The ALICE alignment framework

R. Grosso[a]
for the ALICE Collaboration

[a] *CERN*
Geneva, Switzerland

Abstract
The ALICE alignment framework is described. Based on the ROOT geometry package, it provides the functionality to consistently produce, store, retrieve, and apply the alignment constants.

15.1 Basic objects and alignment constants

The purpose of the ALICE alignment framework is to offer all the functionality related to storing alignment information, retrieving it from the Offline Conditions Data Base (OCDB), and consistently applying it to the ALICE geometry. This functionality is required in order to improve our knowledge of the real geometry by means of the additional information obtained by survey and alignment procedures, without needing to change the hard-coded implementation of the detector's geometry. The ALICE alignment framework is built on the `AliAlignObj` base class and its derived classes; each instance of one of this classes is an *alignment object* and stores the so-called *alignment constants* for a single alignable volume:

- a unique volume identifier,
- a unique global index,
- a delta-transformation.

This is the information to uniquely identify the physical volume to be displaced and to unambiguously describe the delta-transformation to be applied to that volume.

In the following we describe the meaning of these variables, how they are stored and set, and the functionality related to them. A deeper description of the framework providing several examples of usage is given in Ref. [1].

15.1.1 The unique volume identifier

The unique volume identifier is the character string allowing access to a specific physical volume inside the geometry tree. For the ALICE geometry (which is a ROOT geometry) this is the *volume path*, a string containing the names of all physical volumes in the current branch of the geometry tree as a file path does it in a file system tree. For example, the volume path `/A/B_i/.../M_j/Vol_k` identifies the physical volume 'kth copy of the volume `Vol`' by listing its container volumes; going from right to left in the path corresponds to going from the innermost to the outermost containers and from the lower to the upper level in the geometry tree, starting from the mother volume `M_j` of the current volume `Vol_k` up to the physical top volume `A`, the root of the geometry tree.

The unique volume identifier stored by the alignment object is not the volume path but a '*symbolic volume name*', a string dynamically associated to the corresponding volume path at the finalization stage of the geometry; the mapping of symbolic names to volume paths is stored in the geometry itself.
The choice of the symbolic volume names is constrained only by the following two rules:

1. Each name has to contain a leading sub-string indicating its pertaining sub-detector; in this way, the uniqueness of the name inside the sub-detector scope also guarantees its uniqueness in the global scope of the whole geometry.
2. Each name has to contain the intermediate alignable levels, separated by a slash ('`/`'), in case some other physical volume on the same geometry branch is in turn alignable.

Introducing the symbolic volume names as unique volume identifiers stored in the alignment object in place of the volume path has two considerable advantages:

1. The unique volume identifier has no direct dependence on the geometry; in fact, changes in the volume paths reflect changes in the hash table associating the symbolic names to them, which is built and stored together with the geometry. As a consequence, the validity of the alignment ob-

jects is not affected by changes in the geometry and hence is in principle unlimited in time.

2. The unique volume identifier can be freely chosen, according to the two simple rules mentioned above. This allows one to assign meaningful names to the alignable volumes, as opposed to the volume paths, which inevitably are long strings of often obscure names.

The geometry then provides the user with some methods to query the hash table linking the symbolic volume names to the corresponding volume paths; in particular, the user can

- obtain the number of entries in the table;
- retrieve a specific entry (symbolic volume name, volume path) either by index or by symbolic name.

15.1.2 The unique global index

Among the alignment constants we store a numerical index (a short) uniquely identifying the volume to which those constants refer. The 16 available bits of the global volume index are filled by the index of the 'layer' or sub-detector to which the volume belongs (5 bits) and by the 'local volume index', i.e., the index of the volume itself inside the sub-detector (the remaining 11 bits). Limiting the range of sub-detectors to $2^5 = 32$ and of alignable volumes inside each sub-detector to $2^{11} = 2048$, this fits the cardinality of sub-detectors and sub-detectors' modules in ALICE.

The aim of indexing the alignable volumes is fast iterative access during alignment procedures. The look-up table mapping symbolic volume names to indexes is built as a static variable when the first instance of an alignment object is created in memory. The framework allows one to browse easily the layered structure of the look-up table, where each layer corresponds to a set of alignable volumes belonging to the same sub-detector and placed at the same radial distance from the centre of the ALICE detector. The user can query the table for the number of sensitive alignable volumes in each layer, for the layer name corresponding to a given layer index, for the symbolic volume name corresponding either to a global volume index or to a layer index plus a local volume index.

