AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI
INSTYTUT INFORMATYKI

Projekt dyplomowy

Low-latency alignment software for sensor calibration in the CMS-PPS detector at the CERN laboratory

Oprogramowanie na potrzeby kalibracji sensorów w detektorze CMS-PPS w laboratorium CERN

Autor: Mateusz Kocot
Kierunek studiów: Informatyka
Opiekun pracy: dr Leszek Grzanka

Kraków, 2022

Acknowledgements

I wish to express my deepest gratitude to Jan Kašpar and Valentina Avati, who oversaw my work at CERN. You were always ready to help me in every aspect of the project, including writing this thesis.

I am also very grateful to the thesis supervisor, Leszek Grzanka, who co-organises the PPS and TOTEM internships at AGH UST. You introduced me to the vast CERN computing environment and patiently answered all my questions.

The project was partially funded by the Polish Ministry of Science and Higher Education Grant No. MNiSW DIR/WK/2018/13.

Abstract

The Precision Proton Spectrometer (PPS) is a subsystem of the Compact Muon Solenoid (CMS), which is one of the main detectors at the CERN's Large Hadron Collider (LHC). The PPS sensors are hosted in vessels called Roman Pots (RPs), which are frequently inserted and retracted from the LHC beam pipes. Thus, their shifts with respect to the beam are not constant and have to be calculated regularly, in a procedure called 'alignment'. It has already been designed and implemented but is not automated and can work only a sizeable time after collecting the data. In the LHC Run 3, beginning in 2022, PPS will need a faster way of obtaining the alignment results.

The main goal of this project was to reimplement the alignment procedure so that it could operate within 48 hours after data acquisition. To accomplish that, the software uses the Prompt Calibration Loop (PCL) mechanism, and the code follows the structure of the Data Quality Monitoring (DQM) software. At first, the particle tracks detected in selected RPs are preprocessed in parallel in the worker module. Then, the data are merged, and the harvester module produces the alignment results. The software has been integrated into the CMS software framework – CMSSW, and the alignment configuration is effectively delivered via the CMS database – Conditions DB.

The final product meets all the requirements. It has been thoroughly tested and reproduced the results from the old software.

Contents

1. Project goals and vision

1.1. Introduction

CERN – the European Organisation for Nuclear Research – operates the world's largest particle physics laboratory [1]. It was established in 1954 on the France-Switzerland border, near Geneva. Since then, it has made many significant discoveries possible. One of the most spectacular ones was the observation of the Higgs boson in 2012 [2, 3]. That discovery would not have been possible if it were not for the CERN's gigantic complex of particle accelerators, including the Large Hadron Collider (LHC), which is the world's most powerful accelerator [4, 5]. It is located in an underground ring-shaped tunnel (Figure 1.1) with a length of 27 kilometres. Inside, two high-energy particle beams travel in opposite directions in special pipes kept at ultrahigh vacuum. Superconducting magnets and numerous accelerating structures allow the particles to reach a speed close to the speed of light. The pipes intersect at four points around the ring, making the beams collide. These points are called Interaction Points (IPs). Massive detectors have been built at the IPs to understand what happens when two high-energy particles collide. One of them is The Compact Muon Solenoid (CMS) [6, 7] (Figure 1.2).

Figure 1.1: The LHC tunnel (source: Maximilien Brice/CERN)

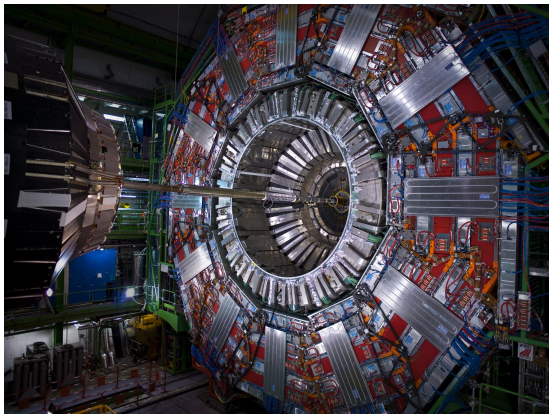

Figure 1.2: The CMS detector (source: Maximilien Brice/CERN)

However, while these central detectors can detect most particles produced in proton-proton collisions, the colliding protons might remain intact in a sizeable fraction of these collisions. They are called leading, forward or scattered protons. The name reflects the fact that these protons scatter to very small angles and thus continue almost 'forward'. They typically loose only a tiny fraction of their energy (several percentage points) in the collision. Because of the small scattering angles (microradians), forward protons remain very close to the LHC beams. They can be detected only after gaining a substantial distance from the beam (millimetres) – only then can the detectors be safely inserted into the LHC pipe sufficiently far away from the beam. Therefore, the detectors must be placed a few hundred metres from the IP. Between the IP and the detector, forward protons remain in the LHC beam pipes and react to the magnetic field of the LHC magnets.

The Precision Proton Spectrometer (PPS or CMS-PPS[1]) [10] can detect forward protons using a near-beam proton spectrometer. The sensors are hosted in vessels called Roman Pots (RPs) [11] (Figure 1.3). They are inserted horizontally or vertically into the LHC pipe, in the

[1]PPS used to be a joint project of CMS and TOTEM [8] (CT-PPS). Currently it is a subsystem of the CMS experiment, and thus it is named CMS-PPS [9].

very forward region on both sides of CMS (LHC sector 45 – anticlockwise of CMS, LHC sector 56 – clockwise of CMS), at about 200 metres from the IP. Their goal is to measure the exact positions of forward protons with respect to the beam.

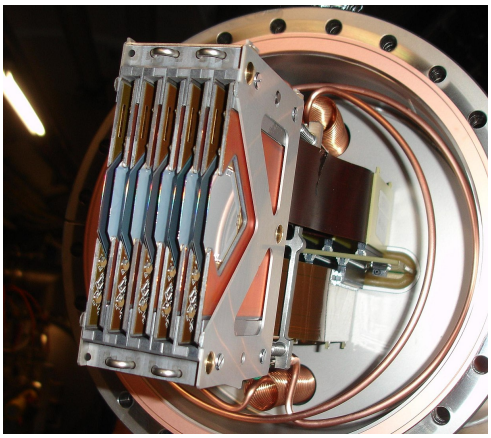

Figure 1.3: A package of sensors hosted in a Roman Pot (source: CERN)

Since RPs are inserted and retracted from the LHC pipes frequently, the positions of the sensors with respect to the beam are not always constant. Moreover, the conditions in the LHC might differ between fills, i.e. operational cycles or periods between injection of particle bunches and dumping them, whose duration can vary from a few hours to one day. There might be subtle changes of the LHC conditions even within a fill; thus, fills are often divided into several runs. For these reasons, the exact positions of RPs with respect to the beam have to be calculated for every run. The process of obtaining these positions is called 'alignment', and its purpose is to find the alignment constants, i.e. the horizontal and the vertical shifts of RPs with respect to the beam.

The alignment procedure usually uses the data from four RPs placed on both sides of IP5. The layout of the LHC is shown in Figure 1.4. Two of these RPs are located in the LHC sector 45 and the other two – in the LHC sector 56. In each sector, the RP closer to the IP is called 'near', and the other one is called 'far'. For historical reasons, occasionally they are also named 'up' and 'down', respectively. The simplified layout of the RPs used in the alignment procedure is shown in Figure 1.5.

It is assumed that x denotes the horizontal axis (pointing out of the LHC ring) and y – the vertical axis (pointing against gravity). These coordinates are used in various plots to denote the particle hits detected in the PPS sensors. Often, the N (x_N, y_N) and F (x_F, y_F) or U (x_U, y_U) and D (x_D, y_D) subscripts are used to denote hits in near and far (up and down) RPs, respectively.

The full alignment procedure consists of two major stages [12]:

1. alignment with the data from a special, low-intensity calibration fill,
2. alignment with the data from a standard, high-intensity fill – also called a physics fill. This step is also referred to as the global alignment.

In a calibration fill, the beam is characterised by low intensity, which means that it consists of fewer particles than a standard fill. Because of that, both horizontal and vertical RPs can be safely inserted very close to the beam, which allows recording additional data essential for calibration. The alignment analysis is done manually by an expert, and the aligned data serve as a reference for the analysis of physics fills.

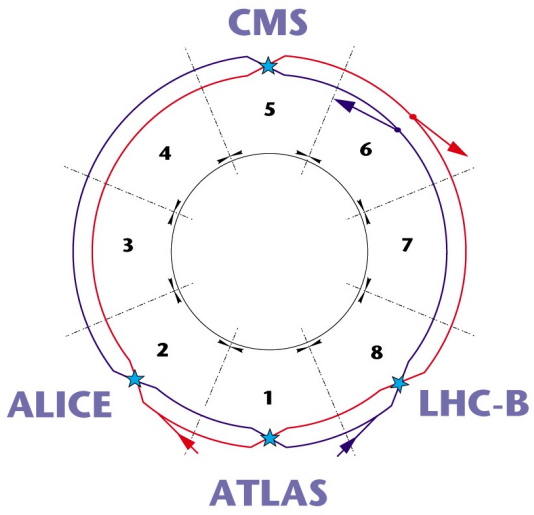

Figure 1.4: LHC layout. It includes the main LHC detectors built at the IPs marked by the blue stars. The digits denote the boundaries of the LHC sectors. For instance, the CMS detector is located at IP5, between the sector 45 and the sector 56. (source: Jean-Luc Caron/-CERN)

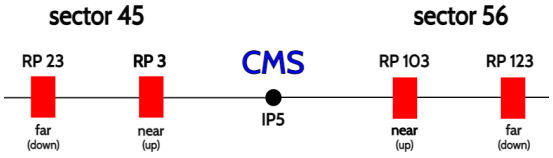

Figure 1.5: Simplified layout of the RPs around IP5 used in the alignment procedure. The RPs are identified by a three-digit number (with leading zeros omitted). The digits, from left to right, stand for the sector (0: s45, 1: s56), the LHC station (0: 210 m from the IP, 2: 220 m from the IP), and the RP type, which, in this case, is always the same. The alignment procedure makes use of two RPs in each sector. They are called 'near' ('up') or 'far' ('down'), depending on their distance from CMS.

This project focused on the second stage, i.e. the alignment of RPs in standard, physics fills, where the fill intensity is much higher. For safety reasons, the distance between an RP and the LHC beam must be larger compared to a calibration fill. Furthermore, in standard fills, only the horizontal RPs are inserted – the vertical RPs are not used for physics measurements. In the procedure, two RPs in each of the two sectors are considered. Since typically there are a few hundreds of physics fills per year, the process needs to be automated.

The physics fill alignment procedure can be divided into the several steps below.

1. Data selection – before the alignment process, data have to be cleaned from the irrelevant events not originating from the IP and thus biasing the measurement.
2. Horizontal alignment – since the physics is the same in all fills, the distributions of the hits observed in an RP are no different too. Therefore, the horizontal alignment matches the hit distributions from a physics fill with the already aligned hit distributions from a calibration fill. After that, the horizontal shift of each RP is obtained.
3. Relative horizontal alignment – it is performed for each arm (sector) separately and calculates the relative position of two RPs (near and far) in an arm. It aims at increasing the accuracy of the horizontal shifts gathered in the previous step.
4. Vertical alignment – it is used to find the vertical shift of each RP.

1.2. Project motivation and product vision

The first version of the alignment procedure was developed for the first running period of the LHC – Run[2] 1 (2009–2013) [13]. Alignment has also been performed for the data gathered during Run 2 (2015–2018) [12, 14]. However, it has only been executed in the offline mode,

[2]The noun 'run' can be understood differently. In this thesis, a capitalised (LHC) Run stands for a running period of the LHC, whereas an uncapitalised run means a part of an LHC fill.

that is a sizeable time after collecting the data. In Run 3, expected to start in 2022 [15], PPS will need a fast way of finding the alignment constants so that the potential of the data can be exploited within 48 hours from their acquisition – in the low-latency mode.

The alignment methods and algorithms have already been developed and implemented for Run 2 [16]. The old software can only operate in the offline mode, however. It also has a low level of configurability, i.e. many parameters are hardcoded, and it is not included in the official code repository of the CMS experiment, CMSSW (CMS SoftWare) [17]. Thus, it is impossible to use it in the CMSSW workflows.

The main goal of this project was to extend the already existing software, methods and algorithms to develop the software that could operate in the low-latency mode (automatic use case). The expected product also had to support the offline mode (manual use case). The software had to be included in CMSSW and be easily configurable. It was expected to save the alignment constants to a database used in the CMS experiment – Conditions Database (Conditions DB). The software was required to save the critical histograms for monitoring purposes and optionally save additional debug plots. **The primary purpose of the product was the alignment of RPs with the data from physics fills**, but it was also required to make use of the already aligned reference data from a calibration fill. To operate in the low-latency mode in Run 3, a reasonable configuration with sensible defaults had to be provided. It was also vital to prepare detailed documentation so that the members of the PPS experiment would know how to configure the procedure and run it properly.

