Manual: Internal tracker alignment for the $g-2$ experiment

Gleb Lukicov [a,1]
[a] *Department of Physics and Astronomy, University College London, London, WC1E 6BT, United Kingdom*

September 8, 2019

Abstract

This document outlines the need for the internal alignment of the tracking detectors, and the methodology of the alignment. The software methods used in the alignment are summarised to allow for the future alignment constants to be established for Run-3 and beyond. Internal alignment of the trackers in Run-1 and Run-2 has been established, with the constants defined as the interval of validity (IoV) in the reconstruction database.

Contents

[1] g.lukicov@ucl.ac.uk

1 Introduction

In order for the tracking detector to reduce the systematic uncertainty on the a_μ measurement and improve the sensitivity to a muon EDM, a precise calibration of the tracking detector is required. One of such calibrations is the internal alignment, which considers the positions of the tracking modules within a station.

Track-based internal alignment of the two stations of the tracking detector was implemented using data from Run-1 and Run-2. A Monte Carlo simulation was also developed to understand the detector geometry, and how the detector geometry affects how well the alignment can be determined, as well as to test the alignment procedure itself. A single figure-of-merit emphasising the need for a well aligned detector is shown in Fig. 1, which highlights that even a relatively small scale of misalignment can have a large impact on the detector performance w.r.t to the beam extrapolation. Additionally, alignment is essential for setting a new limit on the muon EDM. Broadly speaking, an internal misalignment of an element of a tracking detector manifests as a presence of a large residual between a measurement (e.g. DCA of a hit to a wire) and a DCA of a fitted track to that wire. This large residual arises from the fact that the assumed detector position, which was used in the fitting of the track, is not the actual position of that detector. The aim of the alignment procedure is to establish the corrections to the assumed detector position, and hence, minimise the residuals.

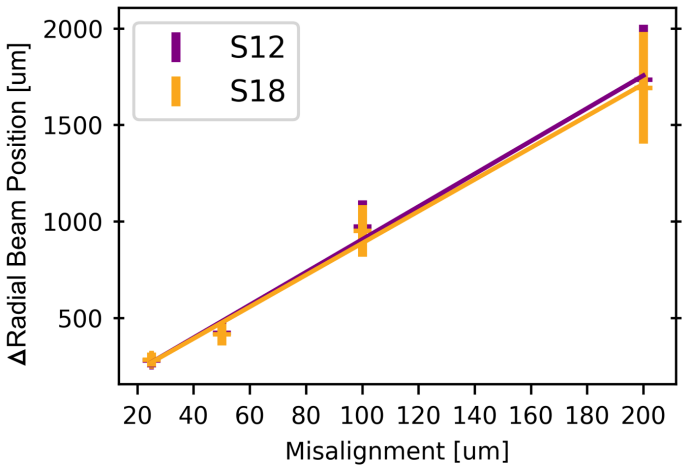

Fig. 1. Detector response to a uniform radial misalignment at different scales w.r.t to the difference in the extrapolated radial position.

2 Methodology of the alignment of the trackers

The chosen alignment framework was `Millepede-II` [1], which utilises a simultaneous fit of many parameters describing the detector geometry and the input data. Under this method correlations between different alignment parameters are taken into account, to produce an unbiased corrections to the position of the detector. The definition of the coordinate system used is shown in Fig 2.

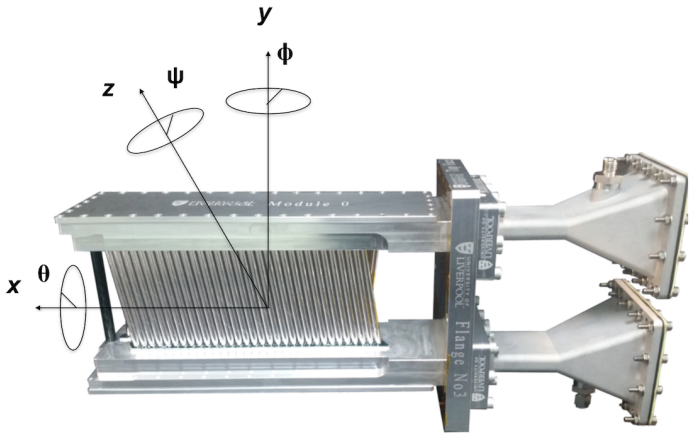

Fig. 2. Convention for misalignments (translations and rotation) in a module coordinate system. The positive radial and vertical shifts are along x and y, respectively.

