design. Although the applications work mostly with online data, logged data is also used in some cases. For the implementation, standard Python libraries (*e.g.* PyQt, pandas, NumPy) have been used, and given the demanding performance requirements of these applications, several optimisations have had to be introduced. Feedback from users, collected during the first year's run after CERN's Long Shutdown period and the 2023 LHC commissioning, will also be presented. Finally, several ideas for future work will be described.

INTRODUCTION

The LHC is renowned for generating substantial amounts of data through particle collisions. However, it is often overlooked that a significant stream of data is generated by the numerous Beam Instrumentation (BI) systems deployed to monitor, control, and ensure the smooth operation of the accelerators.

This paper primarily focuses on two BI systems: the Diamond Beam Loss Monitors (Diamond BLMs) [1] and the Beam Position Monitors (BPMs) [2]. These large systems present serious challenges due to their extensive data production. On the one hand, despite their limited number (17 across all accelerators), Diamond BLMs can buffer millions of samples per cycle. Conversely, while BPMs individually produce less data, their vast quantity (over 1000 deployed in the LHC) contributes to a massive overall data volume.

In response to the lack of software solutions dedicated to addressing these highly demanding systems, a suite of Python applications has been developed under a set of specific mandates. Firstly, the programs must provide the flexibility to monitor all devices through a unified, user-friendly interface. Speed is also essential, not just in terms of data processing and real-time efficiency, but also in expediting system processes such as commissioning, diagnostics, and fine-tuning, which can otherwise become tedious and time-consuming. Lastly, adopting a data-driven approach is imperative to ease maintenance and ensure scalability in the future.

LARGE BEAM INSTRUMENTATION SYSTEMS

Diamond BLM System

The LHC and SPS, along with the SPS transfer lines, are equipped with 17 Diamond BLMs. These detectors, made of diamond crystals with gold electrodes polarised at 500 V, are strategically positioned to offer a time resolution of 1.53 nanoseconds, enabling precise bunch-by-bunch loss measurements. Acquired signals are digitised at 650 MSPS, with the system supporting five parallel acquisition modes logging at 1 Hz and an on-demand mode for time windows of several milliseconds.

Diamond BLMs play a crucial role in analysing beam transfer efficiency and kicker time alignment at injection and extraction lines. In betatron collimation regions, they monitor losses throughout the entire beam cycle, which are generally associated with physical phenomena such as beam-beam interactions, electron cloud effects, and tune drifts, among others.

LHC BPM System

To guarantee the safe and efficient operation of the accelerator, the LHC is armed with over 1000 BPMs distributed throughout the beam line for continuous and precise measurements of the beam's transverse position within the vacuum chamber. BPMs are electromagnetic sensors that non-destructively couple to the electromagnetic field generated by the passing beam. The analogue signals produced by these devices are first processed via analogue front-end electronics located within the LHC tunnel. Subsequently, this processed output is transmitted to the surface-level back-end electronics, where the beam position information is digitised.

Data from each BPM is acquired independently through a dedicated platform, resulting in a large network of devices streaming data at 50 Hz. To mitigate the cost and complexity of the computing infrastructure, nearby BPM acquisition boards are controlled by a shared CPU. This arrangement effectively reduces the number of logical devices recognised by the control infrastructure to 70.

The BPM system is crucial for the LHC operation. Besides continuously streaming the average beam position data, the system offers several other functionalities. For instance, it can capture positions calculated for a selected subset of LHC bunches over a specified number of consecutive turns.

[*] javier.martinez.samblas@cern.ch

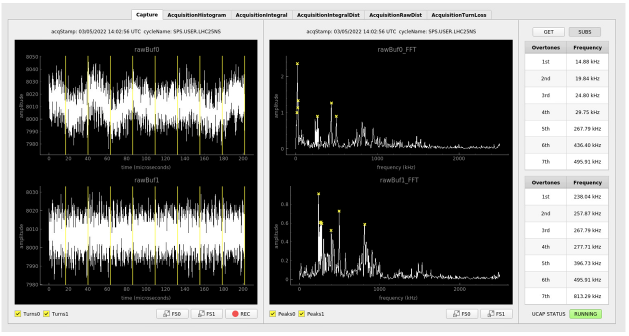
The measurements from the LHC's BPMs are frequently used to optimise the accelerator performance and diagnose anomalies.

