Table of Contents

Alignment in CMSSW using Millepede II

Complete:

Goal of the page

This page should provide basic knowledge how to run the Millepede II algorithm for alignment as implemeted in the CMSSW package `Alignment/MillePedeAlignmentAlgorithm`. So far only track based alignment is used, but the Millepede II program can incorporate survey and laser system data. This is foreseen to be implemented in the CMSSW interface.

Contacts

- Claus Kleinwort claus.kleinwort@desyNOSPAMPLEASE.de
- Gregor Mittag gregor.mittag@desyNOSPAMPLEASE.de
- Matthias Schröder matthias.schroeder@desyNOSPAMPLEASE.de
- the mailing lists cms-millepede@cernNOSPAMPLEASE.ch and hn-cms-tif-alignment@cernNOSPAMPLEASE.ch

Introduction

The Millepede II program is developed by Volker Blobel in close collaboration with the CMS group at the University of Hamburg, but is kept experiment independent. It has two parts:

- **Mille**: This part writes out the experiment specific data into a binary data format: measurements with their uncertainties, derivatives with respect to local (i.e. track) and to global (i.e. alignment) parameters and their integer labels to identify them. This part is fully integrated into CMSSW.
- **Pede**: This standalone program reads in one or several binary files produced with Mille. For proper Millepede II usage, the user is strongly encouraged to have a look at Volker Blobel's Millepede II manual, Claus Kleinwort's Millepe II documentation, and the German *Physics at the Terascale* Millepede II Wiki where further pede development is maintained. In addition, it is worth to have a look into Markus Stoye's PhD thesis CMS TS-2007/017 and the CMS NOTE-2008/008, though this analysis was obtained using the old ORCA framework. Selection of files, fixing of parameters, constraints, the choice of the methods for outlier rejection and for solving the resulting matrix equation as well as other options are steered via text files. This part is implemented in CMSSW as a system call to the standalone pede executable, but it is also possible ro run Pede 'by hand'. The steering files are automaticaly generated within CMSSW, but can be adjusted by hand, too.

The CMSSW interface to the Millepede II is implemented in the package `Alignment/MillePedeAlignmentAlgorithm`. It runs within the `AlignmentProducer`. Depending on a chosen mode, a mille binary file is created, the pede program is executed and/or the result of a previous pede run is read in and translated into CMSSW format. The setup of a Millepede II working environment in a specific CMSSW release is described at SWGuideMillepedeIIProductionEnvironment.

Limitation

The current implementation of the CMSSW interface is designed for use in tracker alignment. Even the best track model implemented so far, the Broken Lines track fit inspired by Volker Blobel is only suited for alignment of the silicon tracker (see SWGuideAlignmentAlgorithms#ReferenceTrajectory and github). The propagation through the magnetic field is done by the `defaultRKPropagator` github, but the transformation of the five helix parameters along the path still use an analytical calculation, assuming a constant B-field between subsequent hit modules. This is not exactly true, especially in the outer TEC disks.

Similarly, alignment of the muon system with its non-constant B-field cannot yet work since the use of the `defaultRKPropagator`, designed for the tracker volume only, is hardcoded and the problem of transforming the helix parameters is even more severe. But in principle it should already work now for B=0. Please contact the author if you want to give it a try...

Usage of Millepede II in CMSSW

To set up a Millepede II working environment in a specific CMSSW release, please follow the recipes at SWGuideMillepedeIIProductionEnvironment.

Modes

The following modes can be chosen in the `algoConfig` (cf. below):

mille
: Running on data and producing a mille binary file.

pedeSteer
: Writing pede steer files.

pedeRead
: Reading results from a previous or separate pede run and converting the alignment parameter result into CMSSW format.

pedeRun
: Running pede on specified mille binaries (includes pedeRead).

pede
: Both pedeSteer and pedeRun.

full
: Both mille and pede.

If you chose a mode that does not write out a mille binary (all but mille and full), you should not run a real event loop, wasting time (CPU and yours). Nevertheless, for technical reasons it is required to run on 1 event, usually with a dummy module only:

```
process.source = cms.Source("EmptySource")
process.maxEvents = cms.untracked.PSet(input = cms.untracked.int32(1))
process.dump = cms.EDAnalyzer("EventContentAnalyzer")
process.p = cms.Path(process.dump)#IOV_Dependent_Alignment
```

The SWGuideMillepedeProductionSystem automatically takes care of that.

Output

Besides the output configured in the `AlignmentProducer`, the mille binary file, the steering files generated for pede and the pede output and log files, one or two ROOT files are created.

1. One ROOT file contains several monitoring histograms. Its production can be deselected, cf. below.
2. The other ROOT file contains several `TTree` with information about the (misaligned) positions and parameters of the chosen alignables. (**Caveat:** This file is **not overwritten** if a job is rerun - so remove/rename it!) Also the number of used hits per alignable is stored. The tree names end with _ referring to different IOVs in case of IOV dependent alignment:
    - **0**: Initial status, i.e. after misalignment (if chosen in `algoConfig`).
    - **N** (with n larger than 0): representing final result with summed hit statistics for IOV N (counting from 1).

Note: The tree `AlignmentParameters_N` does not contain the parameters as calculated by pede, but the re-calculated misaligment parameters with respect to the starting geometry (ideal or an alignment applied from DB), but before application of misalignment tools. The pede parameters are part of the tree `MillePedeUser_N`. **Note**: In very early version of the tool before V00-25-00, these numbers do not refer to the IOV but to the iteration numbers as also used for the iterative alignment algorithms. The iteration numbers are used in the following way:

- **0**: Initial status, i.e. after misalignment (if chosen in `algoConfig`).
- **1**: In mille mode used instead of 0, but including hit statistics. In full mode representing the result.
- **2**: Written only in pede modes in case of merging binary files from separate mille runnings, representing final result with summed hit statistics.

Here is an explanation of some of the variables stored in the trees:

- **Id**: The DetId of the alignable (if the Alignable is a Det), or the DetId of one of the components (if the Alignable is a higher hierarchical object). See DetId.h, SiStripDetId.h and SiPixelDetId.
- **ObjID**: The alignable object identifier, see StructureType.h. The Id together with the ObjID define the aligned object uniquely.
- **Pos**: The position, an array of three numbers, given as x,y,z.
- **Rot**: The rotation matrix, stored as nine numbers (what a waste of space!).
- **Par**: The parameter determined from the `pede` program.

