A graphical user interface for the Millepede production system

Luis Alberto Sánchez Moreno
Centro de Radioastronomía y Astrofísica, UNAM
58089, Morelia, Mexico
Email: l.sanchez@astrosmo.unam.mx

Track-based alignment procedure for CMS involves the determination of $\mathcal{O}(100,000)$ parameters which handle each module position and orientation. Three algorithms have been suggested to address this issue, we will focus on Millepede-II which is a non-iterative method based on linear least-squares fitting that has been successfully used in other experiments. An environment for using Millepede in CMS tracker alignment known as the Millepede production system has been developed, in this paper an implementation of a GUI in Perl/Tk for this production system is described.

1 Introduction

1.1 The CMS detector

CMS (*Compact Muon Solenoid*) is one of the two general purpose detectors [1] located at LHC interaction points. It is designed to cover the full solid angle and most importantly to resist the harsh radiation environment that is expected from the high LHC luminosity.

Both the tracker and the electromagnetic/hadron calorimeters are enclosed inside the superconducting magnet, this solenoidal magnet operates at 4 T and stores 2.7 GJ in its magnetic which is the highest energy achieved at a detector and its crucial to its "compact" design as it bends the trajectories to a relatively small radius, besides the obvious fact that it is large enough to allow both the tracker and calorimeters inside the coil. Great emphasis has been placed in accurate measurement of high p_T particles, specially muons.

1.2 CMS tracker

The tracker is the device responsible for reconstructing the trajectories and momentum of the particles emerging from the *pp* collisions, in particular the CMS tracker is the largest silicon detector ever build and represents a significant advance in detectors technology.

A major requirement has been to achieve at least a 10% accuracy in the determination of the transversal momentum p_T, specially for muons as this allow us to determine secondary vertexes, especially for heavy hadron interactions [3]. This implies that spatial accuracy and the relative alignment of all the planar elements that make up the detector is of utmost importance.

The tracker system consists of a series of concentric cylindrical detectors located inside a solenoid magnet that create an axial magnetic field, this geometry is commonly used in detectors as it produces circular trajectories in the azimuthal plane (r, ϕ).

The detectors form a cylinder by assembling a set of rectangular sub-assemblies with the right orientation. This design is claimed to achieve a 98% reconstruction efficiency for muons which is only possible due to the high granularity in the innermost regions of the tracker.

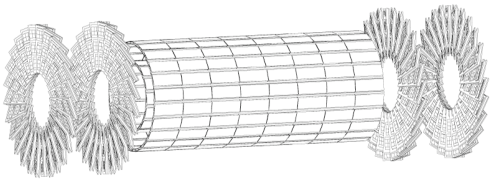

Fig. 1: *The pixel detector is composed by two barrel layers, it is located surrounding the immediate vicinity of the interaction region at 4 and 7 cm at low luminosity and at 7 and 11 cm at high luminosity, while the endcaps go from a radii of 6cm to 15 cm.*

The layout comprise 25,000 silicon sensors having a composite surface of 210 m^2. The diameter is 2.4 m and the length is 5.4 m. The innermost component is the pixel detector which is divided in two structures: the pixel barrel (PB) for transversal trajectories and the pixel endcaps (PE) for forward trajectories. If we move a bit outward we arrive to the silicon strip tracker which is also composed of several subdetectors: the tracker inner barrel (TIB), the tracker outer barrel (TOB), the tracker inner disks (TID) and the tracker endcaps (TEC).

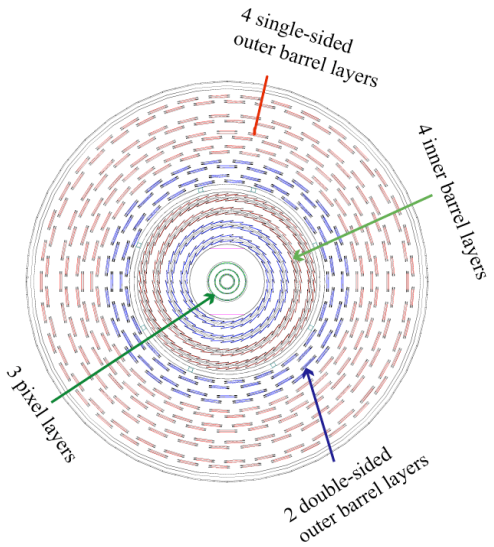

Fig. 2: *Layout for the silicon strip detector which is composed by several sub-detectors, namely the TIB, TOB, TID and TEC.*