ONLINE APPLICATIONS

Online applications are specifically designed to handle real-time data, streaming directly from devices via the Front-End Software Architecture (FESA) [3]; FESA is a CERN-developed framework that provides a consistent approach to the design and implementation of software for equipment controllers. Online applications serve during both commissioning and operational periods, and they have the capacity to monitor and control all devices simultaneously. While designing this software, substantial emphasis was placed on displaying the overall status of the system at a glance, enabling swift and easy intervention in case of issues.

Diamond BLM Expert Application

The main view of the Diamond BLM application presents a summary of all devices, offering a snapshot of the currently running modes along with the most important general information readouts. In addition to the summary views, users can select individual devices to visualise raw buffer plots over time, including turn flags. If a more detailed, bunch-by-bunch visualisation is required, a zoomable extra view is available for in-depth analysis.



Overtones	Frequency
1st	14.88 kHz
2nd	19.84 kHz
3rd	24.80 kHz
4th	29.75 kHz
5th	267.79 kHz
6th	436.40 kHz
7th	495.91 kHz

Overtones	Frequency
1st	238.04 kHz
2nd	257.87 kHz
3rd	267.79 kHz
4th	277.71 kHz
5th	396.73 kHz
6th	495.91 kHz
7th	813.29 kHz

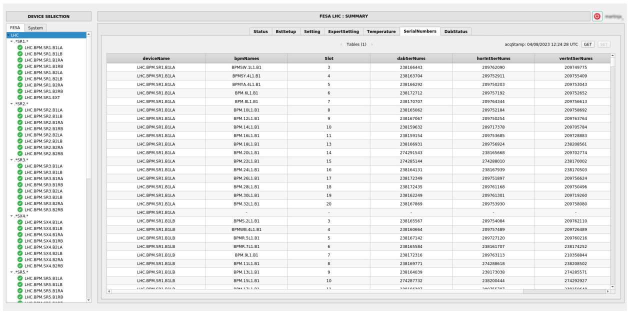
Figure 1: Diamond BLM Capture Window. Left plots depict the losses for both raw buffers, while the right ones display their corresponding FFTs. Turn flags and FFT harmonics are shown in yellow.

As illustrated in Fig. 1, frequency spectrums are displayed alongside the raw buffers. Fast Fourier Transforms (FFTs) are calculated independently on Unified Controls Acquisition and Processing (UCAP) [4] nodes, facilitating the distribution of computational resources; UCAP is a framework designed to enhance the effectiveness of the data processing pipeline within the CERN Accelerator Control System. It is proficient in handling the common "Acquisition - Transformation - Publishing" scenarios. Beyond mere FFT calculations, the applied methodology also incorporates peak detection, the removal of noisy harmonics inherent to the system (*e.g.* 25 MHz), and the integration of spectrums.

Lastly, the application embeds a comprehensive panel that automates the phase-in of the devices. A common challenge with these systems is the potential desynchronisation between the losses signal and the bunch and turn clocks, often due to prolonged cable lengths and processing chain complexity. The implemented algorithm addresses this by adjusting the phase-in parameters, seeking to align the Beam Current Transformer (BCT) pattern with the signal peaks of the first bunch.

LHC BPM Expert Application

The LHC BPM application serves as the central component of the newly revamped Python ecosystem, which also includes a toolkit of offline applications (refer to the subsections "LHC BPM Capture Analysis Tool" and "LHC BPM Auxiliary Tools" for more details). This master application effectively retrieves, processes, and presents the extensive data generated by the numerous BPMs at the LHC.



