LHCb-2005-101
December 12th, 2005.

LHCb VELO software alignment

Part I: the alignment of the VELO modules in their half boxes

S. Viret, C. Parkes, D. Petrie

University of Glasgow

Abstract

Software alignment of the Vertex Locator (VELO) is a critical stage of the LHCb alignment strategy. This note presents a demonstration of a potential algorithm to satisfy the requirements of this stage. A fast track-based software alignment procedure is described. This method is based on non-iterative least squares fitting. The first step of the algorithm, the alignment of the VELO modules in their half-boxes, is fully described and results obtained with simulated events are presented and discussed. The approach described in this document, and the tools developed, are also applicable to the alignment of the other LHCb sub-systems and the global relative alignment of the sub-detectors.

Document Status Sheet

1. Document Title: A procedure for LHCb VELO online alignment			
2. Document Reference Number: LHCb-2005-101			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
Draft	1	December 5, 2005	First version of the internal alignment box.

Contents

List of Figures

List of Tables

1 Introduction

This note describes a fast track-based software alignment procedure to align the LHCb vertex locator (VELO) modules into their respective RF boxes. This method is based on a non-iterative least squares fitting method which utilizes a C++ implementation[1] of the 'matrix-crushing' algorithm Millepede **[1]**. An overview of the stages in the alignment procedure for the VELO detector are presented in section 2 alongside with the precision expected from the system metrology measurements and construction precision. The choice of a non-iterative alignment method for the track-based software alignment of the VELO is motivated and an introduction to the Millepede package provided in section 3. The proposed alignment method, and the results obtained thus far, are then described in detail in sections 4 and 5. Perspectives, summary, and conclusions are given in section 6.

2 The VELO Alignment Context

2.1 VELO Geometry

The VELO consists of 23 stations placed along the z (beam) axis, parallel to the x-y plane. Two of these stations are primarily used for the L0 VETO trigger. Each station consists of two 'modules' which are separated and placed in opposite 'VELO-halves' allowing the detectors to be retracted during LHC beam injection. Each of the 42 standard modules consists of a pair of r and ϕ measuring sensor which are bonded to the same substrate. The four VETO modules consist of a single r-measuring sensor and the alignment of these is not considered here. Figures 1 and 2 describe how the modules are located within the VELO. Details of the VELO detector design can be found in **[3, 4]**.

2.2 Proposal for the VELO Alignment Procedure

The alignment of the LHCb VELO will proceed in a number of stages, which might be divided as proposed below:

1. **Precision Mechanical Assembly:** The modules will be accurately assembled and mounted onto high precision base supports. The expected tolerances are given below in section 2.4.
2. **System Metrology:** The individual modules will be surveyed with a SmartScope and co-ordinate measuring machine at the time of construction. The mounted modules on the VELO-half box mechanical frames will be surveyed with a co-ordinate measuring machine. This survey will provide the initial estimate of the alignment constants of each half-box. The surveys will however be performed at room temperature and pressure, and at a standard humidity level.
3. **Software Alignment:** The software alignment of the VELO can be performed at several different stages of the data-taking and processing:
   - **On-line Alignment:** If necessary the alignment could be run after the insertion of the VELO half-boxes and before standard data-taking begins. This stage will only be necessary if the quality of the previous alignment () is not sufficient to ensure optimal performance of the displaced vertex trigger. It is expected from the high accuracy of the mechanical system that the internal alignment of the VELO modules within their half-boxes would not have to be performed at this stage. The relative alignment of the two half-boxes and the global alignment are expected to bereproducible up to 10 μm level due to mechanical system. However this has yet to be proven and conservatively we reserve this as an option and hence require a reasonably fast software alignment algorithm.
   - **Off-line Alignment for first data processing:** An off-line alignment of the VELO is anticipated to further improve the alignment constants. This alignment could be run easily within the period between data taking and offline data processing, as this period is expected to be significant (see section 4.4.2 of **[2]**). This alignment stage will be largely based on the work presented here.

[1] A standalone version of this code is available from *http://ppewww.ph.gla.ac.uk/LHCb/VeloAlign/VeloExample.html*

- **Off-line Alignment for data re-processing:** It is anticipated that the data will be re-processed at the end of each data-taking year. This re-processing stage offers a further opportunity to refine the alignment constants if necessary.

4. **Alignment Monitoring:** We envisage running a monitoring task, during data taking, that will monitor the quality of the track residuals obtained in the data. This monitoring job could trigger the re-running of the software alignment if required.

2.2.1 *Software Alignment Procedure*

The software alignment of the VELO has two stages.

1. **Internal VELO Alignment:** The internal alignment of the VELO is the topic covered in this note. this is the critical stage of the whole alignment procedure of LHCb. This stage includes the alignment of the VELO modules in their VELO-half boxes and the relative alignment of the two VELO-half boxes. This is the stage discussed in details in this note.

2. **Global Frame VELO Alignment:** The VELO system must then be aligned with respect to the rest of LHCb locating it in the LHCb frame. Although the VELO is the most precise tracking detector in LHCb since the VELO moves the module position cannot provide the LHCb frame reference point. The Global alignment is not discussed here but is a relatively simple task compared with the internal alignment of the VELO system. The techniques developed in this note for the internal VELO alignment, a system with 42 separate modules, can be readily adapted to this task of locating one geometric object within a reference frame.

If the alignment procedure is required to be performed online, then it would be envisaged to proceed as follows:

1. A set of tracks required for the VELO alignment is collected. The number of tracks required is expected to be of the order of 10,000, so this does not require a significant data taking period. Tracks residuals are then checked and alignment is run if significant discrepancies are observed. If not data taking could proceed directly.

2. The VELO internal and global alignment is performed and a new set of alignment constant is produced. This stage is required to take no more than a few minutes and hence not to significantly impact on the physics data taking time of the fill.

3. The new constants are tested with a set of tracks. A residual and track quality checking task using simple monitoring histograms is envisaged (but could be performed during data-taking period).

4. If the new alignment constants have been successfully verified the Conditions Database is updated with the new alignment constants and these are propagated throughout the on-line farm.

5. LHCb data taking then proceeds.

2.3 VELO systems Test - Alignment Challenge and Detector Calibration

The operation of the VELO-half boxes in an Alignment and Calibration Challenge (ACDC) is an integral part of our preparations for physics readiness of the VELO detector. All modules of the fully assembled VELO-half boxes will be readout during a beam test scheduled for summer 2006. Ten modules will be readout simultaneously, and permutations used which cycle through all the modules. This will allow a test of the alignment algorithms on data and will provide the first software internal alignment of the VELO-half boxes in close to final conditions. These alignment constants will be used as the starting reference point for alignment of the VELO system during initial LHC operation.

2.4 Construction and Assembly Precision

The expected mechanical movement and assembly misalignment scales are important for the alignment algorithm development as they define the range within which the track based software algorithm is required to function. They are related to the mechanical precision with which each component will be installed and assembled. The current estimates of these quantities have been used to define the scope of the simulated misalignments used in this document. If, for example, a module could be placed in a half-box with a $20\ \mu m$ accuracy, one could expect that misalignments will be distributed on a Gaussian centered on zero and with a $20\ \mu m$ width.

Table 1 shows the expected construction accuracies for the different subparts of the VELO, for the different possible degrees of freedom (3 rotations and 3 translations).

Component	**Degree of freedom**	**Mechanical estimate**
Relative r-ϕ sensor position in a module	x and y translations	$5\ \mu m$
Module to VELO-half	x, y, and z translations and rotations	$20\ \mu m$ for the translations, $0.5\ mrad$ for the rotations
VELO-half to VELO	x, y, and z translations and rotations	$50\ \mu m$ for the translations, $0.05\ mrad$ for X,Y rotations, $0.5\ mrad$ for Z
Other (sensor deformation, sagging,...)		see [5]

Table 1 *Alignment precision expected from the construction and assembly of the VELO* [6],[7]

In complement to this table, a $10\ \mu m$ reproducibility of the box translations within the VELO is expected [7].

These construction misalignments (with respect to an 'ideal' detector) will be surveyed with a high accuracy during a VELO mechanical survey during and after assembly of the full system. This mechanical survey, which is part of the alignment Challenge described previously, will provide the reference positions of all components. Hence, we can expect to have to deal with smaller values than the construction misalignments even at the very first alignment. However the system will be surveyed at room temperature and atmospheric conditions and, of course, each VELO half-box will move between each LHC fill. In the absence of other information at the current time for the studies presented here the values from Table 1 have been used to set the scale of the considered misalignments: in particular values were needed for the misalignment of modules within the moving boxes, and for the boxes 'mechanically-allowed' degrees of freedom: x and y translations.

2.5 Misalignment Studies

A detailed study of the effects of VELO misalignments on LHCb performance have been performed by the authors in [6]. This study described the effect of module and half-box misalignments. It has been decided to take the module, one r and one ϕ sensor bonded to the same substrate, as the smallest object to align. This choice is justified by the fact that r and ϕ sensors are assembled with respect to each other with high accuracy and that all misalignments, except individual sensor deformations, will affect the two sensors within the same module identically. While the sensor deformation on the final module is still to be evaluated, first studies [5] have shown that the effect of these should arise at second order only.

Study [6] shows that in most of the cases, the tolerable misalignment scale with the existing LHCb software algorithms is larger than the expected mechanical accuracy. The only issue concerns the half-VELO box tilts (their rotations around the x and y axis), these must be well controlled in order to ensure that the LEVEL1 trigger works correctly. However, if mechanical accuracies are as expected, it will mean that VELO will not have to be aligned after each fill. But once again, this point has still to be proven, as many parameters still remain unknown (for example, module movements during box translations).

Another important conclusion of this work **[6]** is that some degrees of freedoms can be neglected to a first approximation. This will be demonstrated quantitatively later in this note (see section 4.2) but for the module misalignments, for example, we could conclude from **[6]** that the LHCb performance is relatively insensitive to Z translations and rotations around the X and Y axes, hence these less important degrees of freedom have been neglected in the study reported here.

3 General Introduction to Alignment Methods

3.1 Introduction

Figure 3 illustrates the simplest alignment problem where the modules have only one degree of freedom - vertical displacements. If the track is reconstructed without taking into account the module misalignments the situation shown on the right plot is obtained: bad quality tracks. The alignment mission is to retrieve the real geometry using the available information, *i.e.* the reconstruction without alignment.

The VELO is not the first detector to be aligned using tracks so there are plenty of efficient alignment methods on the market. However, the VELO geometry of curved strips, the layout of modules along the beam and, not least, the possible requirement for a fast alignment at the start of each LHC fill mean that all existing algorithms will have to be significantly tailored to fulfill these unique circumstances.

3.2 Alignment by Tracks: Iterative versus Non-Iterative Methods

When using tracks, the standard way to obtain the alignment constants is to minimize the residuals. In figure 3 the residual is the distance between the red circle and the blue track intercept with the detector plane. In the VELO it is the distance between the reconstructed cluster (r or ϕ) position and the track intercept point in this sensor. In a perfect detector, without any resolution error, the residual will only depend on the misalignment. Hence, minimizing the residuals will allow us to retrieve the misalignment constants.

More quantitatively, the most general track equation is related to the track measurements by the following relation:

$$\boldsymbol{Y} = f(\boldsymbol{X}) + \boldsymbol{\epsilon}, \tag{1}$$

where $\boldsymbol{Y}$ is a vector[2] of the measurements on the track (eg r, ϕ clusters), $\boldsymbol{X}$ is a vector or matrix of the input parameters of the function f which defines the track, and $\boldsymbol{\epsilon}$ the vector of residuals. The most common way to minimize the residuals is by using the least squares method, this is equivalent to minimizing the following χ^2:

$$\chi^2 = \sum_i (\boldsymbol{Y}_i - f(\boldsymbol{X}_i)^T) V_i^{-1} (\boldsymbol{Y}_i - f(\boldsymbol{X}_i)) = \sum_i \boldsymbol{\epsilon}_i^T V_i^{-1} \boldsymbol{\epsilon}_i, \tag{2}$$

where we sum on all track measurements, and where V_i is the covariance matrix of track state $\boldsymbol{Y}_i$. Now, if $\boldsymbol{Y}$ has n coordinates, the problem is reduced to the solution of a system of n equations:

$$\left(\frac{\partial \chi^2}{\partial Y_k}\right)_{k \in [1,n]} = 0. \tag{3}$$

However, this system could be solved only if the problem is linearizable *i.e.* if one could express the track equation as follows:

$$f(\boldsymbol{X}) = X \cdot \boldsymbol{\alpha}, \tag{4}$$

[2]Throughout this note vectors are denoted in **bold**

where X is now a matrix containing the **local derivatives** of the tracks and $\boldsymbol{\alpha}$ is a vector containing the **local parameters** of the tracks.

In this case, and if we consider in addition that the different coordinates of $\boldsymbol{Y}$ are not correlated (V_i is then a diagonal matrix containing the errors on the coordinates), the solution to the problem is:

$$\boldsymbol{\alpha} = \left(\sum_i X_i^T \cdot V_i^{-1} \cdot X_i\right)^{-1} \cdot \sum_i X_i^T \cdot V_i^{-1} \cdot \boldsymbol{Y_i}. \tag{5}$$

¿From that point it is straightforward to get the residuals.

To illustrate this lets describe a simplified VELO example. A VELO track could be roughly defined as two straight lines:

$$\begin{cases} x_{measured} &= x_{track} + \epsilon_x &= a \cdot z + b + \epsilon_x \\ y_{measured} &= y_{track} + \epsilon_y &= c \cdot z + d + \epsilon_y \end{cases}, \tag{6}$$

so, in the VELO case, one has: $\boldsymbol{Y} = (x_{measured}, y_{measured})$, $\boldsymbol{\epsilon} = (\epsilon_x, \epsilon_y)$, $V = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$, $\boldsymbol{\alpha} = (a, b, c, d)$, and $X = \begin{pmatrix} z & 1 & 0 & 0 \\ 0 & 0 & z & 1 \end{pmatrix}$. In this case there are 4 local parameters, these are the track parameters which are the components of $\boldsymbol{\alpha}$, and 8 local derivatives, which are the components of X.

Looking back at equation (5), it becomes evident that changing the track measurement will modify the residuals. This is the basic principle exploited by **iterative minimization methods** (described on figure 4). Changing the measurement is equivalent to displacing the sensitive area, the idea is to move the module, then fit the tracks with the new measurements and analyze the new residuals. You iterate the method until you reach an acceptable solution, hopefully the minimal set of residuals.

This method is simple and in fact has already proven its efficiency in detector alignment on many occasions but it has a few disadvantages which are potentially important for the VELO.

The first disadvantage is that it may be time consuming, and consequently ineffective for fast online alignment. This can be illustrated by considering the example of the VELO testbeam alignment **[8]** which was performed using an iterative technique. The required time to align the six telescope sensors was about 10 1GHz Pentium CPU hours and 650 iterations were needed. In our case we will only have a few minutes to align 42 modules so the same method is not applicable on a single CPU. However, we do have the CPU power of a 1800 node processor farm available, so such an algorithm is not necessarily excluded by CPU considerations. Furthermore, the number of iterations could be significantly reduced, and consequently the computing time, by using more sophisticated iterative techniques. However in this case we loose the simplicity of the method and the result is an algorithm that is more difficult to setup and tune. Since we require a fast and robust alignment the simplicity of the method is a strong criterion for our alignment decision.

The other significant disadvantage with basic iterative methods is that they are *blind* in the sense that they ignore the relationship between the residuals and the misalignments. The track is biased by the measurement hits it contains but this information is not used in the procedure. The effect of this loss of information that is potentially most damaging is that outliers (incorrect hits) might be more easily propagated through the iterations then leading to biases in the final result. Once again, sophisticated techniques more or less manage to overcome that point but it becomes clear that the problem would be simplified by taking into account directly the relationship between the residuals and the misalignments.

Non-Iterative methods aim to take account of the relationship between the residuals and the misalignments . Instead of simply fitting the track, the track **and** the residuals are fitted simultaneously. To do so, you first need to find a linear relationship between the residual and the alignment constants (as was previously done for the local part of the tracks). The track equation is now given by:

$$\boldsymbol{Y} = X \cdot \boldsymbol{\alpha} + C \cdot \boldsymbol{\Delta}, \tag{7}$$

where $\boldsymbol{\Delta}$ contains the **global parameters**, *i.e.* the alignment constants we are looking for, and C is a matrix containing the **global derivatives**. The problem is then solved exactly as previously. The difference is that the solution now contains not only the local track parameters but also the global alignment parameters. It is not necessary to deal with the residuals directly and you get the track parameters and the misalignment constants in one step.