1.3 Why a silicon detector?

Gas detectors are intrinsically limited in position reconstruction due to diffusion, where the limit on the localization of the drifting ionization is of order 100 μm from drift distances $\sim$ 1 cm. This means that if we want to resolve the main decay vertex from secondary vertex which usually are separated by $c\tau \sim$ 10-50 μm (where τ is the "intrinsic" lifetime of the particles created in the main vertex) we need a detection element with better spatial resolution than proportional wire chambers [2], for this reasons silicon detectors are the first choice for inner tracking detectors.

To see that silicon detectors are indeed suitable to this task, consider the typical values of the "pitch" P, which is the spacing of the electrode strips which collect the signals in silicon detectors, that are usually in the range of 20-100 μm.

Assuming uniform illumination if only which strip is recorded then the resolution is the pitch divided by $\sqrt{12}$ as can be shown with a quite simple calculation.

$$\sigma^2 = \langle (y - \langle y \rangle) \rangle^2 \quad (1)$$

$$= \frac{\int y^2 \, dy}{\int dy} \quad (2)$$

$$= \frac{\int_{-P/2}^{P/2} y^2 \, dy}{\int dy} \quad (3)$$

$$= P^2/12 \quad (4)$$

For a pitch of P=50 μm then the resolution is 15 μm. A more detailed calculation shows that for a drift voltage of 50 V and a drift distance of 300 μm then $\sigma_x \sim 10$ μm.

This is just the resolution we need for an excellent identification of heavy flavour hadrons, which will be produced in great quantities at QCD processes in LHC. For this reason the decision of using silicon detectors for all the tracking system was taken, this also has the advantage of precise momentum measurement through all the measured path of the particles.

1.4 Tracker alignment: an overview

To achieve high precision (the design value is around 10 μm) in particle trajectory determination is of crucial importance to know with high precision the position of every single element of the tracker, unfortunately enough this is impossible to achieve at this level of precision by mechanical means.

At the time of assembly utmost care was taken to place each piece as close as possible to the design position using precision measurements. Also a laser alignment system (LAS) is integrated into the silicon strip tracker, for this purpose some of the silicon detectors in the TEC are transparent to infrared wavelengths and are located at rings 4 and 6 where detectors that measure a laser which follows a trajectory through the TEC-TIB-TOB can measure the position of each element [4] .

Despite that, previous experience has shown that track-based alignment is the optimal method for aligning large detectors, for CMS this is a complicated issue as every module needs 6 parameters (3 for position, 3 for rotation)

which implies around $\mathcal{O}(100,000)$ parameters for the whole detector.

Three alignment algorithms have been implemented. The first one is the Hits and Impact method which minimizes a *local* χ^2 function constructed from the track hit-residuals on the sensor, this is an iterative method and its rather computationally light as it avoids large matrix inversion. The Kalman filter algorithm is another iterative method that avoids large matrix inversion and can use prior information from mechanical alignment surveys and the laser alignment, in this method the speed of convergence depends on the layer. Finally, Millepede is a well tested algorithm which in its previous version has been used successfully at H1, CDF and other experiments. This is a non-iterative method based on linear least-squares fitting and achieves high precision, it involves the inversion of large matrices but this new version is equipped with efficient algorithms for sparse matrices and can handle the problem in reasonable CPU time.

As mentioned before CMS will perform a track based alignment, with data samples coming from [5] :

- **High p_T muons from Z,W**: These are almost ideal tracks for alignment because of their high transversal momentum and little scattering in the tracker material. This requires a powerful enough beam to produce them, so they might not be available as soon as the LHC is commissioned.

- **Cosmic muons**: The advantage of cosmic rays is that they are available before the beams start colliding so we can start the alignment process in advance, they are specially useful for the barrel tracker and the muons detectors.

- **Beam halo muons:** Near horizontal beam halo beam are valuable for alignment of the endcaps, this will be available as soon as the LHC is commissioned with single beams.

