Software Alignment Procedure in FairRoot

An Introduction

Roman Klasen | roklasen@uni-mainz.de

Software Alignment

What is Misalignment?

Every part of our detector has a design position and orientation and an actual position and orientation. The difference between these positions and orientations is called **misalignment**.

It can be expressed as a single 4x4 homogeneous transformation matrix for each part.

What is Software Alignment?

The determination of these matrices using software methods instead of hardware measurements, or:

The model of the real detector geometry in our simulations, which accounts for these misalignments.

Software Alignment II

We distinguish two related but very different concepts:

During Simulation / Pre-Experiment

Generate mc tracks (or similar) using a “wrong” geometry just like a real detector would produce. The tracks will be off w.r.t. to their “real” position. Use this to study how your analysis software handles a realistic, misaligned geometry. This can be done two different ways, see slides 8 and 9.

For Real Measurement Data / During Experiment

Once built, use the alignment parameters obtained from survey etc on *real measurement data*. This accounts for misaligned detector parts and produces reconstructed tracks that are closer to the real tracks than without alignment. ***This is the main goal of software alignment.***

Software Alignment III

__Misalignment Matrix:__

- The matrix that describes the deviation of a part from it’s design position/orientation. It’s impossible to know it with certainty, and is only accessible in simulation.
  Start with ideal Position -> Shift/Rotate to obtain new, misaligned position

__Alignment Matrix:__

- The physically measured parameters for this deviation *in the other direction*. Start at actual, misaligned position -> how do we reach the design position?

That means: $M_{misalignment} \cdot M_{alignment} \approx 1$

Homogeneous Transformation Matrices

Rotation Matrix

$$R = \begin{bmatrix} R_1 & R_2 & R_3 & 0 \\ R_4 & R_5 & R_6 & 0 \\ R_7 & R_8 & R_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation Matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

There are also other matrices for scaling, shearing, mirroring, projections etc, but those are not used for alignment.

Example Matrix path

Multiple transformations can be chained to a single matrix that incorporated all transformations into one single transformation:

$$M_3 \cdot M_2 \cdot M_1 = M_{total}$$

The order of multiplication is important as matrices don't commute!

