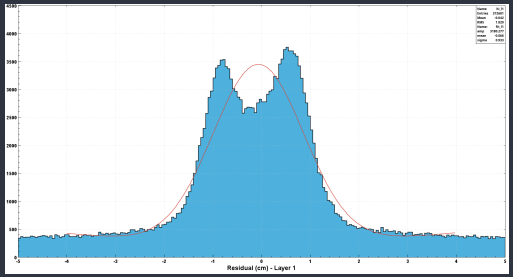
FMT Alignment Software
-
Usage

Bruno Benkel - Raffaella DeVita

December 15, 2021

Context

- This program uses residuals analysis to align the FMT detector.
- A residual is the distance between an FMT cluster and a DC track in the FMT layer's local coordinate system.
- The program works by fitting the residuals distribution in each FMT layer for different shifts or rotations. Then, it plots the means and widths of the fits so that the user can select the minimum for each.



Example of a (bad) residuals distribution.

Program Usage: Setup

Parameters:

- `<file>`: Only positional argument, denotes the path to the file to be used for alignment.
- `-v --var`: String defining the variable to be aligned. Can be **dXY**, **dZ**, **rXY**, and **rZ**. **d** denotes a shift, while **r** denotes a rotation.
- `-i --inter`: Denotes the values of shifts or rotations to be tested. Requires two parameters:
  - Range between nominal position and position to be tested.
  - Step size for each tested value between <nominal - range> and <nominal + range>.
- `-x --dx`: Nominal position for **x** on each FMT layer.
- `-Z --rz`: Nominal rotation for **z** on each FMT layer.
- etc...

This list is not exhaustive. All arguments are specified in the README and in the program's usage.

Program Usage: Setup

For example, if `var == "dZ"`, `inter == "0.2 0.1"`, and `dz == "0.5 0.5 0.5"`, then the values tested for z on each FMT layer are

`(0.3, 0.4, 0.5, 0.6, 0.7).`

If a position or rotation is not specified, it is assumed to be 0 for all FMT layers.

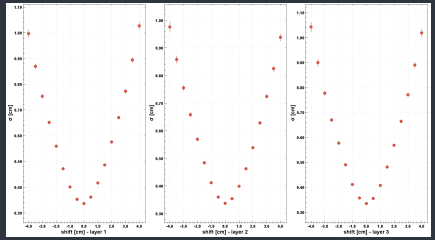
Note: It is recommended to start alignment with the variable that sees the largest misalignment. Considering this, for FMT it's ideal to align **dZ**, then **rZ**, and finally **dXY**.

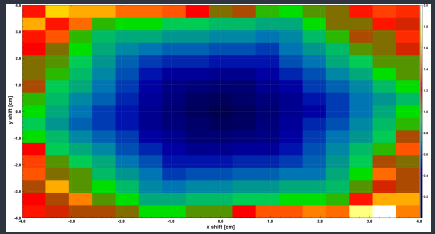
Program Usage: Results

The program produces either a 1D or a 2D plot depending on the type of alignment ran:

- For Z alignment, a 1D plot of the deviation of the Gaussian fit against the shift or rotation applied is shown for each FMT layer.
- For XY alignment, a 2D plot of the average of the means of the Gaussian fits for each layer against the shifts applied. Due to only having 3 layers, there is limited accuracy on XY alignment.

The most accurate shifts and rotations are the ones with the lowest means and deviations, within acceptable error margins. These should be uploaded to the CCDB.





RG-F Example

To exemplify, we're going to take a small peek into the FMT alignment done for RG-F's summer run.
The testing conditions for this are:

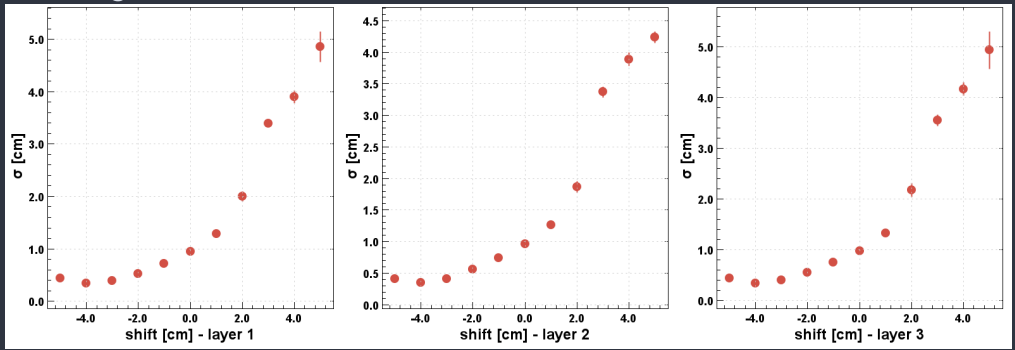
- Alignment was performed on run 12439 — current was 15 nA, energy 2.18 GeV, inbending field.
- Coatjava version 6.5.8 was used.