If calibration parameters (LA and BP corrections) are also fitted, there are additional trees `Calibration_input` and `<type of>Calibration_result_<run>` present, e.g. `SiPixelLorentzAngleCalibration_result_1`. The `input` trees contain the start values of e.g. the LA correction for each Det (module). The `result_<run>` trees contain the fitted corrections to the input values, where `<run>` is the first run number of the respective IOV. The content of the trees is:

- **detId**: The DetId
- **value**: Total LA/BP calibration (input value + fitted correction)
- **delta**: Determined correction to the input value (value = input + delta)
- **error**: Uncertainty on delta. Defined how?
- **paramIndex**: Index of the fitted pede parameter (not the same as the pede label). If one LA correction is valid for several Dets, the same paramIndex appears several times. In case of m IOVs, the ordering is 0,1,2,...m-1 referring to the first parameter in the different IOVs, m,m+1,...m+m-1 to the second parameter in the different IOVs, ... Note that these parameter appear at the very end of the `millepede.res` parameter list with the highest pede labels.

See SWGuideMillepedeIIAlgorithm#Pede_Parameters_and_Geometry_Com below for macros analysing these tree files.

Version and production environment

Version specific information and a description on how to use the production area on the CAF is now available on the page SWGuideMillepedeIIProductionEnvironment.

Configuration

Status: as of version V00-29-10 or CMSSW_6_2_0 (hopefully - please check...)

The configuration of the `AlignmentProducer` is assumed to be sufficiently documented in SWGuideTrackAlignment, so here we concentrate on the Millepede specific parts. To choose the Millepede algorithm your cfg.py has to contain

```
process.load("Alignment.CommonAlignmentProducer.AlignmentProducer_cff")
process.AlignmentProducer.algoConfig = process.MillePedeAlignmentAlgorithm
process.MillePedeAlignmentAlgorithm.<aMillepedeConfigParam> = ... # for each parameter you wa
```

This will pull in the Millepede configuration from Alignment/MillePedeAlignmentAlgorithm/python/MillePedeAlignmentAlgorithm_cfi.py (cf. on github).
The Millepede configuration parameters are the following. A * means that the parameter is `untracked`.

Name	Type	Description	Default	Note
`algoName`	`string`	defines algorithm for `AlignmentProducer`	"MillePedeAlignmentAlgorithm"	
`mode`	`string` *	running mode	"full"	described above
`fileDir`	`string` *	directory for all Millepede input/output files	""	
`binaryFile`	`string`	binary file to write (mille) and/or read (pede)	"milleBinary.dat"	
`treeFile`	`string`	root file to store `Alignable` positions etc.	"treeFile.root"	
`mergeBinaryFiles`	`vstring`	binaries to merge for mode 'pede'	{}	parallel to `mergeTreeFil`
`mergeTreeFiles`	`vstring`	root files to merge hit statistics for mode 'pede'	{}	parallel to `mergeBinaryF`
`monitorFile`	`string` *	root file with monitoring (if empty: no monitoring)	"millePedeMonitor.root"	if empty: no monitoring
`minNumHits`	`int32`	number of hits a track must have on selected Alignables to be considered	7	since refitting might remo after selection
`max2Dcorrelation`	`double`	if xy-correlation of a hit is larger, diagonalise measurements	0.05	only applied for TID and modules
`pedeLabeler`	`PSet`	normal, run or momentum dependent parameters labeling	empty	run dependent labeler automatically chosen if ru dependent parameters co
`pedeReaderInputs`	`VPSet`	`PSet` like 'pedeReader' below to be applied consecutively before running mille for iterative running of Millepede	{}	selected parameters shou
`pedeSteerer`	`PSet`	configuration for steering and running		described below

		pede from within CMSSW		
pedeReader	PSet	configuration for reading pede results		described below
TrajectoryFactory	PSet	track model for local derivatives	BrokenLinesTrajectoryFactory	See SWGuideAlignmentAlgo
surveyPixelBarrel	PSet	to use BPIX survey pictures in alignment		ask Frank Meier...
doubleBinary	bool	use double precision when writing the mille binaries	False	

The pedeSteerer PSet has the following parameters, the meaning of some of them is also explained in the pede section:

Name	Type	Description	Default	Note
steerFile	string	beginning of steering file names	"pedeSteer"	
steerFileDebug	bool *	add debug info to pede steering files	false	makes txt-files large
fileDir	string *	directory for pedeSteer files	""	if empty taken from MillePedeAlignmentAlgorithm.file
pedeCommand	string *	full path to pede executable	"pede_1GB"	might need one with more memory **sh just pede from CMSSW_6_2_0_pre5**
pedeDump string	string *	where pede output is dumped into	"pede.dump"	
method	string	pede method for matrix solution, including iterations	"sparseMINRES 6 0.8"	
options	vstring	option for pede, e.g. outlier rejection	{"entries 50", "outlierdownweighting 5", "dwfractioncut 0.2"}	recommend less entries and downwei with 3 iterations only; all available opt described in the MillePede II manual
Presigmas	VPSet	presigma values for active parameters	{}	described below
minHieraConstrCoeff	double	smaller absolute values of coefficients in hierarchy constraints are ignored	1.e-7	avoids numerical imprecise zeros
minHieraParPerConstr	uint32	ignore hierarchy constraints with less params	2	
applyConstraints	bool		False	

		apply linear equality constraints defined in `constraints`		
`constraints`	`VPSet`	constraints to introduce/avoid certain deformations	{}	only used if `applyConstraints` is set this option is documented in SWGuideMillepedeLinearEqualityCon

The `pedeReader` PSet has the following parameters:

Name	Type	Description	Default	Note
`readFile`	`string`	name of text file with pede results	"millepede.res"	for `pedeReader` must match hardcoded value in pede-code
`fileDir`	`string` *	directory of `readFile`	""	if empty, overwritten by `MillePedeAlignmentAlgorithm.pedeSteerer.fileDir`

The `pedeReaderInputs` PSets must have the same parameters to specify the millepede.res file(s).

Hierarchy Constraints

In the `AlignmentProducer` configuration it is possible to select simultaneously parameters of a composed structure and any hierarchy level of its substructures. This case is automatically detected by the CMSSW interface to Millepede II. Constraint equations are generated and passed to pede via a pede steer file, removing the *additionally* introduced degrees of freedom. In this way the subcomponents' alignment parameters are interpreted with respect to the coordinate system of the larger structure. Note that if the higher level object has inactive parameters (selected via '0', cf. below), the corresponding degree of freedom is not constrained since it is not an *additional* one. If you want to allow e.g. only shifts of a higher level object and select also parameters for its components, the higher level rotations need to be fixed ('f', cf. below) instead of being inactive ('0'). **Note** also that there are cases where a necessary constraint might be missed, as presented by Gero Flucke in a tracker alignment meeting.

IOV-Dependent Alignment

Brief description of the IOV-dependent alignment as implemented currently in the CMSSW Millepede II, written by Frank Meier.

Special Parameter Selections

The selection of alignment parameters is handled in the `AlignmentProducer` and documented at SWGuideAlignmentAlgorithms. Besides the digits 1 and 0 that select and deselect parameters, some letters will be specially interpreted in Millepede II. Note that whatever the special meaning is, the corresponding derivatives will be written to the mille binary file and the special meaning is only used to generate steering files for pede. This means that you can run in pede mode on existing binary files and exchange letters with digit 1 (and vice versa) in the parameter configuration. An exception is taking out an alignable from a hierarchical parametrisation, see below.)