15.1.3 The delta-transformation

The delta-transformation is the transformation which defines the displacement to be applied to the given physical volume. During offline alignment we wish to correct the hard-coded, ideal position of some volume, initially fixed according to the engineers' drawings, by including the survey and alignment information related to those volumes; we say that we wish to align the ideal geometry. With this aim, here we need to describe how the delta-transformations are defined and thus how they have to be produced and applied to the ideal geometry in order to correct the global and local ideal transformations into global and local aligned transformations.

For the representation of the delta-transformation there are several possible conventions and choices, in particular:

1. to use the local-to-global or the global-to-local convention and "active-" or "passive-transformations" convention;
2. to use the local or global delta-transformation to be stored in the alignment object and to be passed when setting the object itself;
3. the convention used for the Euler angles representing the delta-transformation;
4. the use of a matrix or of a minimal set of parameters (three orthogonal shifts plus three Euler angles) to be stored in the alignment object and to be passed when setting the object itself.

The choices adopted by the framework are explained in the remainder of this section.

15.1.3.1 Use of the global and local transformations

Being based on the ROOT geometry package, the framework keeps the 'local-to-global' convention; this means that the *global transformation* for a given volume is the matrix $\mathcal{G}$ which, as in TGeo, transforms the local vector $\vec{l}$ (which gives the position in the local reference system, i.e., the reference system associated to that volume) into the global vector $\vec{g}$, which gives the position in the global (or master) reference system ('MARS'), according to:

$$\vec{g} = \mathcal{G}\vec{l}\,. \tag{15.1}$$

Similarly, the *local transformation* matrix is the matrix $\mathcal{L}$ which transforms a local vector $\vec{l}$ into the corresponding vector in the mother volume RS, $\vec{m}$, according to:

$$\vec{m} = \mathcal{L}\vec{l}\,. \tag{15.2}$$

If, furthermore, $\mathcal{M}$ is the global transformation for the mother volume, then we can write:

$$\vec{g} = \mathcal{G}\vec{l} = \mathcal{M}\vec{m} = \mathcal{M}\mathcal{L}\vec{l}\,.$$

Recursively repeating this argument to all the parent volumes, that is to all the volumes in the branch of the geometry tree which contains the given volume, we can write:

$$\vec{g} = \mathcal{G}\vec{l} = \mathcal{M}_0 ... \mathcal{M}_n \mathcal{L}\vec{l}\,.$$

which shows that the global matrix is given by the product of the matrices of the parent volumes on the geometry branch, from the uppermost to the lowest level.

Let us now denote by $\mathcal{G}$ and $\mathcal{L}$ the ideal global and local transformations of a specific physical volume

(those relative to the reference geometry) and let us put the superscript '^{a}' to the corresponding matrices in the aligned geometry, so that $\mathcal{G}^a$ and $\mathcal{L}^a$ are the aligned global and aligned local transformations, which relate the position of a point in the local RS to its position in the global RS and in the mother's RS, respectively, after the volume has been aligned, according to:

$$\vec{g} = \mathcal{G}^a\vec{l} \quad (15.3)$$
$$\vec{m} = \mathcal{L}^a\vec{l}\,. \quad (15.4)$$

Eqs. (15.3)–(15.4) are the equivalent of Eqs. (15.1)–(15.2) after the volume has been displaced.

There are two possible choices for expressing the delta-transformation; either we use:

- the *global delta-transformation* Δ^g, that is the transformation to be applied to the ideal global transformation $\mathcal{G}$ in order to obtain the aligned global transformation:

$$\mathcal{G}^a = \Delta^g\mathcal{G} = \Delta^g\mathcal{M}\mathcal{L}\,, \quad (15.5)$$

  or we use
- the *local delta-transformation* Δ^l, that is the transformation to be applied to the ideal local transformation $\mathcal{L}$ to get the aligned local transformation:

$$\mathcal{L}^a = \mathcal{L}\Delta^l\,. \quad (15.6)$$

Eqs. (15.5)–(15.6) allow one to rewrite:

$$\mathcal{G}^a = \mathcal{M}\mathcal{L}^a \quad (15.7)$$

as:

$$\Delta^g\mathcal{M}\mathcal{L} = \mathcal{M}\mathcal{L}\Delta^l \quad (15.8)$$

or equivalently:

$$\Delta^g = \mathcal{G}\Delta^l\mathcal{G}^{-1} \quad (15.9)$$
$$\Delta^l = \mathcal{G}^{-1}\Delta^g\mathcal{G} \quad (15.10)$$

to relate global and local alignment.

