ETHZ-IPP Internal Report 2008-11
August 2008

Diploma thesis

Track based alignment of the CMS pixel barrel detector using the Millepede-II alignment algorithm

Frank Meier
frmeier@student.ethz.ch

Supervised by Hans-Christian Kästli[*], Roland Horisberger[*]
and Urs Langenegger[†]

[*]Paul Scherrer Institut
CH-5232 Villigen

[†]ETH Zürich
Institute for Particle Physics (IPP)
Schafmattstrasse 20
CH - 8093 Zürich

.

Abstract

The isolated alignment of the pixel barrel of the CMS detector using a track based approach has been studied. The alignment was performed using the Millepede-II algorithm which is one of the standard algorithms used by the CMS tracker alignment group. To decouple from the surrounding tracker subdetectors, an error enlargement technique has been implemented. The best result for a detector aligned to initial knowledge was a remaining misalignment distribution (with respect to the MC-truth) of $30\,\mu$m RMS in v direction using optimized error scaling parameters.

In the special case of a perfectly well aligned forward pixel detector, remaining misalignments of better than $10\,\mu$m RMS in u have been observed. This report also contains an outlook for further studies.

.

Contents

List of Figures

List of Tables

1 Introduction

The CMS[1] experiment is a multi purpose detector installed around one of the interaction points of the LHC[2] at Cern, Geneva. Its major parts are an inner tracking system of silicon type, an electromagnetic and a hadronic calorimeter, and an outer tracking system for muons. To create the magnetic field of 4 T, a superconducting solenoid is located between the calorimeters and the muon chambers.

The inner tracking detector of the CMS experiment consists of a silicon pixel and a silicon strip detector, both covering a pseudo rapidity range of $-2.5 < \eta < 2.5$ [7]. A track produces at least about 12 hits within the whole tracker, up to 3 of them in the pixel detector [7].

The aim of the presented work is to set up a procedure to align the pixel barrel detector (almost) independently from the strip detector using track based alignment algorithms. This chapter first gives an overview of the pixel detector and available algorithms for alignment. After this, a description of the chosen Millepede-II algorithm follows including the methods used for testing and optimization.

1.1 Tracker

The tracker of CMS is a silicon detector formed by a barrel and an endcap structure. The hit rate per sensitive area of an active detector surface decreases with the distance from the interaction point[3]. It is possible to control the expected occupancy of each channel by choosing an appropriate active area. Therefore, the detector elements that are closest to the interaction point are made of pixels while the outer elements are made of strips.

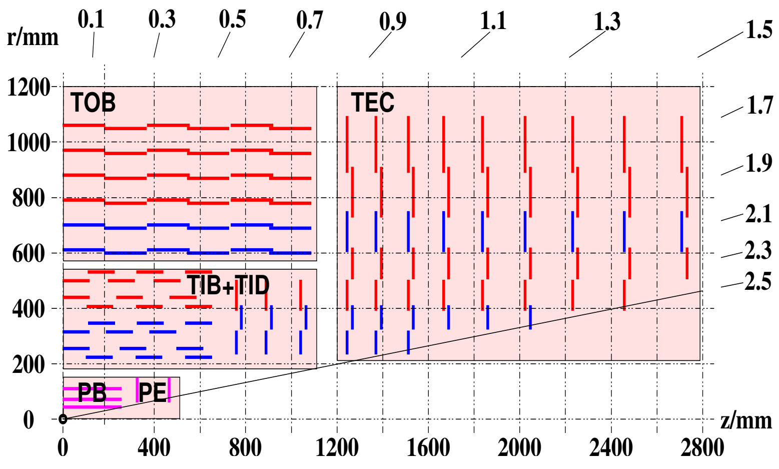


Figure 1: Tracker layout. Shown is one quarter of the tracker in the $r-z$ plane including η coverage. Image taken from [12] and reproduced by kind permission.

[1]Compact Muon Solenoid

[2]Large Hadron Collider

[3]For structures sufficiently far away from the interaction point and no magnetic field, the $1/r^2$-law holds. The beam spot size is $\sigma_{xy} = 15\,\mu$m and $\sigma_z = 5.3\,$cm [10], therefore the central regions of the pixel detector experience a hit rate decrease following $1/r^\xi$ with $1 \leq \xi \leq 2$. As soon as the magnetic field is present, deviations from the $1/r^2$-law occur due to effects like looping tracks from particles with low p_T.

1.2 Pixel Detector

The pixel detector consists of:

- a barrel (BPix): three concentric cylindrical structures with a length of 53 cm and mean radii with respect to the beam of 4.4 cm, 7.3 cm and 10.2 cm respectively.
- a forward detector (FPix): two disks on both ends of the barrel.

The layers of the pixel barrels are formed by modules $67 \times 26\,\mathrm{mm}^2$ in size. Each module has 2×8 readout chips[4] with 52×80 pixels with a cell size of $100 \times 150\,\mu\mathrm{m}^2$ per pixel. The direction of the electron drift is perpendicular to the magnetic field. The resulting Lorentz drift leads to charge spreading over several pixel cells. As the pulse height is an available measurement, the resolution can be enhanced using charge weighting. The resolution has a η dependence, reaching an optimum of $15 - 20\,\mu$m at $\eta \approx 0.64$ (see figure 2).

The main reason for development and construction of a pixel detector was to improve the precision in vertex location measurements, e.g. paramount for precise impact parameter determination and reconstruction of secondary vertices. The knowledge of the true geometry is necessary to achieve the theoretical precision limit of the detector. The deviation of the true geometry from the ideal one is called *misalignment*, the process of measuring this deviation therefore *alignment*.

There are two main sources of misalignment in the pixel detector. First, the mounting precision of the modules is of the order of $100\,\mu$m, about an order of magnitude worse than the resolution of the modules. The second source are movements due to changes in temperature; movements of up to $10\,\mu$m are expected according to [1], being also a source of misalignment in the order of the nominal resolution. Both sources cannot be neglected nor can they be overcome by a suitable change of geometry. The position of the pixel detector needs to be monitored during operation up to at least the nominal spatial resolution, i.e. $\approx 10\,\mu$m.

The tracker consists of subdetectors and each subdetector has a hierarchy in itself. This is also true for the pixel detector, which consists of two half barrels. Each of them is built of three layers. Every layer has ladders on which eight modules are mounted in one row, face to face on the shorter edge.

1.3 Scope of this work

This work concentrates on the isolated alignment of the pixel barrel detector *without* aligning the other parts of the tracker. There are good reasons for doing this: The pixel modules are operated in a zero suppressed mode. Only if there is a signal in a pixel above a certain threshold, the readout circuitry gets in action. As a consequence, the thermal dissipation power correlates with the hit rate, which changes over time during a typical fill run of the accelerator. This is in contrast to the strip tracker having almost constant thermal dissipation power. Deformation of the support structures are likely to be observed by up to ten microns [1]. A procedure of monitoring the alignment of the pixel detector with minimal amount of data taking is required.

[4] So-called half-modules with only 1×8 readout chips exist for ladders where the half-barrels touch each other.

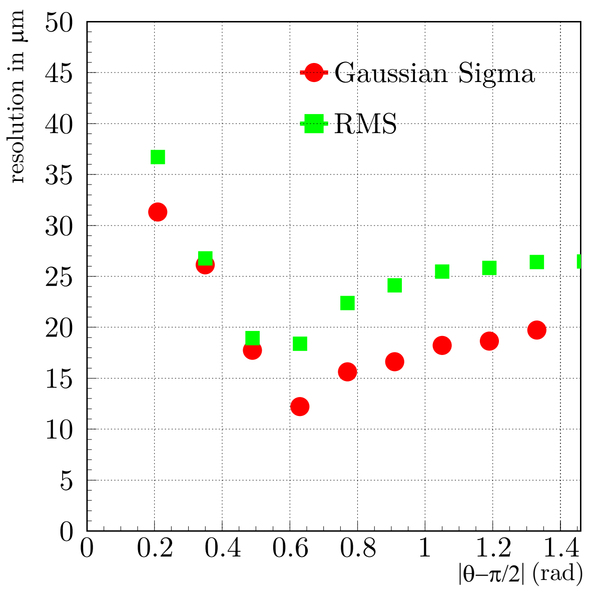


Figure 2: Resolution of pixel detector. The graph shows the resolution in z direction. θ is the angle of the incident particle with respect to the beam ($\theta = \pi/2$: perpendicular incidence on a module on the pixel barrel). The minimum is around $|\beta - \frac{\pi}{2}| = 0.6$, corresponding to a pseudeorapidity $\eta \approx 0.64$. Image taken from [16] with kind permission from the author.

This study is a Monte-Carlo simulation based upon software available in the CMS software framework CMSSW [10]. All results are restricted to the assumption, that the software is a correct implementation of the real behaviour of the detector. The validation of this assumption is beyond the scope of this work. This study has the following goals:

1. Demonstrate the alignment of the pixel barrel detector with as little information from other sources (i.e. strip detector) as possible.
2. Develop a set of working conditions and show limits within which alignment is possible.
3. Give sound rationales for the chosen parameters and the behaviour observed.
4. Give criteria for the quality of the alignment without knowing the truth, i.e. give real world measures of alignment.

2 Principles and methods

2.1 Coordinate system conventions

The *global* coordinate system is defined as follows [10]: The origin is centered at the nominal collision point inside the experiment. The y-axis points upwards and the x-axis points inwards to the center of the collider ring. Consequently, the z-axis points along the beam axis. The azimuthal angle ϕ is measured from the x-axis in the xy-plane, the polar angle θ is measured from the z-axis. Pseudrapidity is defined as $\eta = -\ln\tan\frac{\theta}{2}$.

The *local* coordinate system of an object is defined with respect to its "center of mass" as shown in figure 3. The u coordinate is the one known with highest precision, v the one with least precision. w is perpendicular to the uv-plane. The angles α, β and γ correspond to the movements pitch, roll and yaw respectively.

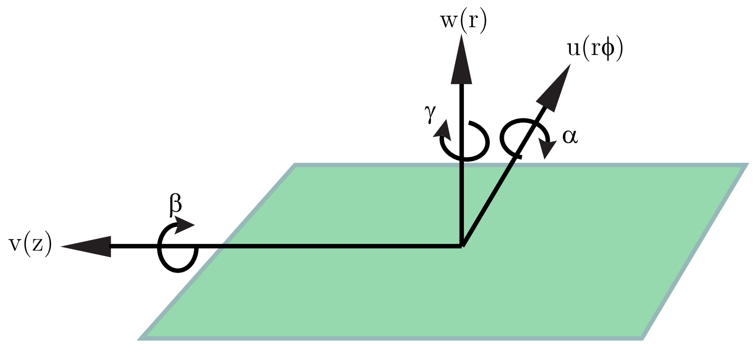


Figure 3: Definition of local coordinates. In brackets: corresponding direction in global coordinate system.

2.2 Track based alignment

While in operation at design luminosity of LHC, every bunch crossing produces about 1000 particle tracks recorded and possibly reconstructed by the tracker [7]. A high level trigger system reduces the rate of events stored down to 100 Hz, each event containing about 4 tracks[5] with a momentum of at least 1.5 GeV/c. Such tracks pass through all layers of the tracker, provided they occur in a suitable η-region. They can be used for track based alignment.

2.2.1 Principles

For the moment, let us assume that we have an ideal detector without multiple scattering. Each track of a charged particle follows a helical trajectory, described by five parameters with respect to a reference point. Whenever such a trajectory penetrates a layer of the tracker, a signal is produced recording one point in space. Such a trajectory is continuous and smooth, all measured points belonging to one trajectory have a normal distributed residual within a width of the nominal detector resolution. Such a residual can be calculated as follows:

[5]Based on the minimum bias MC-samples described in section 3.2. Real data may differ.

$$S = \sum_i \frac{r_i^2}{\sigma_i^2} \quad (= \chi^2) \tag{1}$$

Here S denotes the sum of the residuals, $r_i = |\mathbf{y}_{\text{true}} - \mathbf{y}_{\text{measured}}|$ a single residuum and σ_i the error of the measurement. Here we assume, that measurements follow a Gaussian distribution. As the sum is weighted by the errors, S qualifies the properties of a χ^2.

In an ideal detector with infinite precision and resolution power, S cannot be calculated as the error vanishes. A real detector has a finite resolution, defined by properties of the sensing device itself and by effects like multiple scattering of particles in matter. This makes S a calculable quantity. Now we move to a misaligned detector. The measured points along a trajectory differ by how much the position of a module is shifted away from its ideal position. The sum of the residuals S will be higher than in the ideal case. Observe, that moving from an aligned to a misaligned detector does not change the measurement error σ_i. Minimizing S by adjusting the position data of the modules will lead to a better aligned detector, assuming S is well behaved.

Three algorithms have been implemented in CMSSW. These are:

Millepede-II This algorithm is an implementation of a linear least square fitter assuming a special structure of the data. The main ideas are described in the textbook of Volker Blobel, who is the main inventor of the algorithm. [6] The parameters of the detector together with the parameters of the track trajectories are fitted. As the number of tracks can get very large, say millions, the required matrix inversion in standard least square fitting algorithms is no longer suitable. The special structure of the problem allows a dramatic reduction of the matrix size using linear algebra techniques. This leads to a matrix size determined by the number of detector parameters, which can be inverted by widely available sparse matrix inversion algorithms [5, 12].

Hit and Impact Point Alignment (HIP) This algorithm uses the same principles as Millepede, but it solves for the detector parameters in an iterative manner over each track and detector, keeping the matrix sizes small (typical 6×6 when all degrees of freedom for a single detector module were aligned). [15]

Kálmán Filter This algorithm is based on the widely used approach for track reconstruction in high energy physics experiment. In principle, a Kálmán filter reconstructs a trajectory from an initial point and extrapolates the next point by using least-squares estimates. While propagating along the predicted trajectory, the Kálmán filter accumulates the most valuable data at the current point to predict the next point until the full trajectory has been reconstructed. To smoothen the result, this can be done in both ways, i.e. from the innermost track hit to the outermost and vice versa. In order to use this idea for detector alignment, the measurement used in the algorithm not only depends on the measured data but also on the alignment data. By recursive application, the alignment parameters can be estimated. [13]

Millepede-II has been chosen for this study as being the most versatile algorithm available in the framework with respect to our needs.

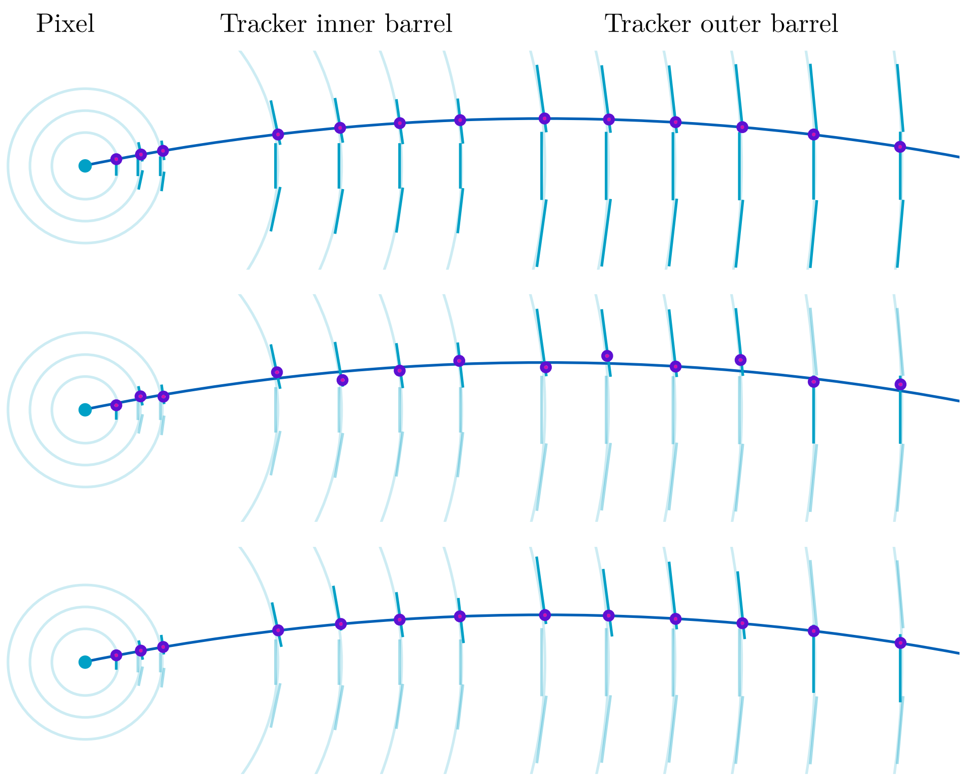


Figure 4: Principle of track based alignment. From top to bottom: Ideal geometry, misaligned detector before and after alignment.
The figures show a schematic view of the CMS tracker in the $r\phi$-plane. Module positions are arbitrarily chosen but the radii with respect to the beam spot are to scale.
A perfectly aligned detector (top figure) records the hits exactly at the true positions up to the inherent precision of the detecting device. Fitting a trajectory to this data will lead to a trajectory close to the real one. When a misaligned detector is operated assuming an ideal geometry (middle figure), the recorded track hit positions deviate from the true trajectory of the track leading to a worse fit. In order to get the correct track hit positions, the actual position of the modules needs to be known. Adjusting the position parameters of the modules in a way that improves the quality of the fitted trajectory leads to a representation of the true geometry of the detector (bottom).
Therefore, the core idea of track based alignment algorithms is to first calculate the summed residuals of each hit in a track between recorded hits and a fitted trajectory. In a second step, the algorithm minimizes the residuals and the known positions converge to the real geometry.