- **Muons from J/Ψ and b hadron decays**: This muons despite lacking high p_T are useful since they will be available since the early phases of LHC when luminosity will be still rather small.

- **Isolated tracks from QCD events:** At low luminosities these tracks are the only option available, evidently they suffer from multiple scattering although they might still be useful for the pixel detector.

2 The Millepede algorithm

Most track based algorithms are based on the χ^2 minimization principle. In the CMS tracker one usual track consists of 20 measurements, a helix track can be determined from only five parameters which implies that it is overdetermined from measurements. Then this measurementes u_m are compared to predictions of the track model. The algorithms try to minimize the residuals between the track hits and the track model.

The Millepede algorithm is a linear least-squares method focusing on certain type of problem where parameters can be separated into *global* and *local*. This method is relevant for tracker alignment based on tracks measurement where we are interested in finding adequate values for global parameters which specify the position of the modules and not in the details of the track itself.

The predicted measurements u_n depend for track j, on the vector of track parameters τ_j and the parameters $\mathbf{p}$ that describe the position, orientation and deformation of the detectors. The residual is then $u_{ij;m} - u_{ij;p}(\tau_j, \mathbf{p})$ making the normalized residual z_{ij} given by:

$$z_{ij} = \frac{u_{ij;m} - u_{ij;p}(\tau_j, \mathbf{p})}{\sigma_{ij}}. \tag{5}$$

Then χ^2 is given by

$$\chi^2(\tau, \mathbf{p}) = \sum_j \left(\sum_j z_{ij}^2(\tau_j, \mathbf{p}) \right) \tag{6}$$

which should be minimized for all τ_j and $\mathbf{p}$. Generally, all the overdetermined parameters are usable for alignment purposes.

Minimizing this function is a complicate issue and we will only sketch it, for more details see [7]. It first involves the linearization of normalized residuals, this essentially consists of creating a a matrix with the values of the residuals and its first order derivatives. The geometric correction parameters $\mathbf{p}$ are known as *global* parameters as they are not specific to a single track or event. The parameter corrections for a track are specific of the event to which the track belongs so the τ_j are known as local parameters. The number of local parameters can be order of

millions, nonetheless we are only concerned with global parameters which are around 50 000.

The next step is matrix reduction, which is mandatory considering the size of the matrix. The resulting matrix only contains global parameters and is only possible because the original matrix has some particular structure.

Constraints are then imposed using Lagrange multipliers and the resulting system should be solved, for this purposes efficient sparse matrix methods are used which allow a computationally feasible task, otherwise the CPU time will be excessive for practical purposes.

Because analysis time is a major concern it is crucial to collect information efficiently in Millepede. The collection of measurements and derivatives from data has been parallelized. For this purposes the program is split in two parts, one part (Mille) produces binary files with the data needed for the alignment procedure and has been interfaced to the CMS softwarem the remaining part (Pede) determines the alignment parameters from the binary input and is a standalone FORTRAN program. This structure also allows Pede to be used in other experiments if required as it is kept experiment independent.

3 The Millepede production system

The Millepede production system (MPS) is a set of Perl scripts developed to allow the user to run a large number of Mille jobs, to fetch the output, and feed it into the Millepede program.

The Millepede II procedure is divided in two main steps: Mille and Pede. The first step, Mille, processes all tracks and hit residuals and prepares the information for the final global fit, which is written out to a "Millepede binary file". The second step, Pede, reads the information from all binary files and performs the global fit that results in the full set of alignment constants.

3.1 Usage of MPS

The main idea of MPS [8] is that it will set up and control a job series performing one Mille-Pede alignment task. Every job series should be set up in its own particular directory, and the MPS commands will create and maintain a subdirectory structure and additional files in this directory. All MPS commands referring to a job series must be issued from the same UNIX working directory. The setup parameters and the current state of each job are maintained in a file named *mps.db*, all job-specific information is stored in a directory tree named *jobData*. For each job a directory jobData/jobNNN is created, where NNN stands for the three-digit job number. The maximum number of Mille jobs MPS sets up in parallel is 999. The Pede job is assigned a special directory jobData/jobm. As a result, an arbitrary number of alignment job series can be run in parallel without interfering, but one should never attempt to set up two job series in the same directory. For example, the *mps_stat.pl* command will always show the jobs related to the job series that was initiated in the same directory.