However, the problem with this method is that we have now correlated all the tracks. By definition, global parameters are common to all the tracks. So, by including them into the fit, the tracks are no longer independent. This means that it is necessary to fit all the tracks simultaneously. Previously to fit a track we had only to solve a system of n equations, where n was the number of local track parameters. Now, the final size of the system of equations is given by:

$$n_{total} = n_{local} \cdot n_{tracks} + n_{global}. \tag{8}$$

In order to get a good accuracy on the global alignment parameters you will need a significant number of tracks. However if we need, say, 10000 tracks, as we have 4 local parameters for each track (in the VELO but it could be far more), you finally get a system of around 40000 equations. The problem thus requires us to invert, in a reasonable time, a 40000x40000 matrix and it is this for which we use the Millepede algorithm.

3.3 Introduction to Millepede

Millepede was developed by Volker Blöbel for H1 alignment **[1]**. To understand how this algorithm works, consider the equation (7) in the simple case of a straight track:

$$y = \sum_j x_j \cdot \alpha_j + \sum_k c_k \cdot \Delta_k. \tag{9}$$

The χ^2 to minimize is given by the relation:

$$\chi^2 = \sum_i w_i \cdot (y^i - \sum_j x_j^i \cdot \alpha_j - \sum_k c_k^i \cdot \Delta_k)^2, \tag{10}$$

with $w_i = 1/\sigma_i^2$ (the measurements should be uncorrelated). As previously explained, all tracks are fitted simultaneously and we have the expression:

$$\chi^2 = \sum_{track,i} w_{track,i} \cdot (y^{track,i} - \sum_j x_j^{track,i} \cdot \alpha_j^{track} - \sum_k c_k^{track,i} \cdot \Delta_k)^2, \tag{11}$$

where $\sum_{track,i}$ stands as $\sum_{track}\sum_i$. Differentiating with respect to the track states allow us to derive the required system of equations and produces the following result (this is the equivalent of equation (5)):

$$\begin{pmatrix} \Delta_1 \\ \ddots \\ \Delta_{n_{global}} \\ \cdots\cdots\cdot \\ \alpha_1^1 \\ \ddots \\ \alpha_{n_{local}}^{n_{track}} \end{pmatrix} = \begin{pmatrix} V_{11} & \vdots & V_{12} \\ \cdots & \cdots & \cdots \\ V_{21} & \vdots & V_{22} \end{pmatrix}^{-1} \cdot \begin{pmatrix} \sum_{track,i} w_{track,i} \cdot c_1^i \cdot y^{track,i} \\ \ddots \\ \sum_{track,i} w_{track,i} \cdot c_{n_{global}}^i \cdot y^{track,i} \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \sum_i w_{1,i} \cdot x_1^{1,i} \cdot y^{1,i} \\ \ddots \\ \sum_i w_{n_{track},i} \cdot x_{n_{local}}^{n_{track},i} \cdot y^{n_{track},i} \end{pmatrix} \tag{12}$$

The four sub-matrices are given by:

$$\begin{cases} V_{11}(k,j) & = & \sum_{track,i} w_{track,i} \cdot c_k^i \cdot c_j^i & (k,j) \in [1, n_{global}] \times [1, n_{global}] \\ V_{22}(k + n \cdot n_{local}, j + n \cdot n_{local}) & = & \sum_i w_{n,i} \cdot x_k^{n,i} \cdot x_j^{n,i} & (n,k,j) \in [0, n_{track} - 1] \times [1, n_{local}] \times [1, n_{local}] \\ V_{22}(k,j) & = & 0 & otherwise \\ V_{12}(k,j) & = & \sum_{track,i} w_{track,i} \cdot c_k^i \cdot x_j^{track,i} & (k,j) \in [1, n_{global}] \times [1, n_{local} \cdot n_{track}] \\ V_{21}(k,j) & = & V_{12}(j,k) & \end{cases} \tag{13}$$

The main point to note here is that the largest part of the matrix is V_{22} and that this is only filled with symmetric sub-matrices of dimensions $n_{local} \times n_{local}$. In addition, those blocks are on the diagonal so that V_{22} is nearly empty and relatively simple to invert, this is the key to the problem. The full solution requires us to invert V, *i.e.* find the four sub-matrices A, B, C, and D such that:

$$\begin{pmatrix} V_{11} & \vdots & V_{12} \\ \cdots & \cdots & \cdots \\ V_{21} & \vdots & V_{22} \end{pmatrix} \cdot \begin{pmatrix} A & \vdots & B \\ \cdots & \cdots & \cdots \\ C & \vdots & D \end{pmatrix} = \mathbf{1} \tag{14}$$

However, in fact, we only need A and B, as this are the only parts required to find the alignment constants. Hence we have:

$$\begin{pmatrix} \Delta_1 \\ \ddots \\ \Delta_{n_{global}} \end{pmatrix} = A \cdot \begin{pmatrix} \sum_{track,i} w_{track,i} \cdot c_1^i \cdot y^{track,i} \\ \ddots \\ \sum_{track,i} w_{track,i} \cdot c_{n_{global}}^i \cdot y^{track,i} \end{pmatrix} + B \cdot \begin{pmatrix} \sum_i w_{1,i} \cdot x_1^{1,i} \cdot y^{1,i} \\ \ddots \\ \sum_i w_{n_{track},i} \cdot x_{n_{local}}^{n_{track},i} \cdot y^{n_{track},i} \end{pmatrix} \tag{15}$$

Inverting by blocks produces:

$$\begin{cases} A &= \left(V_{11} - V_{12} \cdot V_{22}^{-1} \cdot V_{12}^T\right)^{-1} \\ B &= -A \cdot V_{12} \cdot V_{22}^{-1} \end{cases} \tag{16}$$

So the final result is just given by multiplying matrix A by a vector, which is relatively easy to derive, as V_{22} is straightforward to invert. The alignment constants are thus obtained by inverting the matrix A of only $n_{global} \times n_{global}$ dimensions. The central work of the Millepede algorithm is the solving of equation (15).

Millepede manages the solution of this problem based on the user providing the track parameters and the global and local derivatives. The program then does the local fit of each track (V_{22} sub-part), and then updates the matrix A and the final vector, taking into account the global derivatives contributions, which appear in V_{11} and V_{12}. When the loop over all tracks is finished, Millepede performs the A matrix inversion, using the enhanced Gauss pivot method and then deduces the alignment constants, and multiplies it by the correct vector of parameters. If the matrix inversion required is still large we may consider the future use of alternative inversion techniques. A second loop is then performed in order to remove the outlier tracks and the program usually converges after this second loop.

Millepede thus provides a fast and elegant solution to our large matrix inversion problem. It also provides a whole set of methods in order to add constraints to the system. The imposition of constraints is critical to obtaining a robust solution and is discussed for the VELO case in section .

4 VELO Alignment Algorithm

4.1 Introduction

The VELO software alignment algorithm is able to align the modules within each of the two VELO-half boxes and also to align the two half-boxes themselves within the VELO reference frame. The two VELO half boxes will be retracted and reinserted between each LHC fill. However, the alignment with tracks cannot give us information about the absolute box positions in the LHCb frame. This requires a global alignment of the full VELO system with respect to the other sub-detectors, a comparatively simple procedure, and is not considered further in this note.

The VELO alignment procedure naturally divides into two distinct parts:

1. An internal alignment of the modules within each VELO-half box using the residuals on tracks.

2. A relative alignment of the two boxes with respect to each other using primary vertices, module overlaps, and tracks crossing both halves.

An algorithm to perform the first alignment stage is proposed and described in sub-section 4.3. The method is based on the relationship between the residuals and the alignment constants and the first sub-section explains the required equations, for both steps of the alignment procedure. The relative alignment of the two boxes will be discussed in a future note.

4.2 VELO Residuals and Global Parameters

As discussed in section 3.3, the requirement for our alignment procedure is to be able to derive a linear relationship between the residuals and the misalignment constants. Each module has six degrees of freedom and hence six alignment constants (denoted Δ_i) which position it relative to its VELO-half box. In addition, the position of the VELO half box is represented by a further six alignment constants which are denoted Δ_i^b.

In the following subsection, we first find the relations required for the alignment of the modules inside their VELO-half box. Then, in the second part section 4.2.2, we will extend our derivation to the relative alignment of the half-boxes in the VELO frame.

4.2.1 Position of Hits in the VELO Half-Box Frame

Consider a hit in a VELO module, this hit is defined by the coordinates $\boldsymbol{r^b_{hit}} = (x^b_{hit}, y^b_{hit}, z^b_{hit})$ in the VELO-half box frame. The VELO-half box frame is the reference frame for the VELO internal alignment. The same hit is expressed as $\boldsymbol{r_{hit}} = (x_{hit}, y_{hit}, z_{hit})$ in the local sensor frame, with $\boldsymbol{r_{hit}}$ defined as:

$$\boldsymbol{r_{hit}} = R \cdot (\boldsymbol{r^b_{hit}} - \boldsymbol{r^b_0}), \tag{17}$$

where R and $\boldsymbol{r^b_0}$ are respectively the rotation and the translation to get from the global to local frame. For example, if the sensor is perfectly aligned (by which we mean at its ideal position) in the box we would simply have:

$$\boldsymbol{r_{hit}} = \boldsymbol{r^b_{hit}} - \begin{pmatrix} 0 \\ 0 \\ z^b_0 \end{pmatrix}, \tag{18}$$

where z^b_0 is the sensor position on the z-axis. Then, if the sensor is misaligned, the same hit will have a different expression in the sensor's local frame. Let's call $\boldsymbol{r^{new}_{hit}}$ this value, the contribution to the residual due to misalignment is given by:

$$\boldsymbol{\epsilon} = \boldsymbol{r^{new}_{hit}} - \boldsymbol{r_{hit}}. \tag{19}$$

In the most general case, one has 6 degrees of freedom: 3 translations around the x, y, and z axes (called Δ_x, Δ_y, Δ_z respectively), and 3 rotations around the x, y, and z axes (called Δ_α, Δ_β, Δ_γ respectively). $\boldsymbol{r^{new}_{hit}}$ is given by the relation:

$$\boldsymbol{r^{new}_{hit}} = R_{\Delta_\gamma} R_{\Delta_\beta} R_{\Delta_\alpha} \cdot (\boldsymbol{r^{b,new}_{hit}} - (\boldsymbol{r^b_0} + \begin{pmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{pmatrix})) = \Delta_R \cdot (\boldsymbol{r^{b,new}_{hit}} - (\boldsymbol{r^b_0} + \boldsymbol{\Delta_r})). \tag{20}$$

Here we have $\boldsymbol{r^{b,new}_{hit}}$ and not $\boldsymbol{r^b_{hit}}$: since the sensor has moved in the global frame the hit in the global frame has also moved. Hence, we need to derive the new intercept point between the track and the displaced sensor.

$\boldsymbol{r^{b,new}_{hit}}$ and $\boldsymbol{r^b_{hit}}$ belong to the same VELO track, which could be defined by two straight lines in the (x, z) and (y, z) planes (see relation (6)). One thus has (using the notation introduced in relation (6)):

$$\boldsymbol{r^{b,new}_{hit}} = \boldsymbol{r^b_{hit}} + h \cdot \begin{pmatrix} a \\ c \\ 1 \end{pmatrix}, \tag{21}$$

where h is the parameter we wish to determine. By definition $\boldsymbol{r_{hit}^{b,new}}$ belongs to the displaced sensor, which means that its value in the displaced sensor frame, $\boldsymbol{r_{hit}^{new}}$, is orthogonal to the z-axis in that frame. Thus we have:

$$\boldsymbol{r_{hit}^{new}} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 0, \tag{22}$$

which is equivalent to:

$$\Delta_R \cdot (\boldsymbol{r_{hit}^{b,new}} - (\boldsymbol{r_0^b} + \boldsymbol{\Delta_r})) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 0. \tag{23}$$

Assuming that rotations are small we can use a simplified expression for Δ_R:

$$\Delta_R = \begin{pmatrix} 1 & \Delta_\gamma & \Delta_\beta \\ -\Delta_\gamma & 1 & \Delta_\alpha \\ -\Delta_\beta & -\Delta_\alpha & 1 \end{pmatrix}, \tag{24}$$

and to first order:

$$\Delta_R \cdot \boldsymbol{\Delta_r} \approx \begin{pmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{pmatrix}. \tag{25}$$

A straightforward computation leads to the h value:

$$h = \frac{\Delta_z + x_{hit}^b \cdot \Delta_\beta + y_{hit}^b \cdot \Delta_\alpha}{1 - a \cdot \Delta_\beta - b \cdot \Delta_\alpha} \approx \Delta_z + x_{hit}^b \cdot \Delta_\beta + y_{hit}^b \cdot \Delta_\alpha. \tag{26}$$

The last approximation is justified by a binomial expansion to first order. We now have $\boldsymbol{r_{hit}^{b,new}}$, the expression of $\boldsymbol{r_{hit}^{new}}$ is then easy to derive:

$$\boldsymbol{r_{hit}^{new}} = \Delta_R \cdot (\boldsymbol{r_{hit}^{b,new}} - (\boldsymbol{r_0^b} + \boldsymbol{\Delta_r})) = \Delta_R \cdot (\boldsymbol{r_{hit}} + h \cdot \begin{pmatrix} a \\ c \\ 1 \end{pmatrix}) + \Delta_R \cdot \boldsymbol{\Delta_r}, \tag{27}$$

then,

$$\boldsymbol{r_{hit}^{new}} = \Delta_R \cdot \begin{pmatrix} x_{hit} \\ y_{hit} \\ 0 \end{pmatrix} + h \cdot \Delta_R \cdot \begin{pmatrix} a \\ c \\ 1 \end{pmatrix} + \Delta_R \cdot \boldsymbol{\Delta_r}. \tag{28}$$

The z expression cancels, and two equations remains:

$$\left\{ \begin{array}{lcl} x_{hit}^{new} & = & x_{hit} - \Delta_x + y_{hit} \cdot \Delta_\gamma + a \cdot (\Delta_z + x_{hit} \cdot \Delta_\beta + y_{hit} \cdot \Delta_\alpha) \\ y_{hit}^{new} & = & y_{hit} - \Delta_y - x_{hit} \cdot \Delta_\gamma + c \cdot (\Delta_z + x_{hit} \cdot \Delta_\beta + y_{hit} \cdot \Delta_\alpha) \end{array} \right., \tag{29}$$

$\boldsymbol{r_{hit}^{new}}$ is the measurement you will observe and $\boldsymbol{r_{hit}}$ is the value corrected for the misalignments. The expression for the residuals as a function of the measured values is given by:

$$\left\{ \begin{array}{l} \epsilon_x = x_{hit}^{new} - x_{hit} = -\Delta_x + y_{hit}^{new} \cdot \Delta_\gamma + a \cdot (\Delta_z + x_{hit}^{new} \cdot \Delta_\beta + y_{hit}^{new} \cdot \Delta_\alpha) \\ \epsilon_y = y_{hit}^{new} - y_{hit} = -\Delta_y - x_{hit}^{new} \cdot \Delta_\gamma + c \cdot (\Delta_z + x_{hit}^{new} \cdot \Delta_\beta + y_{hit}^{new} \cdot \Delta_\alpha) \end{array} \right. . \tag{30}$$

This expression confirms the results of the VELO misalignment studies: the sensitivity to Δ_z, Δ_α, and Δ_β is proportional to the track slope which is by construction small in the VELO, hence these degrees of freedom are less important.

4.2.2 *Relative position of the half-boxes in the VELO frame*

Let's denote $\boldsymbol{r^V_{hit}}$ the hit coordinate in the VELO frame. This coordinate could be, for example, the primary vertex position. The connection with $\boldsymbol{r^b_{hit}}$ is straightforward:

$$\boldsymbol{r^b_{hit}} = R_b \cdot (\boldsymbol{r^V_{hit}} - \boldsymbol{r^V_0}). \tag{31}$$

If we consider that the VELO and box frames are identical, in perfect geometry we get:

$$\boldsymbol{r^b_{hit}} = \boldsymbol{r^V_{hit}}. \tag{32}$$

Adding box misalignments, we have:

$$\boldsymbol{r^{b,new}_{hit}} = \Delta_{R_b} \cdot (\boldsymbol{r^b_{hit}} - \boldsymbol{\Delta_{r_b}}) = \Delta_{R_b} \cdot (r^V_{hit} - \boldsymbol{\Delta_{r_b}}), \tag{33}$$

which leads to the following relations:

$$\left\{ \begin{array}{lcl} x^{b,new}_{hit} & = & x^V_{hit} - \Delta^b_x + y^V_{hit} \cdot \Delta^b_\gamma + z^V_{hit} \cdot \Delta^b_\beta \\ y^{b,new}_{hit} & = & y^V_{hit} - \Delta^b_y - x^V_{hit} \cdot \Delta^b_\gamma + z^V_{hit} \cdot \Delta^b_\alpha \\ z^{b,new}_{hit} & = & z^V_{hit} - \Delta^b_z - x^V_{hit} \cdot \Delta^b_\beta - y^V_{hit} \cdot \Delta^b_\alpha \end{array} \right. . \tag{34}$$

The expressions for the residuals in the VELO frame, as a function of the box misalignments and coordinates is thus given by:

$$\left\{ \begin{array}{lclcl} \epsilon^b_x & = & x^{b,new}_{hit} - x^b_{hit} & = & -\Delta^b_x + y^{b,new}_{hit} \cdot \Delta^b_\gamma - z^{b,new}_{hit} \cdot \Delta^b_\beta \\ \epsilon^b_y & = & y^{b,new}_{hit} - y^b_{hit} & = & -\Delta^b_y - x^{b,new}_{hit} \cdot \Delta^b_\gamma - z^{b,new}_{hit} \cdot \Delta^b_\alpha \\ \epsilon^b_z & = & z^{b,new}_{hit} - z^b_{hit} & = & -\Delta^b_z - x^{b,new}_{hit} \cdot \Delta^b_\beta - y^{b,new}_{hit} \cdot \Delta^b_\alpha \end{array} \right. . \tag{35}$$

These equations provide the expression for the global derivatives for the box alignment. The main difference with the case considered in the previous section comes from the fact that the z coordinates now have an important effect on the residuals.