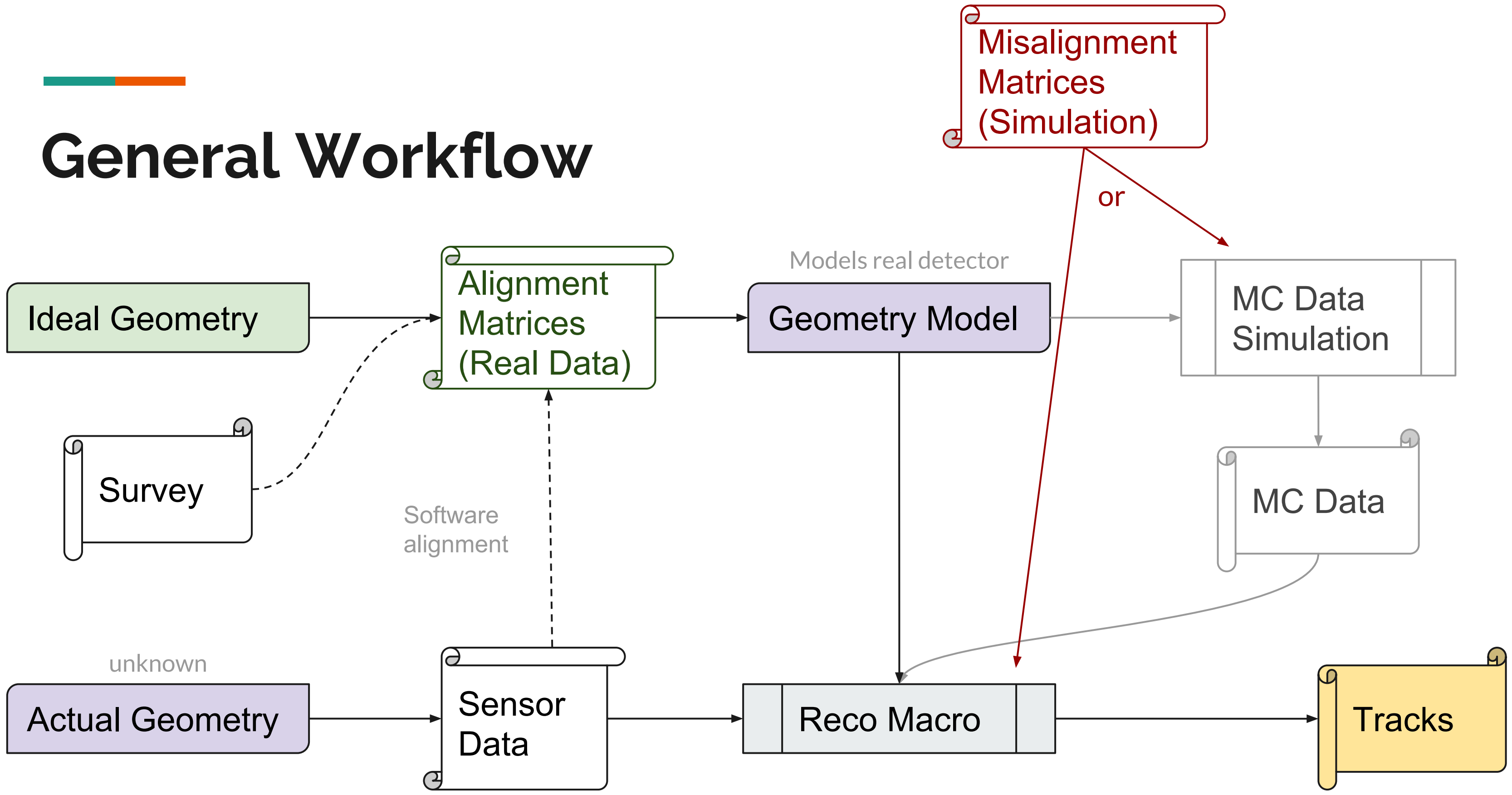
In our specific case, these matrices are:

$$M_{design} \cdot M_{misalignment} = M_{actual}$$

Our software alignment aims to find the deviation matrices for example. The hardware survey team might give you M_{actual} or $M_{misalignment}$. You have to extract alignment matrices from that!

The new AlignmentHandler will always use this system.

General Workflow



Simulation: Shift Detector / Shift Data

Shift Detector

- Realistic Detector Acceptance
- Realistic scenario for Track Finder, Fitter etc.
- Reco Macro need to only load Geometry from TGeoManager (which handles Align.)
- But need to generate MC Data again (esp. If you want multiple misaligned geometries)

Shift Data

- Can use existing MC data
- Wrong detector acceptance may lead to implausible tracks:
- Don’t see some tracks that should be there
- See tracks that can’t be there
- Reco Macros must account for Misalignment

The new AlignmentHandler class can do both.

Simulation: Shift Detector / Shift Data II

But the important thing is: after the Reco step, both methods produce (almost) the same data!

- Shift detector: apply deviation (misalignment) matrices during Simulation *and* Digitization macros
- Shift data: apply *inverse* misalignment matrices to working geometry during Reconstruction macro

The AlignmentHandler class does the actual matrix application for you, you only have to specify the matrices and give them to the handler. See the example code on later slides.

Again: this is for misalignment studies, i.e. how your analysis software behaves when the detector geometry is misaligned. For use on a real detector with real data, see later slide!

This is superseded
By AlignmentHandler!