3.2 List of MPS scripts

The MPS consists of a series of scripts implemented in Perl, essentially every routine controls a different part of the process, from the setup to the retrieval of jobs. The scripts are:

- *mps_setup.pl:* This is the main command that sets up a job series and defines what should be done. After mps_setup.pl, the jobs are in SETUP state.
- *mpsfire.pl:* This routine is the one that actually submits the first nJobsSubmit jobs that are in SETUP state.
- *mps_stat.pl:* This command is used to displays the updated status of all jobs belonging to the job series, making it a monitoring tool. The possible status of a job is further explained in fig 3.
- *mps_kill.pl:* Cancel all or a selection of jobs. The cancelled jobs will be moved to FAIL state.
- *mps_fetch.pl:* Fetch all jobs that are in DONE state, including the Pede job if there is one, and perform checks on the output. After *mps_fetch*, the jobs will be moved either to OK or to FAIL state.
- *mps_retry.pl:* In case one job was suspended or ended with the fail status, this routine allows to retry all or a selection of jobs. The retried jobs will once more go to SETUP state, and can be resubmitted with the *mps_fire* command.

- *mps_auto.pl* Calls *mps_stats.pl* and *mps_fetch.pl* in intervals of 'seconds' seconds to update the statistics of the mille jobs. Then it tries *mps_fire.pl -m* to submit the merge job. Quits if that is successful. This means that it is continually monitoring the status of jobs and fetching the output of finished jobs.

- *mps_save.pl:* It essentially saves all the output of the process, which is usually scattered in multiple files.

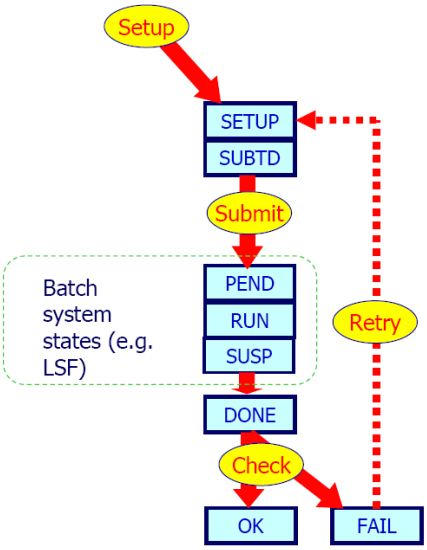

Fig. 3: *We illustrate the situation of jobs status in this diagram. Jobs are first set up (using mps_setup) and then submitted (that's managed by mps_fire). Once the jobs are running is possible to get the status of each one which should pending, running or suspended, after the jobs ends successfully the status set as done, otherwise the user has the option of retrying it.*

4 Implementing a GUI for the MPS

Considering that MPS is written in Perl, the use of Perl/TK for the GUI follows rather directly. It should be added that there are many GUI libraries written for Perl like Gtk2-Perl, although Perl/Tk is probably the most stable one, at least it is the most tested as it has been used for many years in all sorts of projects allowing enough opportunities to iron most of the wrinkles of the code. Another thing that should be mentioned is that the GUI itself can be done in any other language as it only calls for Perl scripts, so it is in principle possible to do it in some fancier library like Qt which also has a fancy IDE.

The decision for Perl/Tk was mostly based on the fact that is an interpreted language that has been ported to practically every platform/operating system so it can run on practically every computer in which it is correctly configured. Because the code is interpreted there is no necessity to recompile binary files if a different computer is used so the user does not have to worry about compatibility issues, this is specially important when a new operating system is installed (or more usually, just updated) in the lxplus cluster[1] which is something to be expected to happen every few years.

For our work we found an extremely useful (and rather unknown) tool for GUI building known as ZooZ [9]. Using ZooZ it's possible to build a rudimentary GUI that can be exported to Perl/Tk and once this foundational code is obtained it's possible to modify it, and because it is written in Perl we can essentially use the same kind of code that is already implemented in MPS.