4.3 Internal Alignment

4.3.1 *Track selection*

The selection of the tracks used in the alignment is important as a full population of the A matrix is required. An ideal track for alignment would cross all the VELO stations and have a sufficiently large slope. This situation is unfortunately not possible in our case and we need to find a compromise.

The first class of tracks to consider is the standard VELO physics tracks which originate principally from the beam spot situated in the center of the VELO. These tracks are angled but they do not cross all of the box. In addition, the modules close to the beam spot will be crossed only at a low radius in the high multiplicity area. We will see in the following discussion that in such a situation the performance of the algorithm will be degraded.

To overcome these problem, we also consider another complementary type of track: those due to beam halo particles[3]. Halo tracks have many advantages for our algorithm: they provide long tracks crossing the whole box; angled tracks are available in all stations; they provide a potentially easier way of studying the overlap region between modules in the left and right boxes. Unfortunately a pattern recognition algorithm for halo tracks is not currently available within the LHCb software. This

[3]We could also imagine to use classic tracks coming from primary vertices significantly displaced in Z.

point is, however, being addressed and a Gaudi implementation of this code should be available soon (see for example **[9]**). First tests using toy tracks produced with a standalone code show encouraging results (see for example **[10]**).

Concerning the standard physics tracks, it is clear that large misalignments are not supported by the basic pattern recognition, which is tuned, until now, for a perfect geometry. This fact is easy to observe with our algorithm, when running with a non-tune pattern recognition the result shown on fig.6 are obtained. Here, the performance of the alignment algorithm is inadequate, this is due to some of the misalignments being sufficiently large than the track hits in the displaced sensor have no chance to fall into the pattern recognition search windows. Hence, we require a pattern recognition algorithm that will take the misalignment into account and work in this area is currently being implemented by the VELO tracking group**[11]**. However, as this misalignment tolerant pattern algorithm is not currently available we rely here on our own private tune of the VELO pattern recognition with reasonably increased tolerances for the search windows.

This privately tuned pattern recognition is one of the reasons why the alignment algorithm is currently not very efficient in the modules close to the interaction point. Indeed, in this high multiplicity area, wrong hit pickup is non-negligible, and this is dramatically increased if, as we are doing, we add misalignments and increase the size of the pattern recognition corridors. A possible solution to this problem would be to add an isolation criteria to the pattern recognition: this was the approach adopted by CDF in their tracking used for alignment **[12]**. This solution has not yet been investigated as this problem is also likely to be significantly reduced by the use of halo tracks. However, a dedicated pattern recognition for use in alignment could be highly beneficial.

4.3.2 Feeding Millepede with (X,Y,Z) space-points

A VELO track consists of a set of (r, ϕ) clusters, and we rely on an r, ϕ cluster pair from a single module as our basic unit for alignment. The r and ϕ sensors are bonded to opposite sides of a substrate and hence are located at different z coordinates. The difference in z co-ordinate must be considered when mapping the cluster positions to an (x, y) coordinate. We thus have to transform (r, z_r) and (ϕ, z_ϕ) into an (x, y, z) point. Standard VELO physics tracks only are nearly linear in (r, z) so it is possible to project the ϕ cluster onto the R sensor since $\phi_{R_{sensor}} \approx \phi_{\phi_{sensor}}$.

However, as the track is never completely (r, z)-linear, a small correction, based on the track extrapolation in the sensors, is applied. Indeed, using those states, one could easily obtain the track ϕ_{track} values when crossing the sensors: $\phi_{track}(\phi_{sensor})$ and $\phi_{track}(r_{sensor})$. If the track is (r, z)-linear one should have $\phi_{track}(\phi_{sensor}) = \phi_{track}(r_{sensor})$. Correction for non-linearity is given by $\phi_{corr} = \phi_{track}(\phi_{sensor}) - \phi_{track}(r_{sensor})$. The ϕ value for r sensor is then easy to derive, as illustrated on fig.5. This correction becomes particularly useful in the case of halo tracks, which may be significantly non linear in (r, z).

4.3.3 Constraining the internal alignment within Millepede

The internal alignment of the VELO modules in their half-box is by definition insensitive to shifts of the position of the whole box. It is necessary to introduce a set of constraints into the alignment procedure to prevent these correlated module movements. This section describes the sort of constraints necessary and methods used to introduce these.

The simplest solution is to control the variation of the alignment constants. As we don't expect them to move much more than the mechanical accuracy, one could add a constraining term to the χ^2:

$$\chi^2_{const} = \chi^2 + \frac{\Delta_i^2}{\sigma_i^2}, \tag{36}$$

where Δ_i is the alignment parameter you want to control, and σ_i the maximal variation you expect for this parameter (currently these are taken from the estimates of the mechanical accuracy of the system). This constraint is straightforward to implement, as you just have to add $1/\sigma_i^2$ to the matrix element $A_{i,i}$. However, one could also fix more powerful constraints with Millepede, using relationships between the alignment constants.

Constraint equations are the best way to prevent global deformations of the system during the alignment. The only possible deformations, in our case, are linear transformations. Translations are the most straightforward example of them but there are four sort of linear deformations, which are summarized on fig.7. In three dimensions, this leads to 12 possible global deformations. However, due to the VELO station structure, we could neglect three of them at the first order: shearing in the XY plane, and scaling of the X and Y axis. In addition, due to the VELO geometry, it will be difficult to distinguish XZ and YZ shearings from Y and X rotations. Thus we choose to consider to constrain only the XZ and YZ shearings, as they are easier to add into our linear system.

Hence we need to constrain 7 possible deformations: Z axis rotation, X, Y, Z translations, XZ and YZ shearings, and Z axis scaling. The way to introduce these constraints in Millepede is explained in detail in **[13]**, consequently we will just briefly describe the basic principles here.

In each box one has to determine 21 alignments constants for each degree of freedom. Let's take the X translation example. Before the alignment, one has:

$$< \Delta_x >= \Delta_X, \tag{37}$$

where Δ_X is a global offset, which couldn't be found by an internal alignment. If we want to avoid global translations along the X axis during the alignment procedure, then we need to fix this constraint equation.

This can be performed by introducing a new parameter Δ'_x related to Δ_x by:

$$\Delta'_x = \Delta_x - \Delta_X. \tag{38}$$

By construction we then have:

$$< \Delta'_x >= 0, \tag{39}$$

which is an equation we could define without any problem...

Using this new formalism, known as the **canonical convention**, we are able to set the 7 constraint equations. One should notice here that the alignment constants we will get from the internal alignment process are the Δ'_i, *i.e.*the 'offset-free' constants. Clearly this is not a problem as the box offsets are only accessible via a box alignment procedure. The description of this procedure will be the subject of a future note.

4.4 Algorithm implementation within the GAUDI framework

As we require reconstructed tracks for our alignment the code described here is currently running within Brunel. The basic program flow is described on fig.9. It refers to the basic steps of a GAUDI algorithm: initialization, execution, and finalization. We will now describe each step in more details.

4.4.1 Initialization

The previous alignment conditions are retrieved during the initialize() method. Algorithm specific initializations are set externally via three job options files:

- **VeloAlign.opts** : contains the VELO geometry information, ie all the parameters defining the sensors to be aligned. It is possible to choose the sensors we want to align. The selection criteria for primary vertices are also set in VeloAlign.opts.
- **TrackStore.opts** : the tracks used for the alignment must satisfy specific criteria. These cuts (minimal number of hits, momentum cut, overlap cut...) are handled throughout the code into a TrackStoreConfig object, which is setup via this job options file.
- **Millepede.opts** : all the information necessary to run Millepede (constraint equations, numbers of DOFs, number of iterations...), are given here.

4.4.2 *Execution*

The execute() method is used for the track selection. This selection is handled by a GAUDI tool: TrackStore via the method TransformTrack(). This method takes a reconstructed track as its input. The output is an AlignTrack, a specific object containing all the necessary information for our alignment algorithm (Millepede ”friendly” coordinates, event number...). If the considered track satisfies the TrackStoreConfig cuts then it is stored in an AlignTrack container.

4.4.3 *Finalization*

The Millepede tool, a C++ translation of the Millepede algorithm, is instantiated here, via the InitMille() method. This initialization takes into account the geometry and configuration requests contained in VeloAlign.opts and Millepede.opts.

After the tool initialization, Millepede is fed with AlignTracks, via the method PutTrack(). This process is quite quick, as the AlignTrack is already in the expected format.

Finally, the global fit is performed via the method MakeGlobalFit(), which produces as its output the internal alignment constants.

This is the end of the internal alignment stage. We then need to find the boxes offsets. This part will be described in a future note, however the method used is quite similar to the one described for internal alignment.

Once the global alignment is performed we update the alignment conditions (of course we first have to check that the results are correct) this is the last stage of the alignment algorithm.

4.4.4 *CPU requirements*

Should the ’fill-to-fill’ software alignment of the VELO be necessary, then their will be strict requirements on the CPU performance. However this alignment will have at its disposal the power of the LHCb trigger farm of approximately 1800 processors. The only other use envisaged for these processors is running monitoring tasks prior to data taking. Hence, it is important to ensure that the alignment algorithm fits comfortably into the available CPU budget. The alignment algorithm adopted here is based on a linearization and matrix inversion by blocks and currently has rather modest CPU requirements. Figure 10 shows the time spent (normalized to a 1 GHz Pentium III) in the alignment algorithm presented in this note as a function of the number of events used. Due to the form of the algorithm adopted the time required by the matrix inversion is relatively small. The majority of the time is spent in populating the matrix and this time grows linearly with the number of tracks used. The matrix population could potentially be performed in parallel on a number of processors with the data combined only for the matrix inversion stage. Such an implementation has not yet been investigated given the small times obtained on even a single processor.

4.4.5 *Planned improvements for possible online utilization*

At this point, the code has not yet been optimized for speed of operation.

For example, the creation of a new Aligntrack object, even if it is really useful for the algorithm, could perhaps be avoided and all the track selection performed in the Finalize() method. Such modifications could reduce the time spent into the alignment algorithm and thus have to be evaluated.

But once again, as we said in section 2.2, the best way to test the code and to optimize it will be the 2006 alignment challenge.

5 Results

5.1 Principle of the study

The results of the internal alignment algorithm have been evaluated using simulation events and are reported in this section.

200 samples of 2000 minimum bias MC events have been produced and propagated through LHCb simulation packages. Each sample had a different set of alignment constants, which were introduced into the LHCb geometry using the recently developed LHCb Geometry Framework described in **[14]**.

The misalignment values have been randomly chosen within a Gaussian distribution centered on 0 and with the resolution σ_{scale}. The different scales are summarized in table 2. Module rotations and translations have been considered. In addition, the constraints equations on the box translations have been tested by introducing box misalignments.

Component	**Degree of freedom**	σ_{scale}
Module	Δ_x, Δ_y, Δ_z	30 μm
Module	Δ_α, Δ_β, Δ_γ	2 $mrad$

Table 2 *Misalignment scales for internal alignment studies*

As previously stated, halo tracks and dedicated pattern recognition weren't available at the time of this study. However, these features are useful mainly for modules which are close to the interaction point (mainly the stations 5 and 6).

The internal alignment algorithm has thus been tested between stations 7 and 20. It is anticipated that the same quality of result will be obtained on all stations using the correct track samples. Preliminary results using backward tracks (stations 0 to 4) are also presented.

5.2 Internal alignment

5.2.1 Effect on misalignments

Figures 11, 12, 13 show the misalignment constants before (full squares), and after (open squares) the step 1 application, for Δ_x translations, Δ_y translations, and Δ_γ rotations respectively. On each figure, the top plot shows the left box result, and the bottom plot the right box one.

The first conclusion is that a good correction of misalignments can be obtained with a relatively modest number of tracks. The results shown here have been obtained with about 15000 tracks for each box. Another interesting point is that relatively large misalignments are corrected without any problem. Rotational biases of a few mrads and translational biases larger than $80\mu m$ are indeed well reconstructed. Such misalignments are larger than the mechanical constraints on the modules positioning, so the algorithm performs over the required range of misalignments.

As expected, the sensitivity to other degrees of freedom is smaller. Hence, if we try to retrieve them the result is not relevant (*i.e.*misalignments constants are not well reconstructed). But in any cases they don't affect or bias the results obtained for the major degrees of freedom. So this is not a concern, at least at the first order.

The robustness of the method is tested in figures 14 and 15. These show the results obtained with the 200 sets of events, for both translations and rotations. The results shown above for a single example data set are confirmed here. The small discrepancies in the rotation plot come from station close to the interaction point (mainly 7 and 8). This effect is expected to disappear using the final track sample for the alignment. Apart from that, all the relevant misalignments are well corrected, even those with a relatively large scale.

A fit of the corrected alignment constants is shown on fig.16. Translational misalignments are corrected to a $2.8\mu m$ accuracy (better than the best possible VELO resolution), whereas a $0.4\ mrad$ precision is obtained for rotation around the z axis. The resolution for the rotations is not as good as for the translations but is already very acceptable as it is a the level of $1/6^{th}$ of a ϕ outer strip. Furthermore, this result is mainly degraded by station 7 and 8, and it is expected to be slightly improved using halo tracks.

5.2.2 Backward areas

An important feature of our basic track selection is that no momentum cut is requested *a priori*, so that the algorithm should work on backward tracks too. This is shown on figures 17 and 18 using the same events samples and applying the algorithm between station 0 and 4 (station 5 is skipped for the same reasons as station 6).

The results concerning the translation parameters are as good as for forward tracks, but the rotational misalignment distributions are a little bit more distorted. This torsion is expected from the canonical convention used for applying constraints **[13]**, and is larger when the number of considered stations becomes small. This could be corrected using a larger amount of tracks, but this is not expected to be a problem with 21 modules.

5.2.3 Effect on residuals

The first objective of the internal alignment is to improve the track reconstruction. The best way to check that is to compare the track residuals before and after the alignment procedure.

One has:

$$\left\{ \begin{array}{lcl} \epsilon_x & = & x_{measured} - x_{track} \\ \epsilon_y & = & y_{measured} - y_{track} \end{array} \right. . \tag{40}$$

In the following, as our detector has an (R, ϕ) geometry, we will present residuals values in R and ϕ directions:

$$\left\{ \begin{array}{lcl} \epsilon_R & = & \sqrt{(x_{measured} + \epsilon_x)^2 + (y_{measured} + \epsilon_y)^2} - \sqrt{x^2_{measured} + y^2_{measured}} \\ \epsilon_\phi & = & atan(\frac{y_{measured}+\epsilon_y}{x_{measured}+\epsilon_x}) - atan(\frac{y_{measured}}{x_{measured}}) \end{array} \right. . \tag{41}$$

Residual values, as a function of R and ϕ are shown on fig. 20 and 21, A global view, for X and Y residual is presented on figure 19. We see that in all cases the mean value of the residuals is centered on zero after the alignment procedure and that the RMS of the distribution is also slightly improved.

The pulls are computed as follows:

$$Pull_i = \frac{\epsilon_i}{\sqrt{\sigma^2_{measured,i} - \sigma^2_{track,i}}} \tag{42}$$

The - sign in the pull definition comes from the fact that the measurement is taken into account in the fit. $\sigma_{track,i}$ is the fitted track error on the i^{th} coordinate. This is obtained from the track parameters by error propagation. Values of $Pull_x$ and $Pull_y$ are presented on figure 22. They are fitted using the sum of two Gaussians. The principal Gaussian has a width of 0.8, which is slightly lower than the expected value of 1. It means that measurement error is slightly over- estimated, which is the case as we are using the value $Pitch_{strip}/\sqrt{12}$ for our measurement error. This point will be improved soon with the utilization of a correctly tuned VELO simulation**[15]**.