Fixing Parameters

Only selected parameters are taken into account when writing mille binary files and pede steer files. Since pede does a refit of the tracks, it might be necessary to pass a parameter with a fixed value to pede. This is chosen via the letters c and f:

- **f**: Fix the parameter at zero.
- **c**: Fix the parameter at the true value, deduced from the difference between the ideal and the current (misaligned) geometry. **Caveats**:
  - ♦ Running on real data or with misalignment via db-object identical to f.
  - ♦ Correction can only be deduced if misalignment is done via configuration file and not via db-object.
  - ♦ Correction is correct only for the lowest level object that is misaligned, i.e. if Dets are misaligned, cannot be used for Rods.
  - ♦ Does not make sense (and will work wrongly) for lower level objects of a hierarchy.

Removal from Hierarchy

It is possible to select alignables that should be highest level in the hierarchy, even if they are in principle part of a higher level structure that also has parameters. The selected alignables are not considered when the hierarchy constraints are build. This means their parameters are still in global frame. This is chosen via upper-case letters:

- **F**: Take out of hierarchy and fix at zero (as lower-case f).
- **C**: Take out of hierarchy and fix at true value (as lower-case c, so restrictions for that are valid as well).
- **H**: Take out of hierarchy, but keep free (as 1).

Note that you **must not** exchange upper- and lower-case letters in case of re-running in pede mode on existing mille binaries: The removal from the hierarchy does not only remove the hierarchy constraint in the steering file, but also prevents writing of the derivatives for the higher level object that the chosen alignable would normally be part of. In addition you cannot mix F, C and H with other selectors except 0: An alignable is always removed from the hierarchy as a whole.

Coordinate System Definition

The coordinate system (three translations and three rotations) must be defined, otherwise the problem is ill-formed and there is no unique solution. This can be done by fixing some alignables. Another solution is to take the coordinate system as the mean of all modules or a part of them. Which modules should be taken can be configured by

- **r**: Select parameter to define coordinate system.
- **s**: Same as **r**, but correct for the mean cff-file misalignment of the chosen modules beforehand (i.e. the misalignment scenario is manipulated).

These alignables selected for the coordinate system must be the highest selected hierarchy level. It is allowed to ignore ('0') or to fix ('f') some of their parametes. Technically the coordinate definition is implemented in the same way as hierarchy constraints of the selected alignables with respect to the global frame. It is not allowed to mix **s** and **r**. **s** should only be used for MC truth-comparison using the treeFile.root. Search for `correctToReferenceSystem` in the log file to see which correction is applied. **Caveat**: This correction works only for misalignment from configuration file and not from db-object.

Assigning Pre-Sigma Values

Assigning pre-sigma values to parameters as used in Markus Stoye's thesis to implement prior knowledge is implemented in the following way: Each PSet in the VPSet Presigma in pedeSteerer must contain `PSet Selector` and `double presigma`. The latter is the pre-sigma value assigned to the parameters selected by the former. Only active parameters (i.e. selected in the `AlignmentProducer`) are considered, others are ignored. The syntax for `Selector` is identical to that of `AlignmentProducer.ParameterBuilder.Selector`. As an example see Alignment/MillePedeAlignmentAlgorithm/python/PresigmaScenarios_cff.py (cf. in CVS), in your configuration file do e.g.:

```
from Alignment.MillePedeAlignmentAlgorithm.PresigmaScenarios_cff import
process.MillePedeAlignmentAlgorithm.algoConfig.pedeSteerer.Presigmas.extend(TrackerShortTermPre
```

The `pede` program has an option to set one common presigma value to all parameters that have no explicit pre-sigma defined. To activate this, add e.g. `presigma 1.e-4` to `pedeSteerer.options` to set it to 1.e-4.

Pede Program

The pede program is well documented on Volker Blobel's Millepede II manual, further updates stated at Millepede II Wiki.

Note: Rest of section slightly outdated!

Some hints for usage within CMSSW are given here. Since CMSSW_2_1_5 pede is also an external tool for CMSSW, providing executables directly available on the command line. Changes with respect to the original version are the following, the applied patch files reside at millepede_2008_08_18.patch and, on top of that for 64-bit architecture, at millepede_64bit_2008_08_18.patch.

- increase allowed file name lengths (500 characters allowed now)
- possibility to steer HUGE cut for rejection of outliers via e.g. `hugecut 50` (but 50 is pede default anyway)
- avoid NAN for 'bad' global correlations (can happen e.g. in case of constraints, now corr = -2 if corr^2 < 0, corr = -1 if corr > 1)
- some lazy pattern comparisons for steering keywords now strict (`Cfiles` and `fortranfiles`)

Remaining known problems:

1. relax convergence parameter `rtol` from 1.E-8 to 1.E-5, otherwise MINRES does (sometimes) not converge and stops after 2001 iterations
2. endless loop for too small matrix (`ovfl return with J = ...` in `pede.dump` file after killing process) should become a program stop
3. funny percentages of used memory for executables with large memory allocation
4. the number of parameters seems to be limited to 46340, see below (unsolved!).

An update for points 1-3 is released with CMSSW_2_2_4, see https://hypernews.cern.ch/HyperNews/CMS/get/tk-alignment/500.html.

See also in hypernews about new millepede II version.

Executable

Please use executables from the CMSSW release or those from Claus Kleinwort at `/afs/cern.ch/user/c/ckleinw/bin/revXX`. You could choose the most recent XX, indicating the most recent SVN revision number.

Note: Rest of section slightly outdated!

The pede program is a standalone executable called from within the CMSSW framework. The absolute path to the used executable is determined by the configuration parameter `pedeSteerer.pedeCommand`. Since it is a Fortran77 program, it cannot do dynamic memory allocation. Instead the size of a large array is defined at compile time. The array is internally used as a kind of heap space for the large matrix and some vectors. If that array is not sufficiently large, pede might end up in an endless loop printing something like `ovfl return with J = 0 62300767 62300767`.

To adjust the size of the preallocated array to the size needed and to the ressources of the used computer, several executables are provided and available in the $PATH, all starting with `pede`. In addition, executables can be found at `/afs/cern.ch/user/f/flucke/cms/pede/versWebEndMay2007patched` (in fact, the executables in there link to the most recent patched version). Suffixes tell you about some properties of the executable:

Suffix	Feature
`_<n>GB`	<n> is the size of the array in GB
`_rfio`	The executable can directly read files starting with `rfio:` from CASTOR, but it requires libshift.so to be installed.
`_64`	The executable requires to be run on 64 bit systems. Without this suffix both 32 and 64 bit systems work.