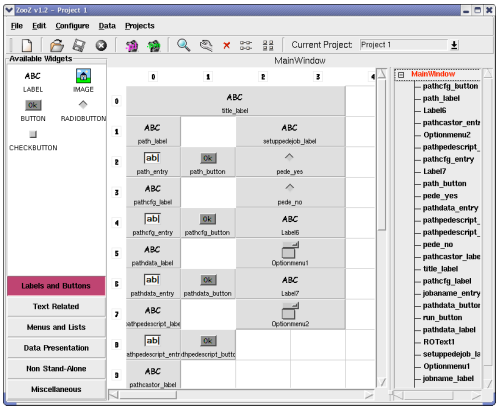

Fig. 4: *ZooZ is a Perl/Tk application that allows the production of GUI's in a simple way inside a graphical environment. The elements of the GUI are available as widgets and their properties can be easily edited. It uses its own format for storing the windows implemented in it but it can easily export it to Perl/Tk which is then possible to modify in any text editor.*

We started implementing an interface for *mps_setup.pl*, that is certainly the most complex, as the other scripts have far less parameters (if any at all). For this task we essentially

[1] Current implementations of MPS *must* run on CERN's lxplus cluster.

needed user input for specifying paths to data, a config file and some scripts, in addition to that the number and the batch queue of jobs should be specified and a name for the batch system is also required.

Our first GUI for this was a rather crude implementation where we only had some input fields for every single of the *mps_setup*, this was essentially the same option as just using the console where the user still needed to write all the parameters in the correct order to get it everything started.

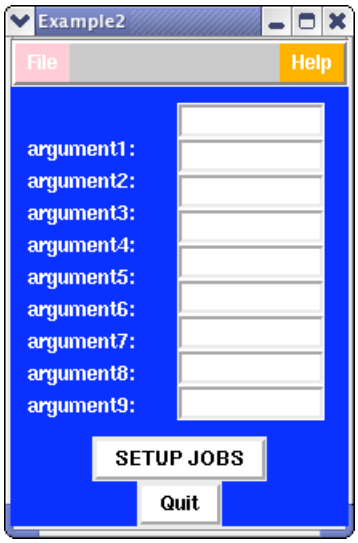

Fig. 5: *Our first attempt of a GUI. It consisted of a collection of input fields for every parameter of MPS, the output was printed in a new window after pressing the SETUP JOBS button.*

After this basic interface had been implemented we designed a new one using ZooZ which allowed us to create a "skeleton GUI" on which we could later add the required functionality. All the paths to user defined directories were changed to a browser input (the user could still manually add a route if desired) and some of the parameters were now available as radio buttons or menus. A new read-only text output field was added to the button of the window allowing immediate visualization within the same GUI so the input and output could be displayed at the same time.

At this stage of development the GUI was more of a "road plan" and little (if any) functionality was actually implemented. The menus were empty, there were no variables associated with the input fields, the buttons did not performed any action when clicked and the output window was not able to fetch anything from the terminal. Despite that it included enough elements (*widgets* in Perl/Tk terminology) to work as a solid base on which to add code to perform the required actions like retrieving the input into variables, present the user a readable menu and presenting the terminal output of MPS in the button part of the window..

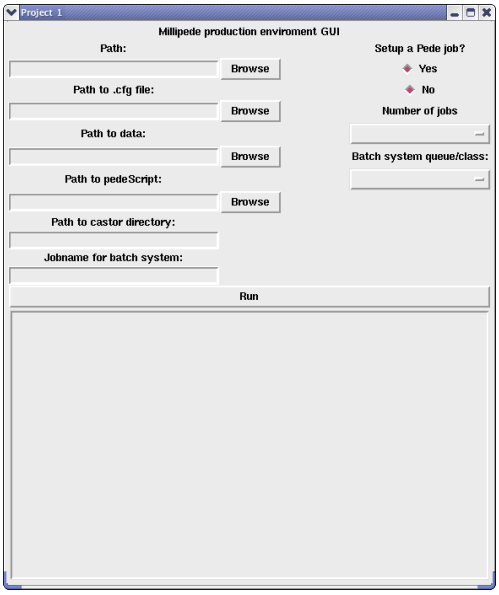

Fig. 6: *The "skeleton GUI" produced by* ZooZ*, the paths to files/directories to which the user has control could now be easily selected in a browser. System paths are intended to be typed in an input field. Finally the parameters controlling the system/queue class and the number of jobs were implemented in menus. The option to set a Pede job was intended to be selected by a radio button. As is evident from the figure, the menus are still empty.*

The next stage of development consisted of actually getting this "dummy" GUI to work, this was essentially achieved by implementing a set of subroutines that managed the task of variable fetching and storing the terminal output in a variable to be displayed. The variables were defined globally which has the advantage of making possible to retrieve their value from any part of the code in case of future modifications that require it.