The alignment object stores as delta-transformation the global delta-transformation; nevertheless both global and local delta-transformations can be used to construct the alignment object or to set it. The reasons for this flexibility in the user interface is that the local RS is sometimes the most natural one for expressing the misalignment, as for example in the case of a volume rotated around its centre. However, the use of the local delta-transformation is sometimes error-prone; in fact, the user has to be aware that he is referring to the same local RS which is defined in the hard-coded geometry when positioning the given volume, whilst the local RS used by simulation or reconstruction code can in general be different. In case the alignment object is constructed or its delta-transformation is set by means of the local delta-transformation, the framework will then use Equation (15.9) to perform the conversion into global alignment constants.

As for the choice of storing a symbolic volume name instead of the volume path as volume identifier, we would also like to make the delta-transformation stored in the alignment objects independent from the geometry, thus keeping their validity unconstrained. This is possible if we store in the geometry itself a matrix for the ideal global transformation related to that volume (this possibility is offered by the class storing the link between symbolic volume names and volume paths, see Section 15.2).

15.1.3.2 Matrix or parameters for the delta-transformation

The global delta-transformation can be saved both

- as a `TGeoMatrix` and,
- as a set of six parameters, out of which three define the translation by means of the shifts in the three orthogonal directions, and three define the rotation by means of three Euler angles.

These two cases correspond to choosing one of the following two `AliAlignObj`- derived classes:

- `AliAlignObjMatrix`: stores a `TGeoHMatrix`
- `AliAlignObjAngles`: stores six double precision floating point numbers;

While storing the alignment constants in a different form, they appear with the same user interface, which allows the delta-transformation to be set both via the matrix and via the six parameters which identify it.

15.1.3.3 Choice for the Euler angles

A general rotation in three-dimensional Euclidean space can be decomposed into and represented by three successive rotations around the three orthogonal axes. The three angles characterizing the three rotations are called Euler angles; however, there are several conventions for the Euler angles, depending on the axes around which the rotations are carried out, right/left-handed systems, (counter)clockwise direction of rotation, order of the three rotations.

The convention chosen in the ALICE alignment framework for the Euler angles is the '*xyz convention*' (see Ref. [2]), also known as *pitch-roll-yaw* or *Tait–Bryan angles*, or *Cardano angles* convention. Following this convention, the general rotation is represented as a composition of a rotation around the z-axis (yaw) with a rotation around the y-axis (pitch) with a rotation around the x-axis (roll). There is an additional choice to fully specify the convention used,

since the angles have opposite sign whether we consider them bringing the original RS in coincidence with the aligned RS ('active-transformation' convention) or the other way round ('passive-transformation' convention). In order to maintain our representation fully consistent with the `TGeoRotation` methods we choose the 'active-transformation' convention, i.e., the opposite convention to the one chosen by the already referenced description of the pitch-roll-yaw angles [2].

To summarize, the three angles — ψ, θ, ϕ — used by the framework to represent the rotation part of the delta-transformation, unambiguously represent a rotation $\mathcal{A}$ as the composition of the following three rotations:

