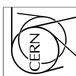
FIRST EXPERIENCE ON PYTHON DEVELOPMENT FOR SURVEY SOFTWARE, ADVANTAGES & DRAWBACKS

SANGLIER Pierre-Adrien, LEWANDOWSKI Przemyslaw, KLUMB Francis, CERN, Geneva, Switzerland

CERN-BE-2023-002
31/10/2022



Abstract

During long shutdown and maintenance periods at CERN, the surveying teams intensively use in-house data processing software, developed in the last decades and mostly written in C++ language.

The accurate measurement of the deviations of hundreds of successive accelerator components with respect to their theoretical positions, in either vertical or radial direction, is part of survey activities. The final smoothing operations consist of mechanically re-positioning some of those components to ensure smooth transitions between elements and to limit optical corrections of the particle beams orbits. RABOT is the survey software currently used to process the measured deviations and provide such smoothed data. Initially designed in Fortran language in the '90s, it had been fully rewritten in C++ ten years later. However, the unnecessary complexity of its code, as well as missing documentation made it difficult to maintain. Based on reverse-engineering methods, it was recently decided to rewrite this essential software using Python language.

The usage of Python as a software development language is a questionable choice from a performance point of view. Indeed, the compiled nature of the C++ language makes it incomparably faster than equivalent algorithms coded in scripting languages such as Python. Nonetheless, there are multiple benefits of choosing Python for simple software development. Amongst them: a more comprehensive syntax and automatized implementation such as memory management and dynamic typing. It can drastically reduce the development time and enhances the legibility for non-expert programmers. As a direct consequence, available resources and development efforts can be oriented towards algorithmic optimization and improvements. Moreover, the development relies on robust and efficient numerical libraries such as NumPy that offer a wide variety of tuneable methods.

The first in-house survey software implemented fully in Python was achieved and performances improved compared to its older C++ implementation.

CONTEXT

The smoothing of an accelerator is the final alignment operation refining the relative positions of its components to avoid substantial displacements between neighboring magnets. Out-of-tolerance positioning could indeed significantly disturb the particle beam.

The CERN surveying teams first measure and compute radial and vertical deviations of the accelerator components with respect to their theoretical positions provided by the beam physicists. Subsequently, they need specific processing software to estimate an ideal smoothed trajectory of the beam through the accelerator from these measured deviations. Regardless of the nominal accelerator geometry (straight line as for beam injecting parts, circular or other shapes), only measured deviations are taken into account in the smoothing process.

The resulting data of this procedure are called "smoothed offsets". They are the radial and vertical differences between measured positions and the estimated smoothed curve representing the trajectory of the beam. Smoothed offsets larger than a fixed tolerance correspond to components to be physically displaced and realigned by surveying teams in the field. This process can be iterative and lead to several smoothing operations while repeating measurements of the successive element positions (see Fig. 1).

A brief history

The first processing software that the surveyors used to compute such smoothed curves and related offsets has been released during the prime time of the LEP accelerator in the 1990s. This program was named PLANE and was written in FORTRAN language.

PLANE relied on least square piece-wise adjustment technique adapted from digital filtering methods. As results for the users, it highlighted the accelerator components to be moved in field to correct the beam trajectory, according to calculated smoothed offsets.

PLANE was a command-line tool requiring additional files generated by various other software that progressively became obsolete. The data processing workflow was quite complex with many different data sources and file formats. To properly maintain and keep developments up-to-date, the software was migrated to a more modern language. PLANE became RABOT around 2010 and was ported to C++. Taking into account new user needs, this modernized version also included new features such as visualization tools and calculation options such as handling weighted input values.

Its documentation mainly describes the software functionalities (input and output values and data formats) which represent obviously the most crucial information for users. However from a developer point of view, the documentation is lacking information. Important employee turnover in recent years has also significantly impeded the knowledge transfer between successive contributors of this project.