2.3 Track parametrization

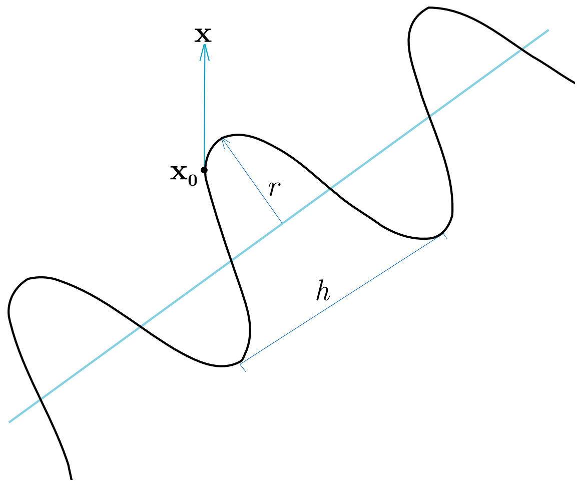


Figure 5: Geometric parametrization of a helix. This is a first approach on how to parametrize a helix: Construct the tangent $\mathbf{x}$ to some chosen reference point $\mathbf{x_0}$. The radius r and the height h are the two other parameters needed to describe the helix. In total, eight parameters are needed for a full description: 3 for the reference point $\mathbf{x_0}$ and 5 for the helix itself $(\mathbf{x}, r, h)$.

An ideal track of a charged particle moving in a magnetic field neglects multiple scattering and does not change its nature throughout the measurement, say due to a decay. Such a track follows a helical trajectory. A helix can be parametrized with respect to a reference point by five parameters. A simple geometric parametrization is shown in figure 5.

Depending on the calculation in question, different parametrizations are available [18].

Curvilinear frame: $(q/p, \lambda, \phi, x_\perp, y_\perp)^T$ [18] This frame can be defined at every point of the trajectory using a local cartesian coordinate system $(x_\perp, y_\perp, z_\perp)$ defined by three orthogonal unit vectors $\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$ and $\hat{\mathbf{t}}$. The vector $\hat{\mathbf{t}}$ is defined as the unit vector parallel to the track, pointing in the particle direction. Using the unit vector $\hat{\mathbf{z}}$ parallel to the global z-direction, the two vectors $\hat{\mathbf{u}}$ and $\hat{\mathbf{v}}$ are defined as

$$\hat{\mathbf{u}} = \frac{\hat{\mathbf{z}} \times \hat{\mathbf{t}}}{|\hat{\mathbf{z}} \times \hat{\mathbf{t}}|} \tag{2}$$

$$\hat{\mathbf{v}} = \hat{\mathbf{t}} \times \hat{\mathbf{u}} \tag{3}$$

Therefore, the $z_\perp$-axis is pointing along the particle direction, the $x_\perp$-axis is lying in the global xy-plane, and the $y_\perp$-axis is perpendicular on the two others, in order to form a right-handed cartesian coordinate system.

The five parameters used in the curvilinear frame are: $x_\perp$ and $y_\perp$ as defined above, q/p being the signed curvature of the track, the dip angle of the particle 3-momentum vector λ, and the inclination angle ϕ between the tangent of the projection of the particle 3-momentum vector into the global xy-plane and the global z-direction.

This parametrization is useful for track reconstruction, as it can be defined at every point along the track using the track length as an evolving parameter. It is the default frame used in CMSSW for track reconstruction.

Perigee frame: $(d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$ [4] As the name suggests, the reference point is the closest approach of the trajectory to the origin of the coordinate system in use. This parametrization is especially useful for vertex fitting.

The definition is given by the following description of a track in three dimensional space:

$$\begin{pmatrix} x_0 + d_\rho \cos\phi_0 + \frac{\alpha}{\kappa}(\cos\phi_0 - \cos(\phi_0 + \phi)) \\ y_0 + d_\rho \sin\phi_0 + \frac{\alpha}{\kappa}(\sin\phi_0 - \sin(\phi_0 + \phi)) \\ z_0 + d_z - \frac{\alpha}{\kappa}\tan(\lambda\cdot)\phi \end{pmatrix} \tag{4}$$

where the vector $\mathbf{x_0} = (x_o, y_o, z_0)^T$ describes the reference point and the vector $\mathbf{a} = (d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$ describes the helix. Assuming the z direction be parallel to the magnetic field, d_ρ is the distance from the reference point in the xy-plane, ϕ_0 is the azimuthal angle to the helix center, κ is the reciprocal transverse momentum, d_z the distance from the reference point in z direction and $\tan\lambda$ the dip angle. κ can be calculated using

$$\kappa = \frac{Q}{p_T} \tag{5}$$

$$\rho = \frac{\alpha}{\kappa} \tag{6}$$

with Q the charge and ρ the signed radius of the helix. The constant $\alpha := \frac{1}{cB}$ depends on the magnetic field.

In a more realistic approach, multiple scattering has to be taken into account. Scattering processes result in an angular straggling which can be treated as an increasing error to measurements further away from the origin of the track.

In the software framework CMSSW, tracks are described by the class `TrackBase.h` using the following information:

- A reference position on the track: (x, y, z)
- Momentum at this given reference point on track: (p_x, p_y, p_z)
- 5D curvilinear covariance matrix from the track fit
  - $\frac{q}{|p|}$ = signed inverse of momentum
  - $\lambda = \pi/2-$ polar angle at the given point
  - $\phi =$ azimuth angle at the given point
  - $d_{xy} = -x \cdot \sin\phi + y \cdot \cos\phi$ – an estimate of the impact parameter in the xy-plane
  - $d_{sz} = z \cdot \cos\lambda - (x \cdot \cos\phi + y \cdot \sin\phi \cdot \sin\lambda)$
- Charge q
- Chi-square and number of degrees of freedom
- Summary information of the hit pattern

Some information is redundant but available for convenience.

2.4 Millepede-II

In the introductory part of this section, formula 1 has been defined as the sum of the residuals and the task is to minimize S. This is in fact a least squares minimization problem. This formula can be rewritten in vector notation as[6]

$$S = \mathbf{r}^T \mathbf{W}(\mathbf{y})\mathbf{r} \tag{7}$$

where $\mathbf{r}$ is the vector of residuals having r_i as its entries. $\mathbf{W}(\mathbf{y})$ is a matrix describing the errors. If the measurements do not influence each other, this matrix is diagonal and has the following form:

$$\mathbf{W} = \mathbf{W}(\mathbf{y}) = \begin{pmatrix} 1/\sigma_1^2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & 1/\sigma_N^2 \end{pmatrix} \tag{8}$$

where $(\mathbf{y})$ is the vector of the $1/\sigma_i^2$. $\mathbf{W}$ is also known as the inverse covariance matrix. In our problem, the residuals depend on a set of parameters $\mathbf{a}$, describing the positions of the detector modules. If $\mathbf{y}$ is the vector of the measured positions and $\mathbf{x}$ the vector of the track parameters, the residuals can be written as

$$r_i = y_i - f(x_i, \mathbf{a}) \tag{9}$$

where $f(x_i, \mathbf{a})$ is the function linking these vectors together, depending on the parameters in $\mathbf{a}$. Assuming a linear dependence, $f(x_i, \mathbf{a})$ can be written as

$$f(x_i, \mathbf{a}) = \sum_j a_j f_j(x_i) \tag{10}$$

and therefore the sum S of the residuals looks like

$$S = \sum_i \frac{r_i^2}{\sigma_i^2} = \sum_i \frac{1}{\sigma_i^2}\left(y_i - \sum_j a_j f_j(x_i)\right)^2 \tag{11}$$

If S is minimal, all the partial derivatives with respect to the parameters a_i will vanish, which leads to a set of equations, commonly known as *normal equations*. These can be written in matrix notation using the following definitions:

$$\mathbf{A} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_p(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_p(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \cdots & f_p(x_n) \end{pmatrix} \qquad \mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} \tag{12}$$

The vector of residuals $\mathbf{r}$ can now be written as

$$\mathbf{r} = \mathbf{y} - \mathbf{A}\mathbf{a} \tag{13}$$

and for the sum S:

$$S = \mathbf{r}^T\mathbf{W}\mathbf{r} = (\mathbf{y}-\mathbf{A}\mathbf{a})^T\mathbf{W}(\mathbf{y}-\mathbf{A}\mathbf{a}) = \mathbf{y}^T\mathbf{W}\mathbf{y} - 2\mathbf{a}^T\mathbf{A}^T\mathbf{W}\mathbf{y} + \mathbf{a}^T\mathbf{A}^T\mathbf{W}\mathbf{A}\mathbf{a} \tag{14}$$

[6]This section is based on [6]

Differentiating this with respect to the parameters a_i and asking the partial derivatives to vanish leads to

$$-2\mathbf{A}^T\mathbf{W}\mathbf{y} + 2\mathbf{A}^T\mathbf{W}\mathbf{A}\mathbf{a} \quad \Rightarrow \quad \underbrace{(\mathbf{A}^T\mathbf{W}\mathbf{A})}_{\mathbf{C}}\mathbf{a} = \underbrace{\mathbf{A}^T\mathbf{W}\mathbf{y}}_{\mathbf{b}} \quad \Rightarrow \quad \mathbf{C}\mathbf{a} = \mathbf{b} \tag{15}$$

which can be solved using standard methods. Observe that only the derivatives of the track parameters are needed. The three parameters in the helix parametrization describing the reference point are constant. Therefore, only the derivatives from the remaining five parameters are assumed to be non-vanishing.

Assuming we want to use this method to align a detector with, for example, 1000 modules. As each module has six degrees of freedom (3 spatial and 3 rotational), we have 6 000 parameters just to describe the detector. For each module hit, 5 parameters are required to describe the trajectory of the track through this particular module. When we now require 100 hits per module, we end up with 500 000 parameters for the hits. The matrix $\mathbf{A}$ has dimension 506 000, which is far too large to be inverted in reasonable time on a current computer. The problem looks forlorn, but one observation may help: we are interested only in the parameters describing the detector, not in those describing the track hits. From now on, the p parameters describing the detector are called *global* parameters $\mathbf{a}$ and those describing the track hits are called *local* parameters α_i, for the i-th track hit. When writing down the matrix $\mathbf{C}$ in ordered form, i.e. assembling all global parameters in the upper left of the matrix, we observe a special structure:

$$\underbrace{\begin{pmatrix} \sum \mathbf{C}_i & \mathbf{G}_1 & \cdots & \mathbf{G}_i & \cdots \\ \mathbf{G}_1^T & \mathbf{\Gamma}_1 & 0 & 0 & 0 \\ \vdots & 0 & \ddots & 0 & 0 \\ \mathbf{G}_i^T & 0 & 0 & \mathbf{\Gamma}_i & 0 \\ \vdots & 0 & 0 & 0 & \ddots \end{pmatrix}}_{\mathbf{C}} \cdot \begin{pmatrix} \mathbf{a} \\ \alpha_1 \\ \vdots \\ \alpha_i \\ \vdots \end{pmatrix} = \underbrace{\begin{pmatrix} \sum \mathbf{b}_i \\ \beta_1 \\ \vdots \\ \beta_i \\ \vdots \end{pmatrix}}_{\mathbf{b}} \tag{16}$$

The square matrix $\sum \mathbf{C}_i$ in the upper left has a dimension given by the number of global parameters p. It is the sum of the contributions of all measurements. The local parameter values from one track hit form a small square matrix $\mathbf{\Gamma}_i$ of dimension 5×5 and are only connected to the global parameters via $\mathbf{G}_i$. Below $\sum \mathbf{C}_i$, the matrix becomes sparse. This special shape and the fact that we are not interested in the local parameters α_i allows the use of block matrix rules for matrix inversion, which, after successive application, leads to a much smaller equation system:

$$\mathbf{C}'\mathbf{a} = \mathbf{b}' \tag{17}$$

with a matrix $\mathbf{C}'$ of dimension $p \times p$ and a vector $\mathbf{b}' = \sum \mathbf{b}_i$. The size of $\mathbf{C}'$ is small enough for inversion using standard numeric algorithms. It is possible to incorporate boundary conditions into this procedure by using Lagrange multipliers.

The algorithm up to now covers the linear case for which the solution will be found in one single step. Further iterations may become necessary in two cases: First, nonlinearities may be inherent in the problem. This can be covered by local linearization and iterations. Second, there may be some measured tracks with erroneous track association, meaning that the track hits do not belong to the same track. Would the residual be calculated just by using the involved track hits, it would stay on a large value almost immune to optimization. Using iterations, this problem can be tackled too: the residuals for each track, i.e. the sum of residuals from all the track hits belonging to one track, can be calculated in the beginning. After the first iteration, the derivatives in the global parameters are known and therefore the residuals can be adjusted. Tracks with high residuals are most likely bad fits and can be discarded using a proper limit as criterion. Narrowing down this criterion with every iteration, all badly fitted tracks will be removed and the problem gets well behaved.

The implementation of the algorithm as provided by the authors consists of two parts: The first of them, `Mille`, is a data collecting routine (available in C and FORTRAN) which prepares the data files for the second part, `Pede`, which performs the calculations and writes the results into text files. It is a standalone subroutine written in FORTRAN. In CMSSW, the module `MillePedeAlignmentAlgorithm` comprises all the necessary instructions to calculate the values for the matrix **A**, which are fed into a C++-version of `Mille` to write the data files in the proper format for `Pede`. The standalone binary for `Pede` is invoked from within the CMSSW framework. A helper routine reads the text files and transforms the results into CMSSW data structures after `Pede` is finished.

The following list shows a subset of the most important parameters for running Millepede:

`alignParams` An array of strings describing the alignment parameters. Consists of the name of the alignable object and a descriptor of the degrees of freedom to be aligned. The descriptor has the form $uvw\alpha\beta\gamma$. Example: `"PixelHalfBarrelDets,111f00"` aligns the pixel barrel on the level of single modules in direction of the local coordinates u, v and w. The angle α will be fixed at its initial value, β and γ will not be optimized.

`pedeSteerer.method` Sets the solver methods for `Pede`.

`pedeSteerer.options` Sets miscellaneous options and selects the outlier treatment for `Pede`:

- `chisqcut first subseq`
  Sets the cutoff for χ^2-values of individual tracks in factors of the χ^2-value corresponding to three standard deviations. The first value is used in the first iteration, the second value therefore in the second iterations. In subsequent iterations the value will be reduced by the square root. Values below 1.5 will be replaced by 1.
- `outlierdownweighting n` Sets the number of iterations to n. After the first iteration, the default weight of a data point (start weight is $1/\sigma_i^2$) is lowered according to the residual and a distribution function. For the second and third iteration, a Huber function is used, in subsequent iterations a Cauchy function is used.
- `dwfractioncut value` Sets the down-weight value. If values are strongly down-weighted during iterations, a cut can be applied to exclude this values from inclusion in the calculation.
- `bandwidth value`: Sets the bandwidth value for the band matrix used by `sparseGMRES`.

2.5 Aligning the pixel detector

The pixel barrel consists of three layers, therefore a track originating from the beam spot delivers three hits at most. One hit gives two results, one measurement in each of the local coordinates u and v. These six values are not enough to obtain the eight parameters of a helix. Three strategies are possible to align the detector:

Full alignment of the tracker: This is the main intention for using an algorithm like Millepede-II. According to latest results, data from an integrated luminosity of $1\,\mathrm{pb}^{-1}$ (about one week of data taking under start-up conditions) is enough to align the full detector with a precision in the pixel barrel of $6\,\mu$m in the most sensitive direction $r\varphi$. [2]

Fully isolated pixel alignment: In principle, the third dimension of each hit is also known up to the mounting precision, leading to 9 measurements. Therefore it is possible to fit a helix.

On the other hand, a straight line is defined by six parameters: Three for a reference point and three for a vector. This assumption would make it possible to align the pixel detector using either 2 or 3 dimensional information. In the latter case, 3 degrees of freedom would be left, leading to a possibly well behaved numerical situation. The disadvantage is that it neglects the true nature of the tracks being helical and not straight. A simple calculation shows that this assumption is only true for very high momentum tracks in the order of TeV/c.

Pixel alignment using tracker information: As the full data from the tracker is available, some selected information may be used to robustify the problem. E.g. momentum information is much more precise using the whole tracker due to the longer track section under observation. Adding this information as an additional parameter, the curvatures of the tracks are fixed, reducing the number of free parameters.

Full alignment is already available but requires a relative large amount of integrated data collected during several accelerator fill cycles. This is not suitable for monitoring thermal movements of the detector which are expected to appear in time frames of minutes or hours. Fully isolated pixel alignment is not feasible as the robustness of the method is questionable at least, as there is only one degree of freedom left per track. In addition, a double cone of the detector centered around the beam spot would *not* be aligned using this approach because all tracks with an $|\eta|$ larger than covered by the outermost layer will produce less than three hits and therefore will not take part in the alignment. Therefore, isolated pixel alignment using additional information is the approach covered by this study.