acqStamp: 04/08/2023 12:24:28 UTC

deviceName	bpmNames	Slot	dabSerNums	horIntSerNums	verIntSerNums
LHC.BPM.SR1.B1LA	BPMSW.1L1.B1	3	238166443	209762090	209749775
LHC.BPM.SR1.B1LA	BPMSY.4L1.B1	4	238163704	209752911	209755409
LHC.BPM.SR1.B1LA	BPMYA.4L1.B1	5	238166292	209750203	209753043
LHC.BPM.SR1.B1LA	BPM.6L1.B1	6	238172712	209757192	209752652
LHC.BPM.SR1.B1LA	BPM.8L1.B1	7	238170707	209764344	209756613
LHC.BPM.SR1.B1LA	BPM.10L1.B1	8	238165062	209752184	209758692
LHC.BPM.SR1.B1LA	BPM.12L1.B1	9	238167067	209750254	209763764
LHC.BPM.SR1.B1LA	BPM.14L1.B1	10	238159632	209717378	209705784
LHC.BPM.SR1.B1LA	BPM.16L1.B1	11	238159154	209753685	209728883
LHC.BPM.SR1.B1LA	BPM.18L1.B1	13	238166931	209756924	238208561
LHC.BPM.SR1.B1LA	BPM.20L1.B1	14	274291543	238165668	209702774
LHC.BPM.SR1.B1LA	BPM.22L1.B1	15	274285144	274288010	238170002
LHC.BPM.SR1.B1LA	BPM.24L1.B1	16	238164131	238167939	238170503
LHC.BPM.SR1.B1LA	BPM.26L1.B1	17	238172349	209751897	209756624
LHC.BPM.SR1.B1LA	BPM.28L1.B1	18	238172435	209761168	209750496
LHC.BPM.SR1.B1LA	BPM.30L1.B1	19	238162249	209761301	209719260
LHC.BPM.SR1.B1LA	BPM.32L1.B1	20	238167869	209753930	209758080
LHC.BPM.SR1.B1LA	-	-	-	-	-
LHC.BPM.SR1.B1LB	BPMS.2L1.B1	3	238165567	209754084	209762110
LHC.BPM.SR1.B1LB	BPMWB.4L1.B1	4	238160664	209757489	209726489
LHC.BPM.SR1.B1LB	BPMR.5L1.B1	5	238167142	209727120	209760216
LHC.BPM.SR1.B1LB	BPMR.7L1.B1	6	238165584	238161707	238174252
LHC.BPM.SR1.B1LB	BPM.9L1.B1	7	238172316	209763113	210358844
LHC.BPM.SR1.B1LB	BPM.11L1.B1	8	238169771	274288618	238208502
LHC.BPM.SR1.B1LB	BPM.13L1.B1	9	238164039	238173038	274285571
LHC.BPM.SR1.B1LB	BPM.15L1.B1	10	274287732	238200444	274292927

Figure 2: Screenshot of the LHC BPM Expert Application. The tree view on the left represents the list of all FESA devices, while the main panel on the right shows the summary view of all BPMs in the LHC.

At the summary level (Fig. 2), the application displays the operational status of all active devices, allows customisable groupings by criteria (*e.g.* by location in the accelerator ring, by BPM type), and facilitates the execution of multi-commands such as concurrent multi-SETs across different Front-End Controllers (FECs) and BPMs. The capability to simultaneously control multiple devices significantly boosts efficiency, leading to considerable time savings in the overall system adjustments and monitoring process.

At the individual BPM level, the application provides real-time visualisations of position data, offering a detailed perspective on each device's performance, along with tables presenting all FESA properties. Moreover, users can access a panel displaying the history of the serial numbers for installed hardware. This allows to easily track past changes in the system, such as the replacement of malfunctioning BPM electronics.

OFFLINE APPLICATIONS

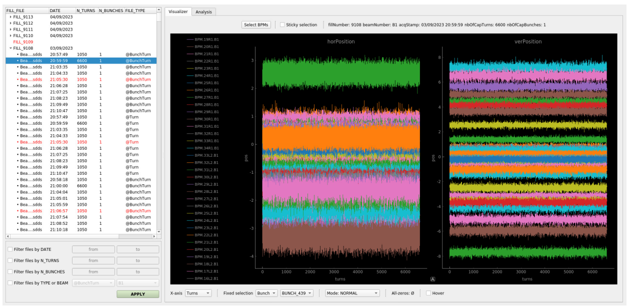
In addition to online applications, offline applications are key for conducting thorough, post-operational analyses of logged data. Offline applications typically read data stored in different formats such as the Hierarchical Data Format V5 (HDF5) [5] or Self-Describing Data Sets (SDDS) [6], as well as data logged in NXCALS [7], CERN's Spark-based logging system.