Software and related algorithms became difficult to maintain due to the lack of information about the code and internal logic. The software was clearly non-perennial for long-term usage.

The recent need to improve very sensitive parts of the software algorithms precipitated the decision to shorten its life cycle and to find alternative development solutions; a

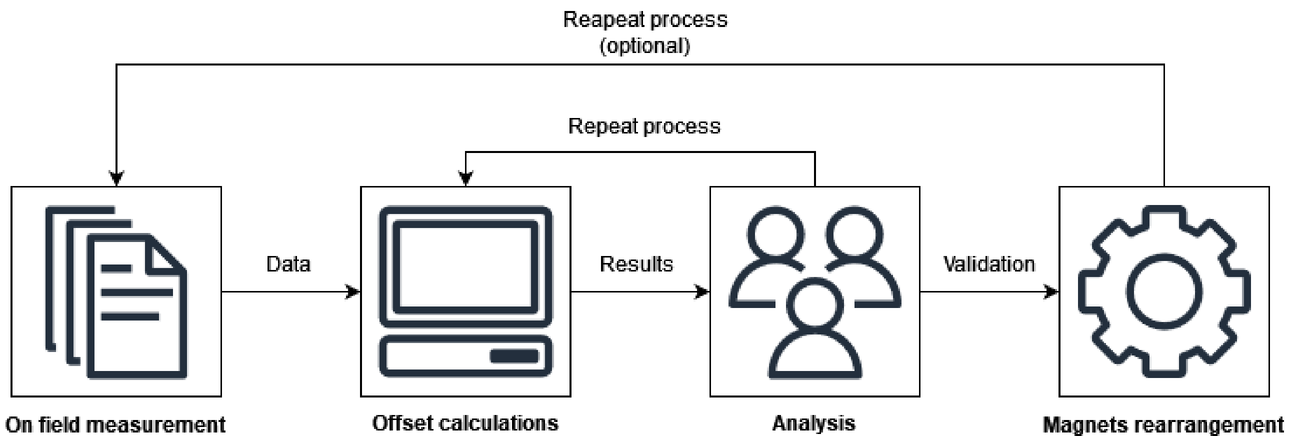


Figure 1: Simplified magnet alignment workflow done by the survey team.

new data smoothing software prototype based on Python language was developed and evaluated in this context.

Algorithm description

From the input data, i.e. a set of measured deviations of accelerator components, the algorithm generates a smoothed curve and provides in the end a list of accelerator components to be mechanically moved in the field (see Fig. 2). This process is always performed separately, either on radial or on vertical deviations. It also treats individual points from measured components, and not the components directly.

The main algorithm is based on successive polynomial regressions, that are applied locally in small consecutive and overlapping segments of the accelerator. Segments are called smoothing windows, that consist of a fixed number of neighboring points measured on the accelerator components. In practice, multiple polynomial regressions with different degrees are applied in each local window, and the "best-fitting" one is selected to calculate the smoothed offset corresponding to the middle point of the window.

The smoothing window slides over a whole accelerator, from one component to the next one, generating the optimized trajectory of the beam from local polynomial regressions. This general principle still remains the basis of PLANE's successors.

The data smoothing process can be described by the following steps or routines:

1. For a given dataset, containing the measured positions of the accelerator components as radial or vertical deviations, a first routine splits the dataset into smaller parts called smoothing windows. The width of these windows is currently determined by a fixed number of measured consecutive points.

2. In each window, the next routine applies polynomial regressions of orders varying from 1 to 5 (see Fig. 4).

3. The following routine compares in each window the polynomial regressions and selects the one with the minimal root mean square error as the “best-fitting” function. It subsequently calculates and saves the offset between the measured position of the window's middle point and its theoretical estimated position on the "best-fitting" curve. This difference is the value of the smoothed offset corresponding to the window's middle point.

4. The last routine selects the point with the highest absolute smoothed offset value and moves it to the corresponding curve position if this value is above a certain tolerance.