The algorithm can be parametrized so that it only optimizes the pixel barrel. In our approach, we can not expect optimum performance as we intentionally refuse to optimize everything besides the pixel barrels. To solve this problem, several options are possible, amongst them are:

Enlarge measurement errors. When calculating the sum for a χ^2, all measurements are weighted by the inverse error. The influence of parameters, which are intentionally left unchanged during the χ^2-minimization, can

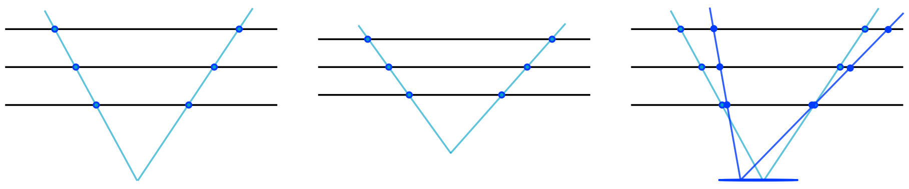


Figure 6: Alignment in w direction. These figures show a cut of the pixel barrel. Assume the figure on the left shows the detector in its nominal shape. Two tracks lead to two hits per layer. If all tracks would have the same origin, the same hit pattern may belong to a shrinked detector as shown in the middle. An alignment in w would be impossible, as the distance between the layers can be arbitrarily chosen. Grace to the spread of the beamspot in z, the origins of the tracks are distributed. This imposes restrictions in radial directions and therefore allows for alignment in w.

be reduced by artificially enlarging the errors by a factor > 1. In the case of isolated pixel alignment, the hit data in the strip tracker would be used as measurements not for optimization. Increasing their errors will downweight their influence in the χ^2-values of individual tracks giving more weight to the measurements from the pixel detector. In other words, the pixel detector is treated as bound to the strip tracker with weak rubber bands formed by the tracks. More details are given in the next section.

Use momentum information. Using the full tracker, good estimates for the track parameters are available from the refitter algorithm. If two or more of these parameters are passed to the alignment algorithm, the track hit information from the pixel detector will suffice to fully describe a helical trajectory.

Both approaches can be implemented in the currently available version of Millepede-II in CMSSW.

The alignment algorithm uses the local coordinate system. Therefore all movements during the alignment procedure are combinations of movements along these coordinates and rotations around them. This choice of coordinate system mimics the expected deviations from the ideal position along a cylindrical shape. On the other hand, the tracks have their origins in the beamspot. If the beamspot would be a point and no magnetic field present, the tracks would follow radial directions. The real beamspot is smeared out, therefore this view does not hold exactly. This situation has implications:

- A movement in the local coordinate u corresponds to a movement in $r\phi$ with respect to the global coordinate system. A movement in (local) w is a movement in (global) r. Therefore, movements in u and w are interdependent.
- Alignment in w (i.e. global r) is only possible because the tracks come from a distributed origin, as explained in figure 6.

2.6 Decoupling with error enlargement

By design, Millepede works best if the algorithm operates on all available parameters for minimization. If not all parameters are free, the χ^2-value of the tracks will have a less prominent minimum becoming almost constant in extreme cases and possibly the overall minimum does not coincide with the best aligned pixel detector. To understand this behaviour, recall formula (1) and split it into two parts: the contribution from parts that are optimized during the procedure ("moving") and parts which are intentionally excluded from being optimized ("fixed"):

$$\chi^2 = \sum_i \frac{r_i^2}{\sigma_i^2} = \underbrace{\sum_k \frac{r_k^2}{\sigma_k^2}}_{\text{moving}} + \underbrace{\sum_l \frac{r_l^2}{\sigma_l^2}}_{\text{fixed}} \tag{18}$$

The sum over the fixed parts of the tracker experiences less changes during the alignment procedure, depending on how strong the misalignments of these parts are. Several cases may occur: a) For very strong misalignment of the fixed parts, the χ^2 is absolutely dominated by their sum. The minimization becomes ill-behaved. b) For a strong misalignment (overall or restricted to the fixed part), the sum dominates the χ^2. The minimum of the χ^2 must not necessarily be at the same parameter set where the moving parts alone reaches its optimal alignment. c) If the misalignment (weighted by measurement error) is within the same order for all parts of the tracker, the minimum of χ^2 is expected to be closer to the optimum. d) If the fixed parts are already close to or at their optimal position, the minimum of the χ^2 is expected to be exactly at the optimal position for the moving parts. This case is close to the situation when all objects take part in the alignment, just carried out separately.

In cases a) and b), the situation seems forlorn as the algorithm will optimize to a ill-behaved or wrong located minimum. Introducing a factor k to intentionally enlarge the errors of the fixed measurements, the χ^2 becomes

$$\chi^2 = \sum_k \frac{r_k^2}{\sigma_k^2} + \sum_l \frac{r_l^2}{k\sigma_l^2} \tag{19}$$

which weights down the influence of the fixed parts. As the algorithm tries to minimize the χ^2, this approach clearly decouples the moving from the fixed parts. The following ranges of operation will be expected:

$1 < k$: The system is still strongly coupled to the fixed part, with $k = 1$ being the same as without increase of the measurement errors.

$1 \ll k \ll \infty$: The system is decoupled enough so that the optimum of the moving parts may be found. For systems with large enough misalignment (and bad alignment performance without decoupling) but not too strongly misaligned, it is expected that the algorithm will find an optimum close to the one of interest in the moving parts.

$k \to \infty$: The system becomes totally decoupled from the fixed parts. If the number of measurements is not enough to fit a track, this will be a underdefined situation and the algorithms behaviour is likely to become unpredictable.

By careful selection of k, a working condition for the algorithm may be found. As the total number of parameters defining such a system is large, finding the best value for k is preferably done by experiment or simulation.

2.7 CMSSW software framework

The software framework is written in C++ and provides data analysis for both Monte Carlo and detector data. All desired action is controlled by configuration files, which are invoked using one executable, `cmsRun`. The framework supplies common data structures and a framework to program modules. Different module types are available:

Filter Checks for criteria. Subsequent modules in the path will only be invoked if the event passes the filter criteria.

Producer Reads data, allows for calculations, alterations and creation of new data.

Analyzer Reads data and allows for calculations, but not for alterations of data.

Looper A special case of a producer. Allows for iterations over events, i.e. defines a loop around the main loop. All alignment algorithm are implemented as loopers in order to iterate.

A module is written as a C++ class (with data structure, constructors, destructor) and consists of at least the following methods:

- `beginJob`: Tasks to be done before event processing.
- Event treatment: This block is named after the type of module, e.g. *analyze* in an analyzer module. This block is invoked for each event.
- `endJob`: Tasks after all events have been processed.

Additional methods are required for loopers: `startingNewLoop` and `endOfLoop`.

Configuration files define the data sources, selects the modules used for a certain task, set parameters and/or redefines them differently from the default settings. The sequence of applied modules is defined in path statements. When `cmsRun` is invoked, the framework instantiates all necessary objects, invokes the `beginJob` methods and starts to loop over the events in the data sources. The event treatment parts of each module are carried out for each module in the defined sequence. After all events, the `endJob` blocks are invoked and the job finishes.

2.8 Monte-Carlo alignment procedure

All results presented here were obtained using Monte-Carlo techniques. The procedure for all alignment studies is straightforward. It uses the following steps, everything supplied by the CMSSW software framework: 1) track generation, 2) track selection, 3) misalignment, 4) refitting, 5) alignment, 6) optional reporting and/or additional iteration.

Track generation is done by using a Monte-Carlo event generator followed by a simulation of the detector. PYTHIA [11] and Geant4 [8] are common algorithms for these tasks. As this takes a reasonable amount of computing time, the tracks for all studies here were taken from previously generated track files specially prepared for alignment studies. [19]

A typical scenario comprised the following steps, all of them carried out within the frameworks defined by a parameter file.

Track selection Some basic cuts and selection criteria can be applied and only tracks, that are fulfilling these, will be used. Typical criteria are:

cut on η, p_T, requiring a minimal number of hits in the tracker and so on. Together with the used track data files, these parameters define the tracks used as input for a particular alignment study.

Misalignment All the tracks used were generated assuming an ideal geometry. In order to simulate a misaligned detector, the geometry of the ideal detector was distorted randomly following distributions described in table 1. The scenario `SurveyLASOnly` is the most pessimistic one. It describes the initial uncertainties after installation and therefore is a realistic approach for the tracker in the begin of regular operation of CMS. This is the benchmark scenario. The scenarios 10, 100 and 1000 pb^{-1} have also been used. They describe the expected situation after a certain time the detector is used in terms of integrated luminosity.

Refitting Tracks are refitted using the same algorithms as in generation. This simulates the detector output under a misaligned situation.

Alignment The alignment algorithm is now invoked. The result is a new set of detector parameters. Depending on the setup, the results are available in root or database files.

This procedures has drawbacks due to idealized assumptions to be kept in mind when it comes to the interpretation of results. In a real world situation the track finding efficiency will drop down and the number of misidentified hits will increase. In addition, the procedure assumes a static tracker throughout data taking. This neglects for example movements due to thermal effects or vibrations. Another drawback is the fact that the data samples available are clean, i.e. there are no events present of other type and they also don't contain any pile-up.

Scenario	SurveyLASOnly		SurveyLASCosmics		$10\,\mathrm{pb}^{-1}$		$100\,\mathrm{pb}^{-1}$		$1000\,\mathrm{pb}^{-1}$	
	spatial	rotational	spatial	rotational	spatial	rotational	spatial	rotational	spatial	rotational
TPB										
Dets	60	270	60	270	60	270	10	45	5	22
Rods	29	20	20	9	10	7	10	7	5	3
PixelHalfBarrelLayers	225/337	10	118/174	9	10	7	5	3	5	3
TPE										
DetUnits	5	100	5	100	5	100	5	11	5	11
Panels	10	200	10	200	10	200	10	22	5	11
Blades	10	200	10	200	10	15	10	15	5	7
HalfDisks	50	1000	50	1000	10	15	10	15	5	7
TIB										
Dets	180	412	180	412	180	412	30	70	10	20
Rods	450	293	275	179	100	65	30	20	10	5
BarrelLayers	750	488	425	277	100	65	15	10	10	5
TOB										
Dets	32	75	32	75	32	75	32	70	18	40
Rods	100	40	100	40	100	40	40	15	18	7
HalfBarrels	60/500	10	60/300	10	60/100	10	20	5	10	2
TEC										
Dets	22	50	22	50	22	50	22	50	22	50
Petals	70	30	70	30	70	30	55	20	40	15
EndcapLayers	60/150	15	50/150	15	60/150	15	30	5	20	5
TID										
Dets	54	250	54	250	54	250	50	230	25	110
TIDRings	185	850	185	850	185	850	50	230	25	110
TIDLayers	350	532	300	456	250	380	25	40	12	20

Table 1: Misalignment scenarios. All values are expressed in RMS. Units are μm for spatial and μrad for rotational degrees of freedom. If two values are given, the first belongs to the $r\phi$-plane, the second to z. Distributions are Gaussian except for TPB Rods and PixelHalfBarrelLayers in the TrackerSurveyLASOnly scenario, where a uniform distribution of ± 50 and $\pm 100\,\mu$m is used, respectively.
Guide to abbreviations: TPB = tracker pixel barrel, TPE = tracker pixel endcaps, TIB = tracker inner barrel, TOB = tracker outer barrel, TEC = tracker endcaps, TID: tracker inner disks.
Sources: [9, 17] and checked against source code in CMSSW

3 Simulations and results

3.1 Software revision

All studies were made using CMSSW version 1.6.x. Work has started in 1.6.7. New releases were used as necessary, the latest being 1.6.12. No substantial changes by the maintainers were observable in the alignment code throughout these versions. This is consistent with the authors observations, as with every change an alignment run has been repeated with no differences in the result between the older and newer version.

Newer releases from the 2.0.x series could not be used, as the data samples required were created with 1.6.x., and they are not compatible with higher releases.

Modifications of the alignment code were necessary. In every case, the changed features were implemented in a transparent and configurable way by the standard CMSSW configuration file mechanism. Therefore, the behaviour of the official release has been available at all the time by not activating the new features in the config files. Consistency to the unchanged behaviour has been checked by repeated runs of setups with known results.

3.2 Data samples

Two types of data samples have been used. Both of them were generated especially for tracker alignment during the CSA07 campaign:[7]

`ALCARECOTkAlZMuMu` Contains pure $Z \rightarrow \mu\mu$-events with a transverse momentum of at least 10 GeV/c. Each event contains exactly two tracks, one per muon.

`ALCARECOTkAlMinBias` Contains miscellaneous minimum bias samples. Tracks have a transverse momentum of at least 1.5 GeV/c with 4 tracks per event on average.

These samples are ideal data sets, i.e. the tracks were reconstructed assuming an ideal tracker geometry. This allows the application of well defined misalignment scenarios and is therefore the right choice for the studies presented hereafter. Samples of AlCaReco-type contain all information needed for tracker calibration and alignment, but generator information has been stripped off.

3.3 Result analysis

The outcome of each simulation has been analyzed using appropriate root scripts. To measure the quality of alignments, the so called remaining misalignment has been calculated. It is defined as the distribution of the differences between the true position and the position after the alignment. Most often, the RMS of this distribution is given. According to the root reference guide, this corresponds to the gaussian σ despite its name. [14]

To correct for global shifts of the detector, a root script developed by Hans-Christian Kästli has been used. It calculates the center of mass and fits the orientation of the barrel. An example is given in appendix D.

[7]CSA: computing, software and analysis. A yearly campaign at Cern for testing the whole workflow as close as possible to real data taking.

3.4 Preliminary studies with standard code

Some runs have been done using unmodified Millepede code, primarily for educational purposes, to get used to the software framework in general and Millepede with its analysis package in particular. I tried to align the pixel on the level of half-barrels and ladders without significant success. Simple analysis led to the hypothesis that the pixel needs to be decoupled from the rest of the tracker. These preliminary studies were quite crude. As all the relevant information is contained in the studies with error enlargement in the limit of error factor 1, no results are given here.

3.5 Studies with error enlargement

To scale the measurement errors for different parts of the detector selectively, the Millepede code has been modified[8]. These modifications allow to specify error scaling factors in the configuration file for each large subunit of the tracker, e.g. TIB or TOB. A factor of 1 denotes no change of the errors, therefore, assigning all parts a factor of 1 leads to the behaviour of the unmodified code. The results of isolated pixel alignment using this code are shown in table 2 and in figure 7. The alignment has been done on module level for all degrees of freedom ($uvw\alpha\beta\gamma$). No information about larger structures have been used. 100 000 events of type $Z \rightarrow \mu\mu$ were used making 200 000 tracks in total. The error scaling has been applied to all subdetectors of the strip detector by the same factor (see later for a judgment of this choice). The error enlargement factor has been enlarged in a logarithmic manner. The following preliminary observations can be made:

- The alignment performance is poor for all scenarios. In some directions the remaining misalignment is even worse after the alignment than before. For example in the LASOnly scenario, the directions u and w experience an increase in remaining misalignment. Only in v direction, a substantial alignment can be observed. In the other scenarios, this is somewhat complementary, as there is an observable alignment in u, but in v and w, only a slight ($10\,\mathrm{pb}^{-1}$) alignment or a decrease (100 and $1000\,\mathrm{pb}^{-1}$) is observable.

  The behaviour in the LASOnly case suggests that u and w are interdependent. This is an expected behaviour, as mentioned in section 2.5.
- The $10\,\mathrm{pb}^{-1}$ scenario shows a clear minimum in all three directions for an error scaling factor of about 50. This suggests that the curve describes the behaviour for the case b) (as described in section 2.6), a misalignment small enough to show a minimum but not necessarily at the optimum for the structure to be aligned. The LASOnly scenario on the other hand clearly matches to the case a). Only in v direction, a minimum is visible at the cost of a stronger misaligned u and w. The 100 and $1000\,\mathrm{pb}^{-1}$ scenarios is case d), where for enlarged errors the information is not sufficient to hold the pixel at its optimum.
- The curves from the scenarios 10 and $100\,\mathrm{pb}^{-1}$ meet each other above an error scaling of about 20. This is not the case for the $1000\,\mathrm{pb}^{-1}$ scenario. This suggests some similarities between the two scenarios. Both scenarios have a structure in common which is almost not present in the

[8]The modified code with annotations can be found in the appendix.

1000 pb^{-1} scenario. The amount of the misalignments applied to the larger structures with respect to the smaller structures is larger in the 10 and 100 pb^{-1} scenario. In the 1000 pb^{-1} scenario, almost all levels have the same misalignment applied. This suggests that the hierarchy of the misaligned structures matters.

- Although a bit contradictory, the results here were done without an error scaled forward pixel. Figure 11 shows why this has been done this way for these initial studies. If the forward pixel is also scaled, the alignment performance increases somewhat but at the expense of shifts in the minima for each direction. As the choice of the error scale cannot be applied with respect to any choice of directions, this is a disadvantage.

When switching to MinBias tracks, the picture changes slightly as can bee seen in figure 9. Again, 100 000 events have been used. A minimal p_T of 2 MeV/c was required. In v, all scenarios start to show a minimum, as can be seen in the insets in the figures.