You can also copy all files `*.h`, `*c`, `*.F` and `Makefile` from the directory and compile your own executable simply typing `make`. The array size is defined in `dynal.inc` and other features are configured in the `Makefile`.

Note:

- The pede method `sparseMINRES` (`sparseGMRES` in older versions) usually requires much less memory and is the only one to work with the full CMS tracker on DetUnit level. But if the percentage of non-zero off-diagonal elements (out-) reaches 67 %, sparse mode needs even more memory - and is slower.
- There is a memory limit for processes to run in a shell on lxplus, so `pede_2GB` and above will be killed immediately. Might be circumvented by `ulimit` command. Interactive and batch nodes of the CAF do not have this limit - at least in queue cmscafspec.

Steering

The pede steering files will be generated by the CMSSW job (C++ class `PedeSteerer`). The strings of the parameters `pedeSteerer.method` (prepended by `method`) and `pedeSteerer.options` are written to the main steering file.

The `method` parameter can be any of the methods described in section 6.6 of the Millepede II manual, e.g. `"sparseGMRES 4 1."` (`"sparseMINRES 4 1."` for pede V02-00-00 and above), `"inversion 6 0.8"` and `"diagonalization 10 0.8"`. Note the different memory and CPU time requirements. (For old for GMRES, do not forget to specify `bandwidth 1`, otherwise there is no preconditioning.)

Possible `options` are described in sections 6.8 (outlier rejection) and 6.9 (further options) of the manual and in the Millepede II Wiki, respectively. `"entries 100"` will tell pede to keep alignment parameters with less than 100 hits at their start value.

Analysing Results

Pede Histograms

The macro readPedeHists.C allows to analyse the `millepede.his` file, containing histogram like informaton about the minimisation process within the `pede` program. The use of the macro is sufficiently documented in the macro itself while the histograms themselves will in future be documented in the millepede manual from Volker Blobel's webpage that then will also provide the macro.

Pede Parameters and Geometry Comparison

The official geometry comparison tool is SWGuideTkGeomComparisonTool.

Some fast ROOT macros to analyse the result stored in the output ROOT file can be found in as Alignment/MillePedeAlignmentAlgorithm/macros. Note that there are some restrictions on usefulness of the plots, see below. Besides methods that compare misaligned geometry and alignment result (these are officially superseeded by SWGuideTkGeomComparisonTool), you can directly investigate the parameters calculated in the pede step and their uncertainties, global correlations, hit statistics etc.

Currently the macros are developed to compare initial misalignment with the MC truth. They work well also for data, but some plotting functions have non-intuitive names for that purpose. The shortest example is:

```
[lxplus434] ~/cms/PlotMillePede % root -l allMillePede.C
root [0]
Processing allMillePede.C...
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/user/f/flucke/cms/Alignment/Mi
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/user/f/flucke/cms/Alignment/Mi
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/user/f/flucke/cms/Alignment/Mi
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/user/f/flucke/cms/Alignment/Mi
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/user/f/flucke/cms/Alignment/Mi
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/user/f/flucke/cms/Alignment/Mi
Info in <TUnixSystem::ACLiC>: creating shared library /afs/cern.ch/user/f/flucke/cms/Alignment/Mi
root [1] PlotMillePede p("treeFile_merge.root"); // second argument defaults to 1 and chooses an
root [2] p.SetSubDetId(3); // choose TIB
root [3] p.SetBowsParameters(true); // bows/kinks parameters (default: rigid body) - must match a
root [4] p.Draw<Tab> // to see what else is available - also p.Scan<Tab> etc.
```

Further documentation can be found in G. Flucke's presentations at the May 2011 Tracker Alignment Workshop (A Quick Look at Millepede Results) and in August 2011 (Millepede and IOV Dependence: Plotting Tools).

Note:

- Methods like `DrawParamResult`, `DrawParam`, `DrawMisVsLocation` plot rigid body parameters of modules after alignment relative to the starting alignment (defined by sqlite/DB), usually labeled as "remaining misal." or so. The curve labeled "misaligned" shows the misalignment on top of the this, i.e. applied in `AlignmentProducer` via cff files. Available (ancient) scenarios are defined in Alignment/CommonAlignmentProducer/python/AlignmentProducer_cff.py , but you are free to define your own using the SWGuideMisalignmentTools. **If in data alignment you do not apply misalignment, the "remaining misali." directly shows the difference to the starting alignment.**
    - If the alignment is an iteration using the `pedeReaderInputs` parameter sets defined above, the applied corrections of previous pede runs are effectively treated as misalignment using cff files.
- Methods like `DrawPosResult` and `DrawPosMisVsLocation` also show the differences relative to the starting alignment, but now in global coordinates. So they are comparable to the geometry comparison tool.
- The methods `DrawPedeParam` and `DrawPedeParamVsLocation` show the bare parameters determined by `pede` - in case of bows and kinks alignment, you have to call `SetBowsParameters(true)` before drawing, otherwise axis labels and units are wrong.
- In case of a hierarchical parametrisation it does not make sense to compare the larger structures, although they are also stored in the root file. Therefore a third argument can be passed to the `PlotMillePede` constructor to select only Alignables of a certain hierarchy level. Default is '0', i.e. the lowest level only.
- **Caveat 1**: In case of aligning larger structures only, the calculated truth of the larger structure does not know about an effective misalignment by a coherent misalignment of its components. So even if perfectly aligned, the tools might show a difference. One can manipulate the cff misalignment to

switch off the lower level misalignent, but then you will not see problems introduced by relative misalignment of these components.
- **Caveat 2**: The scenarios can move the global coordinate system, but the alignment procedure generally cannot recognise that. But the `PlotMillePede` routines compare with ideal geometry. That should only be solved if you do not fix anything but use the **s** flag (before tag V00-16-00 **r** flag) to define the coordinate system, because in that case it is tried to get rid of this overall movement beforehand.

Further helpful methods are:

- `p.SetSubDetId(Int_t subDetId)`, `AddSubDetId(Int_t subDetId)` and `p.SetSubDetIds(Int_t id1, Int_t id2, ...)` to select only alignables of one or several subdetectors.
- `p.SetAlignableTypeId(Int_t alignableTypeId)` to select only alignables of certain types like Det or Rod.
- `p.AddAdditionalSel(const TString& xyzrPhiNhit, Float_t min, Float_t max)` to select only alignables in the given range in x, y, z, r, phi (properly taking into account the +pi/-pi flip) or N(hit).
- `p.SetMaxDev(Float_t maxDev)` to influence the x-axis range of some histograms.

Use the tab completion feature of the ROOT command line to learn about the arguments.

Besides the class `PlotMillePede` there are two other plotting classes:

- `CompareMillePede` might be useful to compare results of two different millepede runs. But note that the alignable selection must be identical.
- `PlotMillePedeIOV` can be used to plot IOV dependences of parameters in case of run dependent alignment.