The combination of all the previous routines represents the main loop of the algorithm. The loop repeats until all estimated smoothed offsets are below the given tolerance.

```
1:  Split Dataset into equal smaller parts called Windows
2:  while Points should be moved do
3:      for each Window do
4:          Apply multiple polynomial regressions
5:          Select the "best-fitting" polynomial function
6:          Save the theoretical middle point position on the
            best-fitting curve
7:          if |theoretical pos. - measured pos.| > Tolerance
            then
8:              Store in memory the point and its position
9:          end if
10:     end for
11:     Within the stored points, move the point with the
        biggest absolute difference to its theoretical position
12: end while
```

Figure 3: Simplified pseudocode of Rabot C++ algorithm

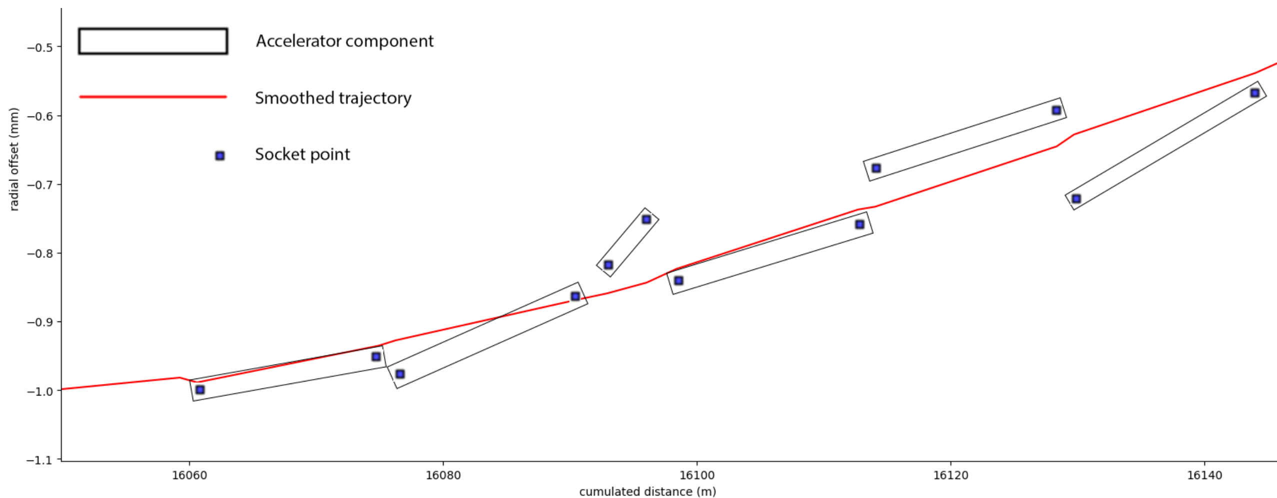


Figure 2: Principle of Rabot calculation algorithm. Socket points are the fiducial points measured on the components.

MAIN OBJECTIVES

Ensuring better maintenance

RABOT is a 30'000 lines of C++ code software that ensures a pretty good separation between visualization methods (data representation) and calculation core. The GUI part has been continuously updated and improved following user needs. This important part of code is based on Qt [1], a popular and well documented GUI framework. On the other side, the calculation core of the code uses no external proven libraries, even for common tasks such as matrix operations or polynomial regression. Moreover, this crucial part of the code was not explicitly documented, nor written using for example self describing variable names. Algorithms have subsequently never been modified nor upgraded in the last decade. The software documentation does not provide helpful information about its code design or theoretical principles.

Implementing new features

As previously explained, the smoothing process relies on a sliding window defined by a fixed number of neighboring components. In practice, its length can vary in terms of metric distances. In case of non-uniform spatial distribution of measured components along the accelerator, the metric length variations can have unexpected effects on the results and lead to inconsistencies.