Table 2: Pixel barrel alignment with $Z \to \mu\mu$ and minimum bias events. Four misalignment scenarios have been studied using 100 000 events. The initial and remaining misalignment have been calculated using the ideal geometries as reference. No correction for global shifts were applied. The remaining misalignment is given from the optimum result, the corresponding error scaling factor is given in the last three columns.
Please note: The *scenarios* refer to the misalignment scenarios as described in table 1 and do not mean that the study has been performed using a data set corresponding to a certain integrated luminosity. Misalignment scenarios reflect the knowledge about the alignment of the detector after a certain time expressed in integrated luminosities.

Events	Scenario	initial misalignment			remaining misalignment					
		RMS / μm			RMS / μm			error scaling		
		u	v	w	u	v	w	u	v	w
$Z \to \mu\mu$	10 pb^{-1}	61.8	58.6	62.4	25.8	26.4	34.0	50	50	50
	100 pb^{-1}	14.5	13.8	14.5	14.1	22.3	19.5	1	1	1
	1000 pb^{-1}	8.8	7.02	8.87	7.05	15.6	9.50	1.2	1	1
	LASOnly	98.2	80.0	97.4	104	55.2	141	1000	200	200
MinBias	10 pb^{-1}	61.8	58.6	62.4	21.8	24.3	27.2	20	100	5
	100 pb^{-1}	14.5	13.8	14.5	14.1	21.7	16.0	1	2	1
	1000 pb^{-1}	8.8	7.02	8.87	7.10	18.9	11.2	1	2.5	1.5
	LASOnly	98.2	80.0	97.4	96.4	39.6	119	500	200	50

The influence of the statistics has also been checked, as is shown in figure 10. Clearly, the situation does not benefit too much from larger statistics above 100 000 events. Therefore, the algorithm seems to find the optimum and the poor alignment performance is an inherent issue in this approach.[9]

3.5.1 Optimization of the error scaling

The results presented in table 2 were obtained using a coupled scheme for the error enlargement, i.e. the error enlargement was the same for all parts, except for the pixel barrel, of course. Some informal runs with different error

[9]A more detailed study has been carried out but the results came in too late to be included here. They can be found in appendix F.

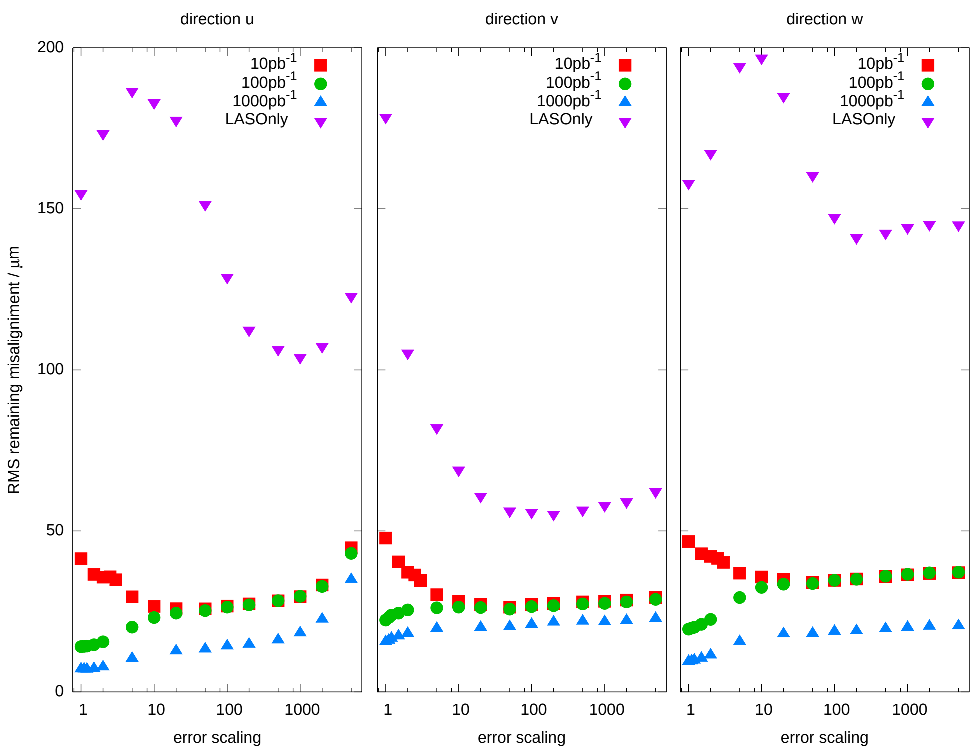


Figure 7: Remaining misalignment of pixel barrel. Alignment with $Z \to \mu\mu$ events and different misalignment scenarios. Shown are the RMS values of the distributions of the remaining misalignment with respect to the MC-truth. All directions shown.

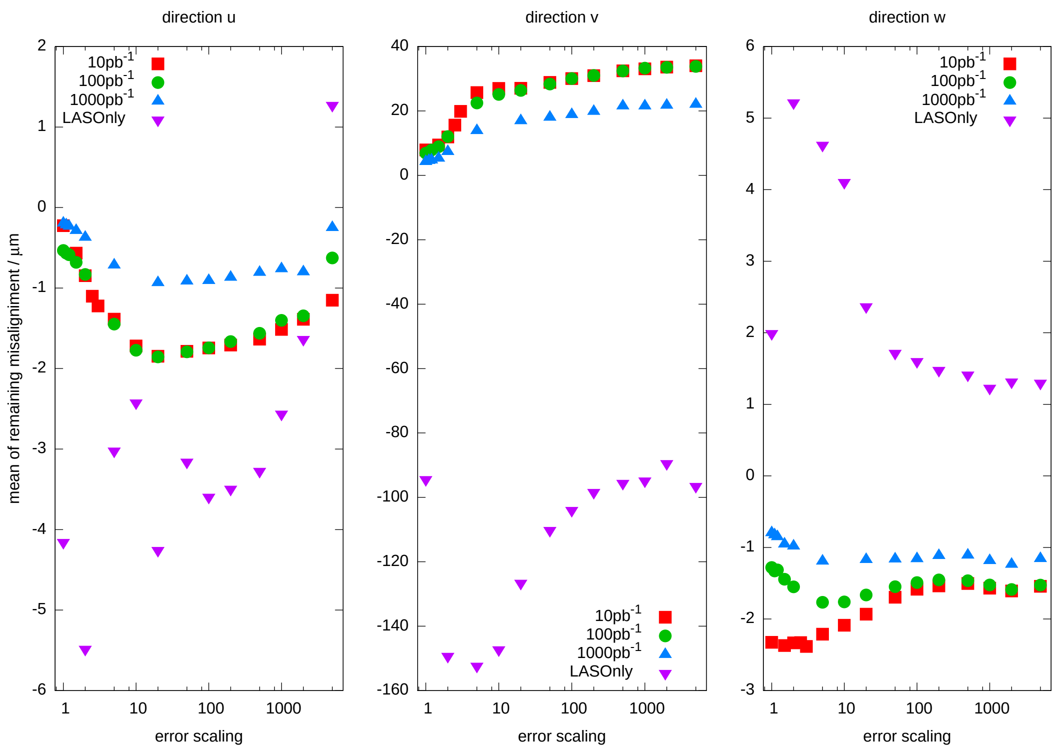


Figure 8: Remaining misalignment of Pixel barrel. The same series as in figure 7. Shown are the means of the remaining misalignment distributions. Note the different scales in the ordinates. The shifts are especially strong in v direction.

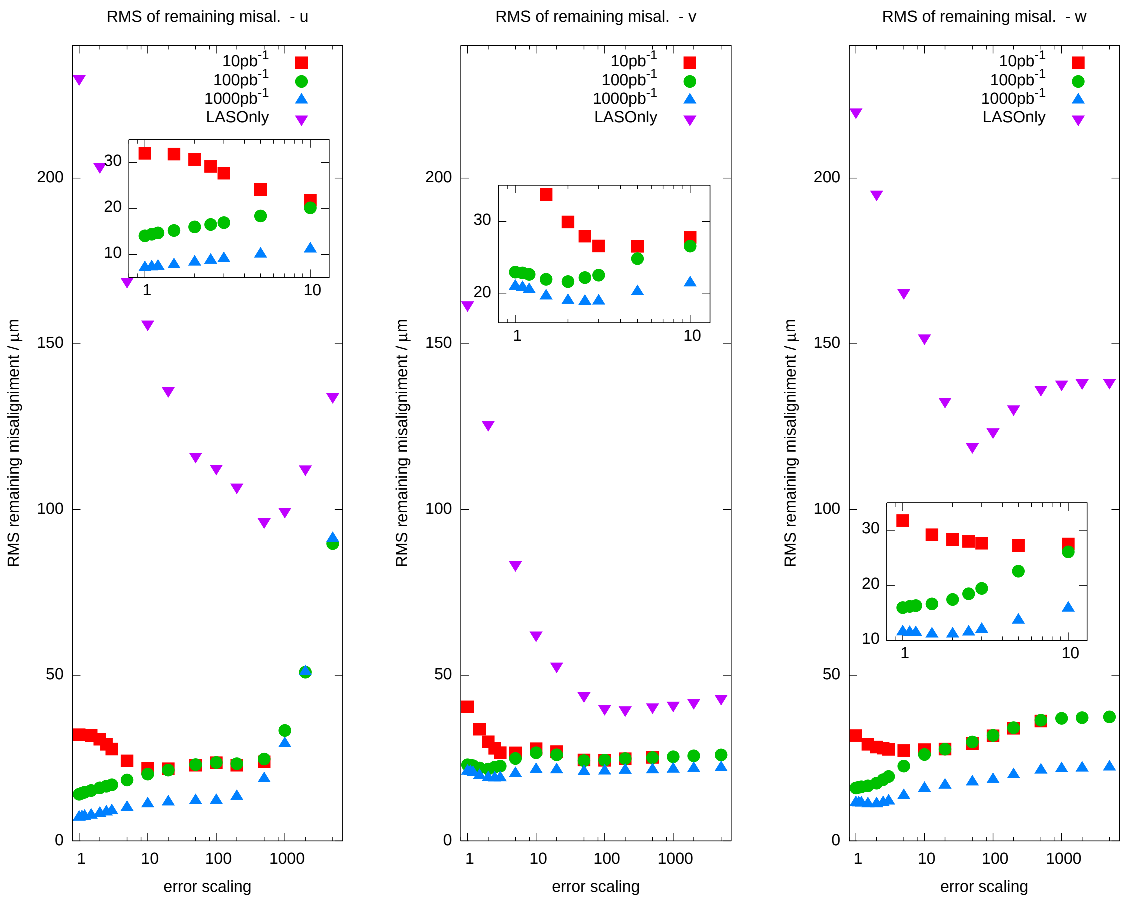


Figure 9: Pixel barrel alignment with MinBias events and different misalignment scenarios.

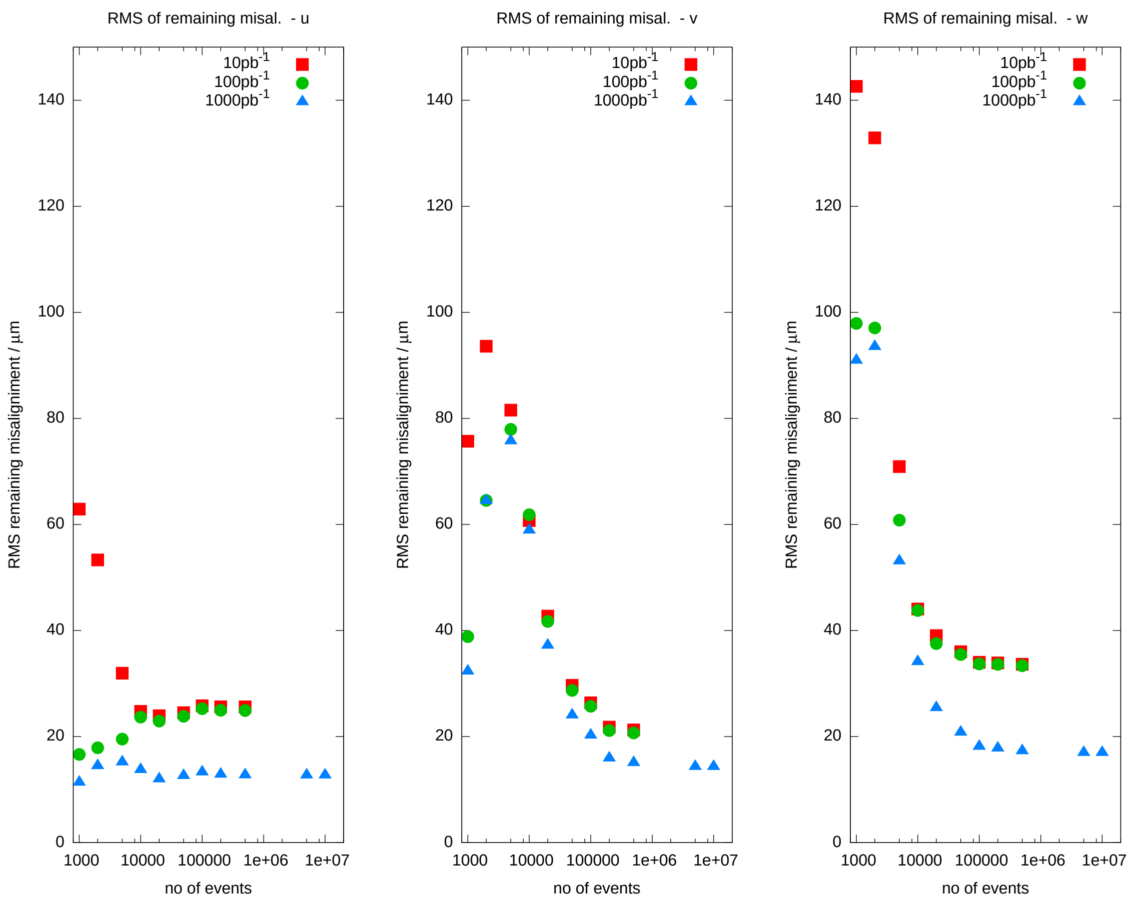


Figure 10: Pixel barrel alignment with $Z \to \mu\mu$ events and different numbers of events selected. Some data points were not calculated due to CPU time and memory restrictions.

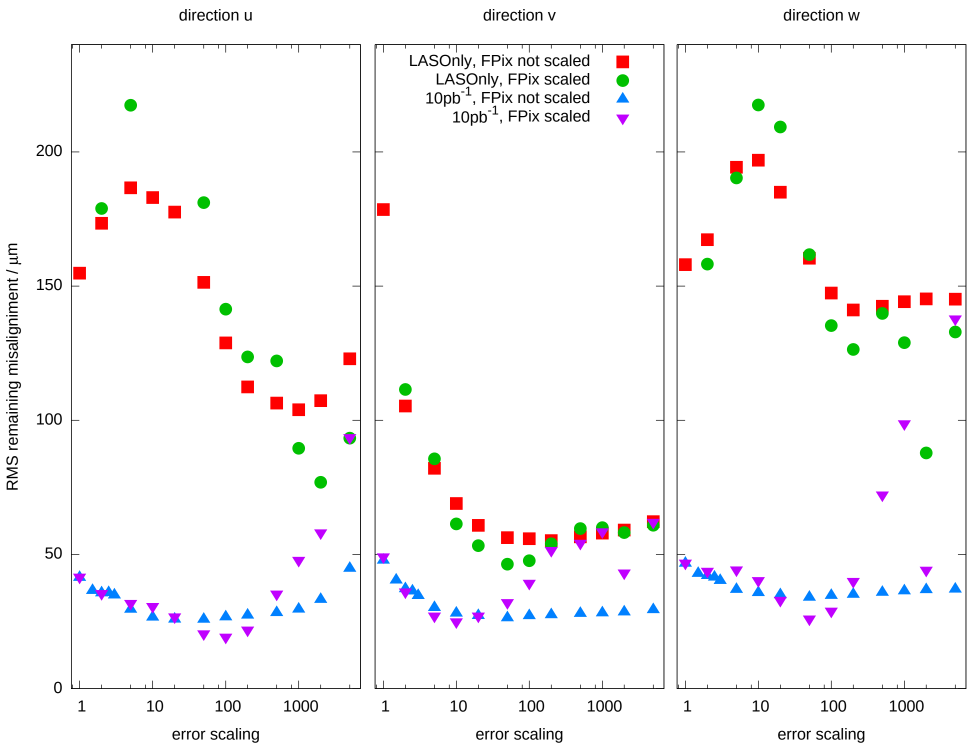


Figure 11: Alignment with and without errorscaled FPix. The data for the cases with unscaled forward pixel are the same as in figure 7.
The series with scaled forward pixel show slightly better alignment performance than the unscaled cases. But the minima are shifted and appear at different values for each direction making it unsuitable to achieve a good alignment in all directions. In the LASOnly scenario, the situation seems to become more unstable, as the curve is not that smooth anymore.

scaling showed that this is not the optimal choice. A short study with an older version of the error enlargement code has been performed manually and showed improvements up the a factor of 2 with respect to the remaining misalignment RMS[10]. This shows that some potential lies in optimizing the error enlargement parameters, but this has to e done for each situation individually, requiring this algorithm to be implemented in software.