1.3. Feasibility study

The project goals can be achieved using the Prompt Calibration Loop (PCL) mechanism. Due to its automated workflows, it can provide low-latency computations. It requires a specific structure of the code, however. Therefore, the software is split into many files in CMSSW. The most important part, i.e. running the alignment algorithms, uses the Data Quality Monitoring (DQM) software [18] that enables effective, parallel data processing.

The input data for the alignment procedure, that is the reconstructed particle tracks[3] detected in RPs are extracted from so-called Events[4] which are C++ container classes for various data related to a particular collision. This model of data processing is named the Event Data Model (EDM) and CMSSW is built around it [19, 20]. Since the alignment procedure needs to be easily configurable, the configuration cannot be delivered as a part of the CMSSW repository – the only way to modify it would be to submit a time-consuming pull request. Therefore, the EventSetup mechanism is used [21]. It provides the auxiliary information needed to process an Event. These information are often the conditions in the LHC, CMS, PPS or other experiments. Therefore, the data provided by the EventSetup mechanism are often called ‘conditions’, and hence the database storing them is named ‘Conditions DB’ [22]. In the case of this project, EventSetup is used to handle the configuration of the alignment procedure. Most importantly, it enables storing the configuration in the Conditions DB where frequent modifications are possible.

To perform the alignment procedure, one has to create a Python configuration file with the definition of a so-called CMSSW process, which includes input, output and modules to run [23]. The whole alignment workflow is shown in Figure 1.6.

[3]Each RP contains a bunch of sensor planes, and thus a particle hit detected in an RP can also be referred to as a (reconstructed) track.

[4]A collision at the IP or the data related to it is also called ‘event’. To avoid confusion, EDM Events are capitalised in this thesis.

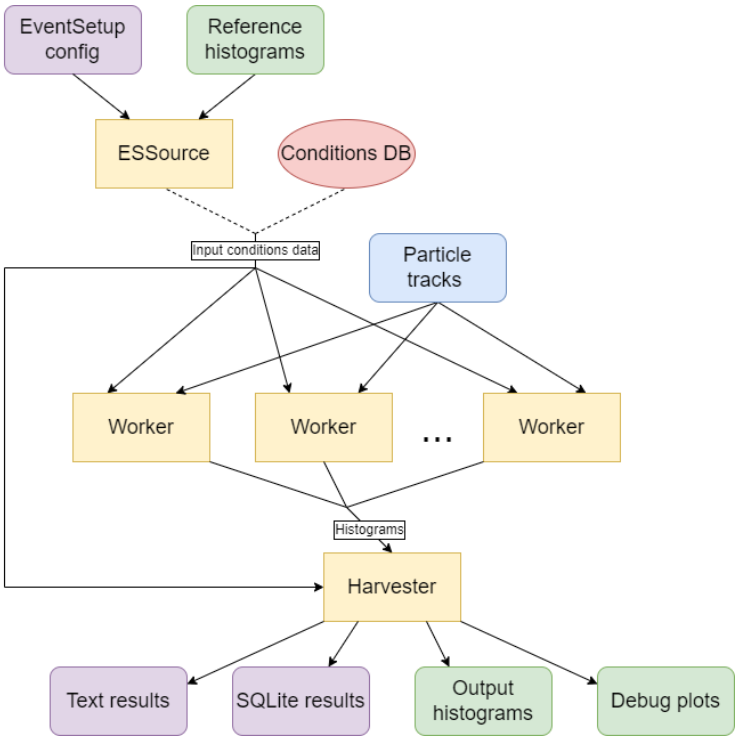

Figure 1.6: Workflow of the physics fill alignment process. Yellow: CMSSW modules; red: external systems; green: ROOT files; purple: SQLite, text and Python configuration files; blue: files containing the EDM data. The dashed lines denote 'or' – the input conditions data are delivered either from the ESSource module or the Conditions DB.

The main workflow elements are described below.

1. **Input conditions data** (configuration). It uses the EventSetup mechanism. The main component of this mechanism is a so-called conditions format, i.e. a container C++ class for conditions data – in this case, the configurable parameters. It includes the setups of both sectors, RP setups, and the parameters of the alignment algorithms. It also contains the vector representations of the reference histograms filled with the already aligned data from a calibration fill. An instance of a conditions format has to be encapsulated in an EventSetup Record. Therefore, a C++ class implementing such a Record had to be developed too. To construct a conditions object, that is an instance of a conditions format stored in a Record, an ESSource module has to be used. It reads a Python configuration file and produces a conditions object that can be used in further steps of the workflow or be uploaded to the Conditions DB. In the alignment procedure, one can either use the ESSource module to construct the conditions object or retrieve it from the DB.

2. **Particle tracks**. They are extracted from the Events stored in the ROOT files selected in the CMSSW process configuration. Most often, the chosen files – and hence the collision data in Events – correspond to one run.

3. **DQM worker**. It is a C++ class that derives from `DQMEDAnalyzer`. It takes the particle tracks and the configuration as input. Then, it processes the data and fills particular histograms on many processing stages. These histograms are later required by the alignment algorithms. A CMSSW process can be configured to run many worker instances in parallel in order to optimise processing large amounts of data and increase throughput. Therefore, filling the histograms in the worker needs to be thread-safe.

4. **DQM harvester**. This C++ class derives from `DQMEDHarvester`. It reads the histograms filled by the worker and performs the alignment analysis. It reconstructs the reference histograms from the input conditions data in order to perform the horizontal alignment. The reference data are not needed for the other two alignment methods: the relative horizontal and the vertical alignment. Another conditions format is used as a container for the results, that is the alignment constants. It had already been implemented before the start of this project.

5. **Output**. In the first place, the alignment results are saved to a text file. They can also be saved to an SQLite file whose contents can later be uploaded to the Conditions DB. Apart from that, the harvester saves a ROOT file with the histograms filled by the worker and a few histograms added by the harvester. Optionally, the harvester can save another ROOT file with more detailed plots that can be used for debugging purposes.

With such a workflow, the software is able to work in the offline mode. It also meets the PCL requirements, which enables it to work in the low-latency mode. Properly used EventSetup mechanism guarantees a high level of configurability. Moreover, it is possible to configure the software to produce the reference histograms using the already aligned data from a calibration fill. Finally, the harvester has an option to save the results to an SQLite file which can later be uploaded to the DB.

The primary programming language of CMSSW is C++. Therefore, the software is developed using this language. CMSSW processes are configured in Python configuration files, however. Example configuration files for both use cases have been delivered with the software. The optimal values of the parameters for the automatic use case are based on the data from Run 2 and Run 3 predictions.

All the histograms filled by the worker module and the debug plots are implemented using the ROOT framework – a data analysis tool used in high-energy physics [24, 25]. The worker produces a ROOT file with histograms, which serves as input for the harvester. The worker can also take the already aligned data from a calibration fill as input and produce a ROOT file containing the reference histograms. The harvester produces a ROOT file with the histograms necessary for monitoring purposes. If specified in its config, it also produces another ROOT file with extra debug plots. The ROOT framework also delivers many functionalities used in the alignment algorithms, e.g. the `TSpline` class for generating a spline interpolation.

The documentation was supposed to be prepared either as a README file in the CMSSW repository or, if it grew into a few separate components, as a page in the wiki system of CMS – TWiki. Those two solutions enable all interested members of the PPS experiment to access the documentation easily.

1.4. Risk analysis

This project was subject to several risks. They are discussed in this section.

1.4.1. Remote work

Due to the COVID pandemic, most of the work was supposed to be done remotely, which could result in communication difficulties. Therefore, weekly online meetings were planned, and, if needed, unscheduled meetings were organised. This solution might not have entirely replaced the standard in-person communication, but it was sufficient for project-related matters.

1.4.2. High entry threshold

The project had a high entry threshold. There were many things to research and learn. Not only was it the theory behind the experiment and the alignment process, but also a non-standard technology stack and CMSSW-specific solutions, such as ROOT, various CMSSW workflows, EDM, DQM, Conditions DB and PCL. Therefore, at the very beginning, three weeks were spent on a general introduction, and later, when some tool was required, an adequate amount of time was spent to understand its rules and functionalities.

1.4.3. Insufficient documentation

Even though CMSSW has been developed for many years and has numerous contributors, its documentation is often imprecise and outdated. For some issues, it does not even exist. Because of that, there was a risk of misunderstanding how some tools, modules or workflows work and not using them correctly. Many emails to experts were foreseen to minimise that risk. They were successfully sent soon enough to allow for a potential delay in answering.

1.4.4. Conditions in Run 3

Run 3 is expected to start in 2022, sometime after the completion of the product. The exact conditions will be unknown until then. It poses a problem and a risk that the software will not work correctly with the provided configuration in the automatic use case. Hence, the configuration for the low-latency mode was not put in the CMSSW repository, but it was included in the more flexible Conditions DB, where the frequency of updates can be higher. However, there is still a high probability of invalid results for the first LHC fills in Run 3, and most likely, these results will not be taken into account.

2. Functional scope

2.1. User characteristics

The software is expected to be used by a few members of the PPS experiment, i.e. most likely physicists with an access to the CERN and PPS resources, with knowledge about the experiment but probably only a basic understanding of the alignment procedure. Therefore, the software and the documentation shall be designed in such a way that new users can be introduced in a reasonable amount of time.

2.2. External systems

In order to be included in the CMS reconstruction workflows, the software shall be integrated into the CMSSW software framework. Therefore, most of the functionalities must strongly depend on its workflows. The framework is built around the Event Data Model. An Event is a C++ class that contains all raw and reconstructed data related to a particular collision and is used to pass the data from one module to another during the processing. CMSSW is divided into many packages, and strict rules define where to upload new source code. It has no graphical user interface (GUI), so all operations, e.g. compiling and running the code, can only be executed in a command line. Main modules are developed in C++. However, to configure them, that is to type the values of their parameters, one has to prepare a Python configuration file. These configuration files are then used in a CMSSW process whose task is to define a single job for CMSSW, which consists of input, modules to run and output. A Python file containing the definition of such a process can be run with a `cmsRun` tool that runs all the selected modules in a chosen order. The software developed within this project shall comply with all these rules. Only then can it be approved and merged in the CMSSW repository on GitHub.

The software shall be integrated with the Conditions DB. It is the CMS database that stores conditions objects, that is EventSetup Records associated with instances of conditions formats containing, for instance, the configuration or the results of some CMSSW modules. Every object in the DB is associated with a so-called tag. The tagging mechanism enables grouping objects of a particular conditions format, for instance, by the type of the data the objects are supposed to take as input. Each tag represents a timeline – the objects within a particular tag are associated with Intervals of Validity (IOVs) which denote the periods during which these objects are valid. Such a solution enables uploading different conditions objects for different LHC fills (or runs). To upload data to the Conditions DB, the user has to generate an SQLite file and ask someone with proper permissions to run a Python script that handles uploading objects to the DB. Retrieving data from the DB is more straightforward since everyone with an access to the CMS resources can configure it within a CMSSW process.

The software shall also work in the PCL. It is a procedure run within 48 hours after data acquisition that provides the calibration and alignment results for the first (prompt) reconstruction of events at IP5 [26]. It requires the software to have a specific structure in the CMSSW framework – it has to be split into a DQM worker and a DQM harvester module.

2.3. Functional requirements

The functional requirements have been divided into the categories below.

2.3.1. Configuration

1. The software has to be prepared to work in different LHC conditions. Therefore, the configuration shall be extensive.

2. The software shall either construct a C++ object containing the configuration using a local Python file or retrieve it from the Conditions DB.

3. The software has to support writing an SQLite file containing a conditions object with the configuration – the CMSSW environment enables uploading the contents of such a file to the DB.

2.3.2. Data selection

1. At the beginning of the alignment procedure, collections of reconstructed tracks with the x and y coordinates of particles detected in selected RPs shall be delivered to the software as input.

2. The software is required to perform the multiplicity selection, i.e. find and skip the events containing too many particle hits in order to make sure the hits from one RP can be properly matched with the hits from the other RP in a sector. The minimum and the maximum number of the tracks detected in an RP shall be configurable.

3. The software must apply the cleaning cuts in order to clean the data and process only forward protons. The cuts shall exploit the correlation between track positions in the near and the far RP in each sector. The parameters of the cuts shall be configurable. Examples are shown in Figure 2.1.