For example in dense areas comprising numerous contiguous measured components the results will be precise enough and the successive regressions are continuous between them. However, in parts of the dataset with fewer elements, the smoothing leads to discontinuous regressions.

Both calculation options, the "metric window" defined by a distance expressed in meters and the "unit window" defined by a fixed number of components, are illustrated in Fig. 5. The top graph shows two unit windows in a non-homogeneous area. Note that the unit window defined in the less dense region is larger than the one defined in regions with higher density. This anomaly can happen multiple times during the loop, resulting in a loss of accuracy. The second graph shows two windows defined by a metric distance. The generated windows cover the same accelerator section in terms of spatial distances. In some configurations, metric window appears to be a more appropriate and realistic model than unit window.

METHODOLOGY

Our main objective for RABOT was to make its algorithm maintainable again. This has been achieved by deeper understanding of its calculation core and by creating proper documentation.

Choice of Python

We initiated a reverse-engineering process and duplicated the software in Python language. Indeed, the high-level syntax of this programming language makes algorithms easier and faster to write and read for further improvement.

Using Python offers many advantages over low-level languages such as C and C++. Low-level languages will still perform better for hardware interfaces, embedded devices or in contexts where performance and accurate resource management are highly required. However, this is not the priority for RABOT where automated high level mechanisms such as memory management with garbage collector can be applied without any inconvenience.

Common numerical operations can be profitably managed by mathematical libraries such as NumPy [2]. NumPy is an open-source library for Python, comparable to MATLAB [3] since both are interpreted at execution time. Graphical interfaces can be developed using PyQt, which is a binding of C++ Qt library. It thus ensures good performance.

Simplicity in programming is the key. “Simple is better than complex” is the third Zen principle [4], stated by Python's creator. In other words, a simple design always takes less time to finish than a complex one. This idea has