LHC BPM Capture Analysis Tool

The LHC BPM Capture Analysis Tool (Fig. 3) is designed for analysing and exploring capture data coming from SDDS files organised per LHC fill. This tool features a file explorer that supports filtering based on criteria such as date, number of turns, and number of bunches, among others. It offers an interface for displaying the data in various forms, enabling x-axis changes to visualise the data per BPM, turn, or bunch (useful for evaluating bunch trains), along with a set of options to inspect the FFTs. Additionally, the tool encompasses a series of outlier detection algorithms employed to identify anomalous or malfunctioning BPMs. Currently, three such anomaly detection algorithms are incorporated:

- **Spike detection:** This method attempts to identify unexpected peaks in the position data by calculating the Z-Score and checking for points that exceed a certain threshold.
- **N-zeros:** Essentially, any BPMs showing all-zeros, or a large range of zeros, in any of their planes indicate potentially corrupted data.
- **Tune comparison:** This approach begins by calculating the FFTs, after DC offset removal. The highest peak is then identified using Jacobsen [8] interpolation for minute precision. This peak represents the tune of the beam and should be consistent for all BPMs within the same plane. Consequently, any BPM with a differing tune is flagged as an outlier.

Figure 4 depicts a typical scenario for a malfunctioning BPM. As demonstrated in the image, the FFT exhibits spectral leakage, highlighted by a peak at the Nyquist frequency. This common scenario often suggests that the FFT might not have been computed accurately, probably due to some form of clipping or distortion in the source data. Upon examination of the horizontal positions, it becomes apparent that



FILL_FILE	DATE	N_TURNS	N_BUNCHES	FILE_TYPE
FILL_9113	04/09/2023			
FILL_9112	04/09/2023			
FILL_9111	04/09/2023			
FILL_9110	04/09/2023			
FILL_9109				
FILL_9108	03/09/2023			
Bea....sdds	20:57:49	1050	1	@BunchTurn
Bea....sdds	20:59:59	6600	1	@BunchTurn
Bea....sdds	21:03:35	1050	1	@BunchTurn
Bea....sdds	21:04:33	1050	1	@BunchTurn
Bea....sdds	21:05:30	1050	1	@BunchTurn
Bea....sdds	21:06:28	1050	1	@BunchTurn
Bea....sdds	21:07:25	1050	1	@BunchTurn
Bea....sdds	21:08:23	1050	1	@BunchTurn
Bea....sdds	21:09:49	1050	1	@BunchTurn
Bea....sdds	21:10:47	1050	1	@BunchTurn
Bea....sdds	20:57:49	1050	1	@Turn
Bea....sdds	20:59:59	6600	1	@Turn
Bea....sdds	21:03:35	1050	1	@Turn
Bea....sdds	21:04:33	1050	1	@Turn
Bea....sdds	21:05:30	1050	1	@Turn
Bea....sdds	21:06:28	1050	1	@Turn
Bea....sdds	21:07:25	1050	1	@Turn
Bea....sdds	21:08:23	1050	1	@Turn
Bea....sdds	21:09:49	1050	1	@Turn
Bea....sdds	21:10:47	1050	1	@Turn
Bea....sdds	20:58:18	1050	1	@BunchTurn
Bea....sdds	21:00:00	6600	1	@BunchTurn
Bea....sdds	21:04:04	1050	1	@BunchTurn
Bea....sdds	21:05:01	1050	1	@BunchTurn
Bea....sdds	21:05:59	1050	1	@BunchTurn
Bea....sdds	21:06:57	1050	1	@BunchTurn
Bea....sdds	21:07:54	1050	1	@BunchTurn
Bea....sdds	21:08:52	1050	1	@BunchTurn
Bea....sdds	21:10:18	1050	1	@BunchTurn

Filter files by DATE from to
Filter files by N_TURNS from to
Filter files by N_BUNCHES from to
Filter files by TYPE or BEAM @BunchTurn B1

fillNumber: 9108 beamNumber: B1 acqStamp: 03/09/2023 20:59:59 nbOfCapTurns: 6600 nbOfCapBunches: 1

X-axis Turns Fixed selection Bunch BUNCH_439 Mode: NORMAL All-zeros: Ø

Figure 3: Screenshot of the LHC BPM Capture Analysis Tool. Left panel displays the file explorer and selector. Plots in the right window represent the horizontal and vertical position data for all BPMs in a file containing 6600 turns.

the acquisition of the signal's negative values is largely incorrect. This anomalous BPM can be identified using either the "N-zeros" or "Tune comparison" algorithms.