Current / Old Method

```
// include in Simulation and Digitization or Reconstruction before init() or hard code to detector
void PndLmdDetector::ModifyGeometryByFullPath() {
  TString volPath;
  for (auto const& entry : fAlignmentMatrices) {
    volPath = entry.first;
    gGeoManager->cd(volPath);
    TGeoNode* n3 = gGeoManager->GetCurrentNode();
    TGeoMatrix* l3 = n3->GetMatrix();              // new matrix, representing real position
    TGeoHMatrix nlocal = *l3 * entry.second;       // from new local mis RS to the global one
    TGeoHMatrix* nl3 = new TGeoHMatrix(nlocal);
    TGeoPhysicalNode* pn3 = gGeoManager->MakePhysicalNode(volPath);
    pn3->Align(nl3);
  }
}
```

That means ***your detector*** class was responsible for misalignment (see FairRoot Tutorial4)

New Method | Shift Geometry

Add this to your Simulation and Digitization Macros:

```
// [...] previous steps
// -----   Reconstruction run   -------------------------------------------
FairRunAna *fRun = new FairRunAna();
// [...] add runtime db, tasks, config files etc
// -----   set misalignement matrices -------------------------------------
std::map < std::string, TGeoHMatrix > misalignMatrices = magicMatrixGetterFunction();            // this is up to you


fRun->AddAlignmentMatrices(matrices); // this is the actual misalignment!

// [...] other settings
// -----   Initialise and run   -------------------------------------------
fRun->Init();                  // call only after AddAlignmentMatrices()!
fRun->Run(0, nEvents);
```

New Method | Shift Data

Add this to your Reconstruction Macro:

```
// [...] previous steps
// -----   Reconstruction run   --------------------------------------------
FairRunAna *fRun = new FairRunAna();
// [...] add runtime db, tasks, config files etc
// -----   set misalignement matrices --------------------------------------
std::map < std::string, TGeoHMatrix > misalignMatrices = magicMatrixGetterFunction();             // this is up to you
bool use_point_transform_misalignment = true;                              // because we need the inverse matrices, might change in the future

fRun->AddAlignmentMatrices(matrices, use_point_transform_misalignment); // this is the actual misalignment!

// [...] other settings
// -----   Initialise and run   --------------------------------------------
fRun->Init();                  // call only after AddAlignmentMatrices()!
fRun->Run(0, nEvents);
```