The `Overlay` feature of the underlying histogram managment tool (used in all these classes) allows to overlay results of different alignment approaches. As an example here the BPIX parameters (including bows) of two alignments, specifying also how many histograms are plotted per canvas:

```
[lxplus434] ~/cms/PlotMillePede % root -l allMillePede.C
root [0]
Processing allMillePede.C...
root [1] PlotMillePede p("/afs/cern.ch/cms/CAF/CMSALCA/ALCA_TRACKERALIGN/MP/MPproduction/mp1320/j
root [2] p.SetTitle("mp1320"); // give a title appearing in histogram titles and legends
root [3] p.GetHistManager()->SetBatch(); // avoid immediate drawing
root [4] PlotMillePede p18("/afs/cern.ch/cms/CAF/CMSALCA/ALCA_TRACKERALIGN/MP/MPproduction/mp1318
root [5] p18.SetTitle("mp1318")
root [6] p18.GetHistManager()->SetBatch();
root [7] p.SetSubDetId(1) // select BPIX
root [8] p18.SetSubDetId(1)
root [9] p.SetBowsParameters()
root [10] p18.SetBowsParameters()
root [11] p.DrawPedeParamVsLocation()
root [12] p18.DrawPedeParamVsLocation()
root [13] for (int i = 0; i< p.GetHistManager()->GetNumLayers(); ++i) { p.GetHistManager()->Overl
root [14] p.GetHistManager()->SetBatch(false); // reset 'batch'
root [15] p.GetHistManager()->SetNumHistsXY(4, 1); // each canvas with 4 hists along x (row), 1 i
root [16] p.GetHistManager()->Draw() // finally draw
```

Millepede Monitoring

The above mentioned cvs directory also contains the macro `PlotMilleMonitor.C` to overlay and summarise certain histograms in the `monitorFile`. Simplest usage:

```
[lxplus434] ~/cms/PlotMillePede % root -l allMillePede.C
root [0]
```

```
Processing allMillePede.C...
root [1] PlotMilleMonitor m("millePedeMonitor_1.root=label1,millePedeMonitor_2.root=label2");
root [2] m.DrawAllByHit();
```

Plots from millepede.res ()

How to understand labels in millepede.res:

(1) Different subdetectors:

PXB: starts from 61

PXF: starts from 17541

TIB: starts from 37021

TID: starts from 121061

TOB: starts from 144401

TEC: starts from 284201

after 700000 - new IOV

(2) Large structures vs module level alignables

Overall, it depends on the HL alignables.

boundaries of certain large structures:

labels = [61, 8781, # TrackerTPBHalfBarrel

17541, 22401, 19961, 24821, 27281, 32141, 29701, 34561, # TrackerTPEHalfDisk

37021, 79041, # TrackerTIBHalfBarrel

144401, 214301, # TrackerTOBHalfBarrel

121061, 132721, # TrackerTIDEndcap

284201, 380121] # TrackerTECEndcap

after 700000 - new IOV

(3) Parameters:

"((label%20-1)%9+1)=="

1-3: translations

1: u (local x or global X in case of large structures)

2: v (local y or global Y in case of large structures)

3: w (local z or global Z in case of large structures)

4-6: rotations

4: alpha (rotation)

5: beta (rotation)

6: gamma (rotation)

7-9: deformations

(4) IOVs:

new IOV starts after 700000, then after 1400000, etc

script which prepares 4 sets of plots from millepede.res

available at private branch here:

https://github.com/eavdeeva/cmssw/tree/B20150609_AddScript_PlotFromMillepedeRes/Alignment/MillePedeAlignm

PlotFromMillepedeRes.C and runPlotFromMillepedeRes.C

pull request 9529 created

to make plot, one has to:

(1) gunzip millepede.res

(2) in runPlotFromMillepedeRes.C

(2.1) uncomment 2 lines which are needed for certain plot

(2.2) write appropriate name and path to millepede.res as the first argument of PlotFromMillepedeRes()

(2.3) at the wish, change strOutdir (directory where plots will be saved)

(3) root -l runPlotFromMillepedeRes.C

runPlotFromMillepedeRes.C has instructions in it and here are the

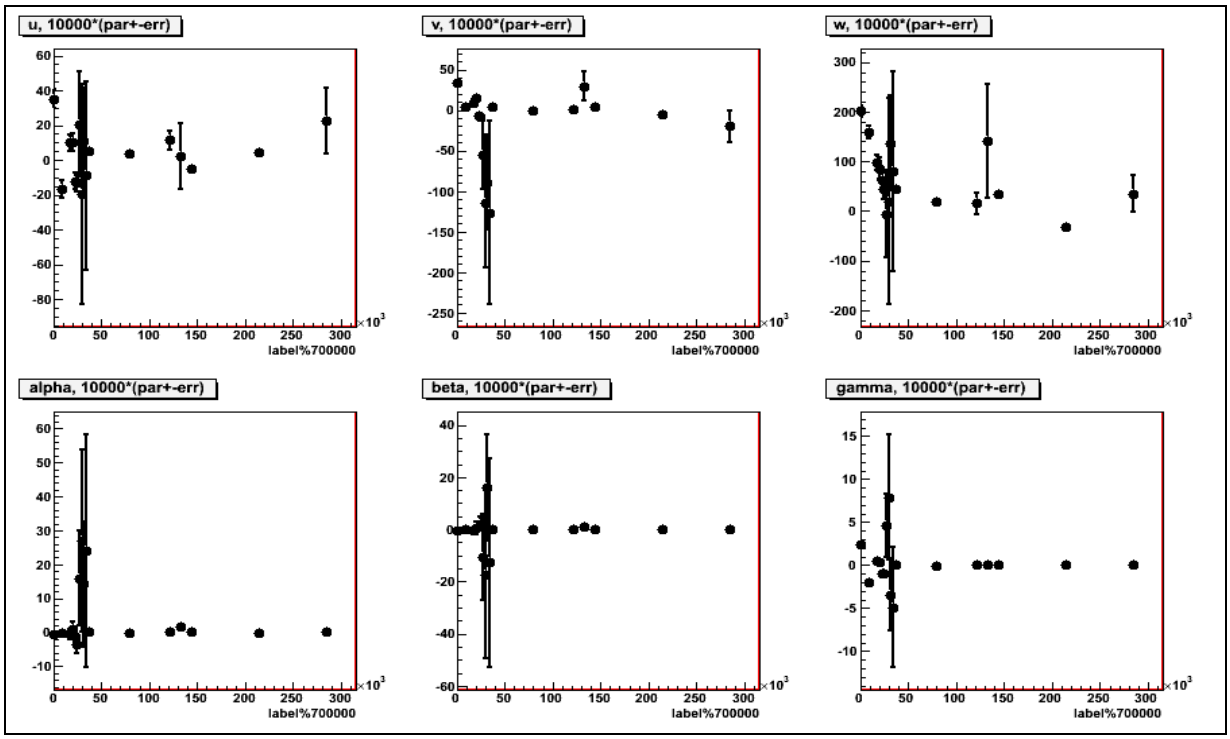
examples of plots produced:

##

to plot parameter values vs labels with errors (you have to have errors in millepede.res, means you had to run it in inversion mode) for 6 alignables (u,v,w,alpha,beta,gamma) for high level alignables only, uncomment these two lines and pass appropriate millepede.res:

```
strVars="label/I:smth2/F:smth1/F:parVal/F:parErr/F:Nhits/I";
PlotFromMillepedeRes("mp1720_millepede.res", strOutdir, strVars, PARSwithERRvsLABEL);
```

this is result on one of the first alignments with 0T collisions on 2015 data:

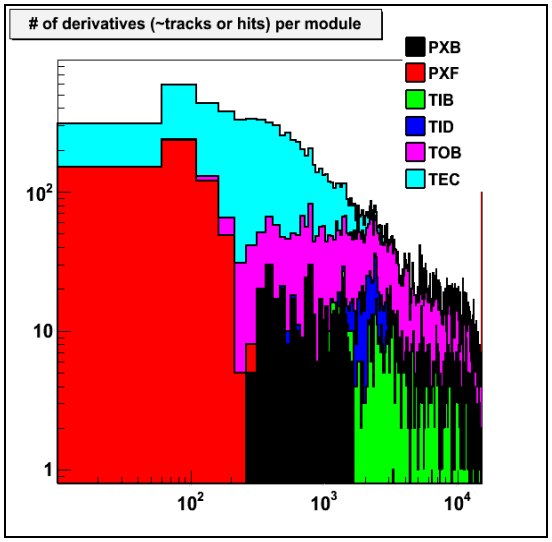


##

to plot number of derivatives corresponding to each module (which is ~ number of tracks passing through module or ~number of hits the module received with all available tracks), uncomment these two lines and pass appropriate millepede.res (if you have errors in millepede.res, make appropriate change in strVars):

```
strVars="label/I:smth2/F:smth1/F:parVal/F:Nhits/I";
PlotFromMillepedeRes("mp1700_millepede.res", strOutdir, strVars, NHITS);
```

this is result of alignment with CRAFT+CRUZET 2015

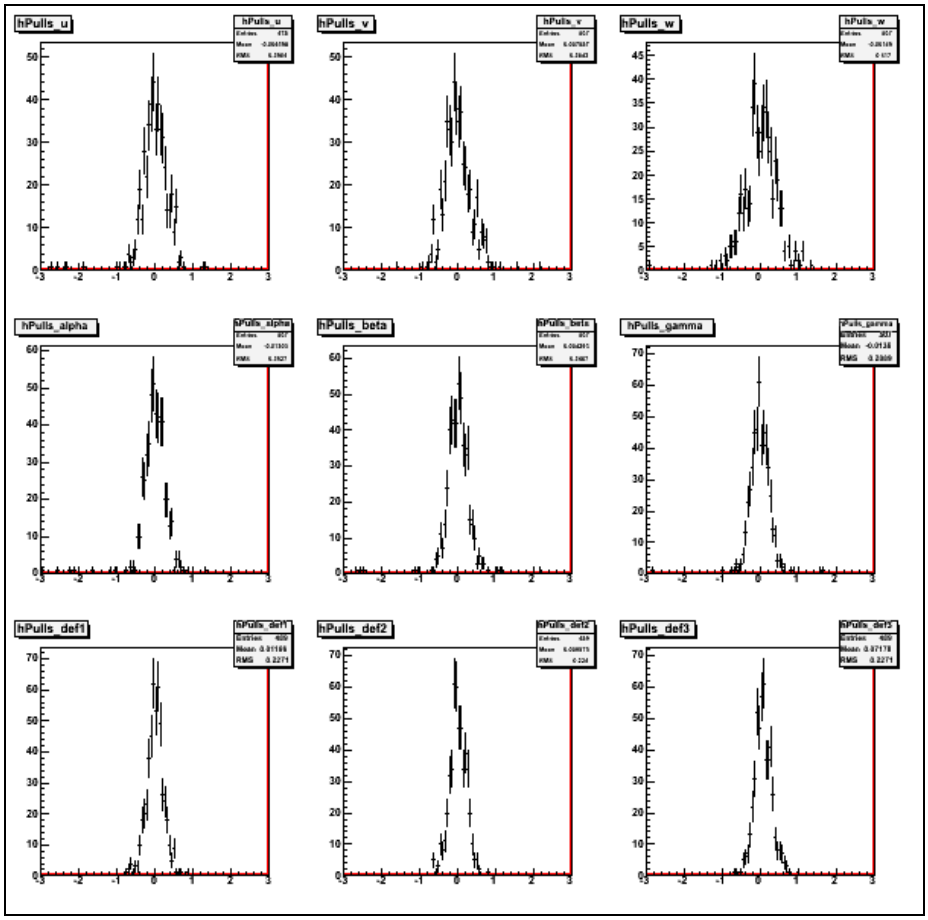


##

to plot pull distributions from millepede.res with errors, uncomment these two lines and pass appropriate millepede.res:

```
strVars="label/I:smth2/F:smth1/F:parVal/F:parErr/F:Nhits/I";
PlotFromMillepedeRes("mp1587_mpInversion_TPBandTPE_jobData_jobm_millepede.res", strOutdir, strVar
```