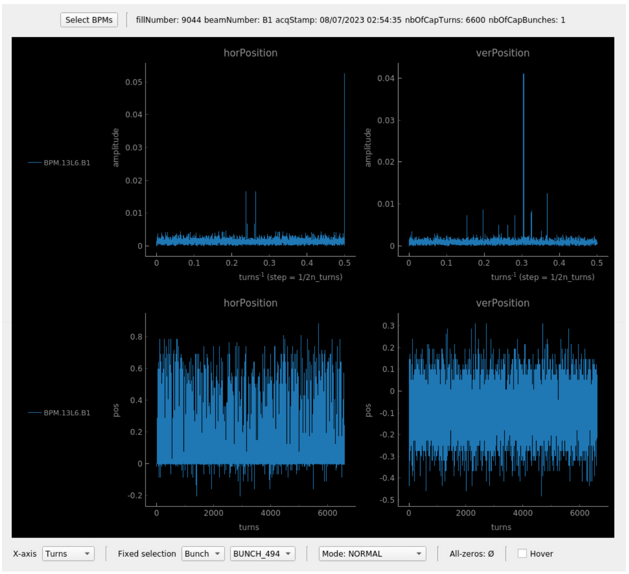
LHC BPM Auxiliary Tools

Apart from the aforementioned applications, the LHC BPM Python ecosystem also includes a set of auxiliary tools: the FIP Expert GUI and the Memory Check Tool. The FIP Expert GUI is principally used to manage Factory Instrumentation Protocol (FIP) devices, which orchestrate various control operations, such as the calibration of the BPMs. Alternatively, the Memory Check Tool performs memory integrity checks for FECs, identifying failing Logical Unit Numbers (LUNs), bits, and their associated memory addresses. Initiating memory checks is made simpler by predefined test templates for both horizontal and vertical planes.

Data Analysis and Visualisation Tool (DAVIT)

In some scenarios, such as those involving certain Diamond BLM configurations that exceed FESA's throughput capabilities, online data processing and visualisation become impractical. In these instances, hierarchical formats like HDF5, which are specifically designed to handle big data, offer an optimal solution due to their comprehensive metadata features that aid in data organisation.

The Data Analysis and Visualisation Tool (DAVIT) is a versatile program designed to address this exact challenge, enabling users to create comprehensive visualisations in a variety of formats, including tables, scatter plots, and single or multi-axis line plots. Leveraging the power of hierarchical structuring, the tool offers users the ability to filter files based on metadata values, allowing for the creation of custom sets of groups and datasets.



fillNumber: 9044 beamNumber: B1 acqStamp: 08/07/2023 02:54:35 nbOfCapTurns: 6600 nbOfCapBunches: 1

X-axis Turns Fixed selection Bunch BUNCH_494 Mode: NORMAL All-zeros: Ø

Figure 4: Anomaly Detection Example. The bottom images illustrate the horizontal and vertical positions for BPM.13L6.B1, while the top images display their corresponding FFTs.

Each resulting group or dataset is treated as a pandas [9] dataframe. This approach simplifies data manipulation, permitting the combination and merging of multiple datasets, as well as supporting built-in operations like matrix transposition and array slicing. It also augments the modularity of the visualisation components, making them independent and reusable, as they only need to handle input dataframes regardless of the data's original source or dimensions. Besides, the application has been extended to fetch data from NXCALS and is expected to handle data from the PostMortem system in future updates.

IMPLEMENTATION

PyQt5 has been selected as the main library for building the expert applications. As a Python binding of the cross-platform toolkit Qt, PyQt is widely used in the industry for large-scale applications, thanks to its mature and stable environment. Moreover, it is gradually becoming a standard at CERN's accelerator teams, reinforcing its suitability for the task at hand.

PyQt is both modular and easily controllable, offering extensive flexibility for application development. It incorporates the model/view design pattern, which separates the GUI's data (model) from its visual representation (view), enhancing coding efficiency. Unfortunately, PyQt's default models are not devised for managing large datasets, so custom models and widgets had to be developed to properly handle the abundant data produced by the BI systems. Notably among these is a tailored table that utilises pandas dataframes as the primary source for its model. It incorporates numerous practical features such as filtering, regular expression searching, natural sorting, and type conversions (*e.g.* decimal to binary). As shown in Table 1, this custom table substantially outperforms the default PyQt table, offering faster loading times and reduced memory consumption. Specifically, the pandas table exhibits a more linear increase in loading times and consumes approximately x4 less memory.