The alignment task can be summarised as a Least Squares Fit (LSF) problem with a very large number of parameters. These parameters can be divided into two classes: global and local parameters. Global parameters (i.e. the alignment parameters) affect all tracks (e.g. radial position of a detector). Local parameters are associated with individual fitted tracks. For example, a straight line fit in 2D will have 2 local parameters: a slope and an intercept. `Millepede-II` solves the linear LSF problem with a simultaneous fit of all global and local parameters minimising a linearised function of a sum of residuals. An objective function, F, is then minimised with respect to a variation of global and local parameters

$$F(\boldsymbol{a},\boldsymbol{b}) = \frac{\partial\chi^2(\boldsymbol{a},\boldsymbol{b})}{\partial(\boldsymbol{a},\boldsymbol{b})} = \sum_{j}^{tracks}\sum_{i}^{hits}\frac{1}{(\sigma_{i,j}^{\mathrm{det}})^2}\frac{\left(r_{i,j}(\boldsymbol{a_0},\boldsymbol{b}_{\mathbf{0},j}) + \frac{\partial r_{i,j}}{\partial \boldsymbol{a}}\delta\boldsymbol{a} + \frac{\partial r_{i,j}}{\partial \boldsymbol{b}_j}\delta\boldsymbol{b}_j\right)^2}{\partial(\boldsymbol{a},\boldsymbol{b})} = 0, \quad (1)$$

where $\boldsymbol{a_0}$ and $\boldsymbol{b_0}$ are the initially assumed geometry and track parameters, respectively, r is the track residual (as shown in Fig. 3), and σ^{det} is the detector resolution. The corrections to the global parameters, $\delta\boldsymbol{a}$, allow for the minimisation of F, and are results of the alignment procedure. These corrections are then added to the assumed geometry of the detector to improve its performance.

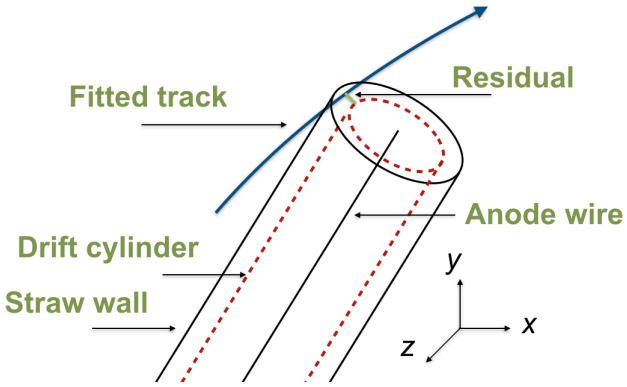

Fig. 3. Visualisation of the drift cylinder, which is defined by the measured drift time in the straw.

3 Alignment results and references

In simulation (MDC1), alignment stability was reached, and alignment convergence within 2 µm and 10 µm radially and vertically, respectively, was established as shown in **DocDB:17551**. Stable alignment results and improved residuals in data, using run 15922, were reached as shown in **DocDB:17722**.

Stability of the alignment constants across 3 intervals of validity (IoV) of Run-1 and Run-2 are shown in **DocDB:19294**.

A more detailed description of the alignment methods is contained in **DocDB:15106** (**see backup also**). More details can also be found in **DocDB:9245 (toy model in 2D)**, **DocDB:11331 (residual derivatives)**, **DocDB:15382 (progress in MDC1)**.

The uncertainty on the beam extrapolation due to the misalignment was estimated in **DocDB:16685**.

A key point to note from the documents listed above is that the alignment is an iterative process. A preliminary alignment constants (i.e. module positions) from iteration 1 are used for re-tracking, with alignment at iteration 2 and beyond taking improved tracks as in input for the LSF. Three iterations are usually enough to reach convergence, which is seen as $\delta \boldsymbol{a} \to 0$. The iterations mentioned here are external to the `Millepede-II` framework, which has its own internal iterations for matrix inversion and LSF; description of `Millepede-II` is beyond the scope of this document, and is contained in the official manual [1].