5.2.4 Effect of track statistics

We expect the alignment result to be improved as a function of the number of tracks. Results obtained in section 5.2 were obtained with 2000 events per job, which corresponds to about 15000 tracks. The same jobs were re-processed but taking 500, 5000, and 10000 events per job. Fits of the final corrected misalignment distribution, as shown on figure 16, were performed. The result of those fits is summarized on figure 23 where resolution as a function of the event statistics is displayed in the case of Z rotation misalignment. We clearly observe the expected improvement, as a function of the statistics.

5.2.5 *Crosscheck of Constraints*

As previously discussed, the track residuals within a box are not sensitive to misalignments of the box, and constraints applied during the internal alignment should prevent any box movement. A clear check of that point is shown on figure 8, where the internal VELO module alignment is run with the box misaligned. The modules are perfectly aligned before the alignment procedure is performed. These plots show the effect of the internal alignment on 30 sets of box misaligned data (1000 events each). On the left are the generated internal misalignment constants, so only zeroes, and on the right the ones found by our internal alignment algorithm. As expected the algorithm doesn't reconstruct any large misalignment. Small distortions arise from the fact that we are not yet using halo tracks, so that the final matrix is not perfectly conditioned. Moreover, these distortions are statistic dependent: the larger the number of tracks, the smaller are the reconstructed misalignments.

6 Conclusion

The alignment of the LHCb VELO proceeds in three stages: a precision mechanical assembly; an accurate metrology of the components and the assembled system; and a software alignment using tracks collected during data taking. The Alignment Challenge and Detector Calibration where the full velo-halves are operated in a test-beam and the first estimate of the alignment constants obtained is also a critical element of the programme. The software alignment has been developed in order to be able to run on a 'fill-to-fill' basis if necessary. The outcome of the alignment will already be physics analysis quality alignment constants. Thus the algorithm could be largely used as a basis for the alignment which will be run on off-line data. This note has described critical elements of the software alignment stage, which has been implemented in the Gaudi framework and studied with simulation events.

As previously stated, the alignment might have to be performed after the insertion of the VELO at the start of an LHC fill and before main LHCb data taking can commence. The alignment will then be performed on a timescale of a few minutes using the CPU power of the CPU trigger processor farm. CPU consumption results have been presented that demonstrate that a single 1GHz processor can produce an effective internal alignment of the VELO on 10,000 tracks in approximately 10 minutes. The processing is readily divisible between CPUs, so the CPU performance of the current algorithm is found to be suitable for this task. This method contrasts with typical iterative techniques of alignment where the CPU alignment time commonly runs into hundreds of CPU hours.

The alignment requires three stages: the internal alignment of the VELO modules in their VELO-half boxes, the relative alignment of the VELO-half boxes and the global alignment of the VELO in the LHCb frame. The relative alignment is the subject of current work and will be discussed in a future note. The global alignment is a comparatively simple task, compared with the other two, and is not considered in this note. A C++ version of the existing FORTRAN 'matrix-crushing' algorithm Millepede has been developed for use as a tool in this alignment and will also be applicable elsewhere in LHCb.

A detailed algorithm to perform the internal alignment of the modules in the VELO-half boxes has been successfully demonstrated in this note. The algorithm relies upon a non-iterative matrix inversion technique. The approach is based on a least squares minimization of track residuals. The technique accounts for the connections between the local track and global alignment constants by simultaneously fitting both (local and global) sets of constants. The technique utilizes tracks from LHC proton-proton collisions and tracks from beam halo particles that traverse the whole detector. Some limitations on the current simulation event tests have been imposed by the fact that beam halo tracking is not yet available in the LHCb software framework. However, the results obtained on forward stations of the VELO show that a $3\ \mu m$ precision (to be compared with the intrinsic hit resolution of the VELO of $> 5\mu m$) were obtained on the relevant translational degrees of freedom (*i.e.*along X and Y axis), and a $0.4\ mrad$ accuracy on rotation DOFs (*i.e.*around Z axis). Further improvements are anticipated using Halo tracks and, if necessary, a dedicated pattern recognition for alignment. The results has been cross-checked by studying the residuals on tracks and the expected improvement demonstrated.

To conclude, this note has demonstrated the feasibility of performing the first critical stage of the VELO software alignment. The results obtained on the internal half-box alignment, notably on the translational degrees of freedom of the modules, fulfill the requirements of LHCb. The CPU requirements of this algorithm are suitable for operation in the LHCb environment. This note sets the strategy for the LHCb VELO online alignment and the group consider this work is on track to provide the required VELO alignment functionality.

7 References

[1] V. Blobel and C. Kleinwort - *A new method for the high-precision alignment of track detectors* - hep-ex/0208021

[2] LHCb Collaboration - *LHCb Computing TDR* - CERN/LHCC 2005-019

[3] LHCb Collaboration - *LHCb VELO technical design report* - LHCb TDR 5 - CERN-LHCC 2001-010

[4] LHCb Collaboration - *LHCb reoptimized detector technical design report* - LHCb TDR 9 - CERN-LHCC 2003-030

[5] C. Parkes, TJV. Bowcock, P. Collins, K. Osterberg - *A study of the curvature of VELO prototype sensors* - LHCb-2001-110

[6] D. Petrie, C. Parkes, S. Viret - *Study of the impact of VELO misalignments on the LHCb tracking and L1 trigger performance* - LHCb-2005-056

[7] H.J. Bulten - *Mechanical Tolerance & Beam Position Information for Velo Alignment* - http://agenda.cern.ch/fullAgenda.php?ida=a054721#s1

[8] I. Tomalin - *Alignment of the 1998 VELO testbeam data* - LHCb-99-032

[9] T. Lastovicka - *VELO Halo/TB Particle Tracking (and Triggering)* - http://agenda.cern.ch/fullAgenda.php?ida=a054721#s1

[10] *VELO alignment with halo tracks* - http://ppewww.ph.gla.ac.uk/LHCb/VeloAlign/VeloApplication_3.html

[11] D. Hutchcroft - *VELO Tracking and Pattern Recognition for Trigger and Offline with Misalignments* - http://agenda.cern.ch/fullAgenda.php?ida=a054721#s1

[12] R. McNulty, T. Shears, A. Skiba - *A procedure for the software alignment of the CDF silicon system* - CDF/DOC/TRACKING/GROUP/5700

[13] R. Mankel - *A 'canonical' procedure to fix external degrees of freedom in the internal alignment of a tracking system* - HERA-b 99-087

[14] J. Palacios - *LHCb Geometry Framework* - https://uimon.cern.ch/twiki/bin/view/LHCb/GeometryFramework

[15] T. Szumlak - *VELO cluster resolution/eta correction* - http://agenda.cern.ch/fullAgenda.php?ida=a054721#s1

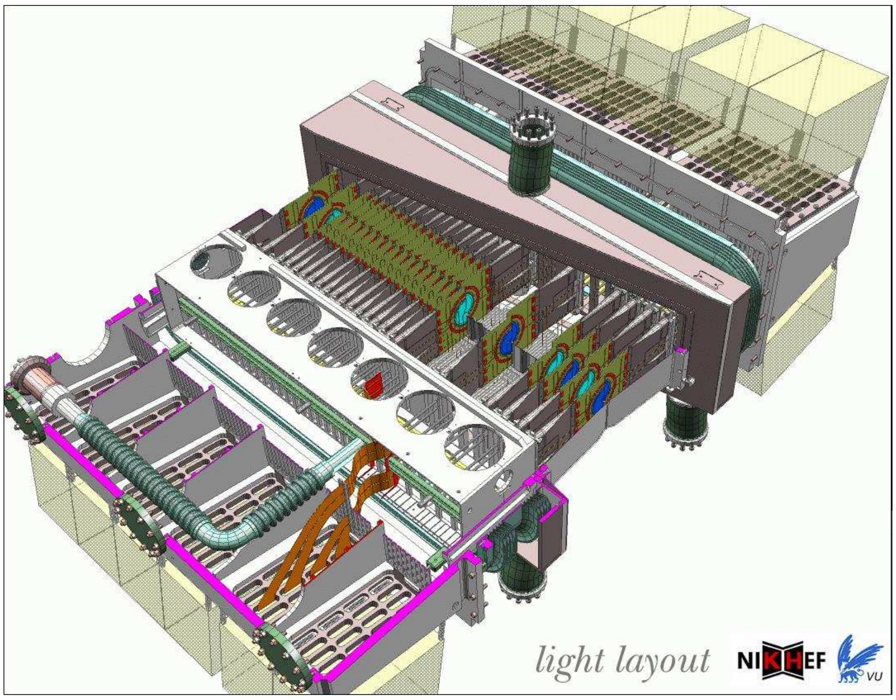


Figure 1 *General view of the VELO, the modules are visible within the two boxes. The sensors are shown in blue.*

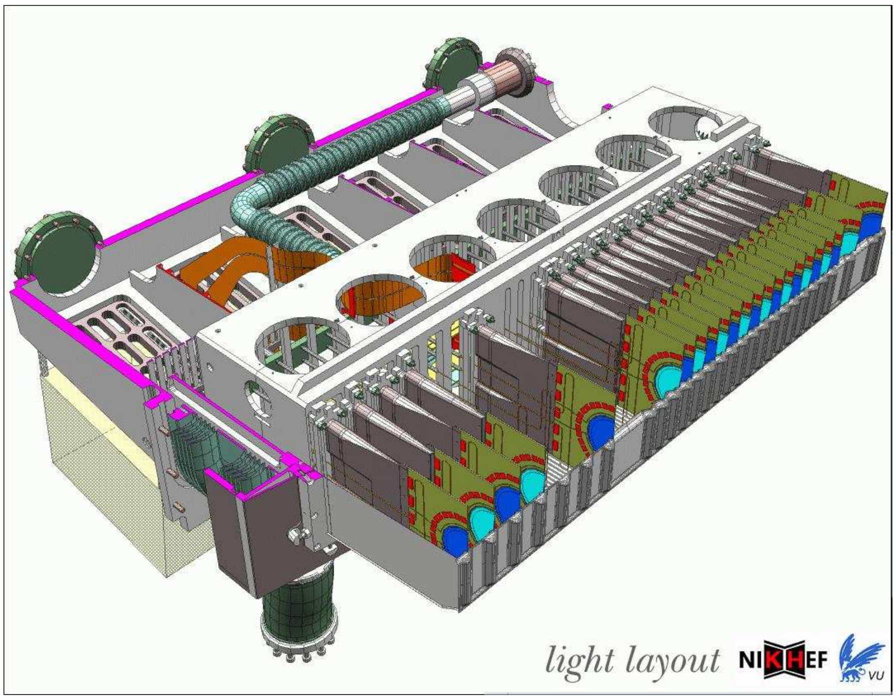


Figure 2 *Detailed view of one VELO box.*

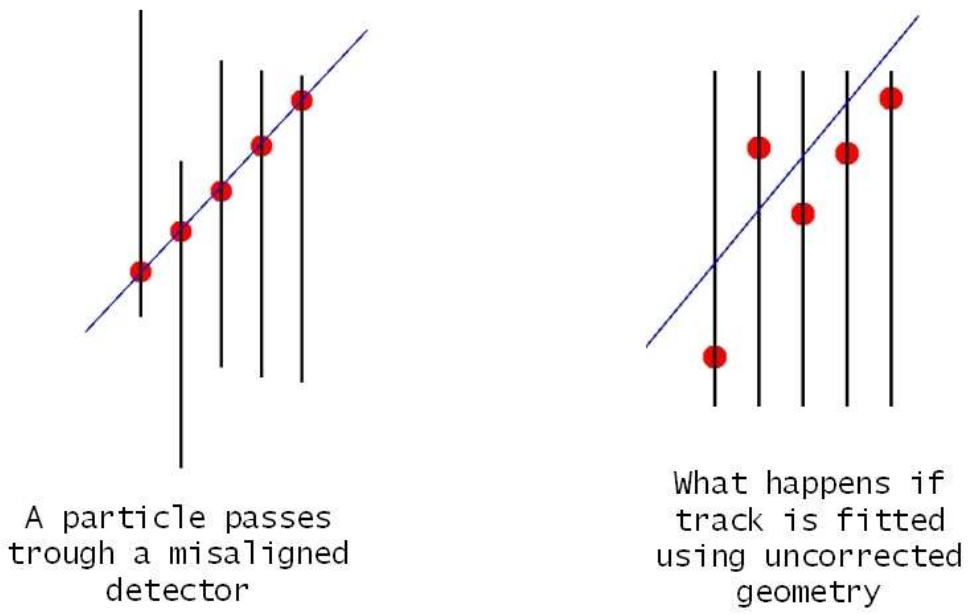

A particle passes trough a misaligned detector
What happens if track is fitted using uncorrected geometry

Figure 3 *The basic alignment problem. The correct module geometry (left) must be retrieved by studying the residuals of the hits (circles) with respect to the tracks.*

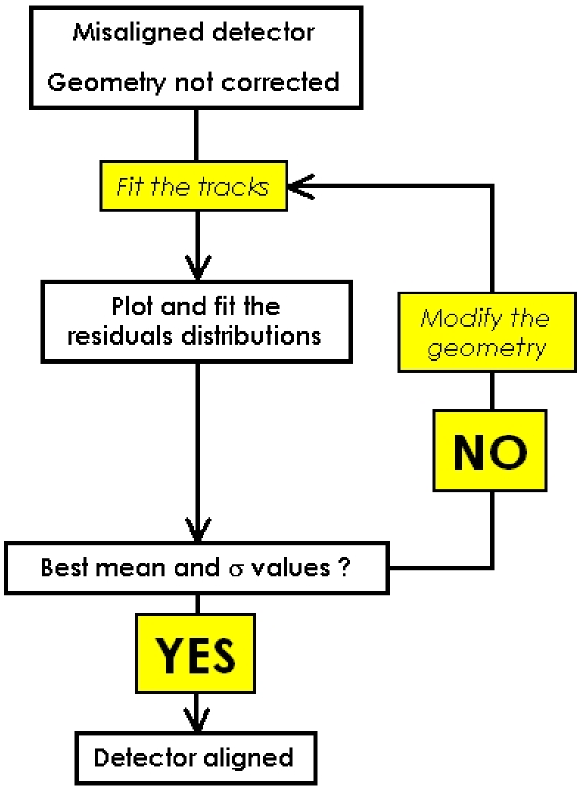


Figure 4 *The basic principle of an iterative alignment technique.*

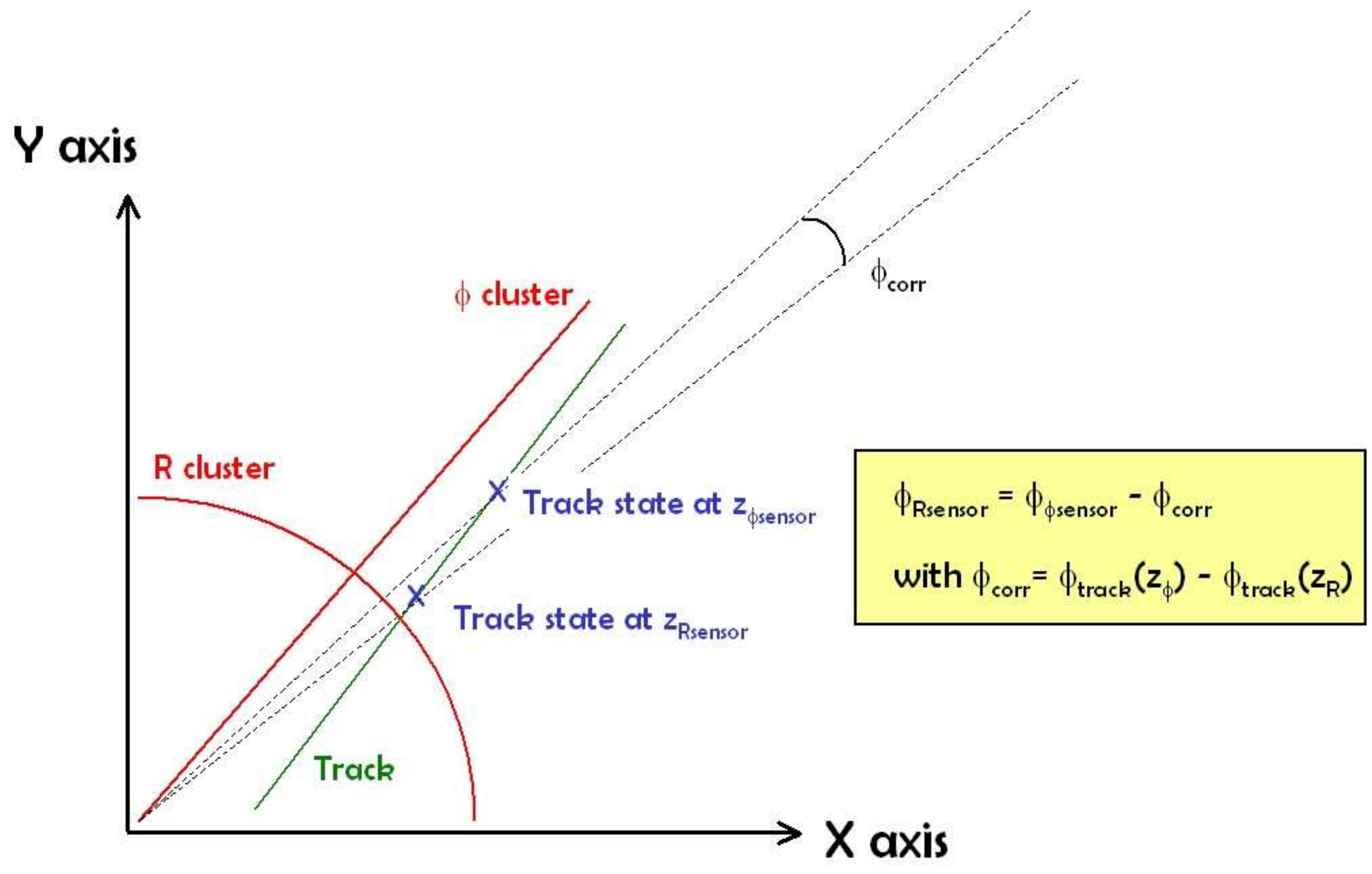

Y axis
X axis
ϕ cluster
R cluster
Track
Track state at $z_{\phi sensor}$
Track state at $z_{Rsensor}$
ϕ_{corr}
$\phi_{Rsensor} = \phi_{\phi sensor} - \phi_{corr}$
with $\phi_{corr} = \phi_{track}(z_\phi) - \phi_{track}(z_R)$