3.5.2 Geometric distortions produced by the alignment

When looking at how the average positions of the detector in space are, strong global movements seem to happen (see figure 8). To investigate this, plots of the remaining misalignment vs. the geographical position have been made. One typical example is shown in figure 12. The following noteworthy observations can be made:

- Plots against z show that shifts towards the end of the barrel are larger than in the center. These regions get aligned by tracks having a large η. Possible reasons may be a significant decline in precision of the measurements for such tracks. This is certainly not the case for the pixel – such an assumption would be contradictory to figure 2. Possible sources for such increased measurement uncertainties are the forward detecting components, namely the endcaps of the pixel, and the strip detectors. This region is also prone to multiple scattering, as cabling for the inner part of the tracker are routed between barrel and endcap structures.
- Plots against φ show superimposed sine and cosine shaped distributions. As these plots were all made against the ideal geometry from the Monte-Carlo truth, a global shift would exhibit such a behaviour.

3.5.3 Influence of the forward pixel

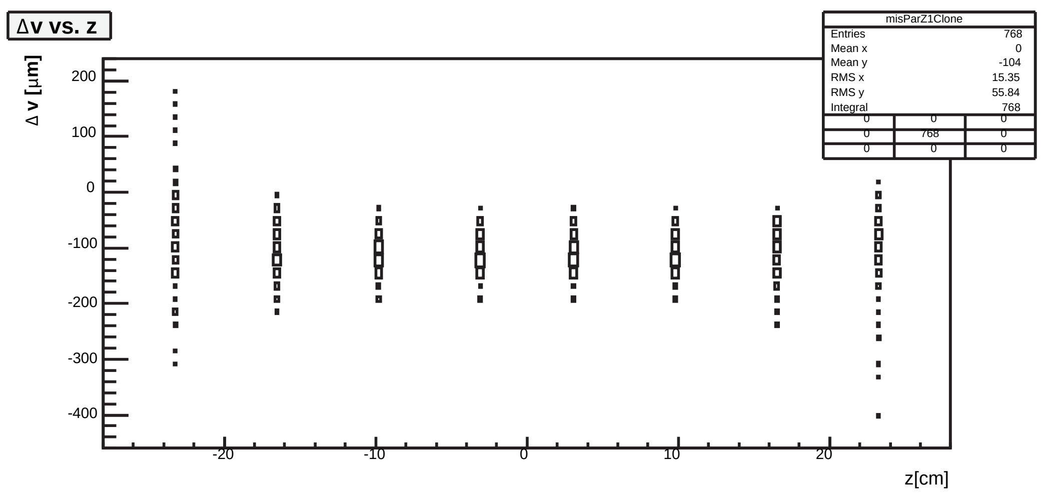
The observations done so far showed clearly that the forward pixel has a strong influence on the pixel barrel alignment. To further understand this effect, a study with scaled misalignment of the forward pixel has been carried out. The results are shown in figure 13, and a digest is given in table 3. These results show, that the forward pixel has a very strong, almost linear influence to the remaining misalignment of the pixel barrel. Only for small misalignments, the influence seems to be less than linear. These results were only used to demonstrate this behaviour.

3.5.4 Fixed forward pixel

The influence of the forward pixel is large. Therefore, studies have been carried out assuming a perfectly well aligned forward pixel.[11] Figure 14 shows the results of a study where the error scale for all parts of the strip detector has been scaled up by the same factor. Using higher factors as 5000 was impossible as numerical errors arose – the algorithm stopped to work due to

[10]See appendix E for more information.

[11]This part presents results created in earlier stages of the work, when we assumed that the influence of the forward pixel is negligible. This lead to some enthusiastic conclusions, as the results were very encouraging. The results shown earlier in this chapter showed clearly that this assumption was wrong. Footing on a unrealistic assumption, these studies enabled me to get a lot of insights which influenced the other studies strongly.



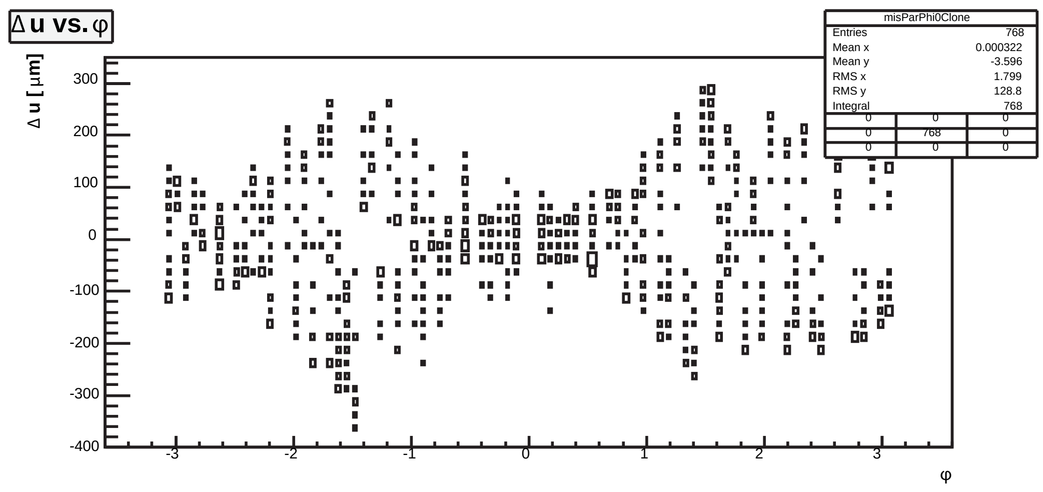


Figure 12: Typical geometric distortions after alignment. These are two typical examples of distortions observed. This is taken from the study shown in figure 7, i.e. LASOnly scenario using $Z \to \mu\mu$ events. The error of the strip detector was scaled by 100. The local movements are plotted against the MC-truth and are not corrected for global shifts.
The plot on the left shows the distortions observed in the local movements in v against the position of the modules in global z-direction. Observe the strong misalignment at the end of the barrel.
On the right, the local movements in u are plotted against the angle φ in the global frame. Two superimposed oscillations are visible. This is an example of a global shift of the detector.
Graphs showing all possible combinations of plots are shown in appendix C.

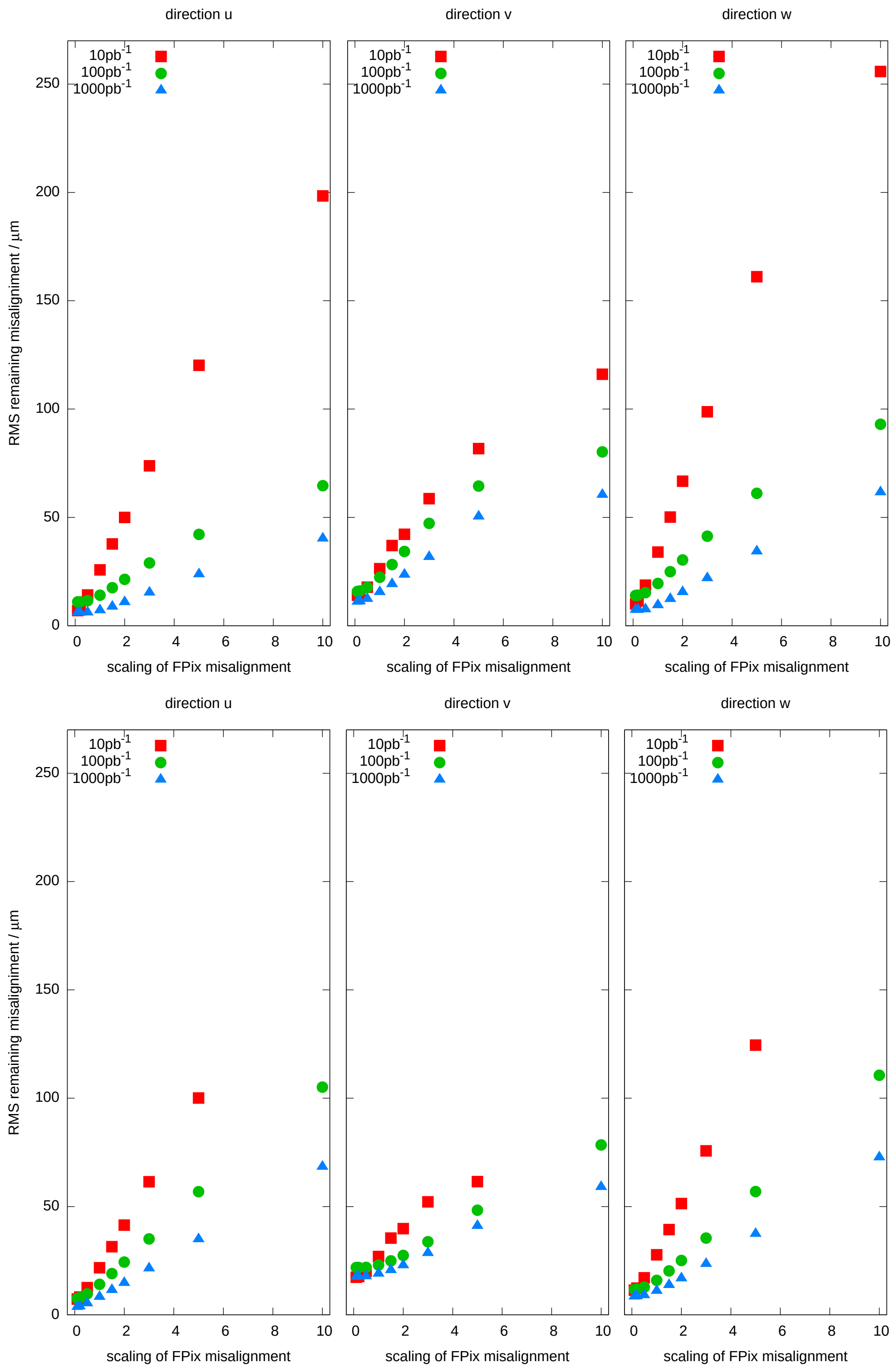


Figure 13: Alignment scaled FPix misalignment. Upper plots show the results using $Z \rightarrow \mu\mu$ events, lower plots show the results using minimum bias events. The scenarios have been as indicated in the legend except that the misalignment magnitude of the forward pixel has been scaled.
The forward pixel clearly has a strong influence.

Table 3: Scaled FPix misalignment. Given here are the estimated slopes of the (almost) linear sections of the plots in figure 13 compared to the misalignments, expressed as the rooted square sum of the misalignment magnitudes as given in table 1. The error scaling for the strip detector was fixed at the following values: $Z \rightarrow \mu\mu$ 50, 1, 1 for 10, 100 and 1000 pb^{-1} respectively and for minimum bias 20, 1, 2. These values were guessed as optimal operating conditions from the results shown in the preceding sections.
The series with minimum bias events show slightly less influence by the misaligned forward pixel. The influence of the spatial and rotational degrees of freedom seem to be mostly orthogonal to each other.

Scenario	slope						rooted square sum of average misalignments	
	$Z \rightarrow \mu\mu$			MinBias				
	u	v	w	u	v	w	position in μm	angles in μrad
10 pb^{-1}	24	16	32	20	13	25	18	224
100 pb^{-1}	7	12	12	11	7	10	18	33
1000 pb^{-1}	4.5	8	6	7	6	6	10	18

division by zero errors. $Z \rightarrow \mu\mu$ and MinBias samples were used, alignment was done in translational degrees of freedom only. Obviously, the decoupling works and produces the desired results. For all spatial coordinates a remaining misalignment of about 10 to 20 μm can be achieved using error scaling in the region of 200 to 500.

Influence of rotational degrees of freedom. Selection of the degrees of freedom for alignment influences the quality of the result. Figure 15 shows the remaining misalignment in the three spatial coordinates for different sets of coordinates allowed to align. The remaining misalignment decreases with more degrees of freedom allowed to align. This behaviour is not surprising: The misalignment scenario applies random movements in position and orientation. Therefore it is reasonable that the remaining misalignment after alignment is smaller when the algorithm is allowed to correct for all movements.

Effect of increased freedom from error enlargement There is one important observation to make: One would expect that the alignment should become better for larger error scaling because the decoupling of the pixel from the strip gets larger. The results contradict this assumption, as in the u-coordinate the remaining misalignment increases above an error scaling of about 500. This could be a sign that too large error scales allow the algorithm to optimize track parameters in an unwanted manner. Millepede does not deliver the optimized track parameters as an outcome, but this does not mean that they are not optimized. In fact they are. High error scaling could lead to some additional freedom in how the track parameters get optimized, as for example tracks with shifted p_T may result in a better fit. This hypothesis needs to be tested.

In order to understand this behaviour, the change of track parameters before and after the alignment were studied. As the optimized track parameters are not available directly from the algorithm, the following approach has been chosen: The track parameters are available as output from the track refitter which runs before the alignment. The resulting optimized geometry from an

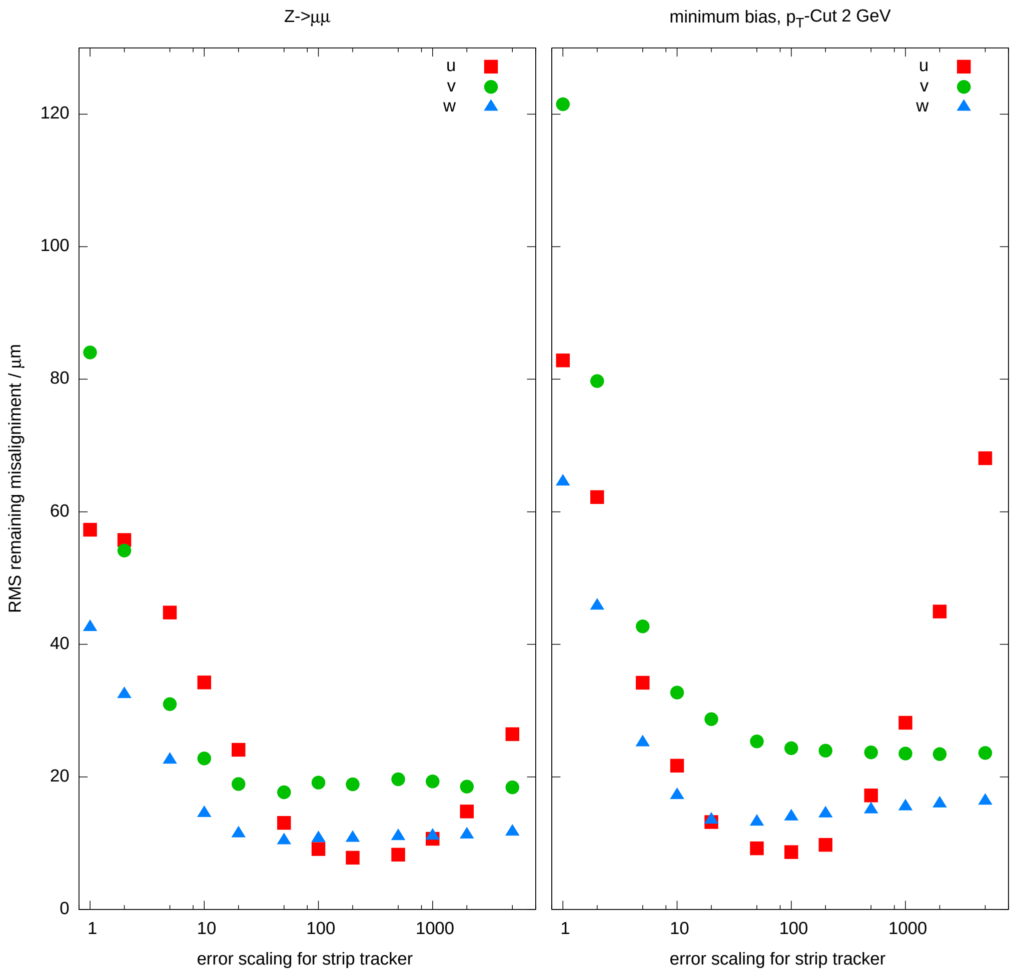


Figure 14: Alignment assuming ideal FPix. In this study, only the spatial degrees of freedom have been aligned. The alignment has been done on the module level, so no higher structures were aligned. 845 412 tracks were used.
Observe the increase in u-direction. As described later and shown in figure 16, this is a consequence of a spread in the p_T-distribution coming from the increased freedom due to the applied error scaling.