4 Software methods

The alignment software is part of the `gm2tracker` project: `gm2tracker/align/`. An alignment software module has been developed as part of the *gm2 art* software framework. This module comes last in the tracking chain, taking the final track-data product (`TrackArtRecords_tracks`) from either data or simulation. The description of the tracking software[2] and data products is contained in a separate manual [2]. The geometry is fully described at the tracking stage and is passed to the alignment module in the `beginRun()` function. The alignment module (`TrackerAlignment`) calculates the residuals of the selected tracks, as well as computes the local and global derivatives **(see backup of DocDB:11331)**. A `C++` infrastructure (`Mille`) is provided [1] to write this information into a binary file (**alignment data**) which is passed to `PEDE` (a `Fortran` executable), which performs a simultaneous LSF via a matrix inversion. The alignment module is also responsible for writing a constraints file (specifying the redundant degrees of freedom), and a steering file (specifying the mathematical methods used by `PEDE`). The final outputs of the `PEDE` algorithm are labels and the corresponding fitted values of the global parameters (module positions) and their errors, if applicable. This algorithm is shown in Fig. 4.

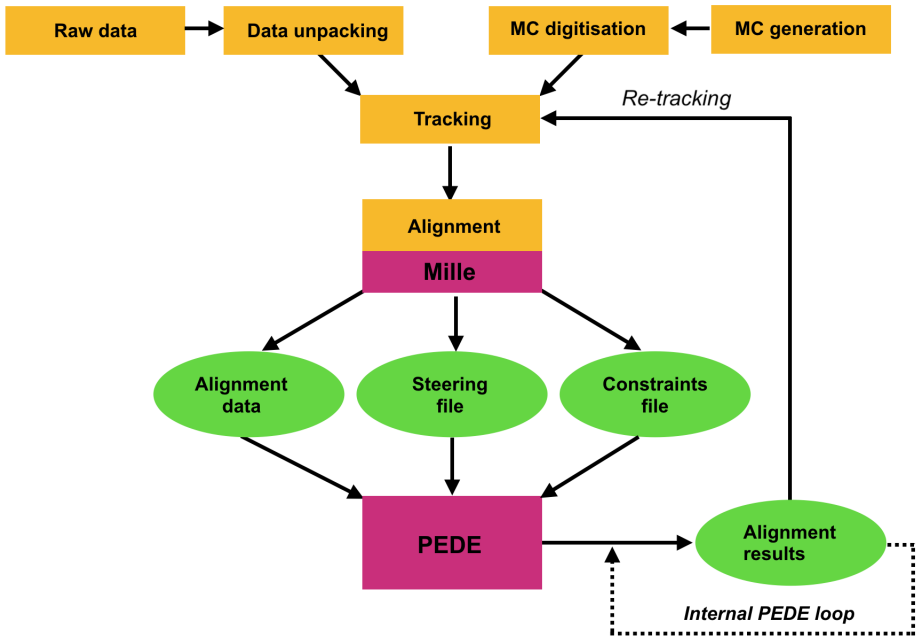

Fig. 4. The schematic of the `Millepede-II` routine (shown in purple) with the *gm2 art* software framework (shown in orange). The final result of alignment is then used in tracking to improve the residuals.

[2]Geane track fitting was developed by Nick

4.1 Alignment module: TrackerAlignment

The *producer* alignment module is controlled by a FHICL file, for example for S12, `trackerAlignment_S12.fcl`. The FHICL file controls the applied cuts on the tracks that will be used for alignment, and specifies the methods that will be used by `PEDE`.

A *dubug* mode is available when not monitoring the alignment over large datasets, with `monitoring : false`. This way, a `ROOT` file with debug plots will be produced. One must also set:
outputCommands : [“drop ∗”, “keep gm2strawtracker::TrackerAlignmentArtRecords_∗_∗_∗”]
in the FHICL file, as well as add plotsPath : [outputArtRootFile], to produce the `gm2tracker_reco.root` file, from which analysis-level alignment plots (`TrackerAlignment.root`) can be produced with `RunAlignmentPlots.fcl`.
The data product for the alignment is described in `gm2dataproducts/strawtracker/TrackerAlignmentArtRecord.hh`.