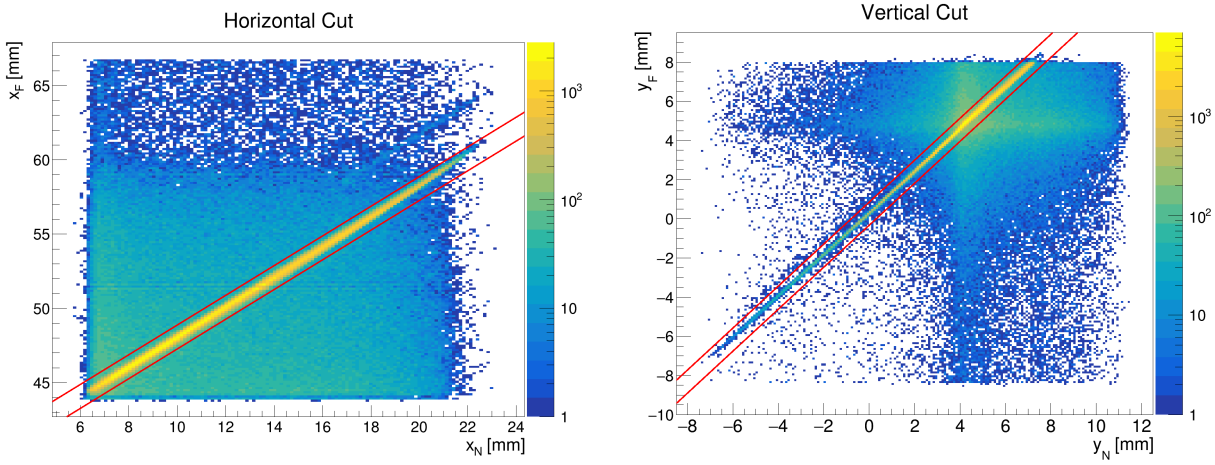

Figure 2.1: Example of the cleaning cuts applied by the old software. The cuts exploit the correlation between horizontal (left) or vertical (right) track positions in near (N) and far (F) RPs. Data from fill 7334

4. The software shall save the hit distribution histograms before processing data, after applying the multiplicity selection, and after applying the cleaning cuts. These histograms are needed in the alignment analysis and are useful for debugging.

5. The software shall save other plots required in the alignment analysis process.

6. The software is required to enable using the data from a calibration fill to produce the reference plots for the horizontal alignment.

7. The minimum sufficient numbers of entries in the histograms shall be configurable.

2.3.3. Alignment analysis

1. The full alignment process shall be based on the already existing software and consist of three steps: horizontal alignment, relative horizontal alignment and vertical alignment. Disabling some of these steps shall be possible.

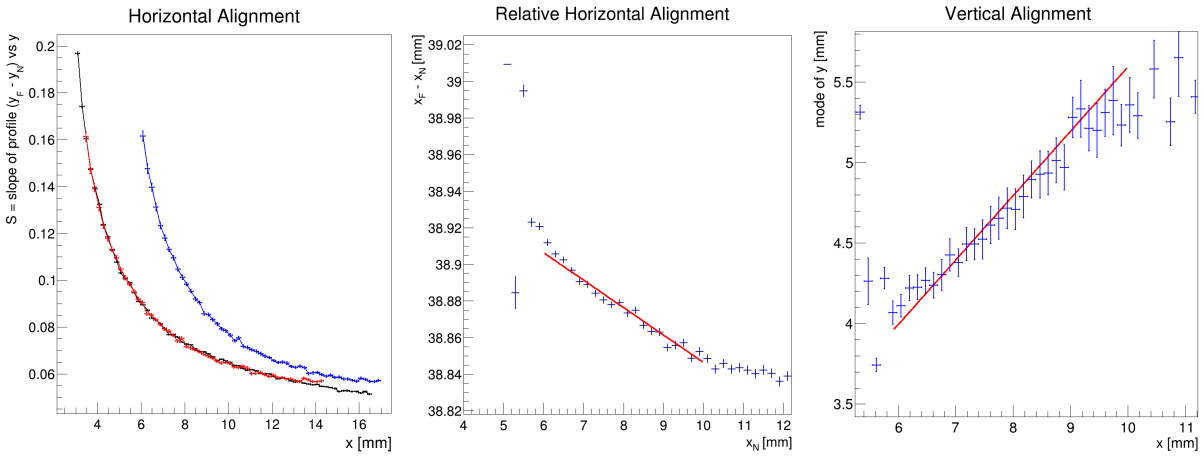

Figure 2.2: Example of the alignment methods; generated by the old software. Left: horizontal alignment (black: reference data from a calibration fill, blue: data from a physics fill before alignment, red: data from the physics fill after alignment), middle: relative horizontal alignment (red: linear fit after fixing the slope), right: vertical alignment (red: linear fit after fixing the slope). Data from fill 7334

2. The software shall perform the horizontal alignment (example: Figure 2.2, left).

   - The software shall match the data from a physics fill with the already aligned reference data from a calibration fill either extracted from a local ROOT file or retrieved from the Conditions DB.
   - The horizontal alignment shall use the x dependence on the S parameter which identifies the slope of the correlation between $(y_F - y_N)$ and y [14]. The dependence is given by the LHC beam optics[5]. It is assumed that the optics is the same in calibration and physics fills, and hence the physics fill histograms can be matched with the reference histograms from a calibration fill.

3. The software shall perform the relative horizontal alignment (example: Figure 2.2, middle).

   - The relative horizontal alignment shall extrapolate the function of $x_F - x_N$ vs x_N to the horizontal beam position and return the relative position of the near and the far RP in an arm.
   - The function must be a line ($y = mx + b$) fitted to the data.

[5]Particles in the LHC oscillate around the ideal trajectory. Quadrupole magnets acting as magnetic lenses are used in order to maintain the beam properties needed for collisions. Because of the similarity to optical lenses, these LHC settings are named ‘optics’ [27].

- The software shall produce two sets of results. The first one shall be calculated using the function with both unknowns (m and b) fitted to the data. In the case of the second set, the slope (m) shall be fixed and set to the value given in the configuration, thus enabling the user to adjust the slope and obtain more accurate results.
- The horizontal beam position shall be delivered either from the configuration or the horizontal alignment (the first step of the alignment process).

4. The software shall perform the vertical alignment (example: Figure 2.2, right).

   - The vertical alignment shall extrapolate the function of mode (most frequent value) of y vs x to the horizontal beam position and return the vertical position of each RP.
   - The function shall be a line fitted to the data, and the software is required to produce two sets of results, as in the case of the relative horizontal alignment.
   - The horizontal beam position and the slope shall be delivered either from the configuration or the horizontal alignment, i.e. the first step of the alignment process.

5. The software shall merge the results obtained from all the methods. Switching between the results obtained with or without the slope fixed shall be configurable.

6. The software shall save the plots used during the analysis for debugging purposes.

7. The software should create a text file comprising the results from all the methods and the final, merged results.

2.3.4. Integration with the Conditions DB

1. The configuration for the alignment procedure shall be delivered either directly from a Python file or from the Conditions DB. The mechanisms responsible for saving and retrieving the data from an SQLite file or the DB must be developed.

2. The software shall store the final alignment results in an SQLite file that will later be uploaded to the DB. A module responsible for retrieving the results from the DB has to be implemented.

2.3.5. Prompt Calibration Loop

The software is required to be included in the PCL in order to work in the low-latency mode. It shall be added to the list of the modules run in the PCL in production, that is in one of the Run 3 reconstruction workflows.

2.4. Non-functional requirements

1. The software shall be integrated into CMSSW and meet all of its requirements.

2. The software is required to be thoroughly tested, which primarily means reproducing the results from the old software.

3. A testing suite in the form of example CMSSW process configurations and expected results must be attached to the software.

4. The software shall be well-documented. The documentation does not have to include the details about the theory behind alignment, but it must describe the configurable parameters and all the developed procedures.

5. DB-related procedures shall be developed and delivered with the software. They must be described in the documentation and properly tested.

6. Data processing in the data selection phase shall be effective and aim at high parallelism in order to be functional in the low-latency mode.

7. Since the exact conditions of Run 3 are not known, the configuration of the alignment procedure shall be flexible, i.e. it must allow for frequent modifications (especially at the beginning of Run 3).

8. A proper default configuration for Run 3 should be prepared.

9. The software should produce detailed logs during its runtime for monitoring and validation purposes.

10. It is required to present the software and the developed procedures to the PPS collaboration and the Alignment Calibration and Database (AlCaDB) group. The presentations should also include questions regarding possible difficulties in the development of the project.

2.5. Use cases

The software shall support two use cases: automatic and manual.

2.5.1. Automatic use case

This use case shall be used in the low-latency mode, that is within 48 hours after data acquisition. The goal of this mode is to gain a first insight into the alignment results and help create the first reconstructions of events. The software shall be run automatically in the PCL. The configuration has to be prepared before Run 3. It must be updated on a regular basis so that the alignment results are as precise as possible.

2.5.2. Manual use case

The manual use case shall be used a sizeable time after the data-taking period, that is in the offline mode. This use case shall enable running the software in many iterations to find the optimal configuration and produce the final results.

3. Selected realisation aspects

3.1. CMSSW ecosystem

There are many CMSSW-specific mechanisms and rules [28] which need to be considered during the development process.

3.1.1. Code organisation

The CMSSW repository is organised in a two-level structure. Modules are included in packages that are subdirectories of subsystems. Each package can therefore be identified by the path: *Subsystem/Package*.

Modules play a crucial role in CMSSW workflows. They can, for example, provide configuration for other modules, gather and preprocess data, or produce results. They can be registered with a simple macro and afterwards used in data processing without any additional imports. By convention, they are placed in the *plugins* folder inside a package. Apart from this folder, a package can also consist of other folders, including:

- *interface* and *src* with declarations and definitions of auxiliary classes or functions,
- *python* with Python configuration files,
- *test* with files used for testing, for instance, unit tests or definitions of example processes.

3.1.2. Data processing

Modules are the building blocks of CMSSW processes. Processes are configured in Python files and can be run with a CMSSW program – `cmsRun`. To run it or other CMSSW executables, one has to set up the CMSSW runtime environment. It can be quickly done with the `cmsenv` command.

In the Python file with a process definition, one can specify [20]:

- which input data to use,
- which modules to execute,
- in what order to execute them,
- values of the parameters used in a particular module,
- where to save the potential output,
- logging policy, that is the minimum message severity level for `MessageLogger`,
- whether to save a file with the logs and, if so, where to save it.

A whole file defining a process can be long, especially when there are many input files or when one needs to specify many parameters of used modules. In such cases, the file can be split into many smaller ones.

3.1.3. DQM software

This project makes use of the DQM software. It enables effective data processing, which is required in the PCL. At first, data are processed in parallel in an analyser module, also called a worker [29]. This module is a class that derives from `DQMEDAnalyzer`. It has to implement two methods: `bookHistograms` and `analyze`. A worker declares (books) all the histograms in the `bookHistograms` method, using the `iBooker` interface implemented by

DQMStore. DQMStore stores the histograms in the form of MonitorElements which encapsulate ROOT histograms. Later, many instances of the worker module run in parallel and fill the histograms via the analyze method. At the end of the job, the histograms are merged and saved in the DQMIO format.

The file is then passed to a harvester module, which has to derive from DQMEDHarvester. It retrieves the data prepared by the worker using the iGetter interface and sequentially analyses them to produce output. The harvester can book and fill additional histograms by using iBooker again. At the end of the harvesting, the contents of DQMStore, that is all the booked and filled histograms, can be saved in the standard ROOT format, which, contrary to the DQMIO format, can be viewed using the ROOT tools.

3.1.4. ROOT framework

ROOT is commonly used in many CMSSW source files. It is an object-oriented data analysis framework, which is used for simple and advanced calculations, processing data, handling histograms and serialising complex data structures in TTree objects. MonitorElements that are used to transport the data from a DQM analyser module to a harvester encapsulate ROOT histograms. The project made use mostly of one and two-dimensional histograms (TH1D and TH2D classes), graphs without or with errors (TGraph, TGraphErrors), and plots called profiles (TProfile) that are used to display the mean value of y for every bin in x axis and gather more statistical data than standard histograms. ROOT provides a simple GUI – TBrowser (Figure 3.1), which can be used to analyse the plots saved by the harvester.