Figure 5 *The diagram illustrates the ϕ correction required to account for non (r, z) linear tracks intercepting the r and ϕ sensors in a module (see text for details).*

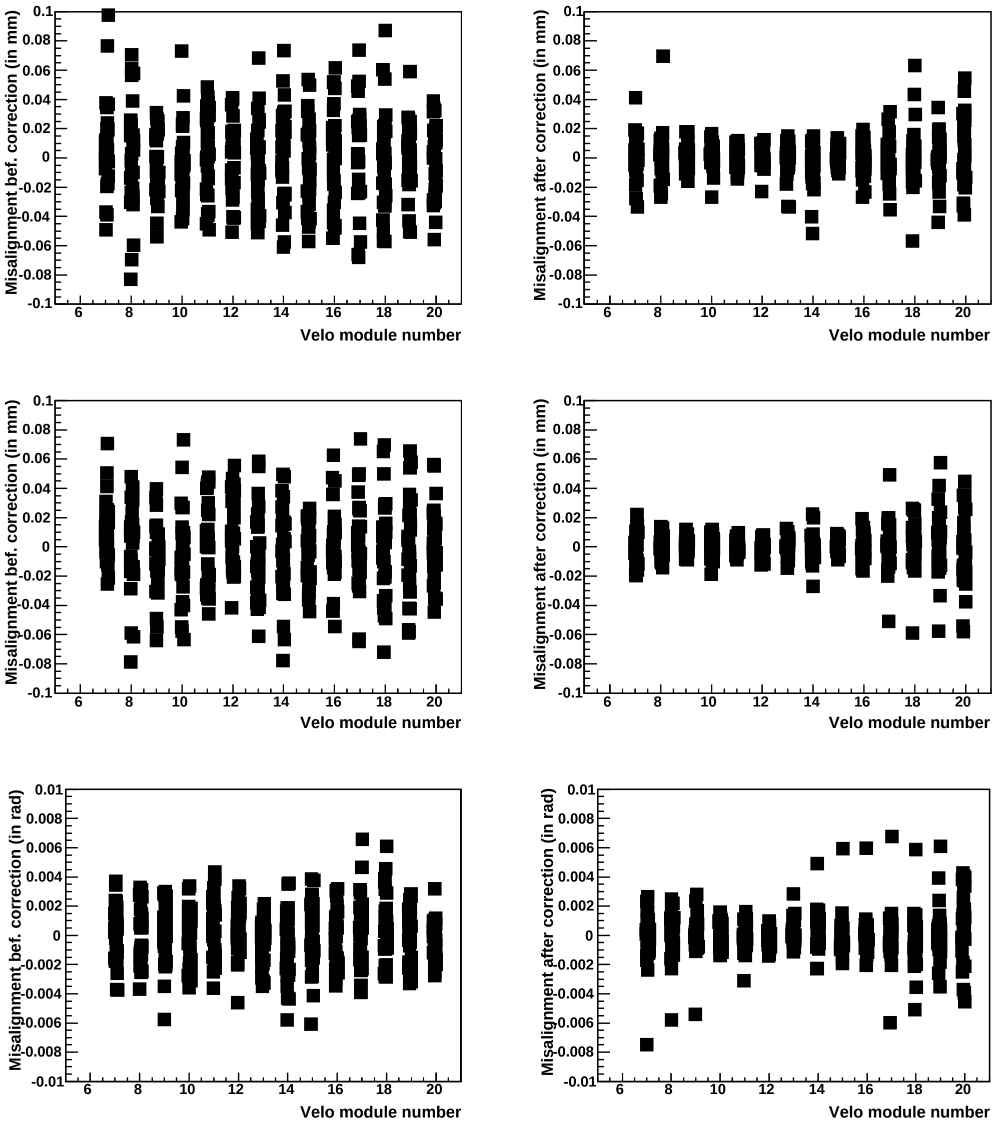


Figure 6 *The results of performing internal alignment using the default non-tuned pattern recognition algorithm. For each station each point correspond to a different set of misalignments. Left plots shows the misalignments constants for X translations (top), Y translations (middle), and Z rotations (bottom) before alignment. Right plots show the same constants after correction.*

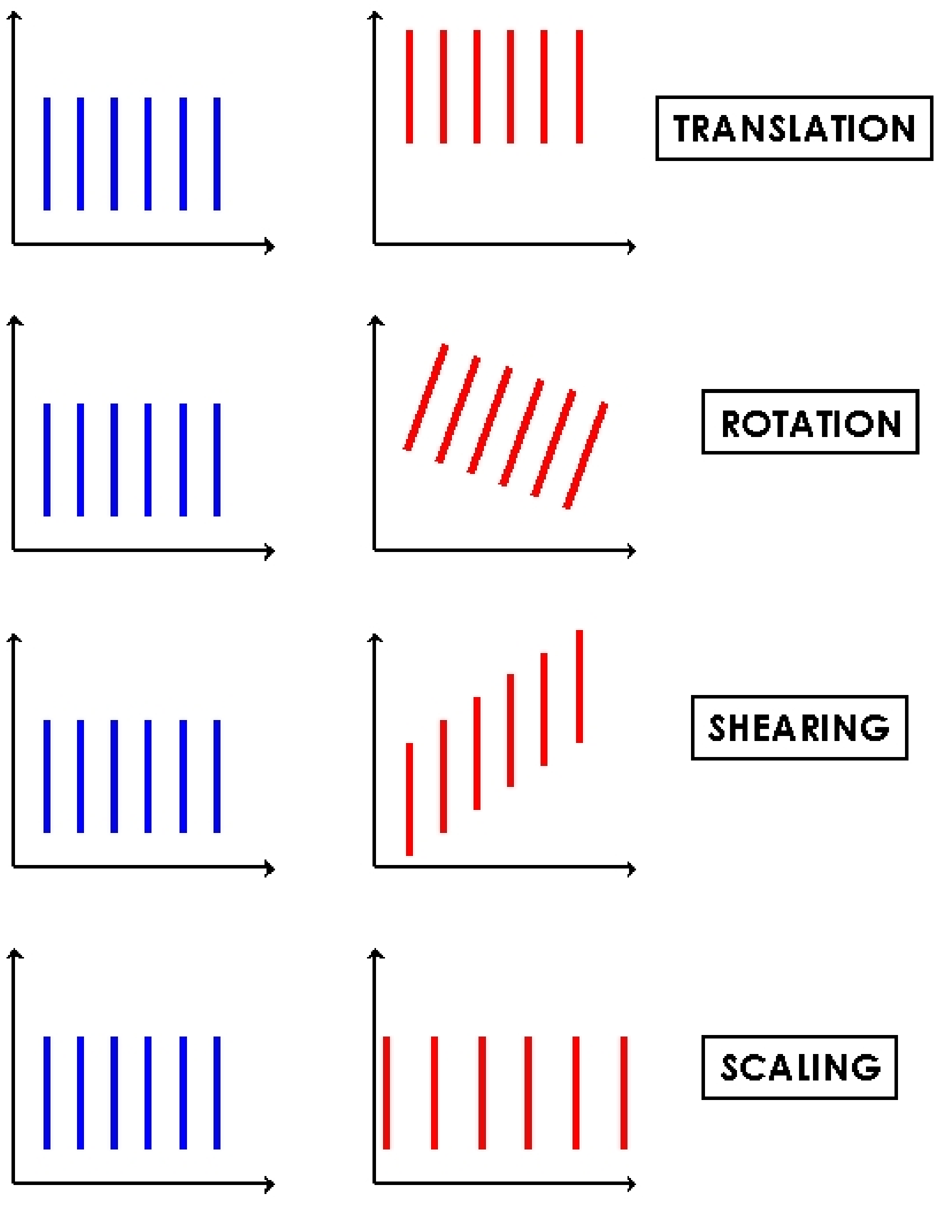


Figure 7 *The four basic types of linear transformations are illustrated.*

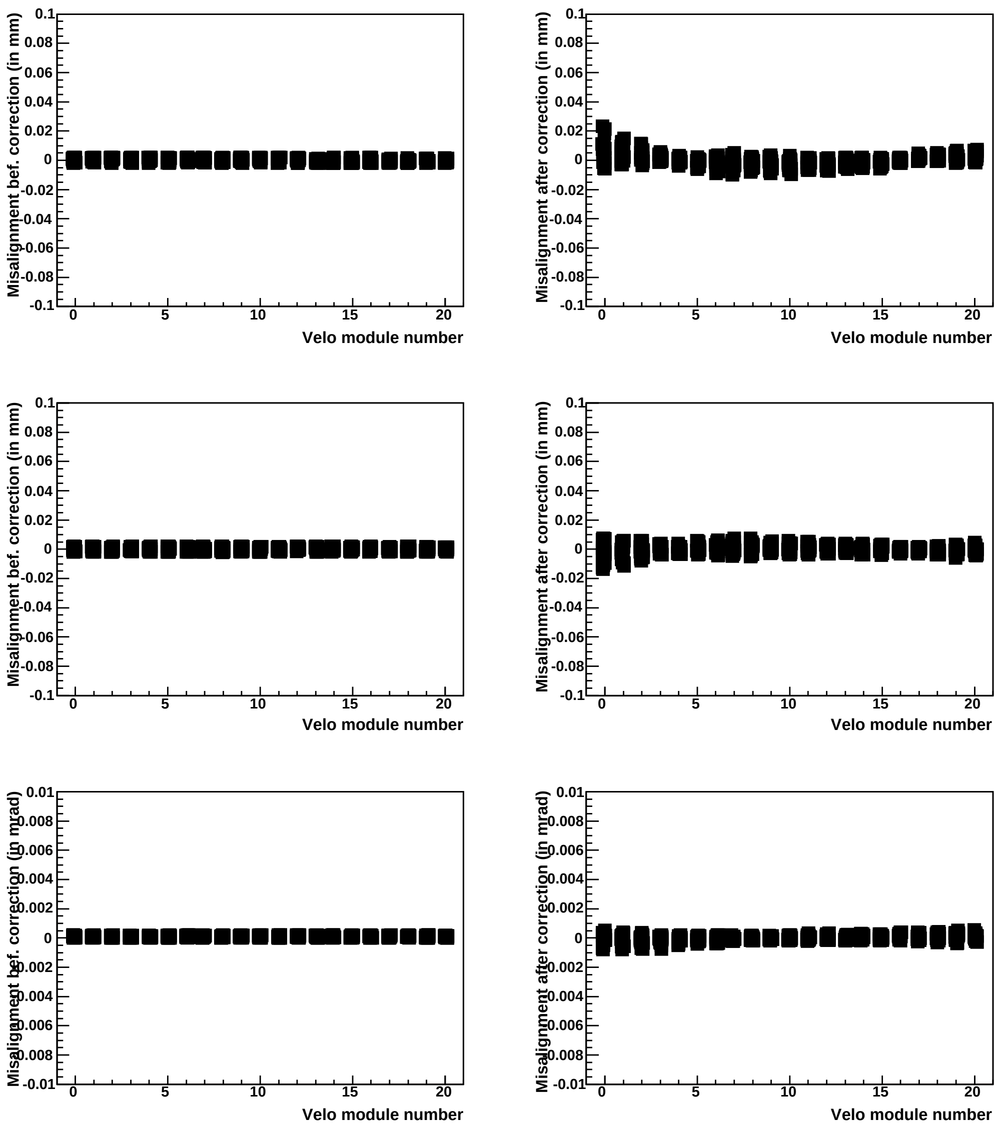


Figure 8 *Internal alignment on an internally aligned box: for dx (top), dy (middle), and $d\gamma$ (bottom) misalignments (the conventions of figure 6 are used).*

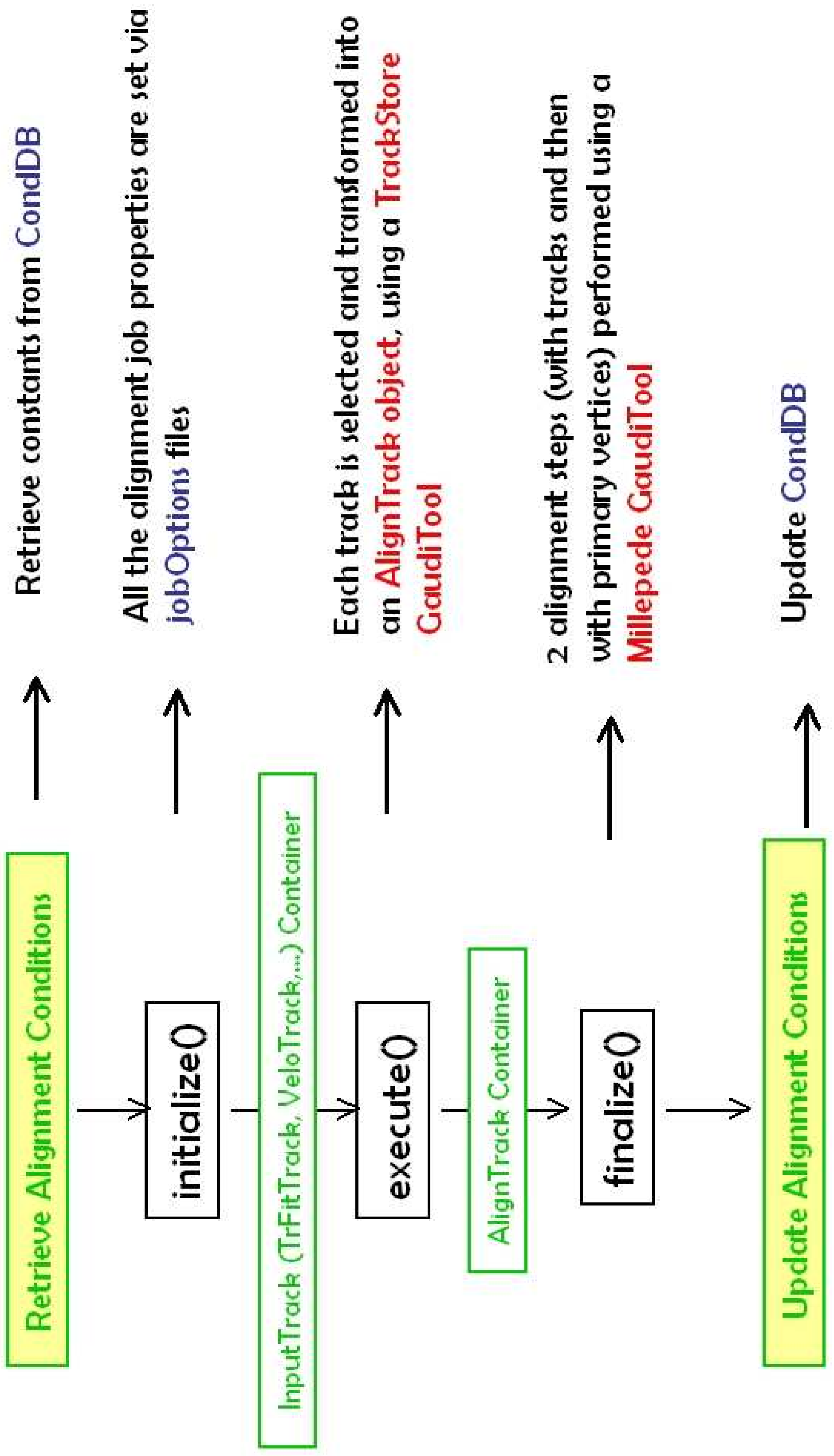

Retrieve Alignment Conditions
initialize()
InputTrack (TrFitTrack, VeloTrack,...) Container
execute()
AlignTrack Container
finalize()
Update Alignment Conditions
Retrieve constants from CondDB
All the alignment job properties are set via jobOptions files
Each track is selected and transformed into an AlignTrack object, using a TrackStore GaudiTool
2 alignment steps (with tracks and then with primary vertices) performed using a Millepede GaudiTool
Update CondDB