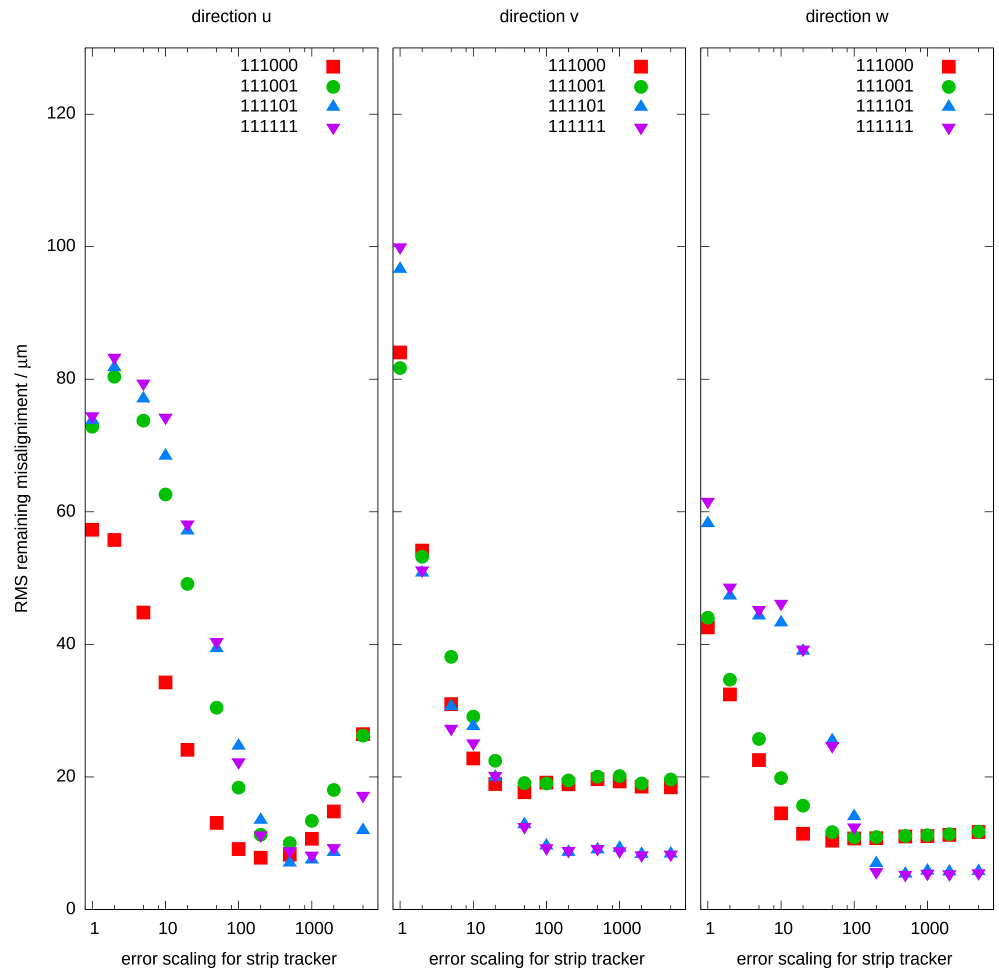


Figure 15: Alignment with ideal FPix, degrees of freedom studied. $Z \to \mu\mu$ events used. The six digit code $uvw\alpha\beta\gamma$ explains which degrees of freedom per module were allowed to align. Reading example: 111001 means alignment done in u, v, w and γ but α and β kept fixed.
Evidently, the more degrees of freedom are allowed to aligned, the better the alignment for higher error scales.

alignment run is stored in a file and used as starting point for a second invocation of the track refitter. The track parameters obtained in this way can be used as estimators for the changed parameters under the assumption that the track refitter after the alignment finds the values, the alignment algorithm applied to the tracks. By design of Millepede, this assumption cannot be proven. The matching between the data out of these two steps is done using the event number and the charge of the tracked particle. Generator based matching is not possible, as the required information is not present in AlcaReco samples. This matching procedure is quite robust for samples like $Z \to \mu\mu$ being clean events with exactly two tracks per event. A loss in matching tracks comes from the fact that high p_T tracks can to some extent observe a change in charge sign if the curvature flips in sign in the refit. This primitive matching procedure is totally unsuitable for MinBias samples, as too many tracks are contained per event and no charge symmetry can be enforced on the data. As a more elaborate matching procedure would have taken too much time to implement with respect to the expected outcome, only $Z \to \mu\mu$-samples were used for such studies.

Figure 16 shows the shift in p_T due to an applied misalignment. For an aligned detector, the spread is small and less than a few 100 MeV/c for most of the tracks. If only the pixel gets misaligned, the shift is less than 1 GeV/c for most of the tracks. But when the whole tracker gets misaligned, the p_T-shift becomes substantial in the order of few GeV/c. This is no surprise, as in a misaligned detector, the curvature of fitted tracks will change. But this is not enough to show that the increase in remaining misalignment RMS with higher error scales is caused by this. The study with increased misalignments will answer this question shortly.

Required luminosity for alignment. Another important question is the number of events necessary to achieve good alignment. A study with both types of events has been done, as shown in figure 17. A value of 200 for the error scaling has been chosen. In order to express the number of events in luminosities or data taking time, the following assumptions have been used:

$Z \to \mu\mu$ The cross section used in Pythia for generation of the events was 1784 pb, the filter efficiency was 0.4633 and the branching ratio is assumed to be 0.03. This leads to a effective cross section of about 25 pb. The following table shows the amount of uninterrupted data taking time required to get 100 000 events:

Luminosity	event rate	time for 100 000 events
$\mathrm{cm}^{-2}\mathrm{s}^{-1}$	s^{-1}	days
$1 \cdot 10^{32}$	$25 \cdot 10^{-4}$	460
$2 \cdot 10^{33}$	$50 \cdot 10^{-3}$	23
$1 \cdot 10^{34}$	$25 \cdot 10^{-2}$	4.6

MinBias Such events are expected to be collected with the trigger rate, which is 100 Hz. 4 tracks with p_T above 2 GeV/c per such event are assumed. 100 000 events are therefore collected in 1 000 seconds.

The overall alignment performance using MinBias events is similar to using $Z \to \mu\mu$, as shown before. Therefore, using MinBias, it is possible to align the pixel barrel detector within 1 000 seconds, *if* the forward pixel is perfectly well aligned. This procedure would be suitable to capture movements of the

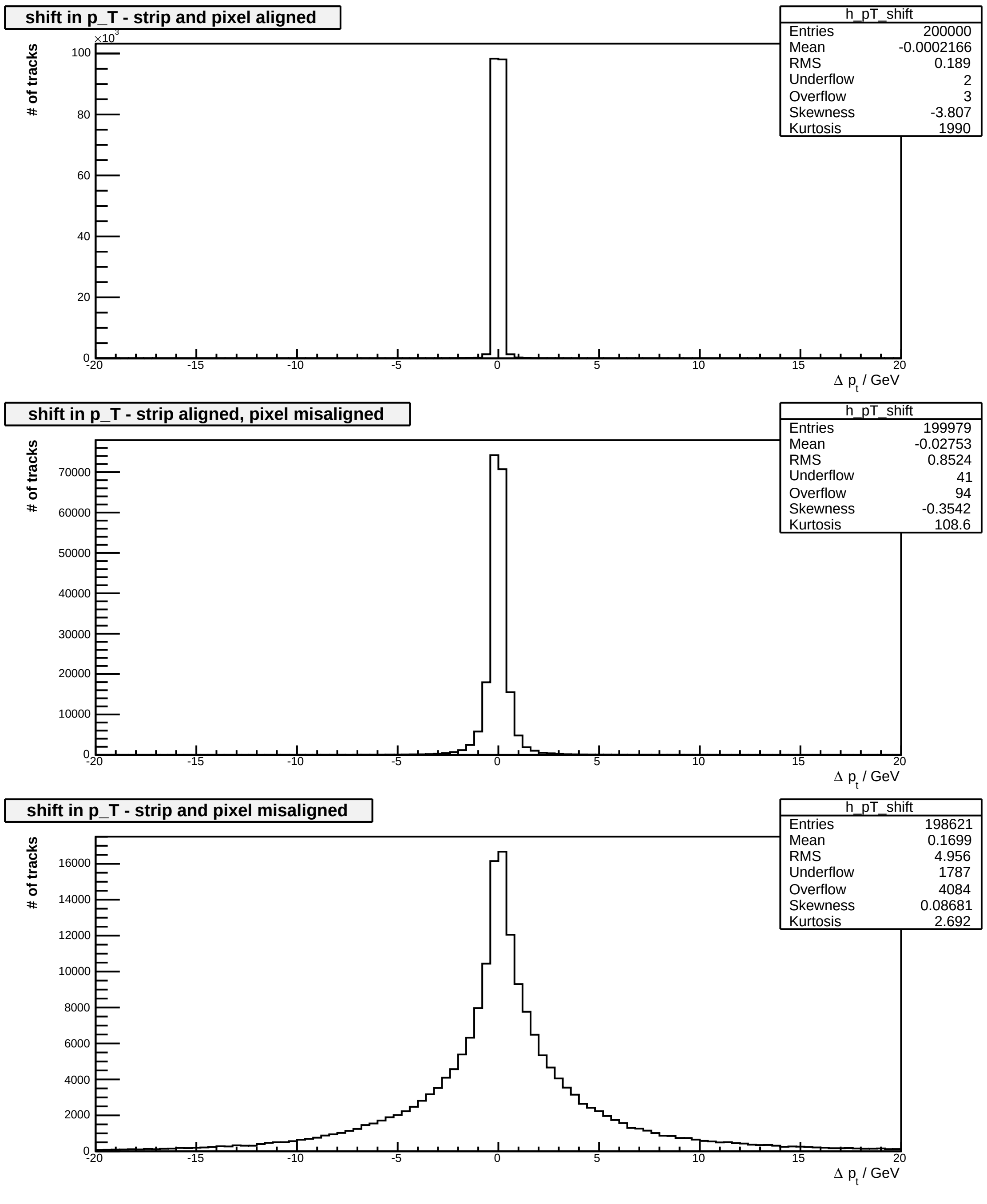


Figure 16: Shift in p_T distribution due to misalignment Shown are the distributions of shift in p_T before and after alignment for error scaling of 200 using $Z \to \mu\mu$ events. Misalignment scenarios were based on the LASOnly scenario. These plots have been obtained by matching the track parameters by event number and charge in the output of the track refitter.

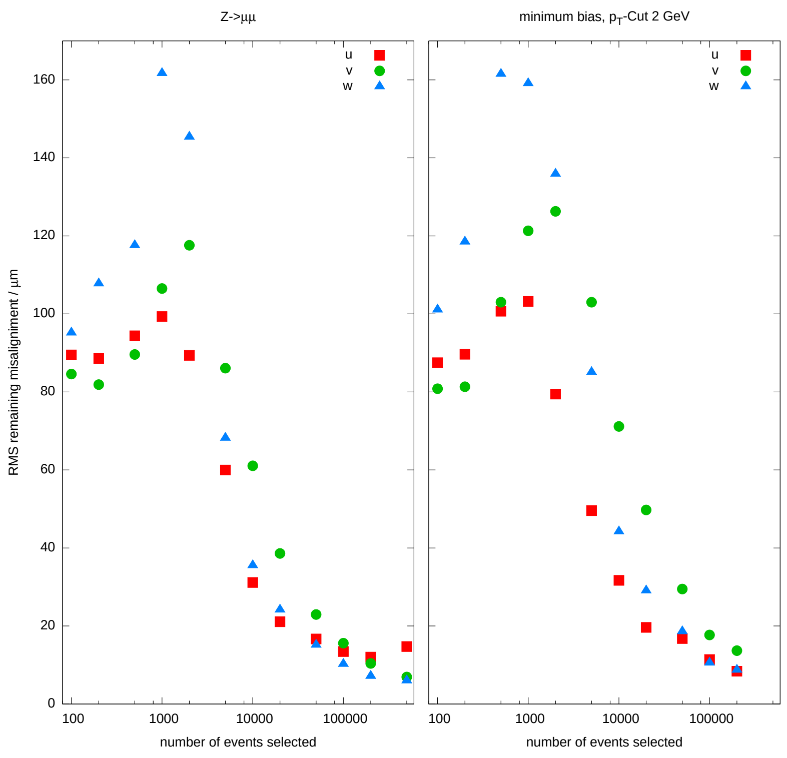


Figure 17: Pixel barrel alignment, remaining misalignment vs. number of events used. In the $Z \rightarrow \mu\mu$-samples there are 2 tracks per event. In minimum bias samples, there are on average 3.5 usable tracks per event when using $\geq 2\,\mathrm{GeV/c}$. The initial increase in remaining misalignment is due to the fact that not all pixel modules got enough hits below about 10 000 events. The algorithm starts its operation around this number of events.

detector due to thermal effects over longer periods of time. A possible scenario could be to take data in slices of 1 000 seconds over a whole beam lifetime and to compare these data with the temperature data collected from the online monitoring system.

Influence of prior alignment. After start-up, the alignment of all parts of the tracker will become better known over time. To simulate this, a series of studies have been performed with scaled misalignments. The scenario was as follows: The strip tracker is assumed to be aligned with a scale in an interval from 0 to 1 with respect to the LASOnly scenario. This scaling has been applied linear to all levels neglecting the knowledge of the misalignment scenarios for higher luminosities. The pixel has always been misaligned as described in the LASOnly scenario. For comparison, an additional study with the whole tracker perfectly aligned has been included.

The results are shown in figure 18. As soon as the error scale is large enough, the initial condition of the pixel detector is wiped out. This is visible for error scales above 100 to 500 (depending on the degree of freedom under inspection). It shows, that the algorithm is robust and has no memory of the initial conditions.

As a side effect the rise in remaining misalignment RMS in u direction can now be understood. First, the increase in remaining RMS for the u-coordinate is present for all initial conditions, the curves merging above a error scale of 500. This means that the effect is induced by the error scaling. The shifts

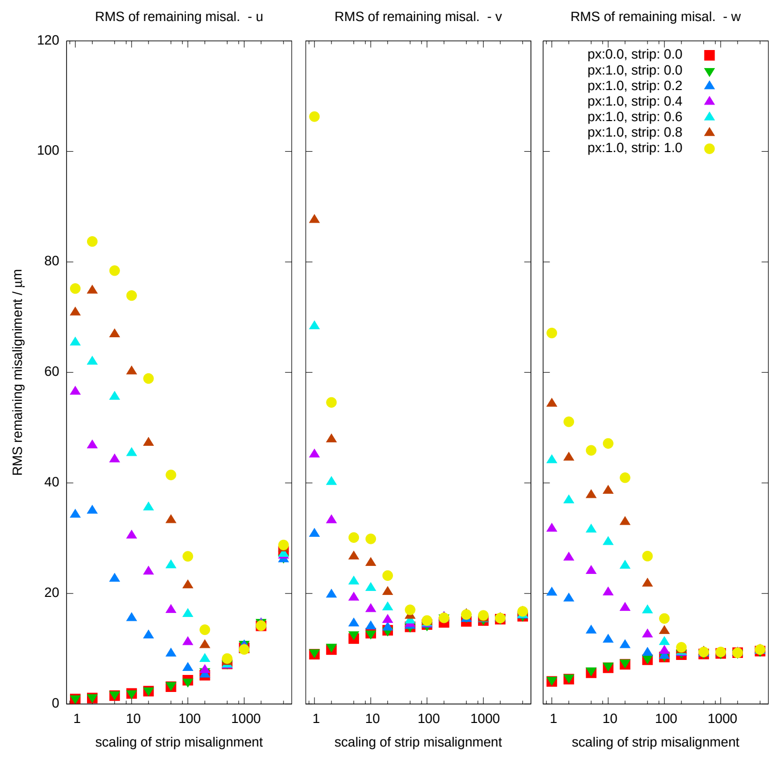


Figure 18: Pixel barrel alignment with $Z \to \mu\mu$ events, remaining misalignment vs. error scaling for different misalignments. The misalignments are given in factors with respect to the LASOnly scenario. Example: px:1, strip 0.4 means pixel barrel fully misaligned as in the scenario, strip tracker misaligned by 0.4 the RMS of the scenario. The forward pixel is assumed to be perfect.
In u there is clearly a minimum visible, around an error scaling factor of 500. This is the same shape as found in fig. 14. Obviously, all curves join together above a specific error scaling factor.

in p_T of these studies are plotted in figure 19. As the measurement of p_T is dominated by the strip tracker, it is not surprising that as soon as the strip tracker gets misaligned, no structure is visible with respect of the error scaling. But in the perfectly aligned detector, clearly an increase in the RMS of p_T-shift is visible, supporting the assumption that the additional freedom imposed by the increased error leads to a change in p_T-distribution. As the u direction is expected to be sensitive to p_T measurements, it is clear that this direction should react most sensitive to changes in the p_T-distribution.

Operating conditions for alignment. The misalignment scenarios from table 1 are based on some assumptions which are thought to be sound. This study shows if the alignment algorithm can also handle larger misalignments than described in the scenarios. The strip tracker has been assumed to be aligned according to the LASOnly scenario but the pixel barrel misalignment is scaled from 0.1 to 5.0. The highest value corresponds to movements of the half barrels of up to 0.5 mm. The results are shown in figure 20. Above a scale of the misalignment of about 2, the remaining misalignment starts to increase. A second iteration has been executed, resulting in an optimal aligned pixel detector for all initial misalignments. The reason for this behaviour is likely that with large misalignments the linearization of the least square problem is no longer adequate. As expected, an additional iteration solves the problem.