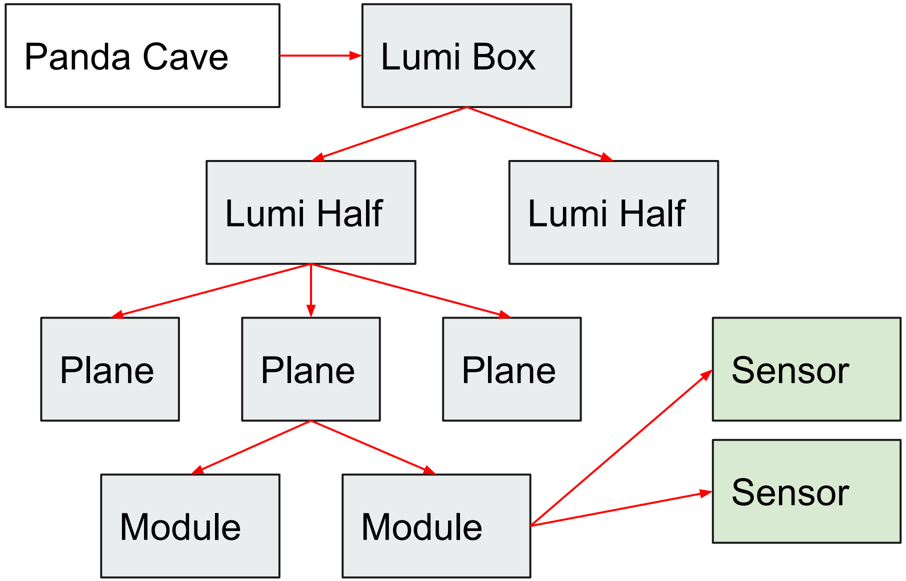
Bonus: For real Measurement Data

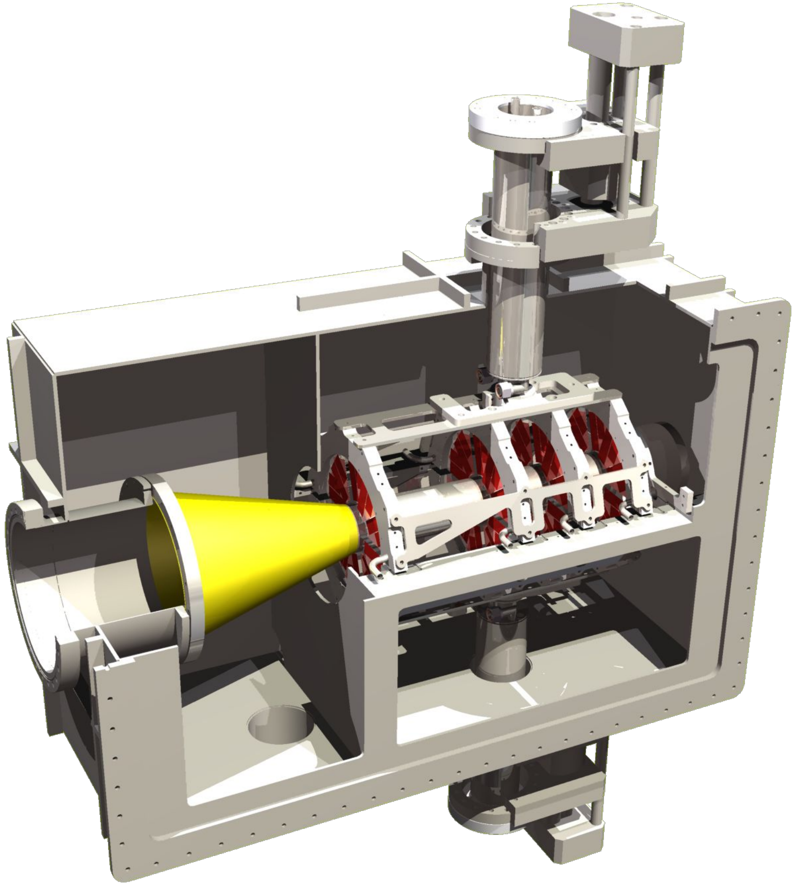
This is for the actual Experiment: like above, just add this to your Reconstruction Macro:

```
// [...] previous steps
// -----   Reconstruction run   -------------------------------------------
FairRunAna *fRun = new FairRunAna();
// [...] add runtime db, tasks, config files etc
// -----   set misalignement matrices -------------------------------------
std::map < std::string, TGeoHMatrix > alignmentMatrices = matricesFromSurvey();                    // physical measurement
bool use_point_transform_misalignment = true;                                   // depends on how you get your alignment parameters

fRun->AddAlignmentMatrices(matrices, use_point_transform_misalignment); // this

// [...] other settings
// -----   Initialise and run   -------------------------------------------
fRun->Init();                    // call only after AddAlignmentMatrices()!
fRun->Run(0, nEvents);
```

Luminosity Detector Example

Alignable Components



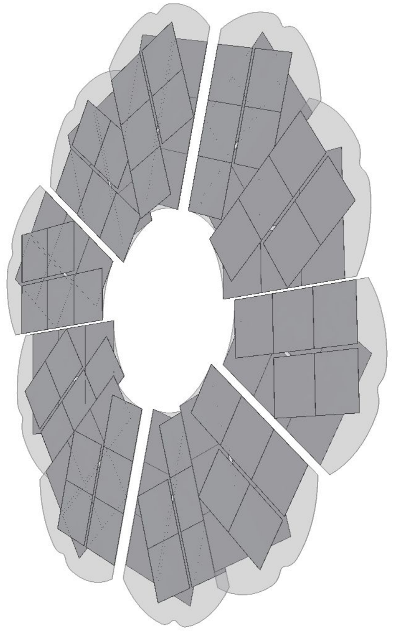


Overlapping Sensor Design

In our geometry, there are overlapping sensors. For these, I create a set of misalignment matrices that rotate and translate each sensor individually and run the entire simulation chain:

- MC simulation
- Digitization
- Reconstruction
- Etc...

We misalign sensors only, and we only shift in xy and rotate about z!