this is result of one of the first alignments with CRUZET 2015

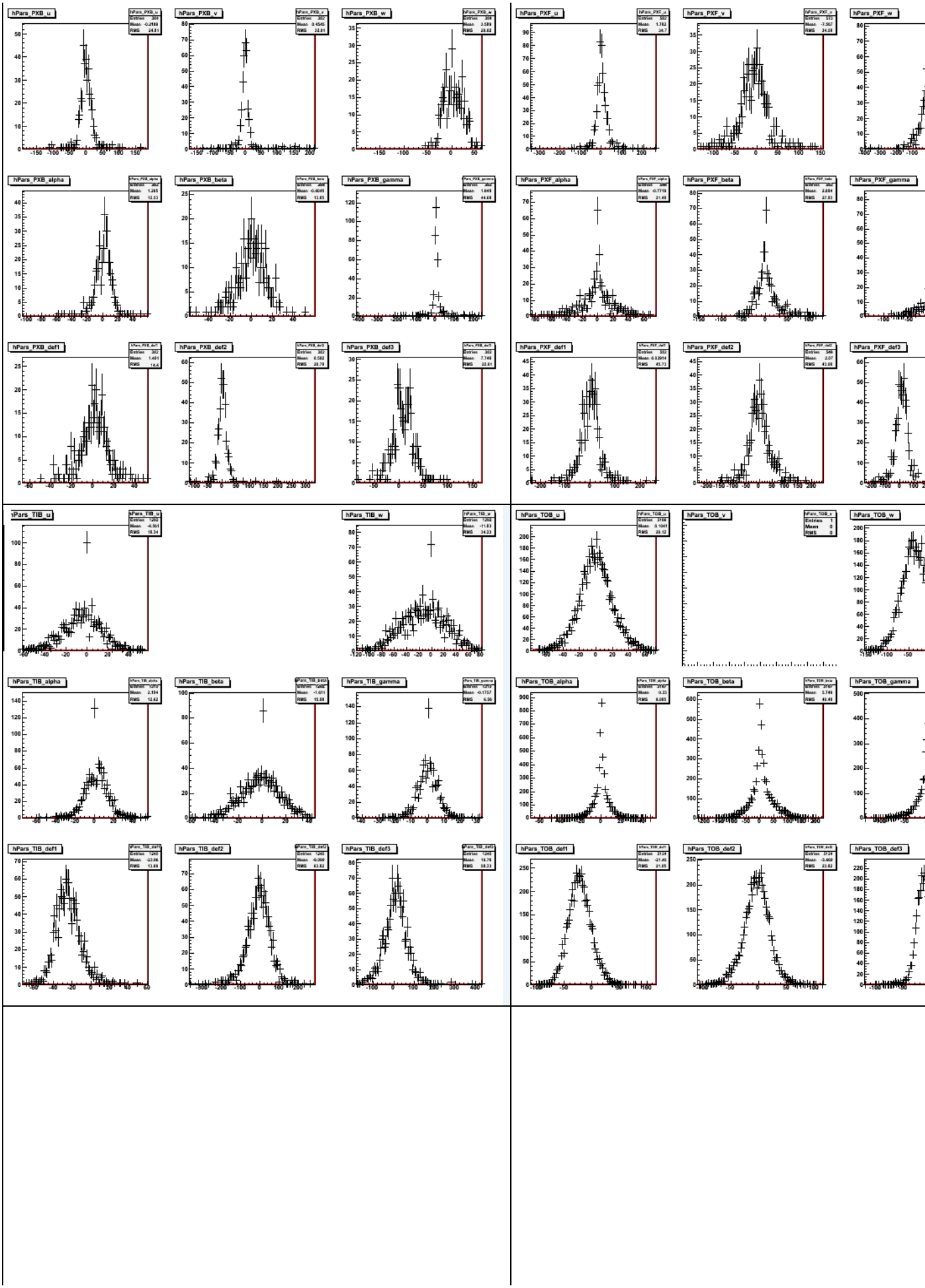


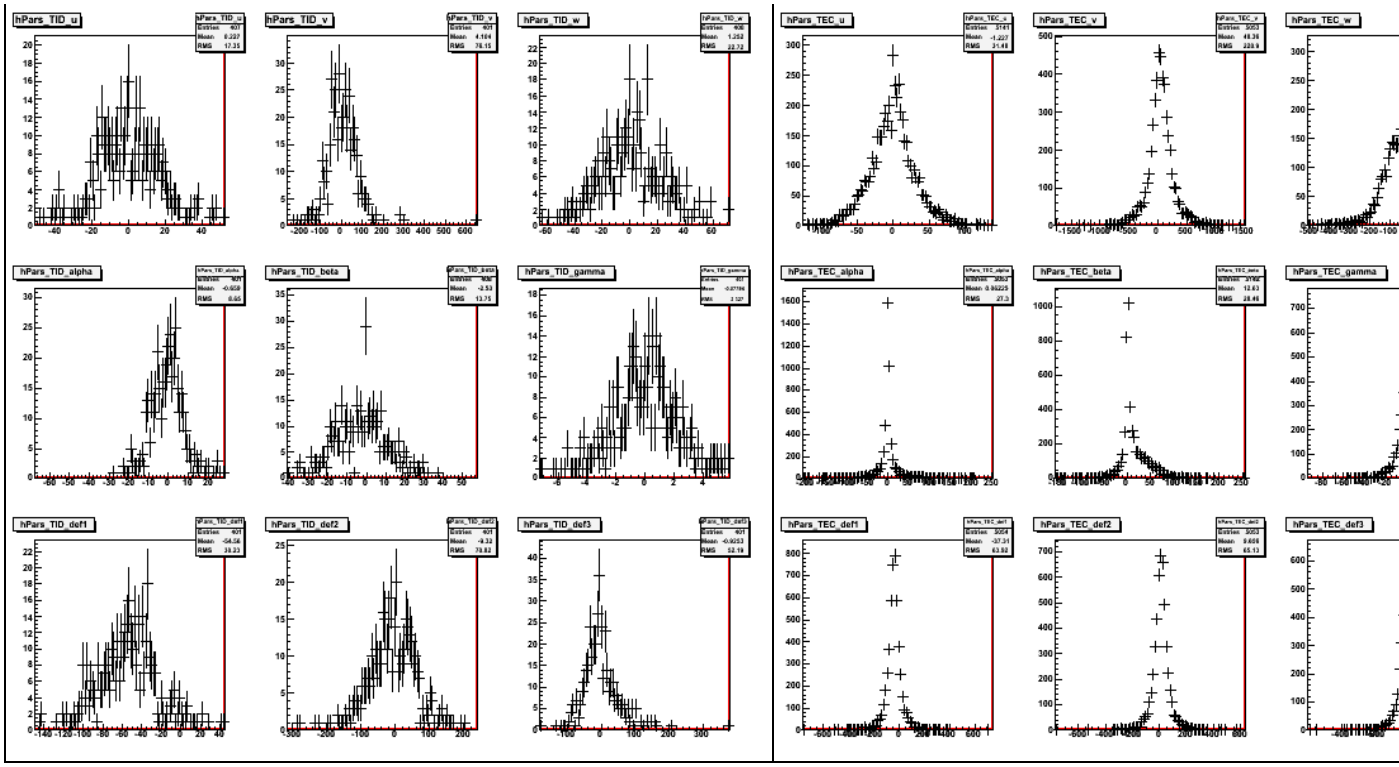
##

to plot distributions of parameters for each subdetector from millepede.res, uncomment these two lines and pass appropriate millepede.res (if you have errors in millepede.res, make appropriate change in strVars):

```
strVars="label/I:smth2/F:smth1/F:parVal/F:Nhits/I";
PlotFromMillepedeRes("mp1700_millepede.res", strOutdir, strVars, PARS);
```

this is result of alignment with CRAFT+CRUZET 2015


hPars_PXB_u
hPars_PXB_v
hPars_PXB_w
hPars_PXF_u
hPars_PXF_v
hPars_PXF_w
hPars_PXB_alpha
hPars_PXB_beta
hPars_PXB_gamma
hPars_PXF_alpha
hPars_PXF_beta
hPars_PXF_gamma
hPars_PXB_def1
hPars_PXB_def2
hPars_PXB_def3
hPars_PXF_def1
hPars_PXF_def2
hPars_PXF_def3
hPars_TIB_u
hPars_TIB_w
hPars_TOB_u
hPars_TOB_v
hPars_TOB_w
hPars_TIB_alpha
hPars_TIB_beta
hPars_TIB_gamma
hPars_TOB_alpha
hPars_TOB_beta
hPars_TOB_gamma
hPars_TIB_def1
hPars_TIB_def2
hPars_TIB_def3
hPars_TOB_def1
hPars_TOB_def2
hPars_TOB_def3