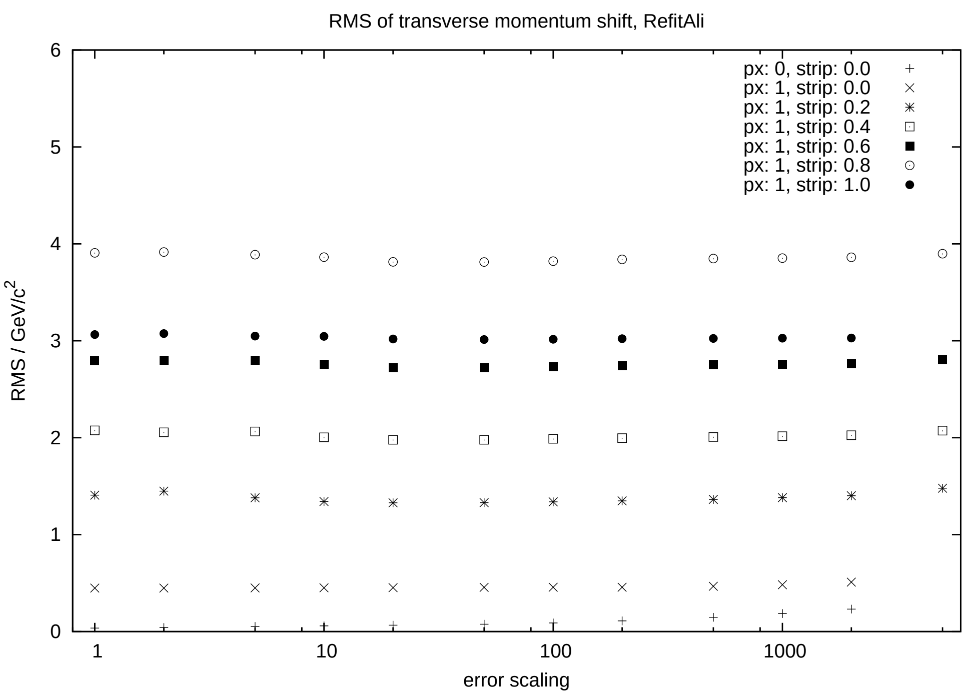


Figure 19: RMS of p_T vs. error scaling for different misalignments. This data comes from the same studies as in 18, see there for explanation of the legend. Shown are the RMS values of the shift in p_T. It is clearly visible that the p_T-measurement is dominated by the strip detector. Only for the perfectly aligned detector an increase in p_T-shift is visible.

3.5.5 Possible use for monitoring the pixel geometry

The very stable behaviour of the algorithm in the case of a perfect forward pixel and large misalignments in the pixel barrel, suggested that even though the alignment performance of the real-world scenario is far from being good, it would still be possible to monitor movements. This would work, if the outer parts (forward pixel and strip detector) project their distortions into the pixel barrel. As the strip detector is expected to be much more stable and less prone to thermal deformations, this assumption probably holds.

If this proposal holds, the alignment algorithm should find the same positions for all alignables (within a certain error bandwidth) without any coupling to the initial misalignment of the pixel detector. Figure 21 shows the results from a preliminary study where the pixel misalignment was scaled linearly. Clearly for misalignment scales below a threshold of 4 to 10, depending on the overall misalignment scenario, the remaining misalignment RMS of the pixel barrel stays almost constant. Between neighboring points with almost no change, the positions have been checked and distributions of less than 1 μm have been observed (average movements). This study has a major methodological flaw, as only the misalignment of the pixel has been scaled in a correlated manner. To further proof the proposal, additional studies with independent misalignments need to be carried out.

3.5.6 Ways to enhance the alignment

Assuming that the reasoning made so far is valid, the following ideas should help to increase the alignment performance:

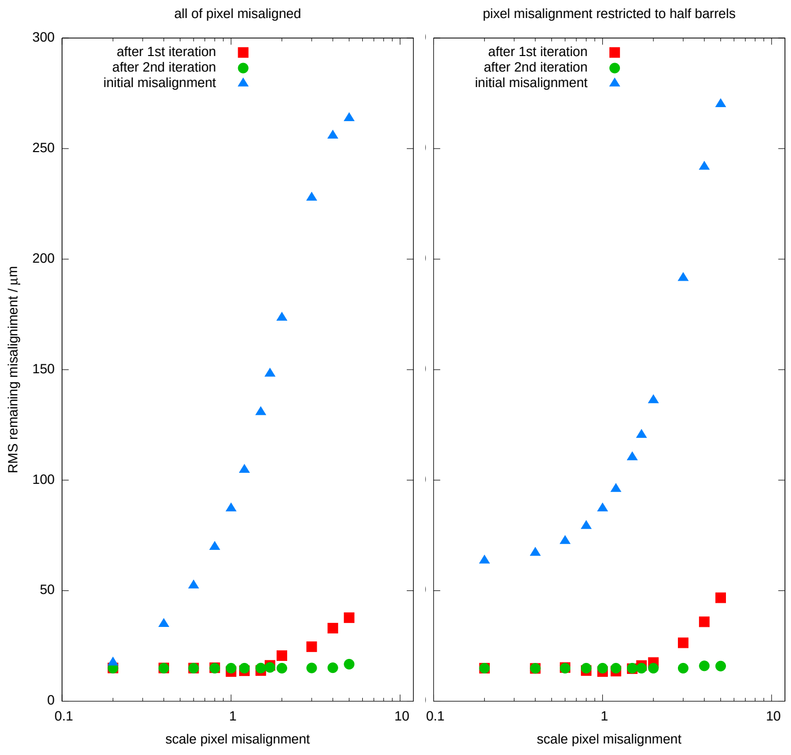


Figure 20: Remaining misalignment RMS in u vs. scale factor of misalignment. This study covers the misalignment factor beyond the LASOnly scenario. Used were $Z \to \mu\mu$-events with an error scaling of 200. Factors larger than 5 led to a breakdown of the algorithm. Only the pixel misalignment was scaled. In the second figure the pixel misalignment scaling was restricted to the half barrels in order to simulate the most realistic case where large movements are likely expected. A factor of 5 in this case corresponds to movements of up to 0.5 mm, see table 1. Above a scale of 2, the alignment algorithm needs two iterations to achieve its full potential, but in any case the detector can be aligned to the same level for each initial misalignment.

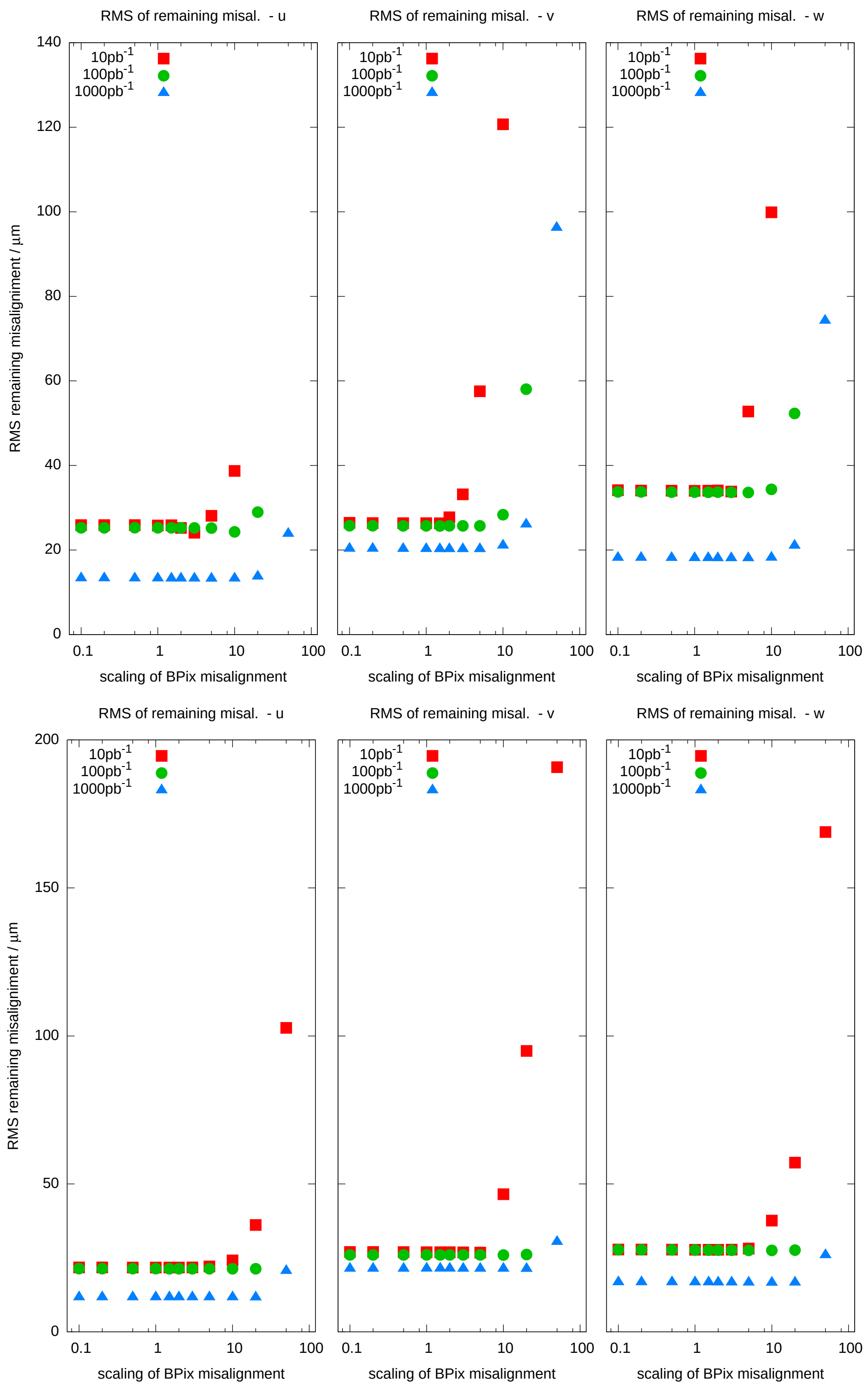


Figure 21: Alignment with scaled pixel barrel misalignment. Upper plots show the results using $Z \to \mu\mu$ events, lower plots using minimum bias events. The scenarios have been as indicated in the legend except that the misalignment magnitude of the forward pixel has been scaled.

Using survey information. For the pixel endcaps, a thorough geometrical survey has been carried out. For the pixel, barrel being a more difficult structure, a survey is more complicated, and therefore much less information is available. The distances between individual modules on the ladders are known up to a precision of $1.2\,\mu$m [3] leading to a survey precision along the ladders of $20\,\mu$m in u and $10\,\mu$m in v.

Survey information has one important drawback. The data was valid at the time when the survey has been carried out. No information is available on how the positions of the module and larger structures change during installation and use.

Improve endcap alignment. A perfectly well aligned forward pixel would indeed solve the problem. Studies with this assumption showed an impressive alignment performance. But this task is still open. In alignment studies carried out by the alignment group, the forward pixel always showed only moderate alignment performance at most. As there are only two large structures per side, unusual tracks like beam halo tracks would be helpful, so there would be events hitting all disks. Unfortunately, no such events will ever be available because these tracks are almost parallel to the beam pipe. Triggering on them is not possible as the hit only the forward pixel.

Using a beamspot constraint. This would enforce the location of the primary vertex of a track to be in the region of the beam spot. Its location, expressed as its "center of mass", can be calculated. Routines for doing so are available in CMSSW. A beamspot constraint can be implemented in two stages of the alignment procedure, namely in the refitter or in the alignment algorithm. The first approach was available in an older version of CMSSW but became orphaned and is therefore useless. The second approach is implemented in a newer version (2.0.x). As there were no compatible data samples available at the time this work was done, this could not be checked. Compatible samples are expected to be available in the near future.[12]

The principle of the implementation is as follows: The track is propagated from the innermost hit to the point of closest approach to the beamspot. At this point, the trajectory parameters are calculated and used as additional hit with an error equal to the beamspot size. As the longest elongation of the beamspot along the z-axis is very large with respect to the xy-plane, the error in the z-direction is regarded as infinitely large. The implementation has two modes of operation. In one mode, the beamspot is treated as a simple constraint, meaning that the primary vertex is required to be in the beamspot. In a second mode, the beamspot becomes the role of an additional alignable.

Using a momentum constraint. As shown in figure 16, the increased freedom for the track parameters imposed by the error enlargement leads to smearing of the p_T-distribution of the tracks. Introducing a momentum constraint for the tracks would the alignment algorithm restrict to geometrical shifts in the track parameters.

[12] A working backport of the version for 2.0.x is now available and first encouraging results are presented in appendix F.

3.6 Studies with use of momentum information

Recall that during a Millepede run, derivatives of the five helix parameters are calculated for each hit along a track. Two of them, the impact parameters, contain locational information of the helix. The other three parameters contain information on the momentum. These are the (scalar) signed inverse of the momentum and two angles. If these three parameters are not passed to Millepede, the algorithm will no longer optimize the momenta of the tracks, which was one of the problems described earlier in this chapter. This behaviour has been implemented. As in this approach the strip information is no longer needed after refitting, only data from the pixel barrel hits were passed to Millepede but demanding that the track has at least three hits. The results are given in table 4. The forward pixel was not misaligned.

Table 4: Preliminary results with fixed momentum.

Event type	remaining misalignment RMS		
	u	v	w
$Z \to \mu\mu$	21.2	13.4	36.1
MinBias	45.8	47.2	79.9

This data compared to what has been achieved in the error scaling case together with a perfectly aligned forward pixel looks worse. This preliminary study has been carried out at the point when we had to decide which path to follow. As at that point the error scaling showed far better results, we postponed any further investigations. A first look at the geometrical distortions showed, that we experience the same cases as already reported for the approach using error enlargement.

For future studies, the implementation of a beamspot constraint seems to be helpful. This follows from the consideration, that in the error enlargement, the pixel barrel is attached to the surrounding structures by weak rubber bands. This constraint, as soft as it may be, ensures that the barrel should not experience too large global shifts or distortions. A beamspot constraint forces the tracks to have at least the primary vertex located at a defined position.

3.7 Monitoring of alignment without knowing the MC-truth

In Monte-Carlo studies, the misaligned geometry is available, the so called MC-truth. In reality, the misaligned geometry is a deliverable not known beforehand. Therefore, when it comes to the real case, another indicator needs to be available to see if the detector is aligned. For the full tracker, such measures are already in use. The most noteworthy is the χ^2 of the refitted tracks. The smaller the distribution of these χ^2 values is, the better aligned the detector. In our case, this would lead to a domination of a badly aligned strip detector, the reason why we implemented error enlargement.

In order to get a measure using pixel barrel data alone, the following approaches are possible:

- Reconstruct helices using three dimensional information from the barrel pixel. This is indeed possible as sketched out in section 2.5 but would re-

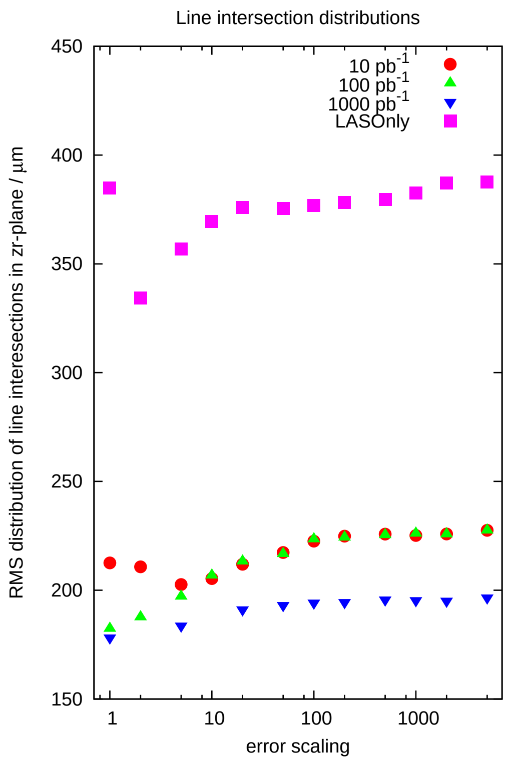


Figure 22: Monitoring alignment using intersecting lines in rz-plane. This plot has been produced using a set with 50 865 $Z \rightarrow \mu\mu$-tracks. The distributions of the distances of the intersections in the rz-plane were histogrammed. The RMS of these distributions is on the ordinate of this plot.
The data used for this plot are the same as for figure 7. The similarities between these two plots are obvious. For example, the two curves for the 10 and 100 pb^{-1} scenarios show the same features and start to merge for values above an error scaling of about 10. One notable exception is the LASOnly scenario, where this plot does not show the same features as the MC-truth.

quire the implementation of a specialized fitting algorithm. Using events with tracks sharing the same primary vertex, the pairwise distance of closest approach can be calculated over all tracks. The better aligned the detector is, the narrower the distribution of these distances will be.

- In the projection to the rz-plane, helices become straight lines. Using the same types of events as in the previous approach, the intersections of these lines can be calculated and histogrammed.
- In the projection to the $r\phi$-plane, helices become circles. The intersections of these circles can be calculated and histogrammed as above.

The two latter methods have been implemented. The approach using straight lines showed good results while the approach using circles resulted in useless distributions. The results from the straight lines using a set of $Z \rightarrow \mu\mu$-tracks are shown in figure 22. The shapes look similar to the ones observed in the corresponding studies where the remaining RMS using the MC-truth has been plotted (figure 7). As the approach suggests, these plots should be sensitive for the remaining misalignment in $r\phi$. This could not be proven using just this data. More studies using restricted alignment in different directions would be necessary.

4 Conclusion and outlook

This study showed, that isolated barrel pixel using the presented approach with error scaling alignment may be possible. The results are reasonable but not overwhelming. For the most realistic scenario at start-up, the LASOnly scenario, the alignment is not better than $40\,\mu$m in the most sensitive direction. An improvement is expected to happen using optimization techniques for the error scaling parameters, but still then $30\,\mu$m is the best for LASOnly (100 000 minimum bias events). A full alignment of the detector achieves far better results but relies on much more integrated luminosity.