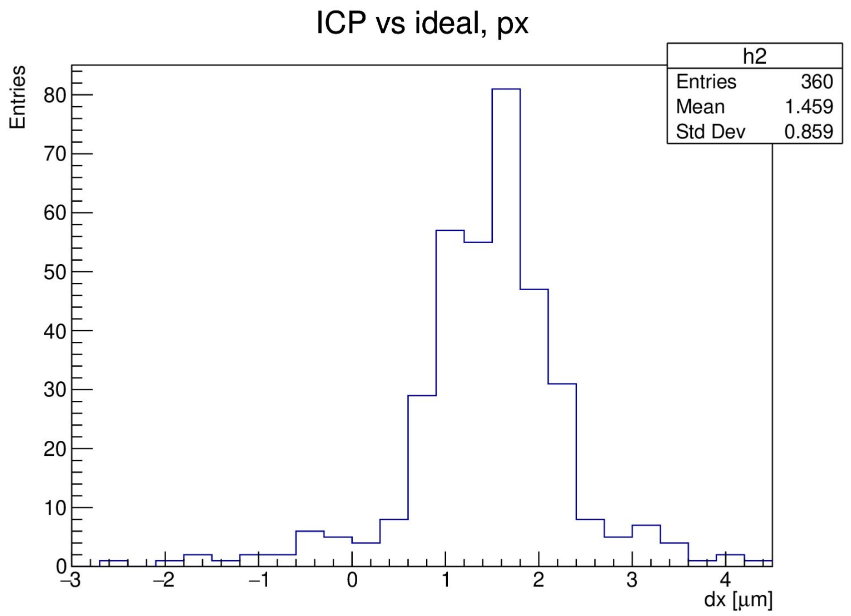
Comparison methodology

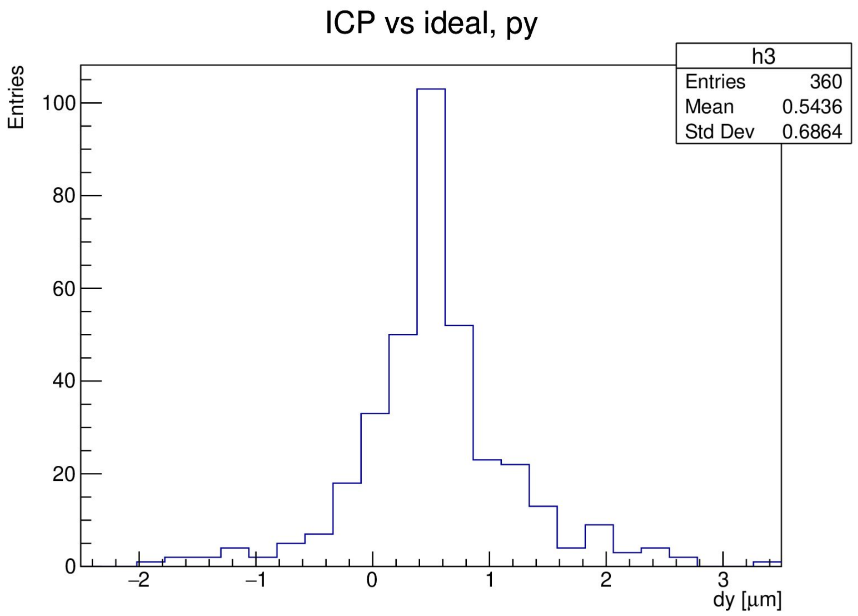
To see if the AlignmentHandler works, and if the two methods yield the same result, we do the following:

- Use the overlapping areas to find the matrices from the front sensor to the back sensor
- Compare this matrix to the design misalignment (which we know from the simulation)
- Do this with both methods, *shift geometry* and *shift data*
- Compare the found matrices from both *shift geometry* and *shift data*