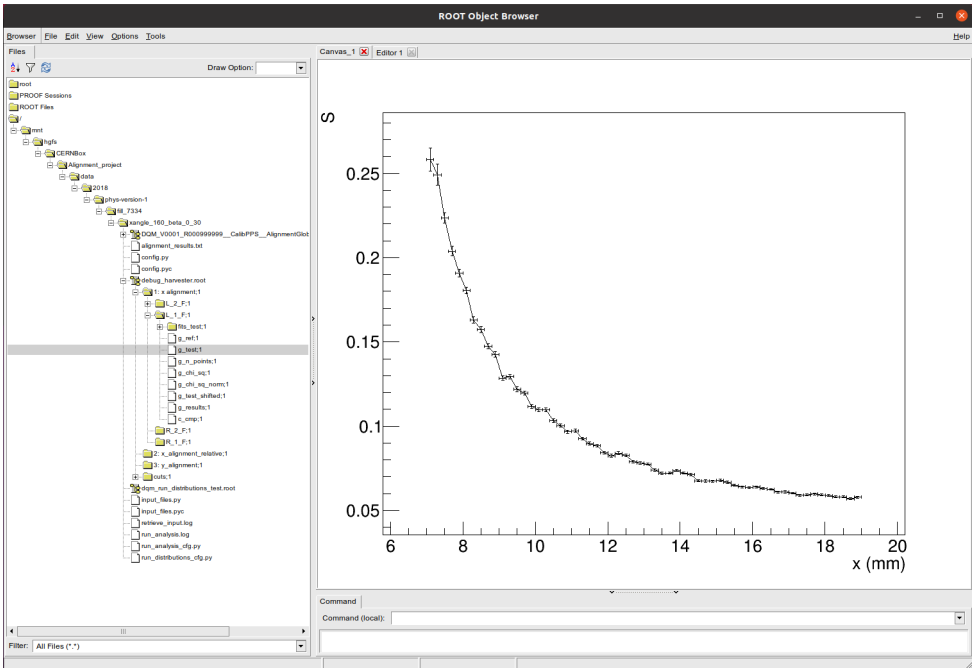

Figure 3.1: TBrowser – the GUI of ROOT

The DQM tools enable booking, filling and using only simple plots, e.g. two-dimensional histograms. Therefore, this project also makes use of ROOT itself, without the MonitorElement encapsulation. If specified in the configuration, the software produces a debug ROOT file, which contains more sophisticated plots used for debugging purposes.

Apart from simple operations on histograms, ROOT enables, for instance, to fit a line or other function to a histogram and plot it on the same canvas. This functionality is used in all of

the alignment algorithms. ROOT also provides other data analysis tools, such as the `TSpline` class, which creates the spline interpolation of provided data. It is used for matching the physics fill histograms with the reference ones in this project.

3.1.5. Testing

CMSSW provides a few ways of testing the software. At first, one can define some example configuration files, put them in the *test* folder of a package, and prepare running instructions. It is not automated at all but is still commonly used. It is the primary testing method for the alignment procedure. The other way is to write a unit test. In CMSSW, such a test is expected to run for a couple of seconds. In some cases, this time is sufficient, but not in the case of alignment; the alignment worker module itself usually runs for a couple of minutes. The last way of testing is preparing a so-called matrix test[6], which is just the definition of a validation workflow consisting of modules defined in Python configuration files. Matrix tests are run frequently. Every pull request must pass these tests, and hence they are also called 'release validation' (RelVal). The logic of these tests is simple – if any changes with respect to the results produced previously are found, an alarm is raised.

3.2. Code structure

The CMSSW rules required the code to be split into many files in many packages. Ultimately, various components of the alignment software have been placed in the subsections and packages shown in Figure 3.2 and listed below. Some minor changes have also been introduced in a few other CMSSW packages.

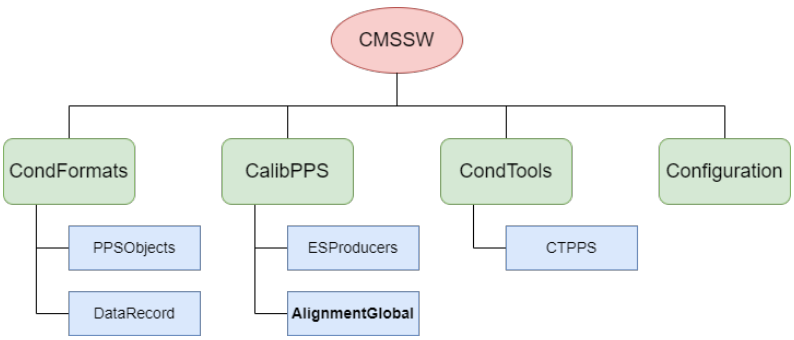

Figure 3.2: Diagram of the CMSSW subsystems and packages added or modified in the project. It does not include a few other packages in which only minor changes were made. The subsystems are marked in green, and the packages – in blue. The name of the project's main package is in bold.

3.2.1. *CondFormats/PPSObjects*

The `PPSAlignmentConfiguration` conditions format is defined in this package. The declaration is in the *interface* folder, and the definition – in the *src* one. This class is a container for the configurable parameters and serves as input to the alignment analysis. To improve readability and represent the dependencies, most of the parameters are grouped in a couple of auxiliary

[6]Each step of a validation workflow can be considered an entry in a matrix. Each row of this matrix corresponds to a different matrix test.

structures. There is also another conditions format in this package which is used by the alignment software – `CTPPSRPAlignmentCorrectionsData`. It had already been implemented before the start of this project. Its purpose is to store the alignment results.

3.2.2. *CondFormats/DataRecord*

The `PPSAlignmentConfigurationRcd` class is defined here. It defines the Record class for the `PPSAlignmentConfiguration` conditions format.

3.2.3. *CalibPPS/ESProducers*

The `PPSAlignmentConfigurationESSource` module, which is defined in this package, creates an instance of the input conditions format and stores it in a Record, from which it can be retrieved during data processing. This module overrides an important `fillDescriptions` method derived from `edm::ESProducer` which is used widely in CMSSW. This function provides the default values for all the parameters. Thus, in most cases, one has to modify only a few essential parameters in a Python configuration file.

3.2.4. *CalibPPS/AlignmentGlobal*

This is the main package of the project [30]. The worker (`PPSAlignmentWorker`) and the harvester (`PPSAlignmentHarvester`) modules are implemented in the *plugins* folder. The *interface* and *src* folders contain some utility functions. Some of the Python files required by the PCL are defined in the *python* folder, and example configuration files that can be used to run the whole alignment procedure are defined in the *test* folder. This package also contains a README file with a basic description of the alignment procedure and a reference to an external TWiki page with the project documentation.

3.2.5. *CondTools/CTPPS*

A few auxiliary modules and their example Python configuration files have been added to this package. Their task is to perform the DB-related operations on the `PPSAlignmentConfiguration` and the `CTPPSRPAlignmentCorrectionsData` objects. At first, they can be used to serialise an object and save it to an SQLite file, which can later be uploaded to the Conditions DB by someone with proper permissions. Other modules can retrieve the objects from SQLite files or directly from the DB.

3.2.6. *Configuration* subsystem

Many files were modified in this subsystem to integrate the software with the PCL and provide a matrix test. As the name suggests, this subsystem does not contain the definitions of modules. It imports them, for example, from the *AlignmentGlobal* package, organises them into workflows, and defines these workflows to be matrix tests or parts of the PCL.

3.3. Alignment algorithms

Four essential algorithms used in the alignment process are described in this section. They were already used in Run 2, but now they were updated and adapted to the DQM software. Since the actual code is complex, only the pseudocode is presented.

3.3.1. Data selection

The worker module starts with booking histograms in the `bookHistograms` method. Afterwards, it takes `CTPPSLocalTrackLiteCollection` objects with particle tracks as input, and for each of them, the `analyze` method is called. In the `analyze` method, the worker extracts the particle tracks from a single `CTPPSLocalTrackLiteCollection` object. Then, the worker processes the data and fills the previously booked histograms. Since the `analyze` method can be configured to run in parallel, it can only use the basic ROOT operation – adding a new entry to a histogram. Other operations, e.g. setting the values of selected bins, are not allowed since they do not support concurrency. The histograms are merged when the processing of all the objects is finished. The pseudocode below presents the data selection procedure in a single sector. It does not include filling every histogram since they are filled at every stage of the process.

```
in: CTPPSLocalTrackLiteCollection trackCollection
data_selection:
    // group the data by RPs
    tracksUp <- near (up) RP tracks from trackCollection
    tracksDw <- far (down) RP tracks from trackCollection

    // multiplicity selection
    if tracksUp.size() < minRPTracksSize
            or tracksUp.size() > maxRPTracksSize:
        return;
    if tracksDw.size() < minRPTracksSize
            or tracksDw.size() > maxRPTracksSize:
        return;

    // applying the cuts
    for trUp tracksUp:
        for trDw in tracksDw:
            cq_h <- trDw.x + cut_h_a * trUp.x + cut_h_c
            cv_h <- abs(cq_h) < n_si * cut_h_si

            cq_v <- trDw.y + cut_v_a * trUp.y + cut_v_c
            cv_v <- abs(cq_v) < n_si * cut_v_si

            cutsPassed <- cv_h and cv_v

            if cutsPassed:
                Fill the final histograms with (trUp.x, trUp.y)
                    and (trDw.x, trDw.y)
```

- `minRPTracksSize`, `maxRPTracksSize`, `cut_h_a`, `cut_h_c`, `n_si`, `cut_h_si`, `cut_v_a`, `cut_v_c` and `cut_v_si` are configurable parameters. The first two parameters stand for the minimum and the maximum number of the tracks per RP in a single event. The `*_a` and `*_c` parameters represent line equations ($y = mx + b$) along which the data selection cuts are applied (see Figure 2.1). The `cut_*_si` parameters represent the widths (root mean square) of track distances from the cut lines. The `n_si` parameter is a multiplicative factor scaling the `cut_*_si` parameters. Together they define the threshold to accept or reject the track combinations.
- The cut is applied along the configurable lines. Only the events within the selected number of σ from the line are selected.

Example histograms generated in various steps of the data selection process are presented in Figure 3.3. The plots produced by the previous software (shown in Figure 2.1) are almost fully reproduced by this software. The only difference is the more strict multiplicity selection in the current default configuration. Since a new filter will be employed in the PCL, the software imitates its behaviour by setting `minRPTracksSize` and `maxRPTracksSize` to 1. Previously they were set to 0 and 2, respectively.

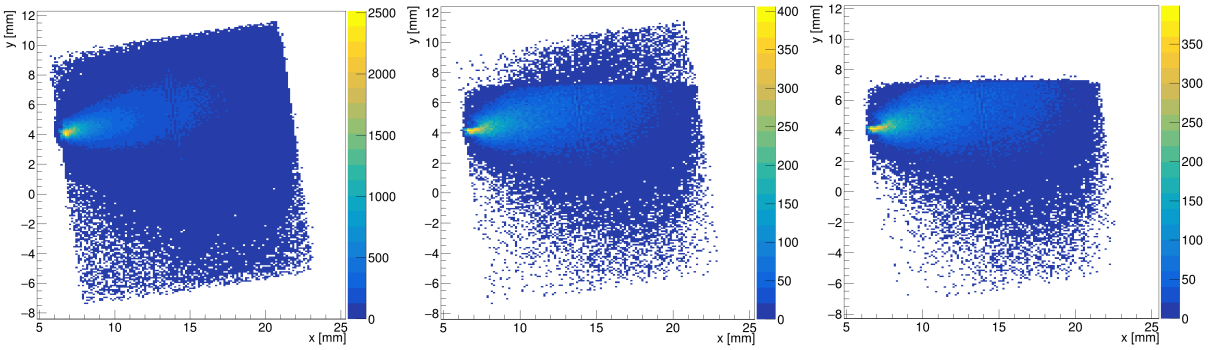

Figure 3.3: Representative examples of the hit distribution histograms created during various steps of the data selection for a single RP. Left: before selection, middle: after the multiplicity selection, right: after applying the cleaning cuts. Data from fill 7334. The cleaning cuts mechanism has also been shown in Figure 2.1.

Apart from processing the data from physics fills, the worker is also used to generate the reference histograms for the horizontal alignment using the already aligned data from a calibration fill. These reference histograms are saved in a ROOT file. The ESSource module can read this file, extract the necessary histograms and transform them into vectors of floating-point numbers. These histograms are then reconstructed and used in the harvester module, in the horizontal alignment.

3.3.2. Alignment

The complete alignment analysis in the harvester consists of the steps below.

```
complete_alignment:
    Read the data prepared by the worker
    Read the input conditions (also containing the vector representations
        of the reference histograms)
    Perform x_alignment to calculate the horizontal shift
    Perform relative_x_alignment to calculate the relative
        horizontal shift
    Perform y_alignment to calculate the vertical shift
    Merge the results from all the methods
```

The horizontal shift value is most often passed from the horizontal alignment to the other methods. Since the sequence of the alignment methods is configurable, it might happen that the horizontal alignment is not the first method in the sequence. In such a case, the shift is taken from the config.

The horizontal and the vertical alignment work separately for each RP but the relative horizontal alignment – separately for each sector. Only one case (out of four RPs or two sectors) is considered below. Examples for all the methods have already been shown in Figure 2.2. The new software can reproduce them.