4.2 Tracking cuts for alignment

The following cuts were used to select tracks, from simulation or data, to be used in the alignment framework:

- **Maximum layers with multiple hits = 0.** Only select tracks that have unambiguous set of hits in a single straw per layer.
- **DCA** $>$ 500 µm**.** A hit with a small DCA is not used in the alignment. This is required due to the derived residual continuity as we cross the L/R boundary.
- **pValue** $>$ **0.005.** Only tracks that has a reasonable fit quality are used.
- **Hits** $>$ **9.** Only using tracks that have hits in at least three modules.
- $\mathbf{P_z/P} >$ **0.93.** Tracks that have large curvature are removed.

4.3 Installing and controlling `PEDE`

The `PEDE`, which a `Fortran` executable, is not part of art. The produced binary data can be passed to a standalone `PEDE` installation on `gm2gpvm`. The instruction to install `PEDE` are located on the **gm2tracker Redmine WIKI**. The `PEDE` is controlled via a steering file, which, as well as the constraints file, are produced automatically via the `TrackerAlignment` module:

```
* g-2 Tracker Alignment: PEDE Steering File
ConstraintFile.txt ! Constraints text file
Cfiles ! Following bin files are Cfiles
Data.bin ! Binary data file
method inversion 10 0.01 ! Method used by PEDE: see Millepede-II manual
```

The constraint equation is given by

$$c = \sum_{i}^{N} l_i \cdot f_l, \tag{2}$$

where c is is the constraint value, l is the global parameter label, f is the parameter factor[3], and the summation is taken over the parameters contributing to the constraint. In the particular case of the alignment of the tracking detector radially and vertically with curved tracks in a magnetic field, five global degrees of freedom must be fixed by the constraints. There are two overall translations, radially and vertically, that must sum to 0, as well as two global angles of rotations, which are also fixed at 0. The global rotation is defined through the centre of the station. This is done to minimise the "lever arm effect". The fifth constraint deals with the radial *bowing effect* due to the radially curving tracks in the magnetic field. The constraint equations can be formulated as follows, for example for station 12:

1. No overall radial translation

$$0.0 = \sum_{i=1}^{N} l_i \cdot 1 = 1211 \cdot 1 + 1221 \cdot 1 + ...1281 \cdot 1, \tag{3}$$

   where, for example, label 1281 corresponds to station number 12, module number 8 and parameter number 1 (i.e. radial shift). The factor of 1 simply means equal weighting for all parameters. The above equation is written in the PEDE format as follows

```
! Radial translational fixing (no overall movement):
Constraint 0
1211 1
1221 1
1231 1
1241 1
1251 1
1261 1
1271 1
1281 1
```

2. No overall vertical rotation

$$0.0 = \sum_{i=1}^{N} l_i \cdot (N + 1 - 2 \cdot i) = 1212 \cdot 7 + 1222 \cdot 5 + ...1282 \cdot -7, \tag{4}$$

[3]See `Millepede-II` manual [1]

```
! Vertical rotational fixing
! around the centre of the station:
Constraint 0
1212 7
1222 5
1232 3
1242 1
1252 -1
1262 -3
1272 -5
1282 -7
```

3. Constraint the radial bowing effect

$$0.0 = \sum_{i}^{N} l_i \cdot (N + 1 - 2 \cdot i)^2 = 1211 \cdot 49 + 1221 \cdot 25 + ...1281 \cdot 49, \tag{5}$$

```
! Bowing effect: parabloic radial term describing the track
   curvature
! around the centre of the station:
Constraint 0
1211 49
1221 25
1231 9
1241 1
1251 1
1261 9
1271 25
1281 49
```

The constrains can be summarised as follows, the radial constraints equation

$$0 = a(x - x_0)^2 + b(x - x_0) + c, \tag{6}$$

and the vertical constraints equation

$$0 = b(x - x_0) + c, \tag{7}$$

where $x_0 = 482.394\,\text{mm}$ is the centre of the station.

The motivation behind constraining the overall translations and rotations is simple: internal alignment should not return the module offsets related to the global movements. These global movements have been independently measured[4] by the global alignment [3]. The constraint on the radial bowing effect deserves a special mention, as it is essentially a "local minima problem": the internal alignment can

[4]Global alignment was established by Joe, Leah, and Horst.

place the tracker modules along a curved path, as the residuals will be unchanged. This radial detector curvature, however, will change the measurement of the momentum of the tracks, and the extrapolated radial position. The detector curvature, therefore, must be measured and eliminated by another means, as described in **DocDB:18625**.