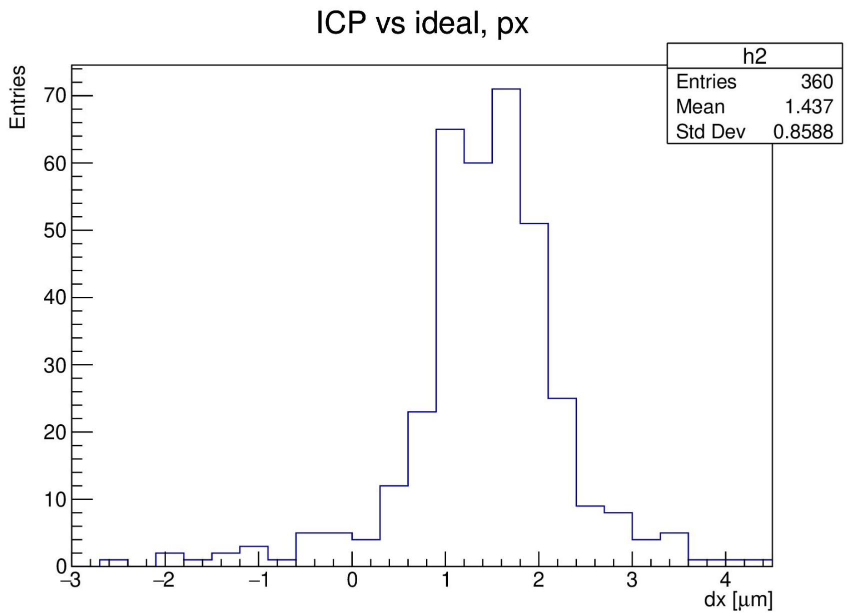
For simplicity, here we'll only look at the translation values of the found homogenous transformation matrices and ignore the rotation for now.

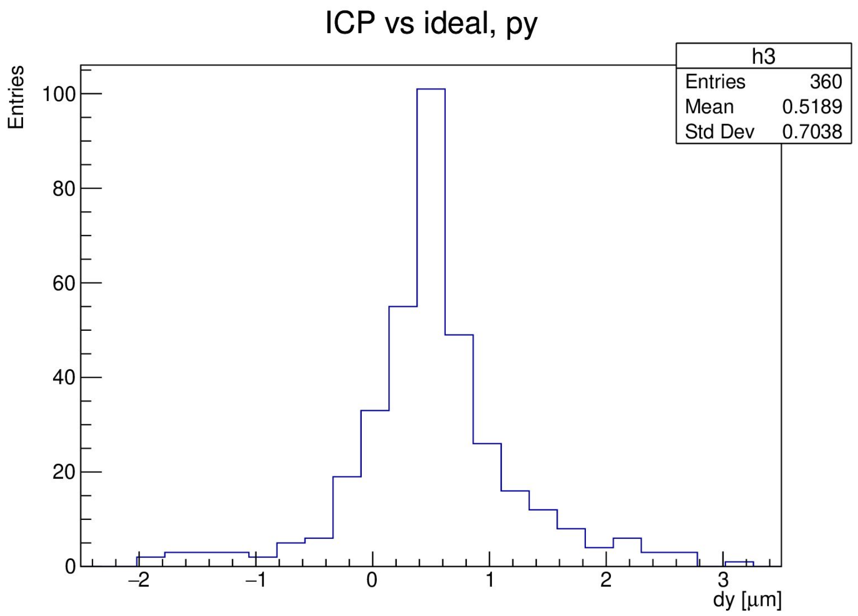
Comparison Overlap Matrices | Shift Geometry