The first alignment algorithm in a regular workflow is the horizontal alignment. It uses the S parameter, that is the slope of the $(y_F - y_N)$ vs y profile, and its dependence on x to match the analysed data (from a physics fill) with the reference data. The $y_F - y_N$ vs y plots for each RP are built using the so-called slice plots created by the worker. They are called 'slice' because they depict only a slice of data, e.g. the data between $x = 49.4\,\mathrm{mm}$ and $x = 49.6\,\mathrm{mm}$. They contain histograms of y within the x ranges (see Figure 3.4, left) and $y_F - y_N$ vs y graphs (see Figure 3.4, middle) from which the fitted slopes are extracted in order to create an S vs x graph (see Figure 3.4, right).

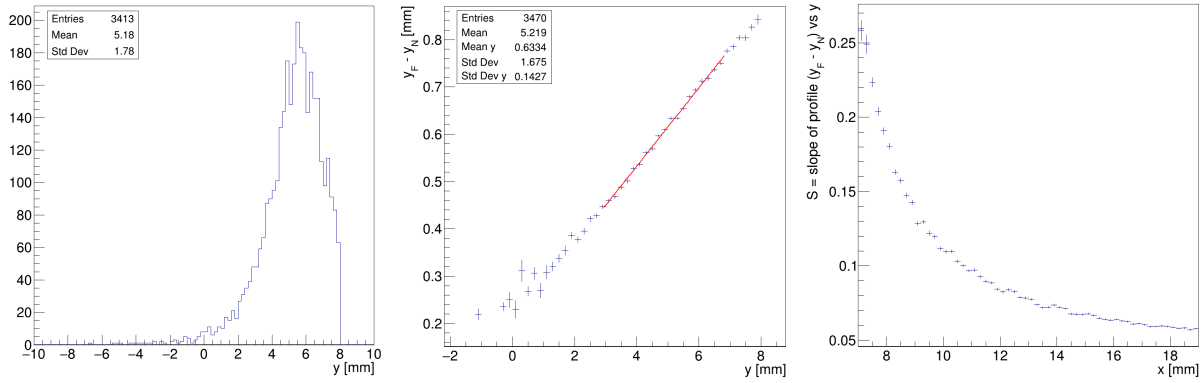

Figure 3.4: Representative examples of various plots used to perform the horizontal alignment. The left and middle ones are the slice plots for x between $49.4\,\mathrm{mm}$ and $49.6\,\mathrm{mm}$. The left one is a histogram of hits in the y axis; the right one is a 2D histogram of $y_F - y_N$ vs y. In this configuration, such plots are created for all the ranges between $x = 46\,\mathrm{mm}$ and $x = 56\,\mathrm{mm}$. Then, all of the slice plots are used to create an S vs x graph on the right. Matching of such a plot with a similar one built using the data from a calibration fill has been shown in Figure 2.2, left. Data from fill 7334

To find the optimal shift, the horizontal alignment matches the analysed data plots with the reference ones by shifting them horizontally (in x axis). It minimises the reduced chi square defined as:

$$\chi^2_\nu = \frac{\chi^2}{n} \tag{1}$$

where n is the number of points and

$$\chi^2 = \sum_i \frac{(R_i - A_i)^2}{\sigma_i^2} \tag{2}$$

with the reference data R and the analysed data A. σ stands for the sum of the squared uncertainties of R and A. Apart from that, the x coordinates of the shifted analysed data graph might not match the reference ones. Thus, the reference graph is interpolated with a spline. This functionality is provided by ROOT.

```
out: sh_x
x_alignment:
    // build graphs
    g <- Use the slice plots to build a graph: S vs x
    ref_g <- Reconstruct the reference graph from the vector representation
    s_ref <- g_ref spline interpolation

    // do the matching
    chi_best <- +inf
```

```
    sh_best <- sh_min
    for sh from sh_min to sh_max with step sh_step:
        g_sh <- shift g by sh
        chi <- calculate chi-squared between s_ref and g_sh
        if chi < chi_best:
            chi_best <- chi
            sh_best <- sh
    sh_x = sh_best
```

The software calculates the uncertainties too. The `sh_min`, `sh_max` and `sh_step` parameters enable adjusting the ranges and the accuracy of the matching.

After the standard horizontal alignment, the relative one is performed. It is called 'relative' because it processes the data from both RPs in an arm and finds relative corrections. It uses ROOT to fit the function `f(x) = [0] + [1] * (x - sh_x)` to the $x_F - x_N$ vs x_N data. `[0]` and `[1]` are the parameters to be found by fitting. Then, it returns the value of the first fitted parameter (`[0]`) which is the extrapolation of the function to `x=sh_x`. Apart from the standard results, the method also returns the 'slope fixed' results. After the fitting, the slope (`[1]`) is fixed to the value provided in the configuration. Then, the fitting is done again. It is useful in the manual mode since it can increase accuracy. However, in cannot be used properly in the automatic mode since the slope is not known before running the alignment procedure, and fixing it might decrease the quality of results. An example of the only plot used in this method has already been shown in Figure 2.2, middle.

Finally, the vertical alignment is performed. The procedure is similar to the relative horizontal alignment, but this time it fits the function to the mode (most frequent value) of the y vs x plot. Again, the value of the first fitted parameter is the result, and the slope can be fixed. This method uses the hit distribution histograms created after the multiplicity selection (example: Figure 3.3, middle) in order to create the mode graph (example: Figure 2.2, right).

When all the results are gathered, they are merged. The vertical shift is calculated only once, but two different methods are used to calculate the horizontal one. Merging is performed for each sector separately.

```
in: d_x_N, d_x_F, d_x_rel_N, d_x_rel_F
out: sh_N, sh_F
merge_sh_x:
    b <- d_x_rel_N - d_x_rel_F
    xCorrRel <- b + d_x_F - d_x_N
    sh_N <- xCorrRel / 2.0
    sh_F <- -xCorrRel / 2.0
```

The input arguments correspond to the shifts of the near and the far RP in a sector, produced by the horizontal and the relative horizontal alignment, respectively. Example results (data from fill 7334) are shown below, edited in the same way the software prints them. There is no way to calculate the rotations of the RPs in the physics fill alignment procedure, but the overloaded insertion (<<) operator of the `CTPPSRPAlignmentCorrectionsData` class streams them to output as well.

```
RP 3: shift (um) x = -3537.6 +- 15.5, y = 3515.6 +- 17.4, z = 0.0 +-
    0.0, rotation (mrad) x = 0.0 +- 0.0, y = 0.0 +- 0.0, z = 0.0 +- 0.0
RP 23: shift (um) x = -41502.4 +- 15.3, y = 4198.7 +- 17.4, z = 0.0 +-
    0.0, rotation (mrad) x = 0.0 +- 0.0, y = 0.0 +- 0.0, z = 0.0 +- 0.0
RP 103: shift (um) x = -2681.4 +- 14.3, y = 2643.1 +- 19.3, z = 0.0 +-
    0.0, rotation (mrad) x = 0.0 +- 0.0, y = 0.0 +- 0.0, z = 0.0 +- 0.0
RP 123: shift (um) x = -41638.6 +- 15.5, y = 3391.2 +- 30.1, z = 0.0 +-
    0.0, rotation (mrad) x = 0.0 +- 0.0, y = 0.0 +- 0.0, z = 0.0 +- 0.0
```

3.4. Configuration

The software is highly configurable. The worker and the harvester have a few configurable parameters, but most of the configuration was placed in a conditions format – the `PPSAlignmentConfiguration` class. Essential parameters are listed and described below. More detailed descriptions have been written in a TWiki documentation page inaccessible without a CERN computing account.

Each of the modules mentioned below implements the `fillDescriptions` method. It provides reasonable default values for all the parameters. Due to that, most parameters do not have to be set manually.

3.4.1. EventSetup

Along with the `PPSAlignmentConfiguration` conditions format, the `PPSAlignmentConfigurationRcd` Record had to be implemented. It is responsible for storing the data and accessing an instance of the conditions format in the worker and the harvester. A Record is filled by an ESSource module, in this case: `PPSAlignmentConfigurationESSource`. It reads a Python config file, creates an instance of a conditions format and stores it in a Record with a selected label. The label is crucial in this project since two instances of the same conditions format are required in the harvester. Because of different labels, these instances can be distinguished from each other. One instance stores the reference data configuration with the reference histograms, while the other one – the configuration of analysed, physics fill data.

Important parameters of the `PPSAlignmentConfigurationESSource`[7] module are shown below.

- `label` – label to distinguish the reference data config and the analysed data config. It is not saved in the instance of the conditions format, but it is used in the `setWhatProduced` function which associates the object with this label.
- `sector_45` and `sector_56` – containers for the sector configurations
- `x_ali_sh_step` – step for the matching in the horizontal alignment
- `min_RP_tracks_size` and `max_RP_tracks_size` – minimum and maximum number of the tracks in each RP
- `matching` – container for the reference dataset parameters; used only for the analysed data config; contains the horizontal alignment shift ranges (`sh_min` and `sh_max`) for each RP and the path of the file with the reference histograms.
- `x_alignment_meth_o` – container for the horizontal alignment parameters; includes the horizontal alignment ranges (the minimum and the maximum x for the matching) for each RP.
- `x_alignment_relative` – container for the relative horizontal alignment parameters; includes the ranges for each RP.
- `y_alignment` – container for the vertical alignment parameters; includes the ranges for each RP.
- `binning` – container for the binning parameters of many histograms booked by the worker, such as: number of bins or minimum and maximum x.

[7] The `PPSAlignmentConfiguration` class is a conditions format, but its instances are created by the ESSource module. Therefore, it is the ESSource module that needs to be configured in a Python file and has configurable parameters.

The configuration of a sector includes the slope value for the relative horizontal alignment, the cut parameters and the configs of both RPs (near and far), which are other containers. They include name, id, the slope value for the vertical alignment, the default value of the horizontal shift and the mode graph parameters (for the vertical alignment).

3.4.2. Worker

The most crucial parameter of the worker is `label`. It enables the worker to run either with the reference (`label="reference"`) or the physics fill data (`label=""`).

3.4.3. Harvester

The harvester has several important parameters:

- `sequence` – determines the order of the alignment methods;
- `overwrite_sh_x` – if set to `True`, the horizontal alignment overwrites the horizontal shift from the configuration;
- `text_results_path` – a path, determines where to save the file with the text results. If it is an empty string, the file is not saved.
- `write_sqlite_results` – if set to `True`, the harvester saves the final results in an SQLite file;
- `x_ali_rel_final_slope_fixed` and `y_ali_final_slope_fixed` – determine whether to use the results with the fixed slope or the standard ones when calculating the final, merged results;
- `x_corr_min`, `x_corr_max`, `y_corr_min`, `y_corr_max` – so-called reasonability ranges for the horizontal and the vertical correction, respectively. If the corrections are not within the corresponding ranges, they are set to 0, as well as their uncertainties;
- `debug` – when set to `True`, the harvester produces an extra ROOT file with the debug plots.

3.5. Integration with the Conditions DB

The software is fully integrated with the Conditions DB. Four classes that enable the serialisation and deserialisation of the conditions objects have been implemented in the *CondTools/CTPPS* package. Example configurations of CMSSW processes have also been prepared. The software has been tested locally using SQLite files with the conditions objects and online, after uploading these objects to the Conditions DB. All of these tests have been successful, i.e. the results from the software used in Run 2 have been reproduced.

4. Work organisation

4.1. Project characteristics

The project was a part of the collaboration between the Institute of Computer Science at AGH UST and the CMS experiment. The development of the project started during a summer internship in 2020. The work was resumed in March 2021, and the project was completed in December 2021.

The main requirements of the project were known from the beginning, but the details were determined over time. Therefore, the nature of the project was primarily incremental. At first, the software core was implemented and merged in CMSSW. Then, the software was integrated with the Conditions DB, and finally, alignment was included in the PCL. However, the code did evolve, and some functionalities were added with time, so the project was also iterative. To sum up, **the project used the iterative and incremental approach**. It enabled starting the work relatively quickly even though not all the requirements were known or understood. Over time, all the requirements were determined, and the software could evolve and expand accordingly.

The project was, in principle, developed only by one person, but it would not be possible to complete it without the help of a few other people:

- Leszek Grzanka – thesis supervisor; along with Maciej Malawski is responsible for the collaboration with the CMS experiment at AGH UST and organises the summer internships, during one of which the project was started;
- Jan Kašpar – responsible for alignment in the PPS experiment; was the main supervisor of the project in PPS and helped the author understand the alignment algorithms and some of the CMSSW ecosystem rules;
- Valentina Avati – supervisor of many software-related activities in the PPS experiment; patiently helped to understand and determine some of the project requirements even though it was seemingly impossible.