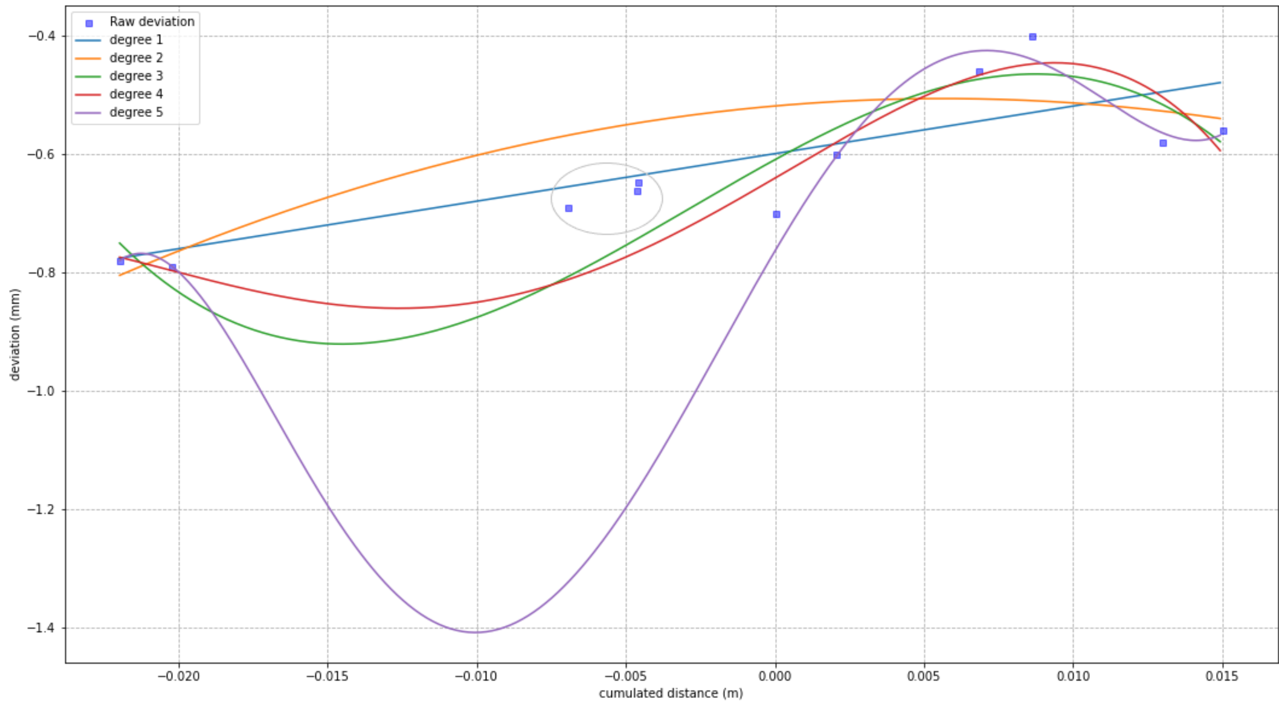


Figure 4: Example of regressions at different degrees for the same window. Circled points weight values are arbitrarily set to 0, they do not contribute to the different regressions but impact the estimated curve.

guided the portage of the existing C++ code to a newer one in Python.

Reverse engineering and porting

A reverse-engineering process has been applied to all methods used by the calculation core. An in-depth look at those routines, combined to the analysis of previous versions of the program, helped understanding most of them. At this first stage of the project, routines were strictly and rigorously translated into Python language and tested one by one.

Using Microsoft Visual Studio [5] tools, such as step-by-step debugging and live-access tools for objects and variables content, a wide range of data has been collected while executing the C++ version of RABOT. Deeper tests have been made on those ported routines using the collected data. Numerical results from a version to another were systematically compared to ensure identical values.

Obviously, since C++ is a compiled language and Python is an interpreted language, translated routines were at first slower than their original ones (see Table 2). Nevertheless, thanks to clearer and readable code, some optimizations could be implemented to simplify the procedures and speed-up calculations. These enhancements will be explained in the next paragraph.

Comparative tests also showed that some original routines relied on numerical approximations. Although the differences of results were not significant for the day-to-day survey and alignment work, considerable time was spent analysing them. To ensure the highest possible precision, ported code has been slightly modified to avoid such approximations, as presented in the next section.

Furthermore, the initial software relied on its own implementation of matrix operations. To provide more standardized behavior and not reinvent the wheel, NumPy library methods were used for common mathematical operations. Results between previous C++ and equivalent Python routines turned out to be strictly identical. Computation time was shorter using NumPy instead of the Python ported implementation of matrix operations. This can be explained by the multiple optimizations available in this widely used numerical libraries.

Once RABOT's code has been ported from C++ to Python and results were strictly compared and analyzed, important efforts were directed towards better documenting algorithms and improvements.

This initial Python software version was not intended to be released in production mode. It served as a basis for further more detailed discussions with users and final decision makers. It was at the same time a proof of concept to decide whether Python could be a pragmatic alternative to some of our developments done or foreseen in C++ environments. As the initial goal was to fully analyze and deeply understand algorithmic parts, this prototype did not provide any advanced GUI feature. A complete and appropriate graphical interface adapted to user needs has been implemented at a further step of the project.