When this was achieved, we decided to implement three additional scripts to the interface: *mps stat.pl,mpsfire.pl* and *mps fetch.pl* which allows us to have an almost full functionality for the whole MPS . This task was quite similar to the implementation of *mps_setup* and it will not be discussed in detail.

The current implementation of the GUI has basic functionality for the central MPS scripts: it is possible to setup jobs, submit them, fetch them once they are finished and monitor the status of the jobs while they are running.

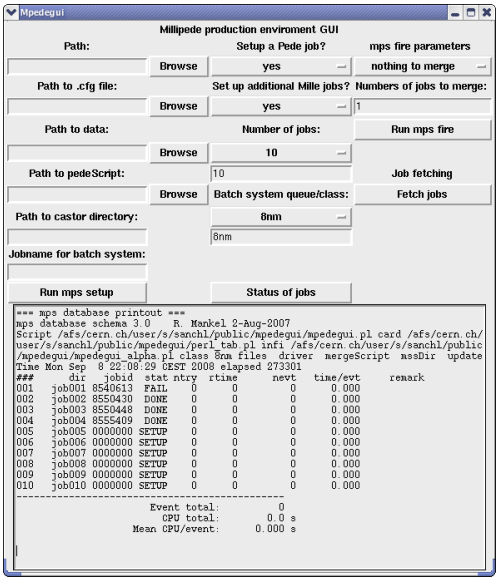

Fig. 7: *The fully functional GUI. The first two columns fetch all the necessary input to setup jobs. The last button of the second column is used to monitor the status of jobs. Once the jobs are set up the third column acquires the necessary information to send and fetch them.*

Acknowledgments

This work was done during DESY Summer Student Program 2008. The author thanks all the people in charge of the program specially Joachin Meyer and Andrea Schrader for organizing an unforgettable experience at DESY-Hamburg.

Financial support for travel expenses came from the particles and fields division of Mexican Physical Society who organizes summer internships for students, with special mention to Heriberto Castilla and Ricardo López.

My supervisor at home, Susana Lizano allowed me to spent two months away at the only time of the year when we can *really* work, it goes without mention all the support from her in all these years.

Finally I am particularly grateful with my supervisors at DESY: Silvia Miglioranzi and Andrea Parenti who have been very friendly and helpful through all the program (even before it actually started) and contributed to a really nice working environment. I just can hope I filled their expectatives for the project!

References

[1] S. Chatrchyan et al. (CMS Collaboration), JINST 3, S08004 (2008). arxiv:0709.3360

[2] Dan Green, *High P_T Physics at Hadron Colliders*, Cambridge University Press, 2005.

[3] Dan Green, *The Physics of Particle Detectors*, Cambridge University Press, 2000.

[4] R. Adolphi, *Construction and calibration of the laser alignment system for the CMS tracker*, PhD. Thesis, RWTH-Aachen, 2006.

[5] F.P. Schilling in *Proceedings of the 1st LHC Detector Alignment Workshop*, 2006.

[6] G. Flucke et al. , *CMS silicon tracker alignment strategy with the Millipede II algorithm. JINST 3, P09002, 2008.* .

[7] Volker Blobel, *Millepede II: Linear Least Squares Fits with a Large Number of Parameters.* Available at *http://www.desy.de/ blobel/mptwo.pdf.*

[8] Reiner Mankel, Andrea Parenti, *CMS Web Documentation: The MillePede Production System (MPS)*, available at *https://twiki.cern.ch/twiki/bin/view/CMS/SWGuideMillepedeProductionSystem#The_mps_setup_Command*

[9] Zooz is available free of charge at *http://search.cpan.org/∼aqumsieh/ZooZ-1.2/ZooZ.pl.*