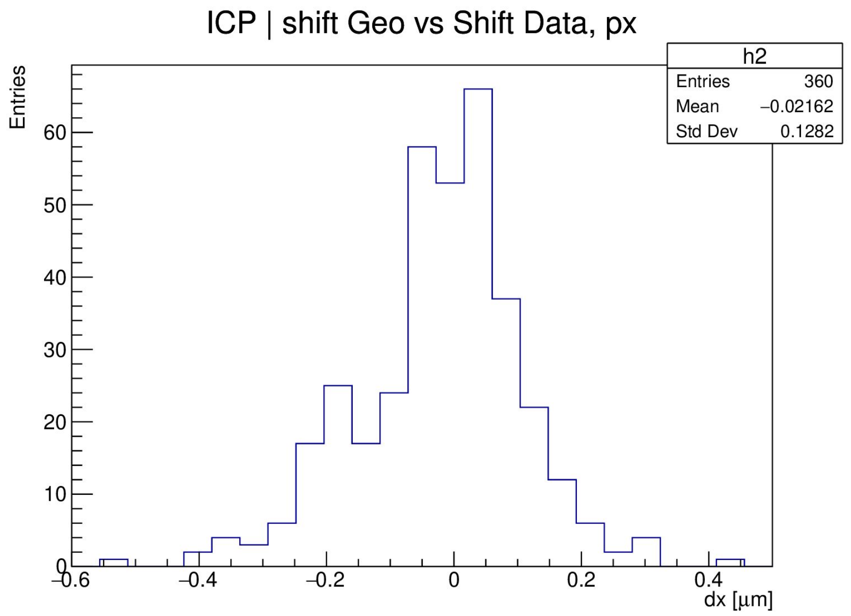


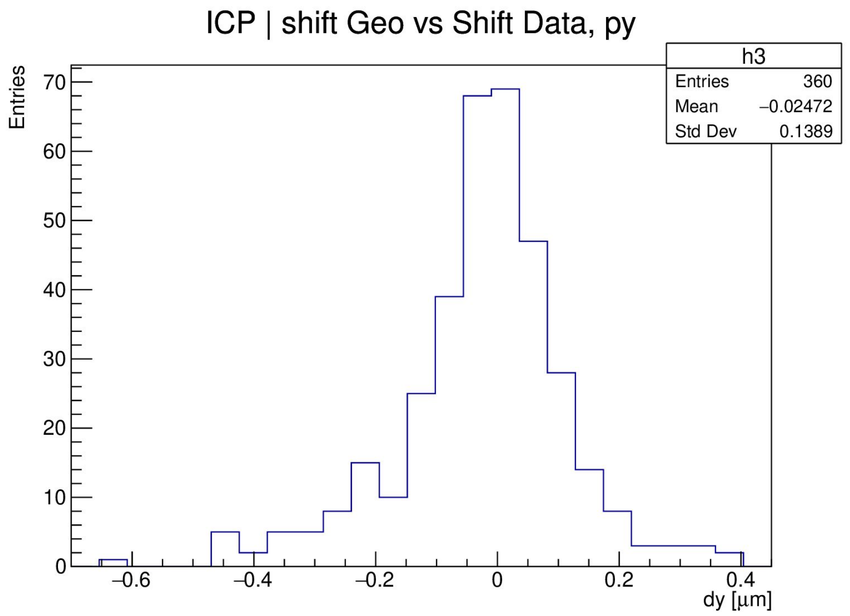
Comparison Overlap Matrices | Shift Data





Comparison Shift Geometry vs Shift Data





Conclusion

We have implemented an easy to use interface to the complex and error prone matrix chain. The user can choose to add misalignment at simulation stage or reconstruction stage.

The two different ways to misalign a geometry produce almost the exact same results. The differences can stem from different sources:

- Unequal amount of data
- Different Detector acceptance

It’s available in FairRoot and therefore readily available to all experiments with minimal changes to current code.

See our Code

You can see an example of the complete process at: **`pandaroot/macro/detectors/lmd`**

Matrix creation:

- `PandaRoot/macro/detectors/lmd/geo/createPndLmdMisalignmentMatrices.C`

Set misalignment matrices in MC data generation:

- `PandaRoot/macro/detectors/lmd/runLumiPixel0SimBox.C`

Or set them in the Reconstruction macro

- `PandaRoot/macro/detectors/lmd/runLumiPixel2Reco.C`

Where is this code?

Pull request submitted to FairRoot, currently in review

As soon as this PR is reviewed, checked and approved, you will find this functionality in FairRoot:

https://github.com/FairRootGroup/FairRoot

This way, no changes are necessary to PandaRoot and every group can start using misalignment code.

Thank you for your attention!