1. a rotation $\mathcal{D}$ by an angle ϕ (yaw) around the z-axis
$$\mathcal{D} = \begin{pmatrix} c_\phi & -s_\phi & 0 \\ s_\phi & c_\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
2. a rotation $\mathcal{C}$ by an angle θ (pitch) around the y-axis
$$\mathcal{C} = \begin{pmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{pmatrix}$$
3. a rotation $\mathcal{B}$ by an angle ψ (roll) around the x-axis
$$\mathcal{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_\psi & -s_\psi \\ 0 & s_\psi & c_\psi \end{pmatrix}$$

which leads to:

$$\mathcal{A} = \mathcal{BCD} =$$

$$\begin{pmatrix} c_\theta c_\phi & -c_\theta s_\phi & s_\theta \\ s_\psi s_\theta c_\phi + c_\psi s_\phi & -s_\psi s_\theta s_\phi + c_\psi c_\phi & -c_\theta s_\psi \\ -c_\psi s_\theta c_\phi + s_\psi s_\phi & c_\psi s_\theta s_\phi + s_\psi c_\phi & c_\theta c_\psi \end{pmatrix}$$

where c_i and s_i denote cos(i) and sin(i), respectively. In the case of alignment, the rotations to be applied are usually very small; in this case we can approximate the sins by the angles and the cosines by 1, and the final expression for the rotation matrix $\mathcal{A}$ is:

$$\mathcal{A} = \mathcal{BCD} = \begin{pmatrix} 1 & -\phi & \theta \\ \phi & 1 & -\psi \\ -\theta & \psi & 1 \end{pmatrix} .$$

15.2 Use of ROOT geometry functionality

The ALICE geometry is implemented via the ROOT geometrical modeler (often referred to as TGeo), a framework for building, browsing, navigating and visualising a detector's geometry, which is independent from the Monte Carlo transport (see Ref. [3] and the dedicated chapter in Ref. [4]). This choice allows the ALICE alignment framework to take advantage of using ROOT features such as its I/O, histogramming, browsing, GUI, etc. However, the main advantage of this choice is that the ALICE alignment framework can provide its specific functionality as a thin layer for managing the additional alignment information, built on top of already existing features for interfacing the geometry.
The ALICE alignment framework takes, in particular, advantage of the possibility:

- to save the geometry to a file and upload it from a file;
- to check the geometry for overlaps and extrusions exceeding a given threshold;
- to query the geometry for the global and local matrix of a given physical volume;
- to make a physical node out of a specific physical volume and change the local and global transformation associated to it, while keeping track of the original transformations;
- to store a hash table of links between symbolic volume names and volume paths which can be queried in an efficient way.

Concerning this last issue, the class representing the objects linking the symbolic volume names and the volume paths also provides the possibility of storing a transformation. This feature turns out to be very useful if it is used to store the matrix relating the RS stored in the geometry (global transformation matrix for that volume) with the RS used in simulation and reconstruction (the two things in general differ).

15.3 Application of the alignment objects to the geometry

The base class provides a method to apply the single alignment object to the geometry present in memory, loaded from file or constructed. This method accesses the geometry to change the position of the volume referred to by the unique volume identifier according to Eq. (15.5). However, this method alone cannot guarantee that the single object is applied correctly; the most common case is indeed the application of a set of alignment objects. In this case the framework has to check that the application of each object in the set does not invalidate the application of the others; when applying a set of alignment objects during a simulation or reconstruction run the framework transparently performs the following two checks:

1. In case of alignment objects referring to physical volumes on the same branch, they have to be applied starting from the one which refers to a volume at the uppermost level in the physical tree (container volume) down to the one at the

lowest level (contained volume). On the contrary, if the contained volume is displaced first, the subsequent displacement of the container volume would change its temporarily correct position;
2. In no case should two alignment objects be applied to the same physical volume separately.

The reason for the first limitation is in short that the position of the contained volumes depends on the position of the container volumes. The reason for the second limitation is that the delta-transformations are relative to the ideal global position of the given volume [see Eq. (15.5)], which then need not to have been previously modified by the application of an alignment object referring to the same volume. The tools used by the framework for checking that the two previous conditions are fulfilled are, respectively:

1. Sorting the alignment objects based on a method which compares the depth of the physical volume to which the given alignment object refers.
2. Combining more alignment objects referring to the same volume before applying them to the geometry.

During a simulation or reconstruction run the user can consistently apply the objects to the geometry, having the two checks described above transparently performed.

An additional check is performed during a simulation or reconstruction run to verify that the application of the alignment objects did not introduce big overlaps or extrusions which would invalidate the geometry (hiding some sensitive parts or changing the material budget during tracking). This check is done by means of the overlap checker provided by the ROOT geometry package; a default threshold below which overlaps and extrusions are accepted is fixed; the TGeo overlap checker favours speed (checks the whole ALICE geometry in a few seconds) at the expense of completeness, thus some rare overlap topologies can eventually escape the check.