Figure 9 *VELO alignment in GAUDI: algorithm description*

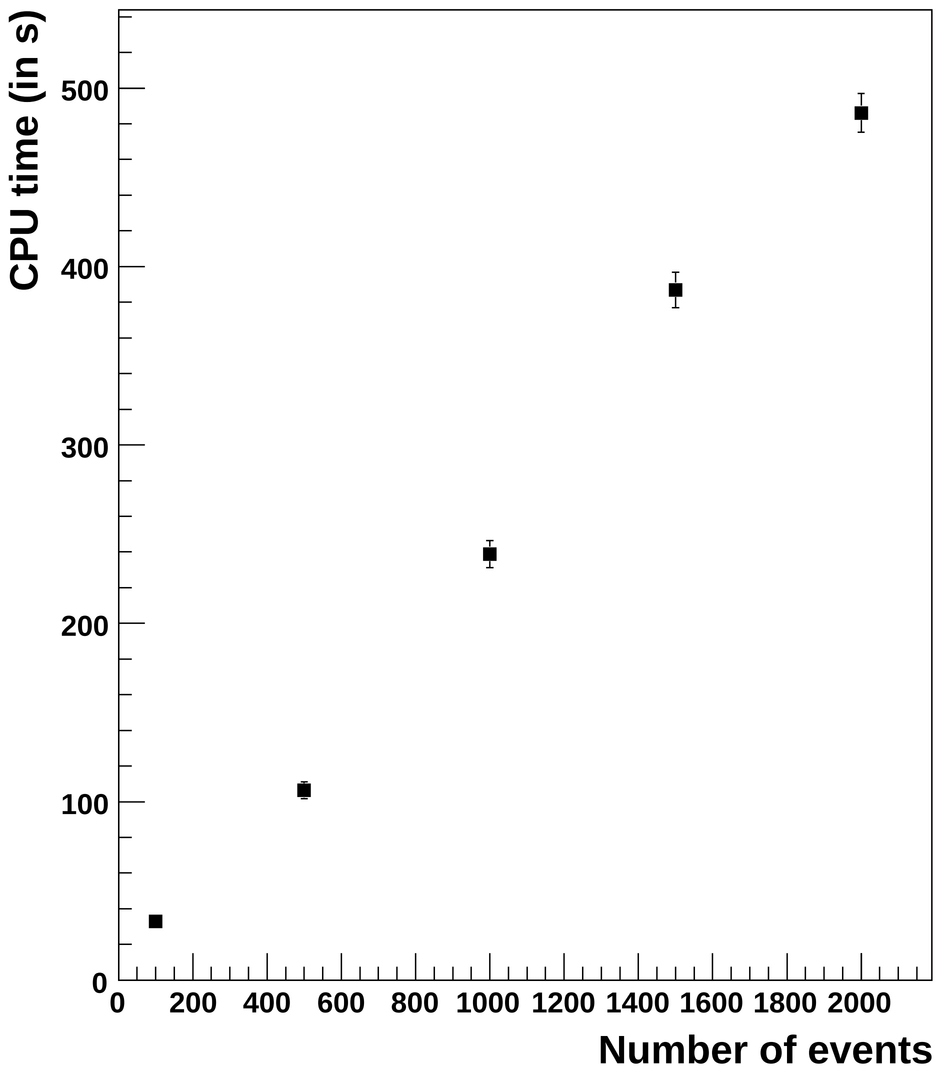
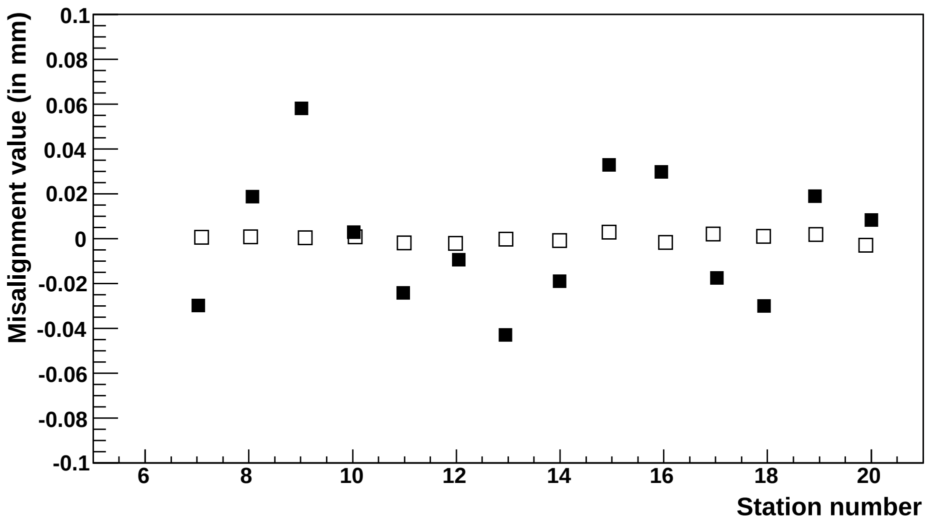


Figure 10 *The computing time, normalized to a single 1GHz Pentium III, necessary to perform the alignment algorithm, as a function of the number of events analysed.*



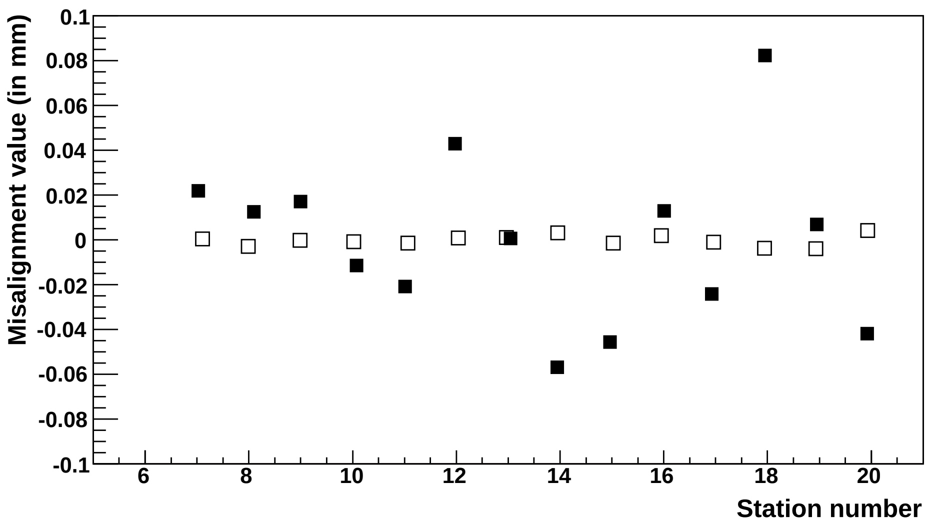
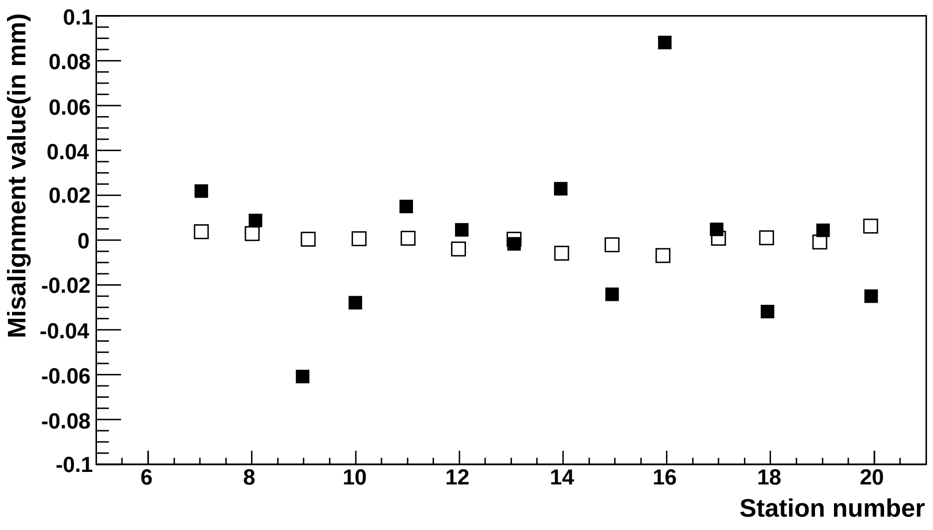


Figure 11 *Internal alignment effect on* dx *misalignments for one particular sets of misalignments. Black squares show the misalignment before correction, whereas open squares show them after internal alignment is applied. Top plot corresponds to the left box, bottom plot is for the right box.*



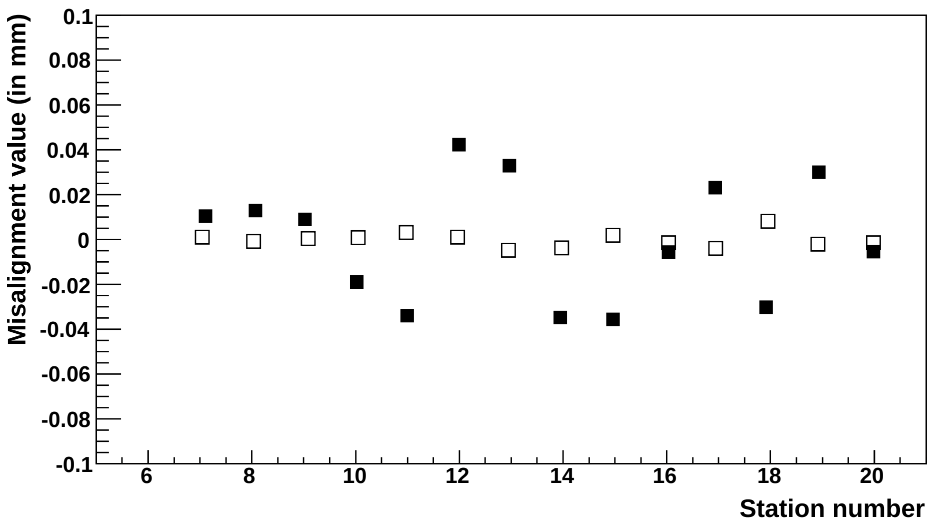


Figure 12 *Internal alignment effect on dy misalignments for one particular sets of misalignments (the conventions of figure 11 are used).*

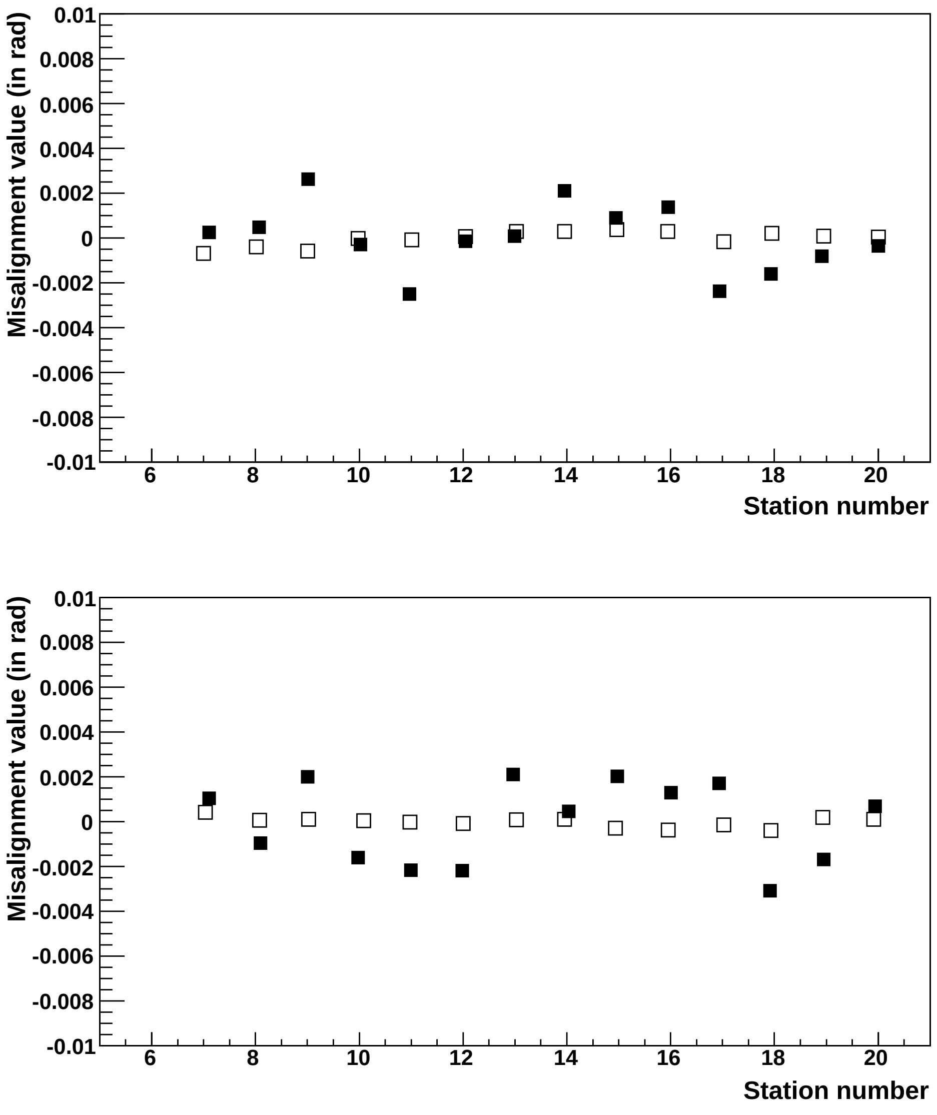


Figure 13 *Internal alignment effect on $d\gamma$ misalignments for one particular sets of misalignments (the conventions of figure 11 are used).*

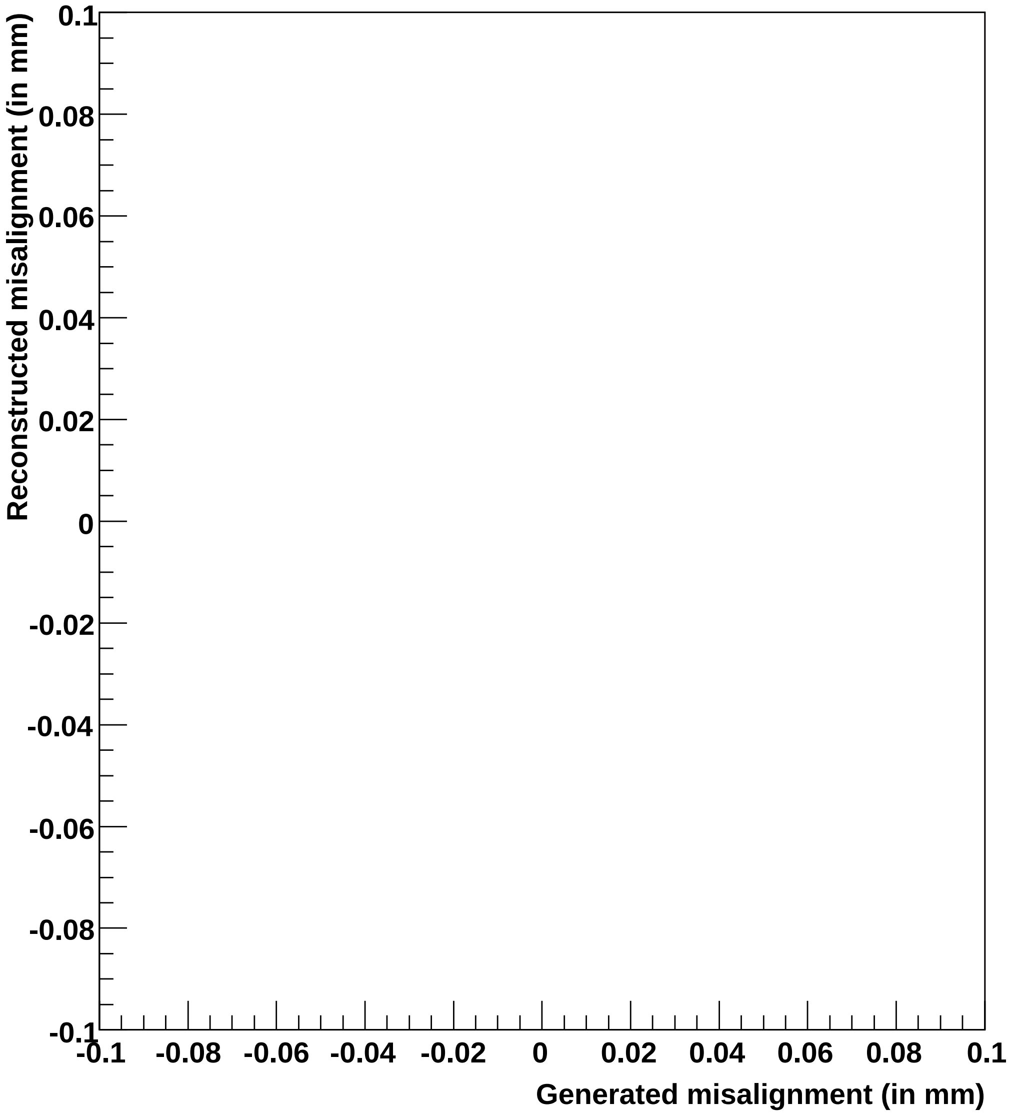


Figure 14 *Internal alignment robustness tests: results for the translations, for dx and dy (middle) misalignments (200 sets of misalignments).*