Table 1: Performance comparison between the default PyQt table (left) and the implemented pandas table (right). Size refers to the number of elements in a table containing only strings.

Size	**Loading Time (s)**	**Memory (MB)**
10,000	0.03 / 0.02	3.87 / 1.02
100,000	0.37 / 0.03	39.96 / 10.97
1,000,000	5.50 / 0.14	411.06 / 104.93
10,000,000	50.76 / 1.53	4033.87 / 1004.49

As for graphics, PyQtGraph was chosen as the main library for generating plots due to its inherent capability of providing rapid and interactive data visualisations. It is particularly efficient when dealing with medium-sized data curves, accommodating up to 1 million samples adequately. However, when handling larger datasets, especially those exceeding 10 million samples such as the data encountered in the Diamond BLMs, its built-in downsampling algorithm begins to lag the applications, causing display freezes and rendering them unusable. Therefore, a custom downsampling algorithm had to be implemented to minimise any performance and freezing issues; Given a signal y composed of n data points, the resulting signal y' of N downsampled points can be computed using the following major steps:

1. **Downsampling factor:** The algorithm starts by determining the downsampling factor, denoted as d. This is the number of data points to be combined into one downsampled point. Mathematically, it can be written as:
$$d = \left\lceil \frac{i_2 - i_1}{\mathrm{N}} \right\rceil \tag{1}$$
where i_1 and i_2 are the starting and ending indices of the plot range, respectively, and N represents the desired maximum number of samples.

2. **Regularisation:** Next, both the range of indices and the downsampling factor are regularised to multiples of a power of 2. This ensures that the same data points are always included when the plot range or zoom level changes slightly. It can be expressed as:
$$d \rightarrow 2^{\lceil \log_2(d) \rceil} \tag{2}$$
$$i_1 \rightarrow \left\lfloor \frac{i_1}{d} \right\rfloor \cdot d \tag{3}$$
$$i_2 \rightarrow \left\lceil \frac{i_2}{d} \right\rceil \cdot d \tag{4}$$
where $\lceil \log_2(d) \rceil$ is the bit length of the initial downsampling factor.

3. **Envelope preservation:** The actual downsampling process begins by dividing the data into chunks of size d. For each chunk, the algorithm computes the minimum and maximum y values. These minima and maxima are then interleaved, meaning that the minimum and maximum of each chunk are placed sequentially. This process forms the downsampled signal y', preserving the envelope of the original data while reducing the number of data points.

4. **Caching:** To further improve performance, the algorithm precomputes the downsampled data for various downsampling factors and stores them in a cache. This allows the downsampling step to quickly retrieve the downsampled data from the cache when the zoom level changes, rather than having to fully recompute it each time.

Finally, it is important to emphasise the significance of multi-threading in the implementation of the applications. For tasks such as data synchronisation, the QThread class from PyQt was preferred, primarily because of its native integration with Qt's event loop and signal-slot mechanism.

Multi-threading is also applied within the Java API for Parameter Control (JAPC) [10] and its Python binding, PyJAPC, which is the library employed to fetch the data from the devices.

COMMISSIONING AND OPERATION EXPERIENCE

The Diamond BLM online application has been a very valuable tool during the commissioning phases following the winter shutdown periods of both the LHC and SPS. Given that detectors are situated across different machines and serve various use cases, they require a multitude of configuration parameters that are inherently error-prone when manipulated manually. Thankfully, the overview panel simplifies the configuration process by swiftly identifying and rectifying malfunctioning detectors. Moreover, the ability to locate and group detectors based on either their names or the machines they are installed in is particularly useful, as it allows for the simultaneous execution of custom commands across multiple detectors. The tool is also regularly used during the year to ensure that all continuous modes are active and disseminating data, a task that previously necessitated the opening and monitoring of different interface windows.

Additionally, bunch-by-bunch losses demand fine-tuned time adjustments between the beam synchronous clocks and the detector data stream to compensate for cable delays. This was previously carried out manually by a firmware expert, making it both time-consuming and relatively error-prone. The incorporation of this functionality into the application has streamlined the process, making it more rapid and reliable.