15.4 Access to the Conditions Data Base

An important task of the ALICE alignment framework is to intermediate between the simulation and reconstruction jobs and the objects residing on the Offline Conditions Data Base (OCDB) [5], both for defining a default behaviour and for managing specific use cases. The OCDB is filled with conditions (calibration and alignment) objects; the alignment objects in the OCDB are currently created by macros to reproduce two possible misalignment scenarios: the initial misalignment, according to expected deviations from the ideal geometry just after the sub-detectors are positioned, and the residual misalignment, trying to reproduce the deviations which can not be resolved by the alignment procedures. The next step is to fill the OCDB with the alignment objects produced from the survey procedures, as soon as survey data are available to the offline. Finally, these objects and those produced by alignment procedures will fill the OCDB to be used by the reconstruction of the real data in its different passes.

The OCDB stores the conditions, making use of the database capabilities of a file system three-level directory structure; the run and the version are stored in the file name. If not otherwise specified, the OCDB returns the last version of the required object and should an object be uploaded, it is automatically saved with increased version number.

The ALICE alignment framework defines a specific default storage from which to load the alignment objects for all the sub-detectors; the user can set a different storage, either residing locally or on the Grid if he has the permissions to access it. The definition of a non-default storage for the OCDB, as well as its deactivation, can also be given for specific sub-detectors only. The user can also just switch off the loading of alignment objects from an OCDB storage, or as a side-effect of passing to the simulation, or reconstruction run an array of alignment objects available in memory.

15.5 Summary

The ALICE alignment framework, based on the ROOT geometry package [3, 4], aims at allowing a consistent and flexible management of the alignment information, whilst leaving the related complexity as much as possible hidden to the user. The framework allows one to:

- save and retrieve the alignment constants relative to a specific alignable volume (automatic retrieval from a Conditions Data Base is handled);
- apply the alignment objects to the current (ideal) geometry;
- obtain from the current geometry the alignment object for a specified alignable volume;
- transform positions in the ideal global RS into positions in the aligned global RS;
- set the objects by means of both global and local delta-transformations.

These functionalities are built on the `AliAlignObj` base class and its two derived classes, which store the delta-transformation by means of the transformation matrix (`AliAlignObjMatrix`) or by means of the six transformation parameters (`AliAlignObjAngles`). The user interface is the same in both cases; it fixes the representation of the delta-transformation whilst leaving several choices to the user which have been explained in this note together with their implementation.

The ALICE alignment framework fixes the following conventions:

- the transformations are interpreted according to the local-to-global convention;
- the delta-transformation stored is the global delta-transformation;
- the three parameters to specify the rotation are the roll-pitch-yaw Euler angles, with the "active-transformations" convention.

The framework also fixes the following default behaviours in simulation and reconstruction runs:

- objects are loaded from a default Conditions Data Base storage, on a sub-detector basis;
- the set of loaded objects is sorted for ensuring the consistency of its application to the geometry;
- the ideal and aligned geometries are saved.

Several choices related to the delta-transformation are left to the user, who:

- can choose to set the alignment object either by passing a `TGeoMatrix` or by giving the six parameters which uniquely identify the global delta-transformation;
- can choose if they want the object to store either the `TGeoMatrix`, using an `AliAlignObjMatrix` or the six parameters, using an `AliAlignObjAngles`;
- can choose if the transformation to be passed is the global delta-transformation or the local delta-transformation; in this latter case the framework converts it to the global one to set the internal data members.

Acknowledgements

I wish to thank Cvetan Cheshkov and Andrei Gheata for the opportunity of working together on the present topic.

References

[1] C. Cheshkov, A. Gheata and R. Grosso, ALICE Internal Note, in preparation.

[2] `http://mathworld.wolfram.com/EulerAngles.html` .

[3] R. Brun, A. Gheata and M. Gheata, *Nucl. Instrum. Methods* **A502** (2003) 676–680.

[4] `http://root.cern.ch/root/doc/RootDoc.html` .

[5] `http://aliceinfo.cern.ch/Offline/Activities/ConditionDB.html` .