4.4 Ancillary scripts

There is a lot of developed infrastructure to simplify the alignment routines in `gm2tracker/align/macros/`. Some of the essential scripts are summarised here:

- `GetRes.py`: Produces residual plots[5] on `trackRecoPlots.root` or `TrackerAlignment.root`
- `GetRes_Iter.py`: Produces residual comparison plots before and after alignment from two `TrackerAlignment.root` files
- `RobustTrackerLaunch.py`: Produces a visual representation of the modules offsets and produces `OffsetsPerModule.fcl` file that can be used for re-tracking
- `Rename.sh`: Renames `.root` files into `.bin` on the returned alignment data files form the grid. See **Grid Wiki: "What if I would like to retrieve output file(s) with different extension? like a binary file (.bin) produced by my jobs?"** and **DocDB:18826**
- `PEDEParallel.py`: Produces multiple steering files and runs `PEDE` in parallel on many runs
- `monitor.py`: Produces the final monitoring plot across many runs

4.5 Run alignment locally

One can start with raw `MIDAS` data, unpack it (`runStrawUnpacker.fcl`), and reconstruct tracks (`RunTrackingDAQ.fcl`), as described in **DocDB:19297**. The alignment can then be run locally on the tracks (e.g. `run15922_tracks.root`)

```
gm2 -c trackerAlignment_S18.fcl -s myDataSet_subrun.root
```

This will produce `SteeringFile.txt`, `ConstraintFile.txt` and binary data file `Data.bin`. To get the alignment correction for these tracks, `PEDE` is run with

```
PEDE SteeringFile.txt
```

If debug mode was selected, as described in Sc.4.1, analysis-level plots can be produced to check the residuals and derivatives via

```
gm2 -c RunAlignmentPlots.fcl -s gm2tracker_reco.root
```

[5] S12 and S18 must be specified for GeanePlots, as in the align/fcl/RunTrackingPlots_align.fcl

4.6 Run iterative alignment on a single run with residual verification

Refer to the **Grid Wiki** on how to *create* and *pre-stage* a SAM dataset.

One can start with raw `MIDAS` data, unpack it (`runStrawUnpacker.fcl`), and reconstruct tracks (`RunTrackingDAQ.fcl`), as described in **DocDB:19294**. Assuming this has been done, or a dataset (e.g. `myDataSet`) containing tracks from, for example, 450 subruns from a single run exists, alignment is done as follows.

1. Submit an alignment grid job (with the default FHICL parameters) for S18:

```
./gridSetupAndSubmitGM2Data.sh --daq --reco --fcl=gm2tracker/align/fcl/trackerAlignment_S18.fcl --localArea --output-dir=/pnfs/GM2/scratch/users/AAA/S18_align --sam-dataset=myDataSet --multipleroot --njobs=450
```

2. Rename files in `S18_align/DATE/data` to `.bin`

```
Rename.sh /pnfs/GM2/scratch/users/AAA/S18_align/DATE/data/
```

3. As `.txt` files are not returned by the grid (one can never have it all!) a local alignment jobs must be run to have the steering and constraint files

```
gm2 -c trackerAlignment_S18.fcl -s myDataSet_subrun.root
```

4. Add full paths of the renamed `.bin` files from `S18_align/DATE/data` to the SteeringFile.txt

5. Run `PEDE`

```
PEDE SteeringFile.txt
```

6. Produce offsets (`OffsetsPerModuleS18.fcl`) and their plots (from the results file `millipede.res`), as shown in Fig. 5. If used as part of an iteration, another option (e.g. for iteration 2) `--offset_file=RunTrackerDAQ_Iteration2.fcl` must be used to supply the previous set of alignment results for comparison.

```
python RobustTrackerLaunch.py --pede_file=millipede.res
```

7. Add the resultant `OffsetsPerModuleS18.fcl` to end of the `RunTrackingDAQ_Iteration2.fcl` and re-track on the grid, defining a new dataset `myDataSetIteration2`

8. Repeat steps $1-7$, until the results from the current iteration are same as the last one

Fig. 5. The alignment results with 10^5 tracks after 2 iterations. The alignment stability is not yet reached, as seen by the alignment results from the previous iteration (shown in black) and current iteration (purple) not converged.