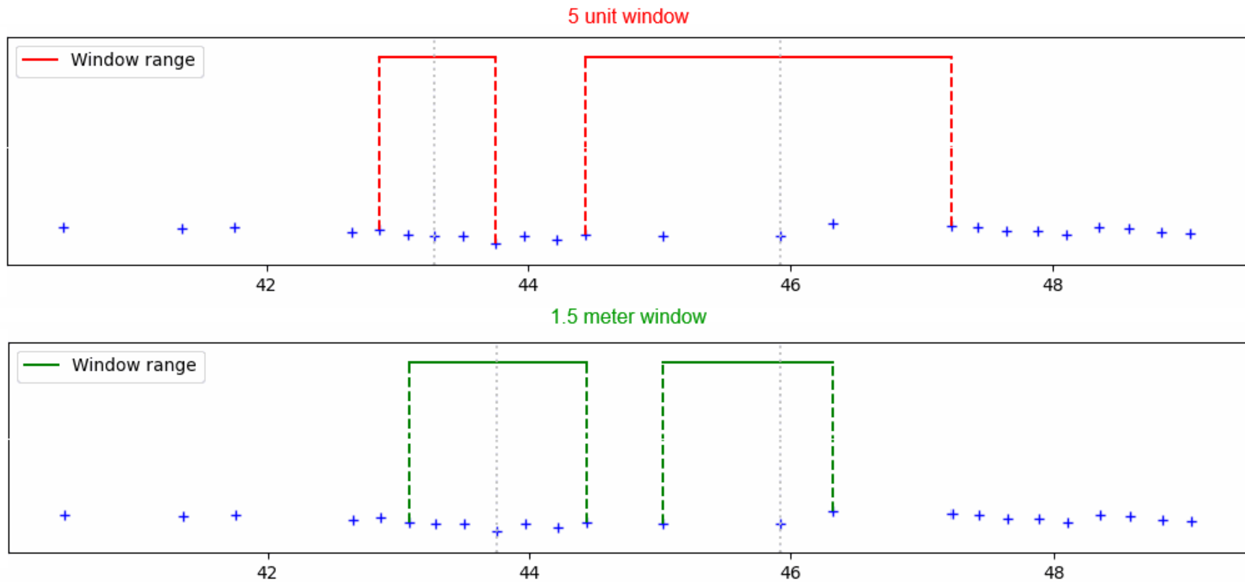


Figure 5: Example of the difference between unit defined windows (top graph) and metric defined windows (bottom graph).

PERFORMANCE OPTIMIZATIONS

The Python software prototype called PyRabot has been enhanced at different levels to become a robust application expected to replace RABOT for future survey activities. It includes new calculations options that have been long requested by the users.

Algorithm enhancement

Among the changes in the algorithms migrated from C++ to Python language, one major modification concerns the management of the weighted input data.

Users have the possibility to assign different weights to the measured positions, depending on the type of accelerator component or the precision of the related observations in the field. As seen in Fig. 4, those weights might have significant effects within the polynomial regression steps and greatly modify the results of the global processing.

In the C++ version, to ignore desired points in the smoothing process, very low weight values (typically 0.001) were set to corresponding measured radial and vertical deviations. Our numerical analysis showed slightly different results by purely suppressing those low-weighted elements from the input dataset. Mathematically results should be identical, but the C++ implementation of the algorithm itself introduced some numerical biases and approximations noticed by users for years but never corrected due to high code complexity. The Python version of the software solves this issue by totally excluding 0-weighted data from the smoothing window and related regressions.

The simplicity with which Python makes it possible to manage and re-arrange lists of objects highly facilitated such quick improvements.

Results

Important optimizations where further performed in the last routine described in Fig. 3 that moves points whose smoothed offset values exceed the tolerance value. In the C++ version, such points are moved one-by-one in the successive loops, the one with highest smoothed offset first, to ensure it will not influence others.

This method only considers the maximal smoothed offset evaluated in the large dataset. It moves the corresponding component to its fitted position before restarting the whole iteration steps described in Fig. 3. By moving simultaneously several components together in each iteration, we considerably decrease the computation time. However this grouping can only be applied for components sufficiently far apart from each other. Moving for example two consecutive components at the same time will not produce identical results than those generated by moving them separately in this iterative process.

Furthermore, our reverse engineering process pointed out a considerable amount of calculations re-done in each iteration while input data remain strictly unchanged. As a matter of fact, sliding the smoothing window over unchanged positions does not affect the various polynomial functions previously estimated by the regression routine.