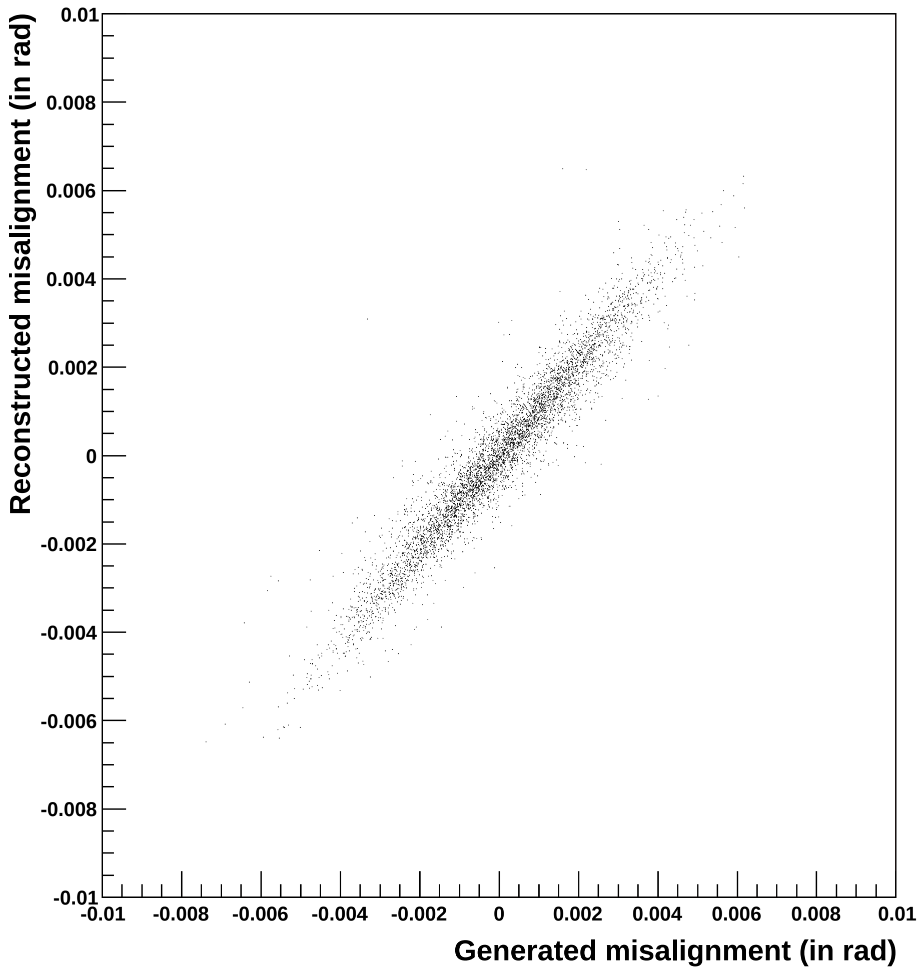


Figure 15 *Internal alignment robustness tests: results for the rotations, for $d\gamma$ misalignments (200 sets of misalignments).*

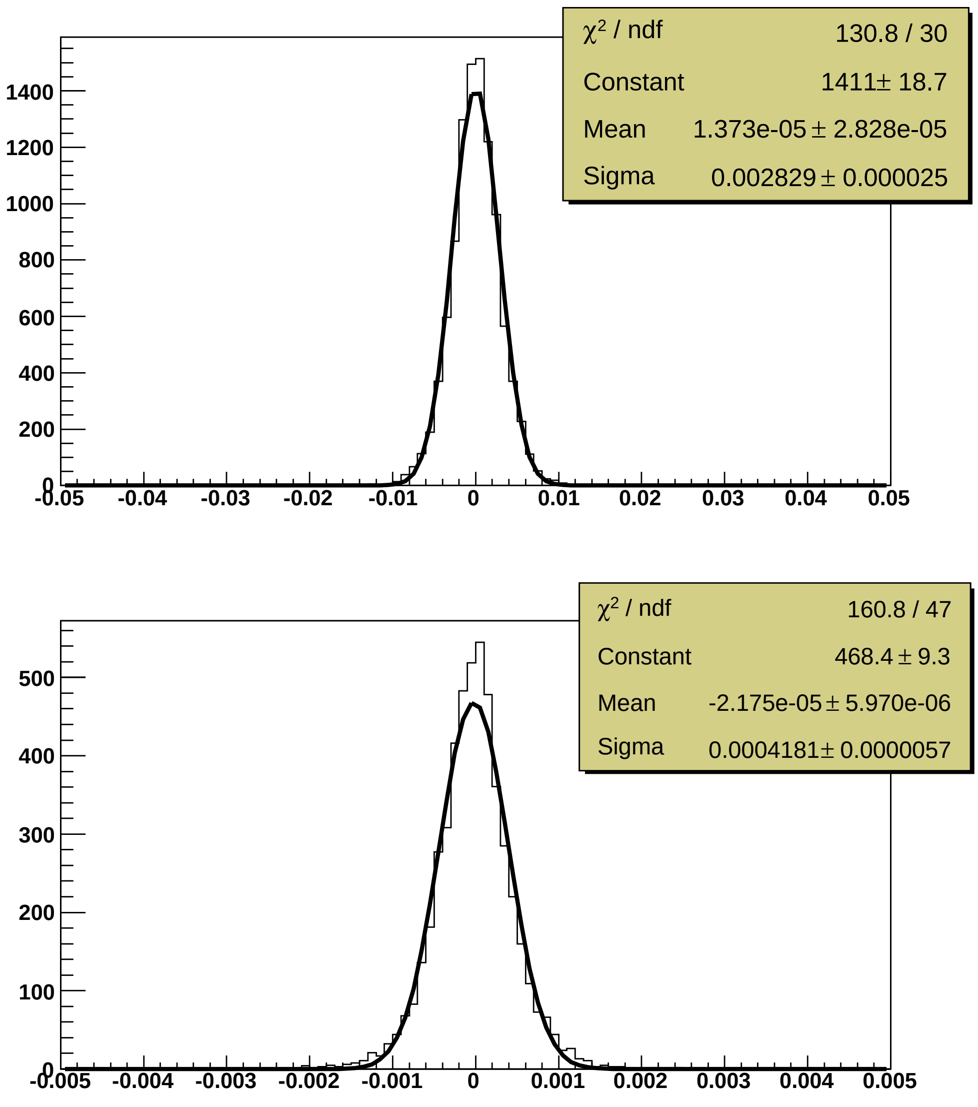


Figure 16 *Internal alignment robustness tests: resolution on the corrected misalignment constants (200 sets of misalignments). Top plot shows the result for the X and Y translations (in mm), bottom plot shows the result for Z rotation (in rad).*

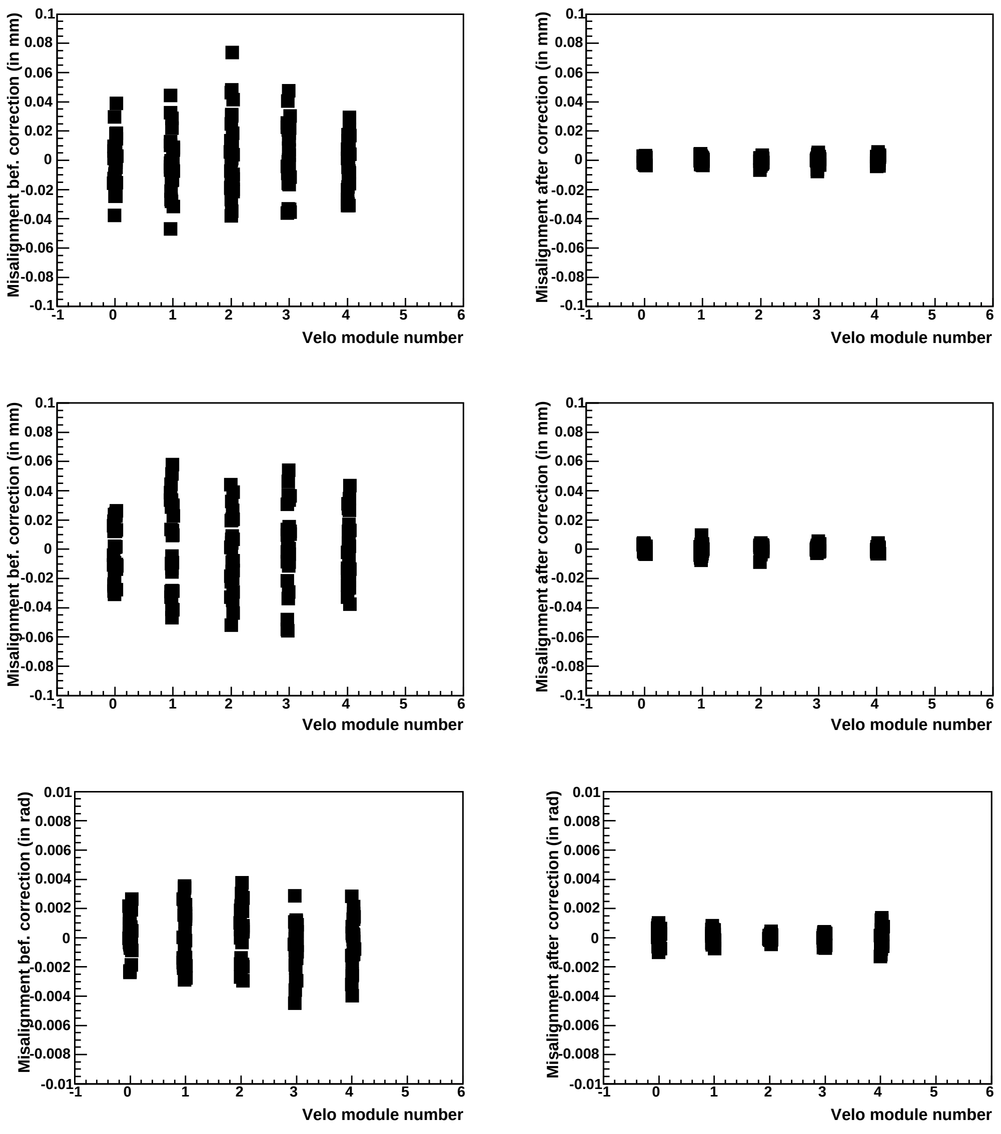


Figure 17 *Internal alignment robustness tests: results for the left box backward part, for dx (top), dy (middle), and $d\gamma$ (bottom) misalignments (the conventions of figure 6 are used).*

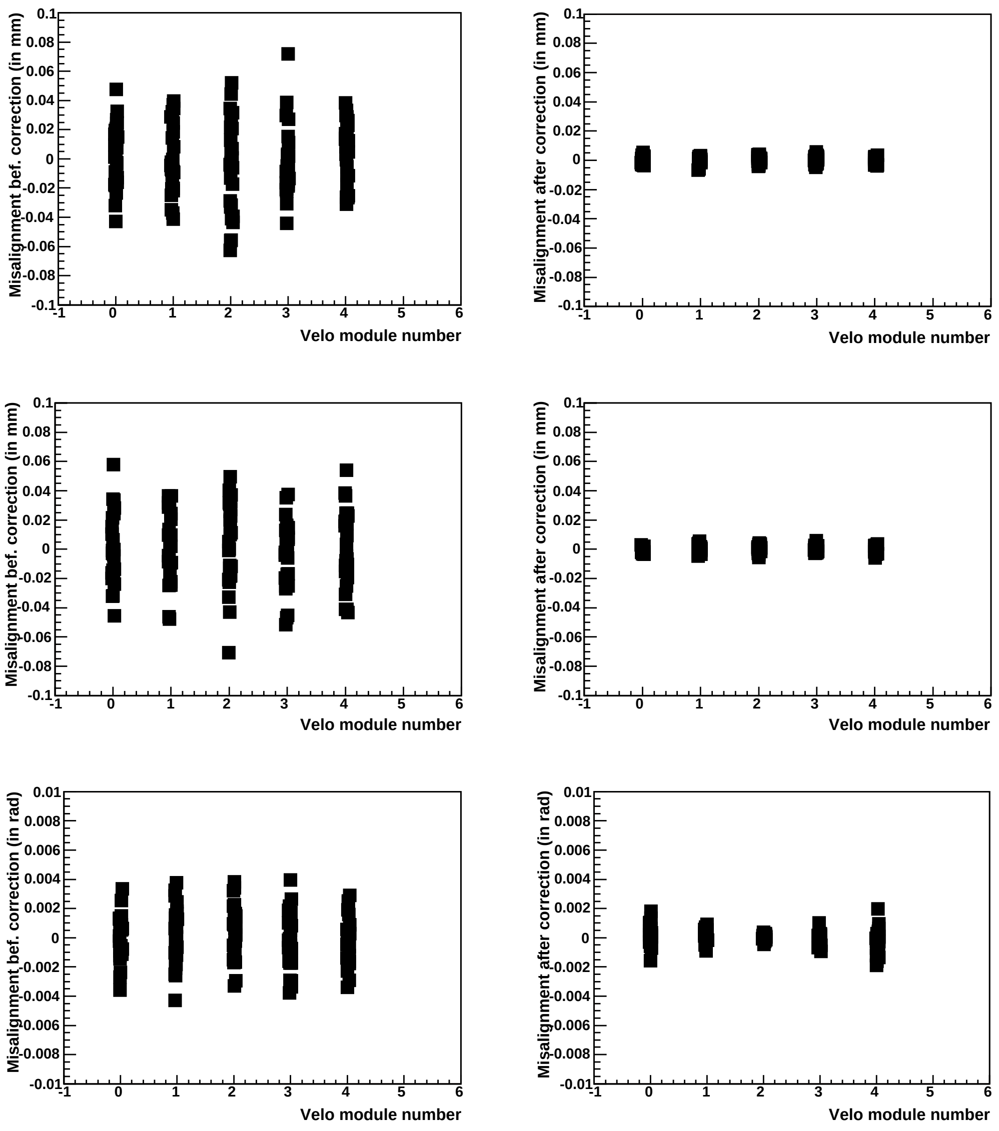


Figure 18 *Internal alignment robustness tests: results for the right box backward part, for dx (top), dy (middle), and $d\gamma$ (bottom) misalignments (the conventions of figure 6 are used).*