9. Produce the final residual comparison plots, for example, for the first and third iteration as shown in Fig. 6. **S12 and S18 must be specified for separate instances of GeanePlots, as shown in `align/fcl/RunTrackingPlots_align.fcl`, to produce residuals separately for each station**. Alternatively, run directly on `TrackerAlignment.root`

```
GetRes_Iter.py --fileN1=Iter1/TrackerAlignment.root --fileN1=Iter3/TrackerAlignment.root
```

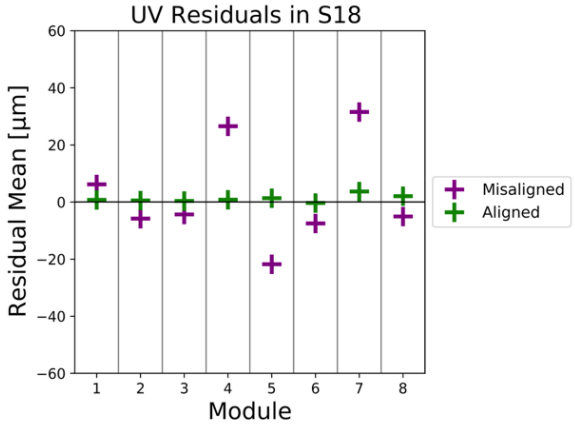

Fig. 6. An improvement in the residual means after the alignment (3 iterations) is clear.

4.7 Run alignment monitoring

The goal here is to check that across an entire dataset or a whole Run (e.g. Run-1), the alignment is stable. For example, for an entire *60-hour* dataset with the dataset name `gm2pro_daq_full_run1_60h_5037A_goldList`:

1. Submit an alignment grid job (with the default FHICL parameters) for S12:

```
./gridSetupAndSubmitGM2Data.sh --daq --reco --fhiclFile=trackerAlignment_S12.fcl --localArea --output-dir=/pnfs/GM2/scratch/users/AAA/gm2pro_daq_full_run1_60h_5037A_goldList/S12/ --sam-dataset=gm2pro_daq_full_run1_60h_5037A_goldList --noifdh_art --onebyone --multipleroot --njobs 5000
```

2. Rename files in `S18_align/DATE/data` to `.bin`

```
Rename.sh /pnfs/GM2/scratch/users/AAA/S18_align/DATE/data/
```

3. Run script that will take care of preparing steering and constraint files for each run in the new folder `Monitoring`

```
python gm2tracker/align/macros/PEDEParallel.py --mode=prepare --inputData=gm2pro_daq_full_run1_60h_5037A_goldList/S12/DATE/data/
```

4. Run the same script in `pede` and then `align` modes to first run `PEDE` and then `RobustTrackerLaunch.py` on each run

```
python gm2tracker/align/macros/PEDEParallel.py --mode=pede --inputData=gm2pro_daq_full_run1_60h_5037A_goldList/S12/DATE/data/Monitoring/
python gm2tracker/align/macros/PEDEParallel.py --mode=align --inputData=gm2pro_daq_full_run1_60h_5037A_goldList/S12/DATE/data/Monitoring/
```

5. Finally, produce a monitoring plot as shown in Fig. 7.

```
python $MP2/monitor.py -s12 S12/ -s18 S18
```

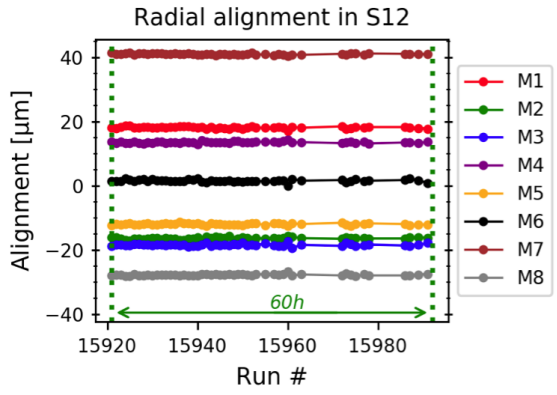

Fig. 7. Radial alignment results in the eight modules of station 12 across 60 hours of $g-2$ data.

4.8 Run alignment to establish constants for a run interval

The methodology is the same as in Sc. 4.6. However, a larger subset of data should be used with ~ 8 runs ($\sim$3 million tracks).