4.2. Work organisation and used tools

Various tools, websites and methods were used in the project. Because of the COVID pandemic, most of the work was done remotely, which only increased the number of them. In this section, they are divided into a few categories.

4.2.1. Task tracking

Since the project was developed only by one person, there was no need for advanced task tracking tools like Jira. In the 2020 phase, the requirements and tasks were provided iteratively, so they did not have to be tracked. In 2021, several significant tasks were considered, and many minor issues and tasks were found throughout the development. A simple list was created in Microsoft Excel to keep track of them. It was stored in CERNBox, that is the cloud of CERN. The list was shared with the group of people mentioned earlier. Apart from the task names and their descriptions, it consisted of a priority estimate (1: low priority, 5: high priority) and a time estimate (1: less than an hour, 2: 1–3 h, 3: 3–6 h, 4: 6–12 h, 5 – more than 12 h). Having created such a list turned out to be a good idea. Most of the time, only high priority tasks were under development. The final months provided the time to address the lower priority tasks. Then, the list was leveraged.

4.2.2. Communication

The author adjusted to the communication methods of the group with which he collaborated. The core was sending emails. The vast majority of them were lightweight and could have been moved to a tool supporting instant messaging like Microsoft Teams. The Mattermost platform was in use at the very beginning, but it was dropped with time. Discord was used to communicate with the thesis supervisor.

Since most of the issues during the development were complex and had many unknowns, online meetings were regular. In 2020, they were organised on the Vidyo platform, but in 2021 it was replaced by Zoom. In the case of longer or more formal meetings, they were planned using the Indico platform that enabled uploading presentations and other materials prior to a meeting or adding minutes. Microsoft PowerPoint was used along with CERN templates to create presentations.

4.2.3. Coding

The project was developed on a computer with Windows 10. Since the ROOT framework is easier to install on Linux, VMWare Workstation Player 16 was employed to run a Ubuntu 20.04 virtual machine.

The project required an editor or IDE supporting both C++ and Python. The author decided to use Visual Studio Code from Microsoft. The basic version of the editor can easily be adapted for most programming languages. Many plugins were employed, which enabled, e.g. syntax highlighting or intelligent code completion (IntelliSense) for both languages. VS Code also includes a neat Markdown editor, which was used to write README files.

The code was formatted in compliance with the CMSSW rules by using Clang-Format.

Configuring the computer so that it could run CMSSW processes would have been difficult. Therefore it was decided to run it on LXPLUS – CERN UNIX computing cluster. A CERN user can log in to this cluster using SSH and their computing account. Having logged in to LXPLUS, they can access their EOS home directory – the disk storage system at CERN with the 1TB quota for every user. The data from one's EOS home directory can easily be accessed online via CERNBox. One can also download a CERNBox client (Figure 4.1) and synchronise a local folder with selected data from their EOS home directory.

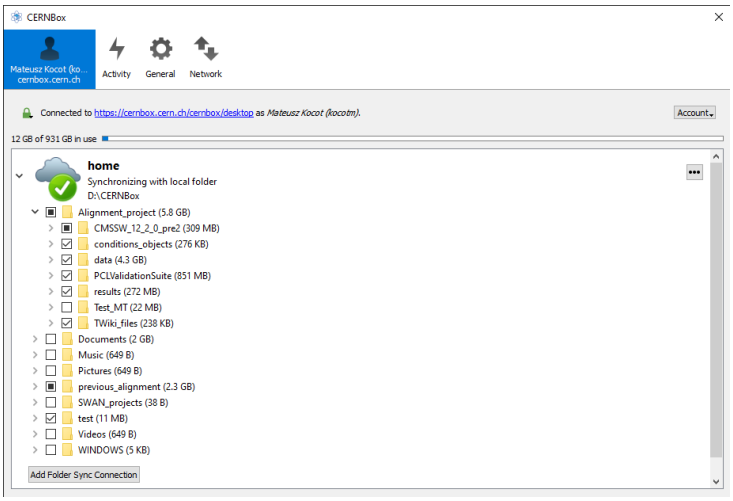

Figure 4.1: CERNBox Windows client

The author chose to work on the code locally, sync it with EOS, and then run it on LXPLUS. Such configuration was almost as convenient as working on the code only on the local PC. The only disadvantage was a bug in the CERNBox client, due to which the code sometimes was not synchronised quickly enough.

At first, the documentation was supposed to be prepared as a README file in the *CalibPPS/AlignmentGlobal* package in CMSSW. However, as the number of configurable parameters grew, it was challenging to keep the file up-to-date. Therefore, in the end, only some basic information was put in the README file, and the complete documentation was moved to a TWiki page. TWiki is an open-source wiki platform used at CERN. The page is accessible only by users with proper permissions.

4.2.4. Version control

The most popular source control system – Git – was used. The Git configuration was stored along with the repository, that is in the EOS space. Another Git repository was used locally to enable VS Code to present current changes neatly.

The repository used Git to synchronise with the remote one on GitHub. According to the CMSSW guidelines, a developer is supposed to fork the official CMSSW repository, create a branch, and submit a pull request (PR) from their fork to the official repository whenever they are ready. In the PPS group, code is developed within the CTPPS organisation on GitHub, so the branches adding the alignment functionalities were created in the CMSSW fork in CTPPS. The auxiliary repository with examples – *pps-alignment-data* [31] – is also hosted in CTPPS.

Many CMSSW branches were made and then merged to the official repository during the development. The *pps-alignment-data* repository was developed only by the author of this project, so it did not require more branches than one. The commit statistics of this repository are shown in Figure 4.2. The chart begins in September 2020, when the repository was created. Afterwards, it lucidly presents the most active development periods.

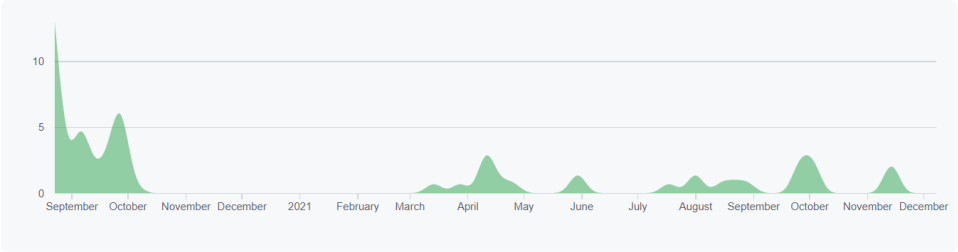

Figure 4.2: The commit statistics of the only branch (*master*) in the *pps-alignment-data* repository, generated in GitHub

4.3. Project phases

The development of the project can be divided into four phases. They are presented in this section.

4.3.1. Phase 1 – summer 2020

The work on the project started during the summer internship in 2020. The first three weeks were spent researching, which included reading and understanding the code from the previous software and adapting to the CMSSW ecosystem. Afterwards, the core of the project was

implemented. It included: the conditions format `PPSAlignmentConfig` (which was later replaced with `PPSAlignmentConfiguration`) with its Record and ESSource module, the `PPSAlignmentWorker` and the `PPSAlignmentHarvester` classes with example configuration files, and the first version of the *pps-alignment-data* repository.

Weekly meetings for the reports of all the interns were scheduled, and if necessary, other, less crowded meetings were organised. The most important ones are listed below:

1. 2 July – introduction meeting with brief information regarding the project and internships in general,
2. 16 July – finished the main part of the research and began the implementation of the project,
3. 22 July – first major problem was solved – the `PPSAlignmentConfig` class had lacked a macro registering the class in the CMSSW framework, and hence CMSSW had not compiled successfully,
4. 28 August – an email concerning a problem with reading reference data was sent to the DQM team, created the *pps-alignment-data* repository,
5. 3 September – solved the problem from the previous week,
6. 17 September – added saving some important debug plots also in the DQMIO format.

At the end, on 30 September, a final meeting was organised. It consisted of final reports from all the interns. The alignment summer project was completed, but there was still much to do to prepare the alignment procedure for Run 3. Therefore, the author was offered to continue working on the project.

There was only one pull request to the official CMSSW repository in this phase. It included all the CMSSW code developed during the internship and was eventually merged more than a month after the internship ended – on 6 November. For the list of all the project's pull requests to CMSSW see Appendix A.

4.3.2. Phase 2 – March–June 2021

The second phase started in March 2021, a few months after the internship completion. The work was mainly focused on integrating the code with the Conditions DB, adding alignment to the PCL, and implementing the matrix test. Initially, some experiments were done to ensure that the software would work correctly with a new event filter that was supposed to be employed in Run 3.

There was a weekly slot for meetings, but they were organised only when needed. The most important ones are listed below:

1. 25 February – first meeting in the new phase, a few old requirements were updated, some new were added including adding alignment to the PCL and need for an investigation concerning the new filter,
2. 16 March – finished the investigation on how the new filter impacts alignment and presented the results,
3. 21 May – uploaded first payloads to the Conditions DB and successfully retrieved them from the DB.

No pull requests were submitted during the second phase.

4.3.3. Phase 3 – July–September 2021

At the beginning, the main task was validation of the alignment matrix test. This task required creating a so-called Candidate Global Tag. Due to some delays in email communication and the issue described below, the candidate GT was not created until October.

In the second week of August, the author was on-site at CERN, and then the project advanced a lot. First of all, it turned out that the `PPSAlignmentConfig` class had not been registered correctly in CMSSW, so a pull request fixing it was submitted. Another PR was created to backport these changes to one of the older versions of CMSSW. It was required since some integration tests used that particular version at the time. Those two pull requests were quickly merged – on 13 and 16 August, respectively. At that point, the next task was to update the `PPSAlignmentConfig` class.

However, after a few days, it turned out that the `PPSAlignmentConfig` class could not be modified as it already had some objects in the Conditions DB. Therefore, it had to be replaced with a new class – `PPSAlignmentConfiguration`, which meant that two previous pull requests turned out to be redundant. The new class was, in principle, a copy of its predecessor, but it included some necessary updates, mainly new configurable parameters. The class, along with its registration in the Conditions DB, was added in September.

In this phase, there was one important online meeting – the AlCaDB group (Alignment Calibration and Database) meeting on 30 August. The author presented the assumptions of the alignment of the PPS detectors for Run 3 and the road map of the pull requests for the following months. During that meeting, it was decided to name the new class `PPSAlignmentConfiguration` instead of `PPSAlignmentConfigRun3v1`.

4.3.4. Phase 4 – October–December 2021

In the first part of the last phase, the code developed in the second phase was finally used to add alignment to the PCL and add the alignment matrix test. The next PR consisted of these changes and other updates, including refactoring, fixing some bugs and replacing `PPSAlignmentConfig` with `PPSAlignmentConfiguration` in the worker and the harvester. However, it quickly turned out that the matrix test was faulty. It did not crash but returned no results. Since it was not crucial for this PR, the PR was merged into CMSSW on 20 October. The fix was implemented quickly and was merged in ten days. During the rest of the phase, minor tasks accumulated over time were successfully marked as done in the TODO list. All of these changes were merged in the last PR on 7 December.

The documentation was completed in November. Most of it was written in this phase. However, two sections – descriptions of the configurable parameters and integration with the Conditions DB guide – were already prepared in the previous phases. Since all the requirements were precise, there were no important meetings in that period.

4.4. Significant obstacles

A few significant obstacles were encountered during the development of the project. They are briefly described in this section.

4.4.1. Updating the `PPSAlignmentConfig` class

In the first and the second phase, the `PPSAlignmentConfig` class was updated even though it had objects in the DB, so it could not be updated in CMSSW. When this was discovered,

there had already been two unnecessary pull requests merged registering that class in the DB and backporting it to the previous CMSSW version. If the integration of CMSSW modules with the Conditions DB had been appropriately documented, that issue could have been avoided.

4.4.2. Bug in the validation workflow

Implementation of the alignment matrix test was based on the matrix test of another component. It turned out that the implementation was faulty – there was a problem with passing the data from one step of the test to another. Because of the poor documentation of the PCL, it was challenging to find the issue. In the end, the solution was found by looking at many other matrix tests, one of which proved to be similar to the alignment matrix test and work correctly. The fix was implemented and could later be applied in other PPS projects added to the PCL.

4.4.3. Issues with fitting in the vertical alignment

Among other things, ROOT is used to fit lines to two-dimensional graphs in the vertical alignment. In early 2021 a bug concerning those fits was found. Sometimes, when alignment was performed on low-statistics (few events) data, fits applied by ROOT were odd, e.g. the slope of the line was equal to 1000 even though it should have been around 1. Since the problem was rare, it was not prioritised and was not solved until the fourth phase. Many solutions were considered, but the one that fixed the issue was found by chance. Once, while experimenting with the code, the graph (`TGraphErrors`) was replaced with a histogram (`TH1D`). This modification solved the problem entirely – the fits became reasonable. The effect is shown in Figure 4.3. The reason behind the odd fitting in the `TGraphErrors` class was not thoroughly investigated, but it might have been caused by handling the points with small uncertainties incorrectly.