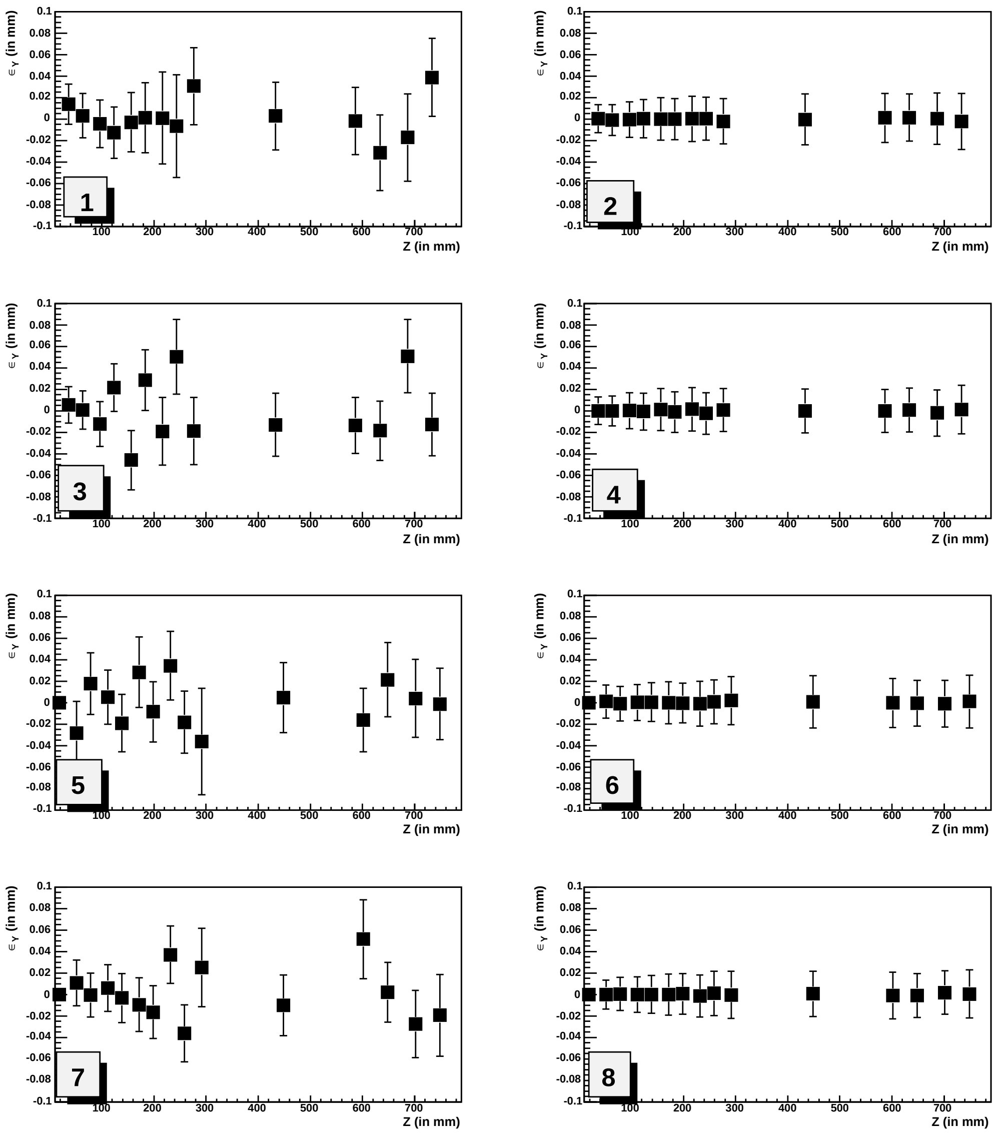


Figure 19 *Track residuals summary plot, as a function of Z position of the modules. Left plots are before internal alignment, right plots after. The plots 1,2 show the X residuals for the left half-box. The plots 3,4 show the Y residuals for the left half-box. Plots 5,6,7 and 8 show the same results for right half-box. Error bars correspond to the RMS of the residual distributions.*

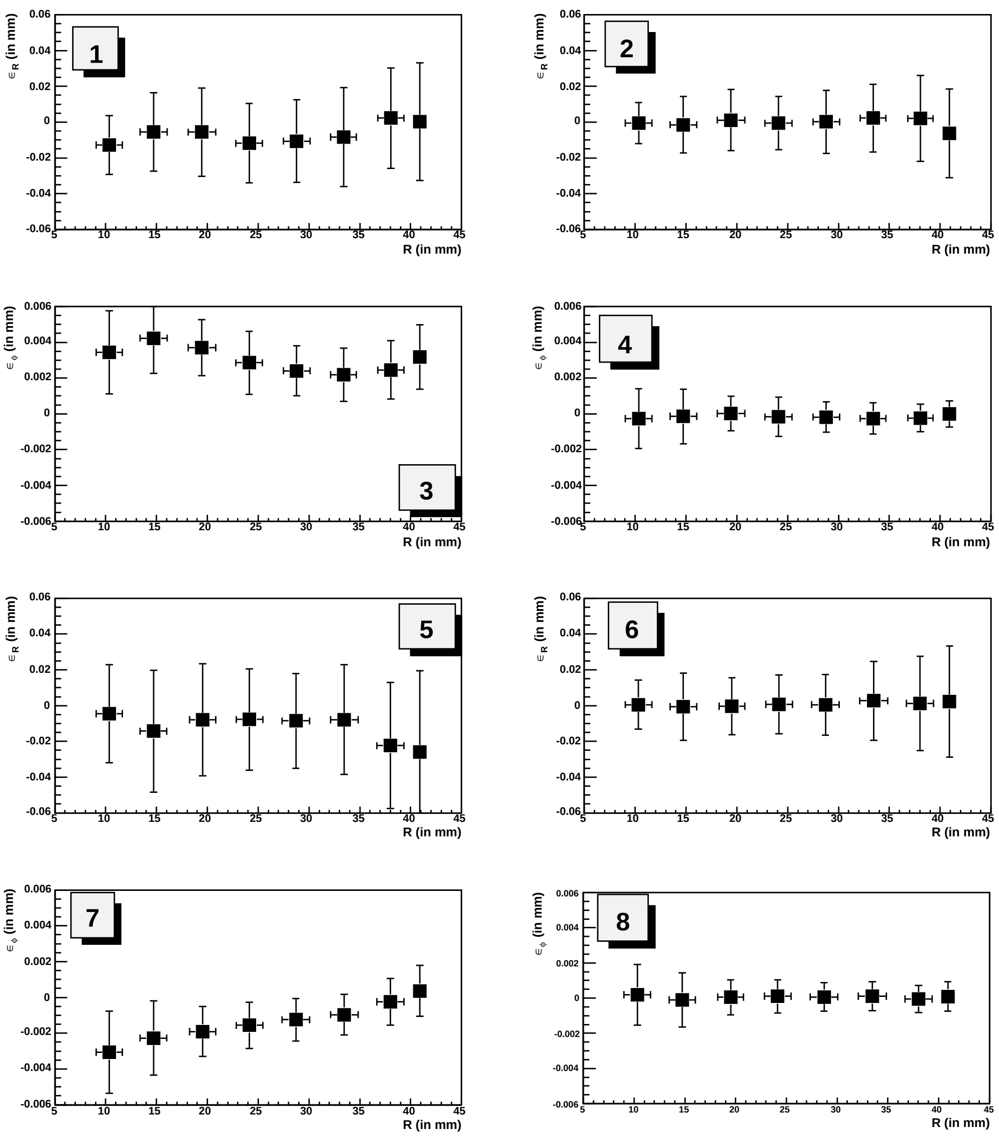


Figure 20 *Track residuals as a function of R, for one particular station (station 14) and one particular set of misalignments. Left plots are before internal alignment, right plots after. Are shown left box R and ϕ residuals (resp. 1,2 and 3,4), and right box R and ϕ residuals (resp. 5,6 and 7,8). Error bars correspond to the RMS of the residual distributions.*

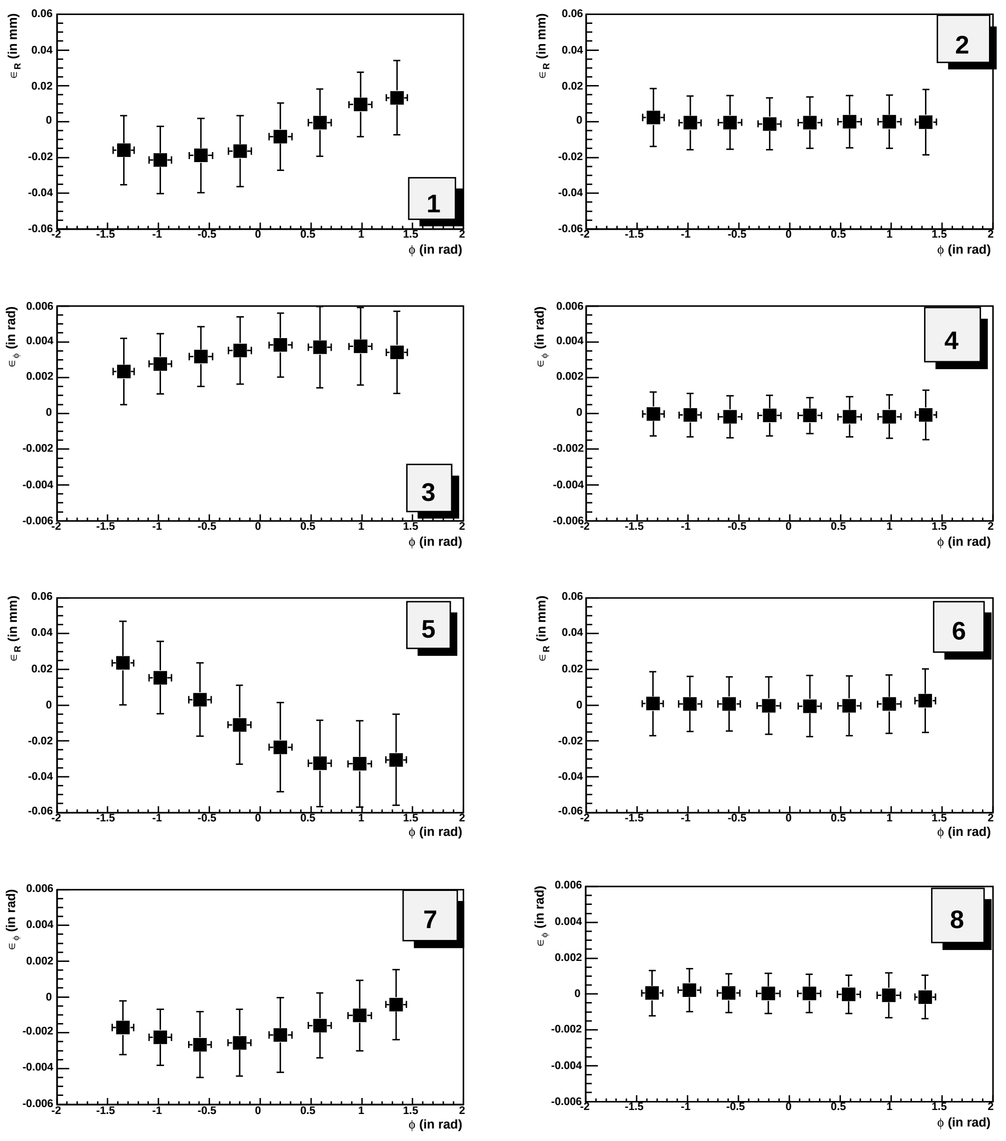
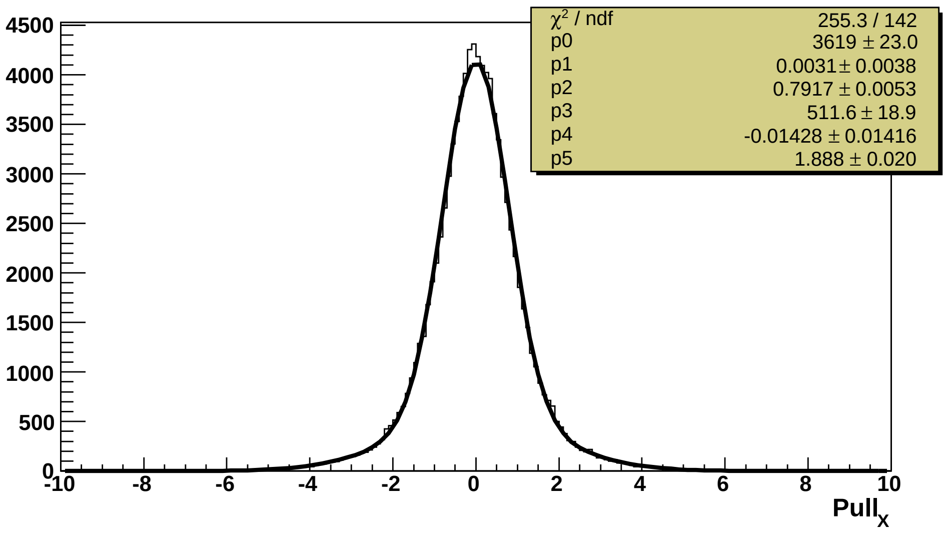


Figure 21 *Track residuals as a function of ϕ (same run and same set of misalignment than previous figure). Left plots are before internal alignment, right plots after. Are shown left box R and ϕ residuals (resp. 1,2 and 3,4), and right box R and ϕ residuals (resp. 5,6 and 7,8). Error bars correspond to the RMS of the residual distributions.*



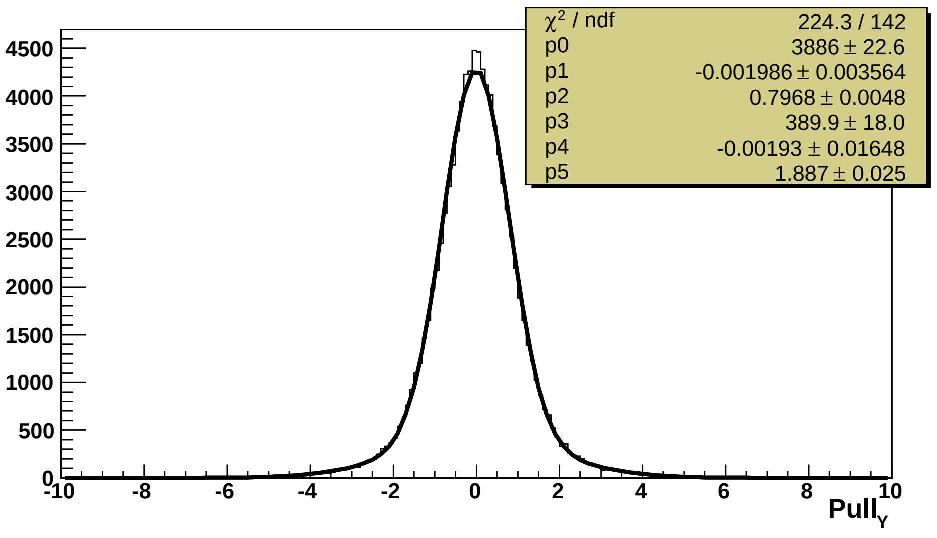


Figure 22 *Pulls of X (top) and Y (bottom) track residuals, computed after alignment. Distributions are fitted using a double gaussian. Gaussians resolutions are given by parameters p2 and p5, and means are given by p1 and p4*

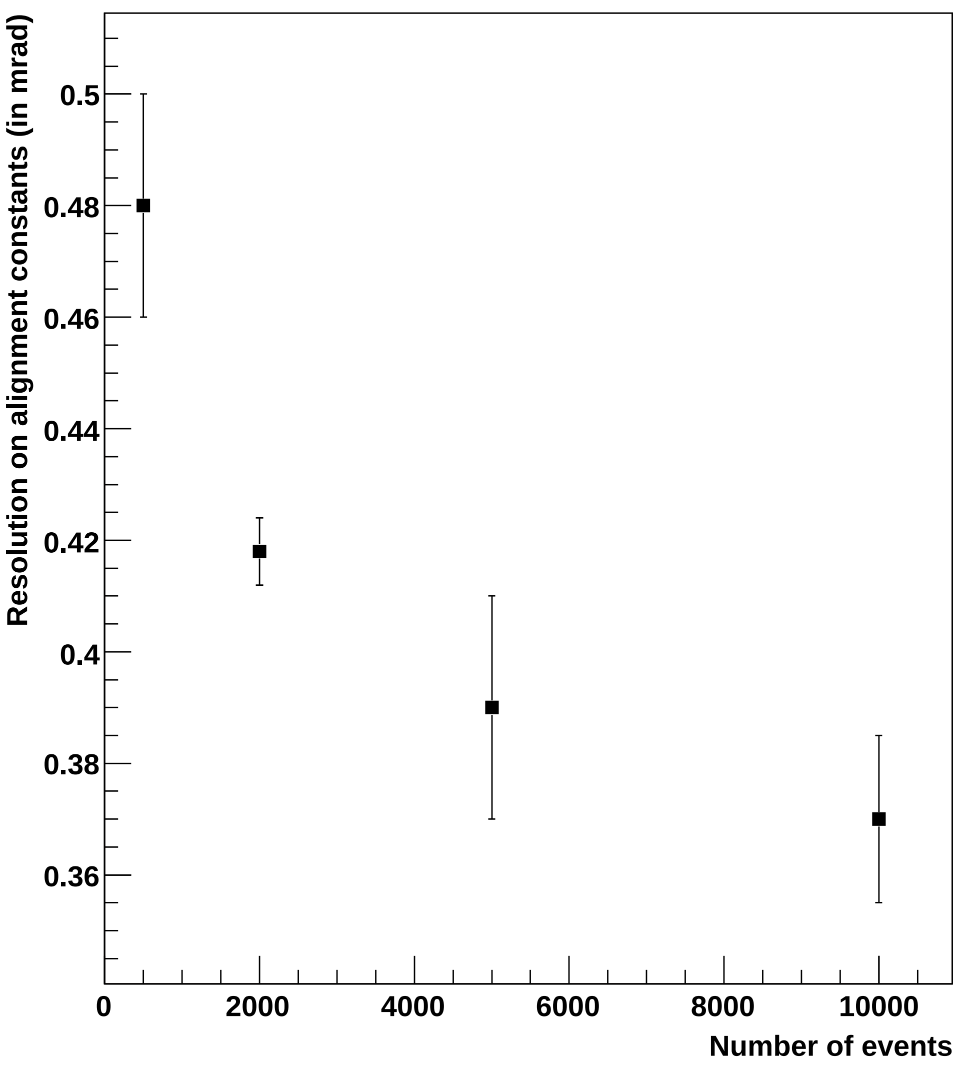


Figure 23 *Effect of number of events on internal alignment performance: Z rotation example.*