RG-F Example: dZ

First, we run **dZ** alignment. We run the script with:

```
./run.sh
/home/twig/data/code/jsw/recon_data/out_clas_012439_unaligned.hipo
-n 100000 -v dZ -i 5.0 0.5
```

And we get:

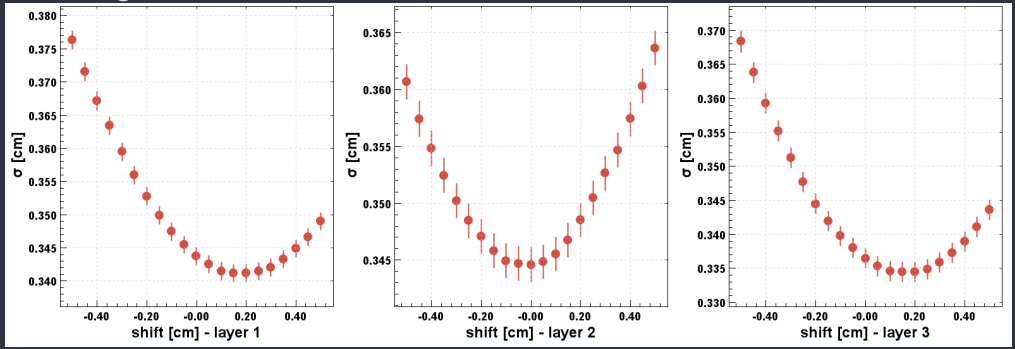


RG-F Example: dZ

Now we know that the z shift for each layer is about -4.0 cm. Knowing this, we run the script with:

```
./run.sh ...
-z -4.0 -4.0 -4.0 -n 1000000 -v dZ -i 0.5 0.05
```

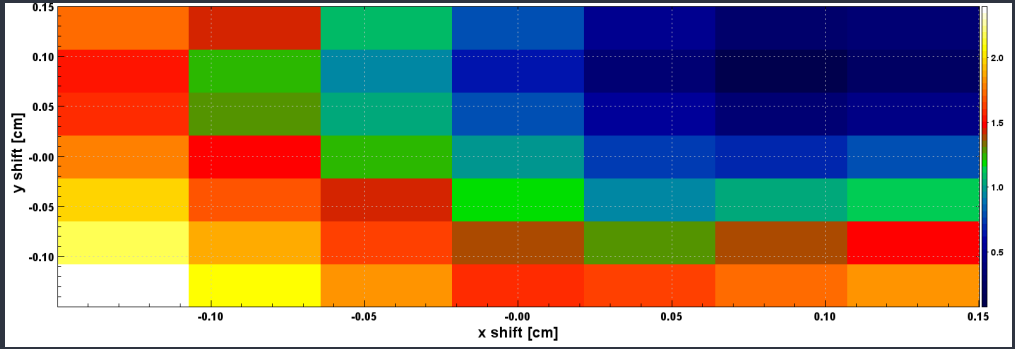
Obtaining:



RG-F Example: dXY

Now we know the z shift for each layer, and the error bars tell us that this is as good a result as we'll get. We'll skip **rZ** and move on straight to **dXY**.

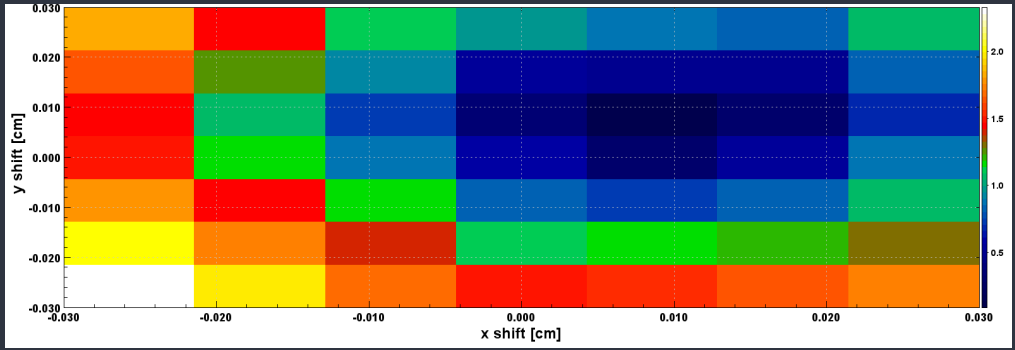
```
./run.sh ...
-z -3.75 -4.05 -3.85 -Z -0.50 -0.50 -0.40 -n 100000
-v dXY -i 0.15 0.05
```



RG-F Example: dXY

Finally, we squeeze in a bit more detail for **dXY**.

```
./run.sh ...
-z -3.75 -4.05 -3.85 -Z -0.50 -0.50 -0.40 -n 100000
-x 0.10 0.10 0.10 -y 0.10 0.10 0.10 -v dXY -i 0.03 0.01
```



RG-F Example: Results

Remembering that CCDB variables are stored in mm, we write a text file detailing the shifts and rotations found as:

```
# sector layer component deltaX deltaY deltaZ rotX rotY rotZ
  0      1     0         1.1    1.1    -37.5  0.0  0.0  -0.50
  0      2     0         1.1    1.1    -40.5  0.0  0.0  -0.50
  0      3     0         1.1    1.1    -38.5  0.0  0.0  -0.40
  0      4     0         0.0    0.0      0.0  0.0  0.0   0.0
  0      5     0         0.0    0.0      0.0  0.0  0.0   0.0
  0      6     0         0.0    0.0      0.0  0.0  0.0   0.0
```

Note: The program takes about 25 seconds to process 100.000 events per tested shift in an average CPU.