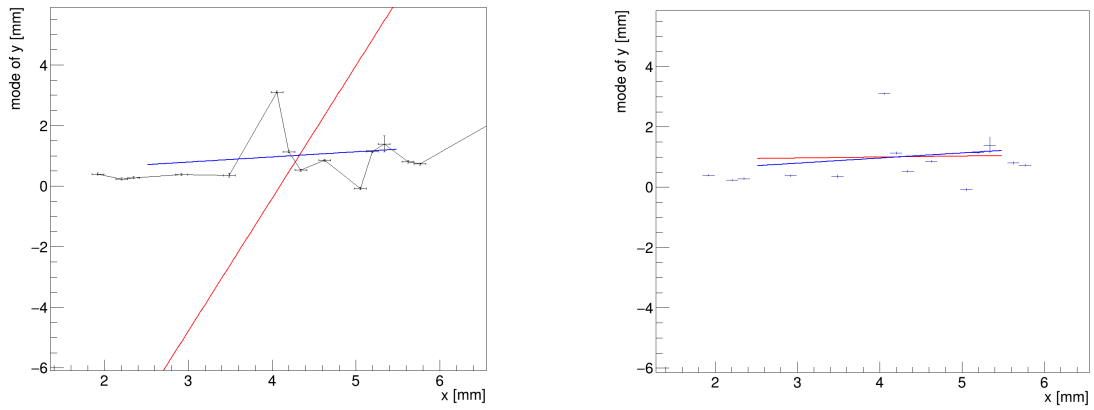

Figure 4.3: The effect of solving the odd fitting issue in the mode graph (the vertical alignment); the blue line was fitted with the slope fixed, the red line with the slope free. Left: before fix, right: after fix – the red line more reasonably fitted to the data. Data from fill 6300

5. Project results

5.1. Final product

The final product fulfils all the functional and non-functional requirements. Some of the realisation details were described in the third chapter. More general conclusions are provided in this section.

5.1.1. Configuration

The software is highly configurable. All the required parameters have sensible default values and can easily be adjusted in CMSSW Python configuration files. The configuration can be delivered locally using the `PPSAlignmentConfigurationESSource` module. It can also be prepared earlier and saved in an SQLite file. Such a file can be uploaded to the Conditions DB. The software can be configured to retrieve the configuration from the DB. The documentation includes proper instructions and examples.

5.1.2. Data selection

At the beginning of the alignment procedure, the `PPSAlignmentWorker` module reads particle tracks and performs the data selection. The previous software employed the multiplicity selection to avoid analysing crowded events. It utilised two parameters – the minimum and the maximum number of the tracks detected in an RP. They were always set to 0 and 2, respectively. To mimic the new filter intended to be used in the PCL, these two parameters have been set to 1 so that only one track in each RP is permitted. The impact of this modification on selected events and results has been investigated.

First of all, the plots created after applying the filter, but before applying the cuts, are much more clear, as in Figure 5.1.

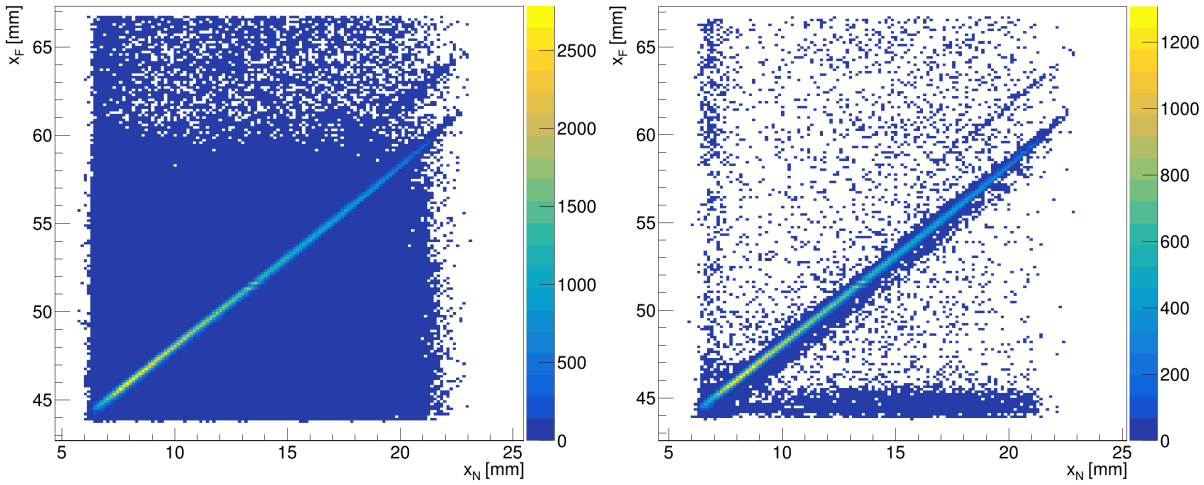

Figure 5.1: Comparison of the multiplicity selection in the previous software (left) and the new (right). Now, the software mimics the filter employed in the PCL which results in more clear plots. Data from fill 7334

Because of the stricter data selection, the alignment results are also different. The differences for the horizontal alignment are shown in Table 5.1. Even the biggest difference ($80\,\mu\mathrm{m}$) is

acceptable. Figure 5.2 shows the plot for a fill-RP configuration with even bigger horizontal shift difference. The small shift in the plot is also acceptable.

	old [μm]	new [μm]
RP 3	-3500	-3470
RP 23	-41490	-41570
RP 103	-2630	-2580
RP 123	-41770	-41740

Table 5.1: Comparison of the horizontal alignment results after the old data selection and the new – mimicking the new filter. Data from fill 7334

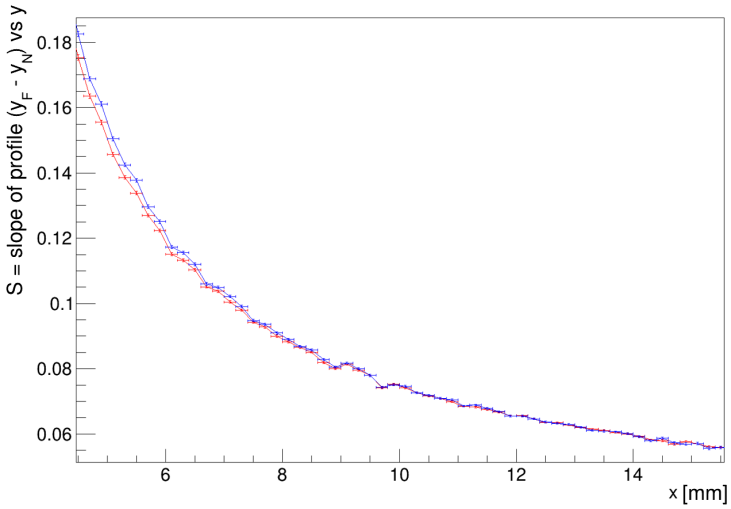

Figure 5.2: Comparison of the S vs x plot created during the horizontal alignment in the previous software (red) and the new (blue). Data from fill 6554, RP 3 – this configuration had the biggest difference between the horizontal alignment results. Little shift between the blue and the red curve is acceptable.

Similar experiments have been run for the two other algorithms. The filter turned out to have little impact on the relative horizontal alignment results. The differences in the vertical alignment results were more significant but still acceptable. What is important, the results made sense after comparing the corresponding mode graph plots.

5.1.3. Alignment analysis

The software enables running all three required alignment methods. The default and suggested order is: horizontal alignment, relative horizontal alignment, vertical alignment. By modifying one of the parameters, the order can be changed. Many tests were performed, and each reproduced the results from the previous software. There were only little differences stemming from the stricter data selection. The comparison of calculated alignment constants with their uncertainties was done by checking the differences in the text files with the results produced by the software. The plots were compared by drawing them on the same canvas in TBrowser (example in Figure 5.2). Comparison of example text results is shown in Table 5.2.

At the end, the results from all the methods are merged. These final results can be saved to an SQLite file by setting one of the harvester parameters to `True`. Much attention has been paid to saving many illustrative debug plots to an extra ROOT file. These plots proved valuable many times during debugging.

	horizontal		rel. horizontal		vertical	
	old [μm]	new [μm]	old [μm]	new [μm]	old [μm]	new [μm]
RP 3	-3480	-3470	18984	18986	3492	3460
RP 23	-41490	-41570	-18984	-18986	4289	4200
RP 103	-2630	-2580	19479	19483	3036	2961
RP 123	-41770	-41740	-19479	-19483	3357	3586

Table 5.2: Comparison of representative example results from the old and the new software (without uncertainties). Small differences derive from slightly different reference data and more strict data selection in the new software. The relative horizontal and the vertical alignment results without fixed slope. Data from fill 7334

5.1.4. Integration with the Conditions DB

The `PPSAlignmentConfiguration` class is fully integrated with the Conditions DB. As every object needs to be tagged in the DB, a convention consistent with the tag naming rules has been settled:

- *PPSAlignmentConfiguration_reference_v1_express* for the reference conditions objects (*PPSAlignmentConfig_reference_Run3_v1_express* for Run 3 PCL),
- *PPSAlignmentConfiguration_v1_express* for the physics fill conditions object (*PPSAlignmentConfig_Run3_v1_express* for Run 3 PCL).

Incrementing the number in the version segment (*v1*) might be used in some cases to update the object associated with the tag instead of overwriting it.

A few `PPSAlignmentConfiguration` (and earlier `PPSAlignmentConfig`) objects have been uploaded to the DB for test purposes. The evolution of the class and used tags can be viewed in the Conditions DB Browser service (Figure 5.3).

Tag	Time Type	Object Type	Synchronization	Insertion Time	Description
PPSAlignmentConfig_reference_test_v1_prompt	Run	PPSAlignmentConfig	any	2021-04-30 19:08:07 (UTC)	New Tag
PPSAlignmentConfig_test_v1_prompt	Run	PPSAlignmentConfig	any	2021-04-30 19:10:40 (UTC)	New Tag
PPSAlignmentConfig_v1_prompt	Run	PPSAlignmentConfig	any	2021-08-06 21:05:27 (UTC)	New Tag
PPSAlignmentConfig_reference_v1_prompt	Run	PPSAlignmentConfig	any	2021-08-06 21:09:05 (UTC)	New Tag
PPSAlignmentConfiguration_v1_express	Run	PPSAlignmentConfiguration	hlt	2021-10-01 22:02:43 (UTC)	New Tag
PPSAlignmentConfiguration_reference_v1_express	Run	PPSAlignmentConfiguration	hlt	2021-10-01 22:05:37 (UTC)	New Tag
PPSAlignmentConfig_Run3_v1_express	Run	PPSAlignmentConfiguration	any	2021-11-05 17:44:37 (UTC)	New Tag
PPSAlignmentConfig_reference_Run3_v1_express	Run	PPSAlignmentConfiguration	any	2021-11-05 17:47:24 (UTC)	New Tag

Figure 5.3: List of all the tags containing 'PPSAlignmentConfig' in their names, generated in the Conditions DB Browser service on 15 December 2021

5.1.5. PCL

Alignment has been configured to run in the PCL. A matrix test that uses most of the modules and the files used in the alignment PCL workflow has been added. It produces reasonable plots and results. Adding a new workflow to the PCL required to provide memory consumption plots for each of the employed DQM modules. Such plots were created using the *PCLValidationSuite* tool [32] and included in the description of one of the pull requests. They are shown in Figure 5.4.

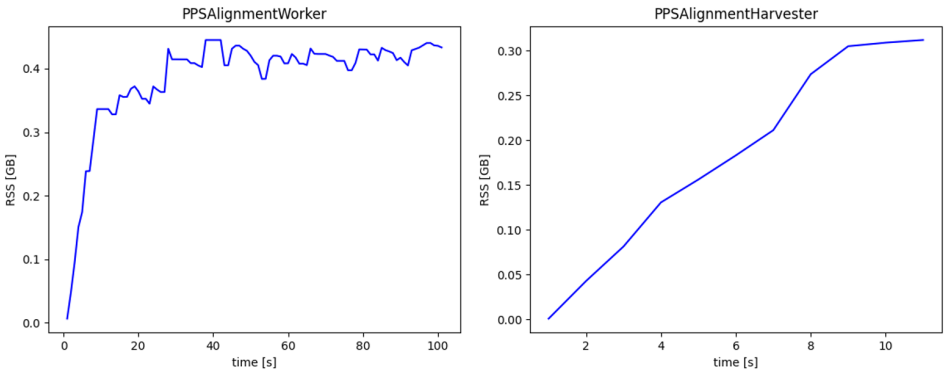

Figure 5.4: Memory consumption plots for the worker and the harvester. RSS stands for the resident set size which is the RAM memory portion consumed by a process.