##

Parsing of millepede.res in CMSSW ()

The MillePedeFileReader can be used to parse millepede.res files and access the information from within CMSSW modules. This implementation is specific to the alignment of the pixel large structures in the prompt calibration loop. It accesses the alignment results for these structures and compares with pre-defined limits on movements/rotations and their significance to determine the need for an updated alignment in the PromptReco. Millepede.log is also read to access the number of tracks used.

See here and here for example usage.

Collected Monitoring Info and Fit Results (Under Construction)

A new system to collect all relevant monitoring information and fit results and display them conveniently in different formats is being developed. The developing documentation can be found at SWGuideMillepedeIIAlgorithmResultAnalysis

Millepede Production System

An environment to easily submit many parallel mille jobs and combine their results in a single pede job is documented in the SWGuideMillepedeProductionSystem.

Introductory and Theory Reading

1. Markus Stoye's PhD thesis: CMS TS-2007/017.
2. The Millepede II manual (though a bit redundant after the thesis).
3. Documentation about the global parameters used by CMS tracker alignment:
    1. Karimaki parametrisation of the modules as rigid bodies: CMS CR-2003/022.
    2. Extension to bowed sensors: alignment presentation.

4. Documentation about the `BrokenLinesCoarse` and `BrokenLinesFine` track model for the local parameters:
    1. Inspired by V. Blobel's broken lines track fit in 2D, see also NIM A, 566 (October 2006), pp. 14-17.
    2. Extension to full 3D tracking by Claus Kleinwort as presented in CMS alignment meetings: 17-Sep-2009, 2011-01-20, 2011-02-17 and also partly reported in the conference reports CMS CR-2010/089 and CMS CR-2010/168.
    3. Full documentation expected to appear here.
5. The first CMS paper was on tracker alignment with CRAFT08 (not yet with BrokenLines and with an old Millepede II with only up to 46340 alignment parameters).
6. See also various educational talks given at the CMS Tracker Alignment Workshop 2011 at DESY: The alignment framework (A. Mussgiller), Alignment parametrization (G. Flucke), Track Based Alignment as Global Fit (G. Flucke), ReferenceTrajectory based on General Broken Lines (C. Kleinwort), Track Model Implementations (A. Mussgiller), Pede the Millepede II Solver (C. Kleinwort), The Millepede Production System (J. Behr), A Quick Look at Millepede Results (G. Flucke - with update from August 2011: "Millepede and IOV Dependence: Plotting Tools").

Description for Full Tracker Alignment

CSA08 was a first real test of a full scale alignment in CMSSW. See talks (M. Stoye) at: http://indico.cern.ch/conferenceDisplay.py?confId=35584 for the CSA08 concept and details in http://indico.cern.ch/conferenceDisplay.py?confId=32205

If you are familiar with SWGuideMillepedeProductionSystem you will still need the corresponding cfg to reproduce the results. A special CMSSW version (with beamspot constraint) for the cfg is needed or simply comment out the following two lines:

replace MillePedeAlignmentAlgorithm.beamspot = false

replace MillePedeAlignmentAlgorithm.aliBeamspot = false

More details can be found here about the use of the BeamSpotInsideMillepede.

Then it should work with any clean version, if you also add the Presigma.cff below.

/afs/cern.ch/user/m/mstoye/scratch0/MPS/CSA08II_allAngle_NoAB_Reco/alignment_markusset_PreSig.cfg : minBias.cfg

/afs/cern.ch/user/m/mstoye/scratch0/MPS/CSA08II_allAngle_NoAB_Reco_JPSI_mus/alignment_markusset_PreSig.cf : JPsi_singleMu

/afs/cern.ch/user/m/mstoye/scratch0/MPS/CSA08II_allAngle_NoAB_Reco_Zmumu/alignment_markusset_PreSig.cfg : Zmumu

/afs/cern.ch/user/m/mstoye/scratch0/MPS/II_CSA08_allAngle_NoAB_cosmics_RECO/alignment_markusset_PreSig.c : cosmics

/afs/cern.ch/user/m/mstoye/scratch0/MPS/CSA08II_allAngle_NoAB_Reco_muon11/alignment_markusset_PreSig.cfg : mu11

for the pede job:

* alignment_merge.cfg: pede

in addition you need a special presigmas cff:

* PresigmaScenarios.cff: presigmas

The full CMSSW code used in CSA08 can be found at: /afs/cern.ch/user/m/mstoye/scratch0/CMSSW_2_0_7

In /afs/cern.ch/user/m/mstoye/scratch0/MPS/ are actually the steerfiles used (S156) and in /afs/cern.ch/user/m/mstoye/scratch0/MPS/datasets datalist files as they can be digested by MPS. Once the CSA08 are reproduced with a newer CMSSW version this part should be updated.

Review status

Reviewer/Editor and Date	Comments
Main.flucke - 25 Apr 2007	created page
Main.flucke - 26 Apr 2007	added missing cvs co
Main.flucke - 30 Apr 2007	update of tags for AlignableDetOrUnitPtr
Main.flucke - 03 May 2007	comments from V. Blobel, refitter patch
Main.flucke - 10 May 2007	new MillePedeAlignmentAlgorithm tag fixing cf?s
Main.flucke - 18 May 2007	CMSSW_1_3_3 and new tags for more features
Main.flucke - 21 Jun 2007	New set of tags with more features (e.g. reference frame)
GeroFlucke - 17 Dec 2007	Remove obsolete versions, mention new hierarchy in 16X, add link to production system
GeroFlucke - 27 Feb 2008	document new features for 200
GeroFlucke - 03 Apr 2008	add pede section, more info about versions 16X to 20X, more info about analysing result
GeroFlucke - 20 Oct 2008	Mention pede as part of release, remove info about old versions before CMSSW_2_0_X
GeroFlucke - 17 Apr 2009	Adjust to python configuration and tags V00-18-08 and above

Responsible: FrankMeier
Last reviewed by: Gero Flucke

- alignment_markusset_PreSig.cfg: atest

This topic: CMSPublic > SWGuideMillepedeIIAlgorithm
Topic revision: r99 - 2016-07-18 - GregorHellwig