Two major improvements have thus been performed to drastically improve the execution time:

- The first improvement is to move multiple points at the same time in each iteration step of the loop, after ensuring that they do not influence each other (i.e.: moved points should not be part of the same regressions, so at least spaced twice the size of the smoothing window).
- The second improvement consists in systematically keeping all calculated smoothed offsets in a temporary

data storage (computer memory cache). It makes them directly accessible within the next loop and avoids unnecessary identical data processing. The trade-off for this speed improvement comes at the cost of increased memory usage. Considering largest datasets processed in the LHC accelerator complex (several thousands of components), this increase remains nevertheless acceptable.

Table 1: Files Specifications

	File 1	File 2	File 3	File 4
Total	13170	770	3870	5370
WS	61 (0.5%)	11 (1.5%)	11 (0.3%)	11 (0.2%)
Moved	150 (1.1%)	100 (13%)	27 (0.6%)	63 (1.1%)

Total = total number of points.
WS = window size.
Moved = number of moved points.
Values indicated with % are proportion represented regarding the total number of points in the file.

Table 2: Computation time (value in second)

Algorithm	File 1	File 2	File 3	File 4
Rabot C++	806	10.2	26	27
Rabot Python ported	2103	38	53	173
Rabot Py optimized	235	2.7	6	7.9
Rabot Py opti + cache	38	1	2.3	3.2

Applying those two upgrades in the Python version, execution time drastically reduced by a factor of 10 in average when comparing to C++, see Tables 1 and 2. The ease of Python syntax allowed us to focus on the theoretical part, instead of spending time on pure code implementation.

CONCLUSION AND PERSPECTIVES

A new version of RABOT has been developed in a few months for dedicated CERN survey activities, which includes all the functionalities of previous versions. It also includes several new features and an upgraded algorithm that is faster than the previous C++ one, while being written in Python.

By focusing on the survey business core of the code and not on the languages subtleties, it has been released in a short amount of time. The total number of lines of code has been drasticaly reduced from 30.000 in C++ version to 2.000 of well documented lines in PyRabot. It is more human-readable for both developers and users, which allows us to discuss the code with non-expert programmers. It is also easier to learn and use, so developers can be more efficient and users can understand how the algorithm is implemented in the software. And from the organization and management point of view, the increasing popularity of Python language [7] allows more flexibility to recruit new graduate students in the future.

Using Python for survey development was at first a trial and became a proof of concept. This project size was ideal to do it and allowed to evaluate Python for GUI and calculations core. This PyRabot example concludes that Python can be used for rapid application development. It seems to be a viable solution for projects where performance is not critical. Moreover, Python is supported now at CERN for machine accelerator control [8].

Again, the main reason we achieved those upgrades that quickly is the Python high-level syntax. It makes developing, reading and modifying to the needs of the users easier and faster. Alternative theoretical approaches could also be tested easily, basing calculations on NumPy library, that allows the software to adapt quickly if needed. Optimizing the algorithm and correcting potential errors are also easier. Based on NumPy facilities, other theoretical approaches could also be implemented, evaluated and compared in the future such as adaptive splines [9] or B-splines with penalties to fully automate the process.

REFERENCES

[1] Qt `https://www.qt.io/`

[2] NumPy `https://numpy.org/`

[3] MATLAB `https://www.mathworks.com/products/matlab.html`

[4] Zen of Python `https://en.wikipedia.org/wiki/Zen`$_{o}fPython$

[5] Microsoft Visual Studio `https://visualstudio.microsoft.com/`

[6] GitLab `https://about.gitlab.com/`

[7] Programming language popularity ranking `https://www.tiobe.com/tiobe-index`

[8] Acc-Py environment and packages `https://abpcomputing.web.cern.ch/guides/accpy/`

[9] Vivien Goepp, Olivier Bouaziz, Grégory Nuel. Spline Regression with Automatic Knot Selection. 2018. hal-01853459