5.1.6. Non-functional requirements

The software has been merged into CMSSW in a few pull requests. Each of them met the requirements and was eventually approved by reviewers.

Many test configurations have been prepared. Each of them works as expected and reproduces the corresponding results from Run 2. One of these tests has also been placed in the *CalibPPS/AlignmentGlobal/test* folder in CMSSW. The rest of them have been uploaded together to a new GitHub repository inside the CTPPS organisation – *pps-alignment-data*. It consists of many example configuration files, mainly for 2018 data. The files can easily be switched to generate an SQLite file or retrieve the data from it. The necessary input can be retrieved from the Conditions DB, too.

A TWiki page has been created for the user documentation. It consists of an introduction to the problem, descriptions of the modules and their configurable parameters with the default values, the DB-related procedures with explanations and some general guidelines.

DQM worker modules can be configured to use many threads. Filling the histograms in the alignment worker is thread-safe and hence data processing is effective. Both the worker and the harvester produce detailed logs. Every log has a proper severity assigned – *Info*, *Warning* or *Error*. In the process configuration, one can select the logging threshold and, for instance, make the software generate only error logs.

5.1.7. Use cases

The software supports the two use cases described in the second chapter. The automatic use case is equivalent to running alignment in the PCL. The manual use case is supported, too. By modifying the configurable parameters, one can obtain the most accurate results. The software produces lots of debug plots. They can be analysed to determine which parameters need to be modified to increase accuracy.

5.2. Conclusions

The project has been completed successfully. Each requirement has been met, and the software works as expected. Many obstacles have been encountered, including remote work, poor documentation and a few significant bugs stemming from the complexity of the CMSSW environment. In the end, all of them have been overcome.

Even though the project has been completed and deployed in CMSSW, it still has not been used in production. The LHC Run 3 is being launched in the first half of 2022, which will be the final test for the software. The code is flexible, and many parameters which now are redundant might turn out necessary. Therefore, the configuration will inevitably have to be updated at least a few times at the beginning of Run 3.

Personally, the project was a great adventure for me. I had an opportunity to work for one of the most prominent institutions on Earth and meet fantastic people. My software is a part of the Large Hadron Collider, which makes me feel excited. I also managed to learn a lot. Understanding the complex CMSSW ecosystem was difficult, but now I find myself more confident in new environments. Apart from that, I improved my coding and algorithmic skills. Dealing with various obstacles encountered during the development improved my problem-solving skills. Last but not least, the project required discussing many complex issues via emails or online meetings. Over time, I learned how to deliver detailed information effectively.

Appendix A. List of the project's pull requests to CMSSW

1. PR #31728: *PPS: global alignment*. URL: `https://github.com/cms-sw/cmssw/pull/31728`
2. PR #34844: *PPS Alignment – bug fix*. URL: `https://github.com/cms-sw/cmssw/pull/34844`
3. PR #34845: *PPS Alignment – bug fix (backport)*. URL: `https://github.com/cms-sw/cmssw/pull/34845`
4. PR #35174: *Added PPSAlignmentConfiguration – a new conditions format for PPS global alignment*. URL: `https://github.com/cms-sw/cmssw/pull/35174`
5. PR #35631: *Added PPS Alignment to the Prompt Calibration Loop*. URL: `https://github.com/cms-sw/cmssw/pull/35631`
6. PR #35874: *Fixed the PPS alignment relval – 1042*. URL: `https://github.com/cms-sw/cmssw/pull/35874`
7. PR #36257: *PPS global alignment – reformat*. URL: `https://github.com/cms-sw/cmssw/pull/36257`

Glossary

alignment The process of finding the positions of RPs with respect to the LHC beam. This project focused on the global alignment, that is the alignment of RPs with the data from physics fills. The main steps of the global alignment are: data selection, horizontal alignment, relative horizontal alignment and vertical alignment. Its goal is to calculate the alignment constants, that is the horizontal and the vertical shifts of RPs with respect to the beam.

arm An LHC sector clockwise (s56) or anticlockwise (s45) of CMS

CERN The European Organisation for Nuclear Research

CMSSW (CMS SoftWare) The software framework of the CMS experiment [17]

Compact Muon Solenoid (CMS) A general-purpose detector at the LHC, built at IP5 [6, 7]

Conditions Database (Conditions DB) The database of CMS [22]; stores conditions data, i.e. the objects containing the auxiliary information needed in the processing of Events. Each object in the DB is associated with a tag – a label used, e.g. to group the objects with similar purposes. Each object within a tag is valid only in its Interval of Validity (IOV).

conditions format A container C++ class for conditions data (e.g. the alignment configuration) used by the EventSetup mechanism

Data Quality Monitoring (DQM) A subsystem of CMSSW dedicated to monitoring and validation of acquired data. The DQM workflow is used in this thesis. The main components of this workflow are worker and harvester modules.

ESSource A CMSSW module that reads a Python configuration file, produces an instance of a conditions format and associates it with a Record.

Event A C++ container class for all raw and reconstructed data related to a particular collision. It is used to pass the data from one module to another during the processing in CMSSW. This data processing model is named the Event Data Model (EDM) [19, 20]. In this thesis, the uncapitalised 'event' means a collision or the data related to a collision at the IP.

EventSetup A mechanism that provides auxiliary information needed to process an Event [21]. In this project, this mechanism is used to handle the alignment setup.

forward/leading/scattered protons Protons scattered to small angles with respect to the LHC beams

harvester A `DQMEDHarvester` module that reads the histograms produced by the worker module and performs the alignment analysis.

Interaction Point (IP) A place in the LHC, where the two beam pipes intersect and particles can collide (interact).

Large Hadron Collider (LHC) The world's most powerful particle accelerator [4, 5]. It can accelerate particle beams to a speed close to the speed of light.

LHC fill An operational cycle or a period of time between an injection of particle bunches and dumping them; can be divided into several runs. Two fill types are considered in this thesis: calibration fill (a special low-intensity fill) and physics fill (a standard LHC fill).

LHC Run A running period of the LHC; Run 1: 2009–2013; Run 2: 2015–2018; Run 3: expected to start in 2022

LHC sector The LHC is divided into 8 sectors. The CMS detector is located between two sectors – sector 45 anticlockwise of IP5 and sector 56 clockwise of IP5.

low-latency mode Within 48 hours after data acquisition. It implies the automatic use case of the software.

matrix test A validation workflow of a CMSSW component. The matrix tests are used to test CMSSW releases, and thus the mechanism is named 'release validation' (RelVal).

offline mode Sizeable time after collecting data. It allows employing the manual use case of the software.

Precision Proton Spectrometer (PPS or CMS-PPS) A CMS subsystem with the goal to study forward protons [10]. It used to be a joint project of CMS and TOTEM (CT-PPS).

Prompt Calibration Loop (PCL) A low-latency workflow used in the reconstruction of CMS events

Record (EventSetup Record) An instance of a conditions format has to be encapsulated in an EventSetup Record in order to be used in data processing or be uploaded to the Conditions DB.

Roman Pot (RP) A vessel inserted into the LHC pipe [11]; can host the PPS sensors. The alignment procedure uses two RPs in each of the two sectors. The ones closer to the IP are called 'near' ('up') and the other ones – 'far' ('down').

ROOT A data analysis framework used in high-energy physics [24, 25]

TWiki The wiki system of CMS

worker A `DQMEDAnalyzer` module that fills histograms with data and can be configured to run in parallel. After being filled, the histograms are analysed in the harvester module.

References

[1] *CERN: About*. URL: https://home.cern/about (visited on 04/04/2021).

[2] The ATLAS collaboration. 'Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC'. *Physics Letters B* 716.1 (2012), pp. 1–29. DOI: 10.1016/j.physletb.2012.08.020.

[3] The CMS collaboration. 'Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC'. *Physics Letters B* 716.1 (2012), pp. 30–61. DOI: 10.1016/j.physletb.2012.08.021.

[4] Lyndon Evans and Philip Bryant. 'LHC machine'. *Journal of instrumentation* 3.08 (2008), S08001. DOI: 10.1088/1748-0221/3/08/S08001.

[5] *CERN: LHC*. URL: https://home.cern/science/accelerators/large-hadron-collider (visited on 04/04/2021).

[6] The CMS collaboration. 'The CMS experiment at the CERN LHC'. *JINST* 3.08 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004.

[7] *CERN: CMS*. URL: https://home.cern/science/experiments/cms (visited on 05/04/2021).

[8] The TOTEM collaboration. 'The TOTEM Experiment at the CERN Large Hadron Collider'. *JINST* 3.08 (2008), S08007. DOI: 10.1088/1748-0221/3/08/S08007.

[9] The CMS Collaboration. *The CMS Precision Proton Spectrometer at the HL-LHC – Expression of Interest*. CERN-CMS-NOTE-2020-008. 2021. URL: https://cds.cern.ch/record/2750358.

[10] The CMS and TOTEM collaborations. *CMS-TOTEM Precision Proton Spectrometer*. CERN-LHCC-2014-021. TOTEM-TDR-003. CMS-TDR-13. 2014. URL: https://cds.cern.ch/record/1753795.

[11] Marco Oriunno et al. *The Roman Pot for The LHC*. MOPLS013. Proceedings of EPAC 2006, Edinburgh, Scotland. URL: https://accelconf.web.cern.ch/e06/PAPERS/MOPLS013.PDF.

[12] Jan Kašpar. *Alignment of CT-PPS detectors in 2016, before TS2*. 2017. URL: https://cds.cern.ch/record/2256296.

[13] Jan Kašpar. *Elastic scattering at the LHC*. PhD thesis. 2011. URL: http://cds.cern.ch/record/1441140.

[14] The CMS and Totem Collaborations. *Proton reconstruction with the Precision Proton Spectrometer (PPS) in Run 2*. CERN-CMS-DP-2020-047. 2020. URL: http://cds.cern.ch/record/2743738.

[15] Stephane Fartoukh et al. *LHC Configuration and Operational Scenario for Run 3*. CERN-ACC-2021-0007. 2021. URL: https://cds.cern.ch/record/2790409.

[16] *GitHub code repository for the alignment analysis in 2018*. URL: https://github.com/CTPPS/analysis_ctpps_alignment_2018.

[17] *CMSSW repository on GitHub*. URL: https://github.com/cms-sw/cmssw.

[18] M. Rovere. 'The Data Quality Monitoring Software for the CMS experiment at the LHC'. *Journal of Physics: Conference Series* 664.7 (2015), p. 072039. DOI: 10.1088/1742-6596/664/7/072039.

[19] C. D. Jones et al. 'The new CMS Event Data Model and Framework'. *Proc. CHEP* (2006). URL: https://indico.cern.ch/event/408139/contributions/979800.

[20] *CMSSW framework – main rules and features*. URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookCMSSWFramework (visited on 23/06/2021).

[21] C. D. Jones. 'Access to non-Event data for CMS'. *Proc. CHEP* (2006). URL: https://indico.cern.ch/event/408139/contributions/979797.

[22] S. Di Guida et al. 'The CMS Condition Database System'. *Journal of Physics: Conference Series* 664.4 (2015), p. 042024. DOI: 10.1088/1742-6596/664/4/042024.

[23] R. Wilkinson, B. Hegner and C. D. Jones. *Using Python for Job Configuration in CMS*. CMS-CR-2009-109. 2009. URL: http://cds.cern.ch/record/1196114.

[24] Rene Brun and Fons Rademakers. 'ROOT – An object oriented data analysis framework'. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 389.1-2 (1997), pp. 81–86. DOI: 10.5281/zenodo.3895860.

[25] *ROOT*. URL: https://root.cern/.

[26] David Futyan. 'Commissioning the CMS alignment and calibration framework'. *Journal of Physics: Conference Series* 219.3 (2010), p. 032041. DOI: 10.1088/1742-6596/219/3/032041.

[27] Jaime Maria Coello De Portugal – Martinez Vazquez. *Local Optics Corrections in the HL-LHC*. PhD thesis. 2021. URL: https://cds.cern.ch/record/2778023.

[28] *CMSSW – naming, coding, and style rules*. URL: http://cms-sw.github.io/cms_coding_rules.html (visited on 21/06/2021).

[29] *DQM services – core*. URL: https://github.com/cms-sw/cmssw/tree/master/DQMServices/Core (visited on 02/11/2021).

[30] *CalibPPS/AlignmentGlobal package*. URL: https://github.com/cms-sw/cmssw/tree/master/CalibPPS/AlignmentGlobal.

[31] *pps-alignment-data repository on GitHub*. URL: https://github.com/cms-sw/cmssw.

[32] *PCLValidationSuite repository on GitHub*. URL: https://github.com/mmusich/PCLValidationSuite.