As soon as a module is physically swapped in a station, or removed even for a short time, a new IoV, and new constants, are required. However, the non-swapped modules should have the same offsets and set as *fixed*[6] in the new *parameter file*. That is, we will only perform alignment on the swapped modules. For alignment with a single module, since the number of degrees of freedom is reduced, a constraint file is no longer necessary. Tracking must be performed without loading any alignment constants. The new steering file is

```
* g-2 Tracker Alignment: PEDE Steering File
ParameterFile.txt ! Non-swapped modules are fixed here
Cfiles ! Following bin files are Cfiles
Data.bin ! Binary data file
method inversion 10 0.01 ! Method used by PEDE: see Millepede-II manual
```

While the newly added parameter file fixes all other modules at their previously established positions, and only performs alignment on, for example, module 6 (N.B. alignment counts modules from 1, due to the underlying Fortran code in *PEDE*, convention used in tracking software starts counting from 0, as is sensible)

```
PARAMETERS
1811  0.0153  -1
1812  -0.0711  -1
1821  -0.0182  -1
1822  -0.0919  -1
1831  -0.0107  -1
1832  0.0645  -1
1841  0.0543  -1
1842  0.0767  -1
1851  -0.0534  -1
1852  0.0581  -1
1871  0.0752  -1
1872  0.018  -1
1881  -0.0413  -1
1882  -0.1705  -1
```

For iteration 2 onwards, the above offsets must be added to the `RunTrackingDAQ.fcl`, together with the newly derived offsets for module 6 (in this example), and the parameter file updated with 0.0 for all fixed modules instead of their original offsets.

[6] Parameter factor of -1

5 Alignment constants and the reconstruction database

The alignment constants are now used in the production database (`gm2_conditions_prod : tracker_align_internal_module`). The service that implements the alignment into the tracker geometry is the `gm2ringsim/strawtracker/StrawTrackerCadMesh_service.cc` via (e.g. for S12 radial shift) `strawModuleRShift12`, where offsets are given as an array of 8 numbers. To write the newly derived alignment constants, as described in in Sc. 4.6 and Sc. 4.8 (also see **DocDB:19505**):

1. The `gm2db/utils/FhiclToJsonFileWriter.py` script is used to produce a JSON file from the FHICL file

```
python FhiclToJsonFileWriter.py --fcl
OffsetsPerModuleS12S18_Run1_Period_1_of_1_15921_19027.fcl --ana
tracker_align_internal_module --iov run_range --path
tracker_align_json
```

   where OffsetsPerModuleS12S18.fcl is the combined FHICL file for S12 and S18, with the added (e.g.) firstRun: 15921 and lastRun: 19027 lines. The above command is run once per IoV (per FHICL file) to produce a JSON file. For reference, all 3 IoV FHICL files from Run-1 and Run-2 are stored in `gm2tracker/align/alignmentConstants_Run1_Run2`. The produced JSON files are stored in `gm2tracker/align/tracker_align_json`

2. The `gm2db/conDBScripts/LoadDataDevDB.py` script writes JSON files into the **development** DB:

```
python LoadDataDevDB.py --action=write --input=
TrackerAlignment.txt
```

   where `TrackerAlignment.txt` is a file of file paths to the JSON file(s) from step 1 (see `https://cdcvs.fnal.gov/redmine/projects/g-2/wiki/Development_Database`). If running for the first time, create the tables:

```
python LoadDataDevDB.py --action=create --input=
TrackerAlignment.txt
```

3. Tag the DB commit via (e.g.)

```
python LoadDataDevDB.py --action=tag --input=
TrackerAlignment.txt --tag=v9_28_00
```

4. Test the new constants, and ask a production team member to replicate the tables into the **production** DB

```
python LoadDataProdDB.py --folder=
tracker_align_internal_module --prodTag v9_28_00 --devTag
v9_28_00
```

References

[1] V. Blobel, *Software alignment for tracking detectors*, Nucl. Instrum. Methods A, **556**, 5 (2006).
[2] N. Kinnaird, *E989 Note 184: Geane Fitting Documentation*, **DocDB:8102**, (2017).
[3] J. Price, *Global alignment*, **DocDB:16063**, (2019).

Acknowledgements

I would like to thank the entire tracker group for their immense support in this project! In particular, I would like to express gratitude to Mark, Becky, James, Joe, Brendan, Saskia, Tammy and Nick for their advice.