Regarding offline processing, DAVIT has eased the exploration of the extensive on-demand Diamond BLM capture files generated during custom Machine Development (MD) studies. This has led to considerable savings in post-analysis time.

As for the LHC BPM application, the newly developed expert software ecosystem very quickly proved to be fundamental for validating the system after maintenance activities, identifying broken channels, and diagnosing complex performance issues. The tools provide experts with not only a broad overview of the entire BPM system, but also the ability to examine critical low-level details of individual channels within a user-friendly interface. The many built-in data analysis features enable most problems to be studied directly from the GUI. Furthermore, the software simplifies the export of raw data to all standard data formats, facilitating exhaustive offline analysis.

FUTURE WORK

Despite operating within a largely generic environment, applications will entail regular maintenance to keep compatibility with FESA changes as well as updates to other libraries. Moreover, these applications are in a state of perpetual evolution, significantly driven by user needs across the various commissioning and operation periods.

A prime area for enhancement lies within the LHC BPM Capture Analysis Tool, which is scheduled to be upgraded with two new anomaly detection algorithms based on Isolation Forest and SVD dominance analysis. DAVIT might also be expanded to handle new file types (*e.g.* Parquet) and data sources (*e.g.* PostMortem).

CONCLUSIONS

Managing Beam Instrumentation systems poses a challenge due to the substantial number of devices and the vast data volumes they continuously collect. The suite of applications introduced in this paper successfully addresses this challenge by delivering rapid overview information, facilitating data manipulations, enabling automatic anomaly detection, and providing comprehensive visualisations. Performance techniques, such as downsampling and multithreading, have been explored and will be used for future Python developments in other accelerator systems.

REFERENCES

[1] E. Calvo Giraldo *et al.*, “The Diamond Beam Loss Monitoring System at CERN LHC and SPS”, in *Proc. IBIC’22*, Kraków, Poland, Sep. 2022, pp. 202–206. `doi:10.18429/JACoW-IBIC2022-TU2C2`

[2] E. Calvo-Giraldo *et al.*, “The LHC Orbit and Trajectory System”, in *Proc. DIPAC’03*, Mainz, Germany, May 2003, paper PT08, pp. 187–189.

[3] M. Arruat *et al.*, “CERN front-end software architecture for accelerator controls”, in *Proc. ICALEPCS’03*, Gyeongju, Korea, Oct. 2003, pp. 342.

[4] L. Cseppentö and M. Büttner, “UCAP: A Framework for Accelerator Controls Data Processing @ CERN”, in *Proc. ICALEPCS’21*, Shanghai, China, Oct. 2021, pp. 230–235. `doi:10.18429/JACoW-ICALEPCS2021-MOPV039`

[5] M. Folk, A. Cheng, and K. Yates, “HDF5: A file format and I/O library for high performance computing applications”, in *Proc. of Supercomputing*, vol. 99, pp. 5–33.

[6] M. Borland, “A self-describing file protocol for simulation integration and shared postprocessors”, in *Proc. PAC’95*, Dallas, TX, United States, May 1995, vol. 4, pp. 2184–2186. `doi:10.1109/PAC.1995.505492`

[7] J. Wozniak and C. Roderick, “NXCALS - Architecture and Challenges of the Next CERN Accelerator Logging Service”, in *Proc. ICALEPCS’19*, New York, NY, USA, Oct. 2019, pp. WEPHA163. `doi:10.18429/JACoW-ICALEPCS2019-WEPHA163`

[8] E. Jacobsen and P. Kootsookos, “Fast, accurate frequency estimators [DSP Tips & Tricks]”, in *IEEE Signal Processing Magazine*, May 2007, vol. 24, no. 3, pp. 123–125. `doi:10.1109/MSP.2007.361611`

[9] W. McKinney, “Data Structures for Statistical Computing in Python”, in *Proc. SCIPY’10*, Jun. – Jul. 2010, pp. 56–61. `doi:10.25080/Majora-92bf1922-00a`

[10] V. Rapp and W. Sliwinski, “Controls middleware for FAIR”, in *Proc. PCaPAC’14*, Karlsruhe, Germany, Oct 2014, paper WCO102, pp. 4–6.