The special case of a perfectly well aligned forward pixel detector showed that the approach using error enlargement works fine under certain – somewhat unrealistic – boundary conditions. Despite that fact, this result shows that a well aligned forward pixel has a strong influence to the alignment approach using error enlargement. Improving the alignment of the forward pixel is clearly an option.

A lot of open questions still exist. Monitoring the relative movements of the pixel barrel is probably possible, but this will require further studies. The optimization of the error scaling parameters shows an enhancement of the results, but needs to be implemented into the alignment code. Remaining global shifts are also an issue, which needs further investigation – the tool already available is very helpful to analyze the output of alignment procedures but cannot straightforwardly be integrated into the alignment procedure. Constraining the momentum needs to be further studied as well as the potential of a beamspot constraint.[13]

5 Acknowledgments

This work would have been impossible without the help of the following people:

- Hans-Christian Kästli, Roland Horisberger and Urs Langenegger for making this work possible and their excellent supervision. It was a pleasure to work with you.
- Gero Flucke for help in introducing me to CMSSW and Millepede and answering a lot of questions by email.
- Silvan Zenklusen for providing me a style file for root to make the ugly looking default much better.

[13]As latest results show, a beamspot constraint improves the result. See appendix F.

References

[1] *Status BPIX deformation studies*, 2007. Talk given at the Pixel General Meeting on 30 April 2007, available on http://indico.cern.ch/.

[2] *Alignment: CSA08, TIF and Beyond*, 2008. Draft of talk given at Cyprus CMS Week on 22 June 08, available at http://indico.cern.ch/.

[3] *Final BPix Survey*, 2008. Talk given at the Pixel General Meeting on 10 July 2008, available on http://indico.cern.ch/.

[4] Unknown author. Introduction to helical track manipulations. Technical report, JLC Physics Group, KEK, Tsukuba 305, Japan, 1997. Downloadable from http://www-jlc.kek.jp/subg/offl/.

[5] Volker Blobel. *Millepede II – Linear Least Squares Fits with a Large Number of Parameters.* Institut für Experimentalphysik, Universität Hamburg, 2007. Draft, downloadable at http://www.desy.de/~blobel/Mptwo.pdf.

[6] Volker Blobel and Erich Lohrmann. *Statistische und numerische Methoden der Datenanalyse.* B.G. Teubner, Stuttgart, Germany, 1998.

[7] The CMS collaboration. The CMS experiment at the CERN LHC. *Journal of Instrumentation*, 3(S08004), 2008.

[8] The Geant4 collaboration. http://geant4.web.cern.ch/geant4/.

[9] N. de Filippis, T. Lampén, F.-P. Schilling, A.Schmidt, and M. Weber. Update of misalignment scenarios for the CMS tracker. CMS Internal Note CMS IN 2007/036, CMS collaboration, 2007.

[10] D. Acosta et.al. CMS Physics Technical Design Report: Vol I: Detector performance and software. Technical Report CERN/LHCC 2006-001, CMS TDR 8.1, CMS collaboration, 2006.

[11] Torbjörn Sjöstrand et.al. http://home.thep.lu.se/~torbjorn/Pythia.html.

[12] G. Flucke, P. Schleper, G. Steinbrück, and M. Stoye. A study of full scale CMS tracker alignment using high momentum muons and cosmics. CMS Detector Note CMS NOTE-2008/008, CMS collaboration, Institut für Experimentalphysik, Universität Hamburg, 2008. Draft.

[13] R. Frühwirth, T. Todorow, and M. Winkler. Estimation of detector alignment parameters using the kalman filter with annealing. *J. Phys. G*, 29:561–574, 2003.

[14] ROOT Reference Guide. http://root.cern.ch/root/Reference.html.

[15] V. Karimaki, A. Heikkinen, T. Lampen, and T. Linden. Sensor alignment by tracks. *arXiv:physics/0306034v2*, 2003.

[16] D. Kotlinski and S. Cucciarelli. Pixel hit reconstruction. CMS Internal Note CMS IN-2004/014, CMS collaboration, Paul Scherrer Institut, Villigen, Switzerland, 2004.

[17] T. Lampén, M. Weber, N. de Filippis, and A. Schmidt. Misalignment scenarios for the startup conditions of the CMS tracker. CMS Note CMS IN-2007/061, CMS collaboration, 2007.

[18] A. Strandlie and W. Wittel. Propagation of covariance matrices of track parameters in homogeneous magnetic fields in CMS. CMS Note CMS NOTE 2006/001, CMS collaboration, 2006.

[19] The CMS tracker alignment group. Twiki web page. https://twiki.cern.ch/twiki/bin/view/CMS/TkAlignmentMC.

Appendix

A Code changes

All the changes were done in `MillePedeAlignmentAlgorithm.cc`. Only the relevant methods are presented in the following listings.

Error scaling: While looping over all alignables, the current alignable is accessed using the pointer `alidet`. The *switch*-statement after line 47 applies the error scaling depending on `subdetId` and what is defined as scaling factors in the configuration file. In line 57 the method `callMille` is invoked, where the scaling factor for the error, `sigmaScale`, is passed. In line 13 of this function (listing 2) the error scale is applied.

Momentum constraint: This is implemented in function `callMille` (listing reflst.errscal3) in the lines after 19. If the boolean parameter *forceMomentumConstraint* is set to true, only the two parameters containing locational information are passed to the data set used for the optimization. In addition, the data used for the optimization can be further restricted using the parameter *omitStripDetector*. If set to true, only data from the strip detector (barrel and endcap) is used (see line 19 in listing 1).

Pixel hit data: To reconstruct the lines and circles for monitoring the alignment as described in section 3.7, pixel hit data is required. When the parameter *writePixelGlobalHits* is set to true, a root file is written containing the required data. Listing 3 shows the relevant part from the initialize method where the root file and its data structure is generated.

Listing 1: Method `addGlobalDerivatives`

```
1  int MillePedeAlignmentAlgorithm::addGlobalDerivatives
2  (const ReferenceTrajectoryBase::ReferenceTrajectoryPtr &refTrajPtr, unsigned int iHit,
3   const TrajectoryStateOnSurface &trackTsos, AlignmentParameters *&params,
4   unsigned int &nPxHits)
5  {
6    params = 0;
7    theFloatBufferX.clear();
8    theFloatBufferY.clear();
9    theIntBuffer.clear();
10
11   const TrajectoryStateOnSurface &tsos =
12     (theUseTrackTsos ? trackTsos : refTrajPtr->trajectoryStates()[iHit]);
13   const ConstRecHitPointer &recHitPtr = refTrajPtr->recHits()[iHit];
14   // get AlignableDet/Unit for this hit
15   AlignableDetOrUnitPtr
16    alidet (theAlignableNavigator->alignableFromGeomDet(recHitPtr->det()));
17   // suppress propagation of strip tracker data to mille by config
18   if(theConfig.getParameter<bool>(”omitStripDetector”)
19    && alidet->geomDetId().subdetId() > 2) return 0;
20   const bool is2DHit = this->is2D(recHitPtr);
21
22   // FRANK: Abschnitt um Hitdaten in globalen Koordinaten zu erhalten
23   if(theConfig.getParameter<bool>(”writePixelGlobalHits”)
24    && alidet->geomDetId().subdetId() == 1)
25   {
26       LocalPoint myPoint = recHitPtr->localPosition();
```

```
        AlignableSurface myAliSurf = alidet->surface();
        GlobalPoint myGlobalPoint = myAliSurf.toGlobal(myPoint);
        if (nPxHits < 3) // protect from case when there are more px hits by error
        {
          pgh_x[nPxHits] = myGlobalPoint.x();
          pgh_y[nPxHits] = myGlobalPoint.y();
          pgh_z[nPxHits] = myGlobalPoint.z();
        }
        nPxHits++;
    }

    if (!this->globalDerivativesHierarchy(tsos, alidet, alidet, is2DHit,// 2x alidet, sic!
                            theFloatBufferX, theFloatBufferY, theIntBuffer, params)) {
      return -1; // problem
    } else if (theFloatBufferX.empty()) {
      return 0; // empty for X: no alignable for hit
    } else {
      // experimental FRANK : blow up errors depending on object hierarchy
      double sigmaScale = 1.0 ;
      // now scale sigmas according to objectId
      switch ( alidet->geomDetId().subdetId() )
      {
          case 1: sigmaScale = theConfig.getParameter<double>("sigmaScalePXB"); break;
          case 2: sigmaScale = theConfig.getParameter<double>("sigmaScalePXE"); break;
          case 3: sigmaScale = theConfig.getParameter<double>("sigmaScaleTIB"); break;
          case 4: sigmaScale = theConfig.getParameter<double>("sigmaScaleTID"); break;
          case 5: sigmaScale = theConfig.getParameter<double>("sigmaScaleTOB"); break;
          case 6: sigmaScale = theConfig.getParameter<double>("sigmaScaleTEC"); break;
      }
      // rest  still  the same but invoking overloaded method
      this->callMille(refTrajPtr, iHit, kLocalX, theFloatBufferX, theIntBuffer, sigmaScale);
      if (is2DHit) {
        this->callMille(refTrajPtr, iHit, kLocalY, theFloatBufferY, theIntBuffer, sigmaScale);
        return 2; // 2D information used
      } else {
        return 1; // 1D information used
      }
    }
}
```

Listing 2: Method `callMille`

```
// experimental - overloaded version for blowing up errors
void MillePedeAlignmentAlgorithm::callMille
    (const ReferenceTrajectoryBase::ReferenceTrajectoryPtr &refTrajPtr,
    unsigned int iTrajHit, MeasurementDirection xOrY,
    const std::vector<float> &globalDerivatives, const std::vector<int> &globalLabels,
    double sigmaScale)
{
  const unsigned int xyIndex = iTrajHit*2 + xOrY;
  // FIXME: here for residuum and sigma we could use KALMAN-Filter results
  const float residuum =
    refTrajPtr->measurements()[xyIndex] - refTrajPtr->trajectoryPositions()[xyIndex];
  const float covariance = refTrajPtr->measurementErrors()[xyIndex][xyIndex];
  const float sigma = sigmaScale * (covariance > 0. ? TMath::Sqrt(covariance) : 0.);

  const AlgebraicMatrix &locDerivMatrix = refTrajPtr->derivatives();

  // hack for forcing a form of momentum constraint
  int localDerivsSize = locDerivMatrix.num_col();  // standard behaviour
  if (theConfig.getParameter<bool>("forceMomentumConstraint") )
     localDerivsSize = 2; // limit to two parameters for constraint

std::vector<float> localDerivs(localDerivsSize);

 if (!theConfig.getParameter<bool>("forceMomentumConstraint") )
{ // normal behaviour as in Geros code
for (unsigned int i = 0; i < localDerivs.size(); ++i) {
  localDerivs[i] = locDerivMatrix[xyIndex][i];
}
}
else // this is an experimental feature:
     // just let two local parameters pass to mille for optimisation
{
  for (unsigned int i = 3; i < 5; ++i) {
    localDerivs[i-3] = locDerivMatrix[xyIndex][i];
  }
}

// &(vector[0]) is valid - as long as vector is not empty
// cf. http://www.parashift.com/c++-faq-lite/containers.html#faq-34.3
theMille->mille(localDerivs.size(), &(localDerivs[0]),
        globalDerivatives.size(), &(globalDerivatives[0]), &(globalLabels[0]),
        residuum, sigma);
 if (theMonitor) {
  theMonitor->fillDerivatives(refTrajPtr->recHits()[iTrajHit],localDerivs,
          globalDerivatives, (xOrY == kLocalY));
  theMonitor->fillResiduals(refTrajPtr->recHits()[iTrajHit],
                 refTrajPtr->trajectoryStates()[iTrajHit],
                 iTrajHit, residuum, sigma, (xOrY == kLocalY));
 }
}
```

Listing 3: Part of method `initialize`

```

// FRANK: Abschnitt um Hitdaten in globalen Koordinaten zu erhalten
if(theConfig.getParameter<bool>("writePixelGlobalHits"))
{
    pxGlobalHitsFile = new TFile(
     (theConfig.getParameter<std::string>("writePxGlobalHitsRootFile")).c_str(),
     "RECREATE");
    pxGlobalHitsTree = new TTree("Tpxhit","TreePxHits");
    pxGlobalHitsTree->Branch("run", &pgh_run, "run/i");
    pxGlobalHitsTree->Branch("event", &pgh_event, "event/i");
    pxGlobalHitsTree->Branch("x", &pgh_x, "x[3]/D");
    pxGlobalHitsTree->Branch("y", &pgh_y, "y[3]/D");
    pxGlobalHitsTree->Branch("z", &pgh_z, "z[3]/D");
    pgh_run = 1; // dummy run number as iEvent is not available
    pgh_event = 0;
}
```

B Sample configuration file

The following example files are given as a general reference to see what the configurations were. These two files were used in study number 266, which was used in the production of the LASOnly-curve in figure 7

Listing 4: Sample config file

```
# ********************************************************************
# Configuration file for alignment using Millepede
# Annotated version
#
# Frank Meier

process Alignment = {

    # ********************************************************************
    # Message Logger

    # include "FWCore/MessageLogger/data/MessageLogger.cfi"
    service = MessageLogger {
        untracked vstring destinations = { "alignment" }# {, "cout" }
        untracked vstring statistics = { "alignment" } #{, "cout" }
        untracked vstring categories =
             { "Alignment", "LogicError", "FwkReport", "TrackProducer"}

        untracked PSet cout = {
            untracked string threshold = "DEBUG" # "ERROR"
            untracked PSet FwkReport = { untracked string threshold = "ERROR" }
            untracked PSet TrackProducer = { untracked string threshold = "ERROR" }
        }
        untracked PSet alignment = {
            untracked string threshold = "DEBUG"
            untracked PSet INFO = { untracked int32 limit = 10 }
            untracked PSet WARNING = { untracked int32 limit = 10 }
            untracked PSet ERROR = { untracked int32 limit = -1 }
            untracked PSet DEBUG = { untracked int32 limit = -1 }
            untracked PSet Alignment = { untracked int32 limit = -1}
            untracked PSet LogicError = { untracked int32 limit = -1}
        }
    }

    // service = Tracer { untracked string indention = "$$"}

    # ********************************************************************
    # Initialize magnetic field
    include "MagneticField/Engine/data/volumeBasedMagneticField.cfi"

    # ********************************************************************
    # Ideal geometry and interface
    include "Geometry/CMSCommonData/data/cmsIdealGeometryXML.cfi"
    include "Geometry/TrackerNumberingBuilder/data/trackerNumberingGeometry.cfi"


    # ********************************************************************
    # Track selection for alignment - apply some basic cuts on data sample
    #
    include "Alignment/CommonAlignmentProducer/data/AlignmentTrackSelector.cfi"
    # --> module AlignmentTracks

    replace AlignmentTracks.src = ALCARECOTkAlZMuMu
    # Note: values in comments correspond to defaults from .cfi
```

```
replace AlignmentTracks.ptMin     =  10.             #  10.
replace AlignmentTracks.ptMax     =  999.            #  999.
replace AlignmentTracks.etaMin    = −3.0             # −2.4
replace AlignmentTracks.etaMax    =  3.0             #  2.4
# replace AlignmentTracks.phiMin  = −3.1416         # −3.1416
# replace AlignmentTracks.phiMax  =  3.1416         #  3.1416
replace AlignmentTracks.nHitMin   = 8                #  8
replace AlignmentTracks.nHitMax   = 99               #  99
replace AlignmentTracks.chi2nMax = 999999.          #  999999.
# replace AlignmentTracks.applyNHighestPt = false  #  false
# replace AlignmentTracks.nHighestPt = 2            #  2


# ********************************************************************
# Alignment producer − defines what to be aligned
#
include "Alignment/CommonAlignmentProducer/data/AlignmentProducer.cff"
# −−> looper AlignmentProducer
# includes also other modules:
# − Scenarios.cff: block MisalignmentScenarioSettings
#   + a bunch of scenarios as blocks
# − MillePedeAlignmentAlgorithm.cfi: block MillePedeAlignmentAlgorithm
# − AlignmentParameterStore.cfi: PSet ParameterStore

replace AlignmentProducer.tjTkAssociationMapTag = TrackRefitter
                        # default is TrackRefitter

# Defines what to align
# 0: do not align, 1: align,
# f: fix parameter at zero, c: fix parameter at true value
replace AlignmentProducer.ParameterBuilder.Selector = {
    vstring alignParams = {
        "PixelHalfBarrelDets,111111",   # pixel as dets
        "PXEndCaps,ffffff",              # let everything else be fixed
        "TOBHalfBarrels,ffffff",
        "TIBHalfBarrels, ffffff ",
        "TIDs, ffffff ",
        "TECs, ffffff"
    }
}

# blow up of sigmas
replace MillePedeAlignmentAlgorithm.sigmaScalePXB = 1.0
replace MillePedeAlignmentAlgorithm.sigmaScalePXE = 1.0
replace MillePedeAlignmentAlgorithm.sigmaScaleTIB = 1.0
replace MillePedeAlignmentAlgorithm.sigmaScaleTID = 1.0
replace MillePedeAlignmentAlgorithm.sigmaScaleTOB = 1.0
replace MillePedeAlignmentAlgorithm.sigmaScaleTEC = 1.0

# misalignment settings
replace AlignmentProducer.doMisalignmentScenario = true # default: false
include "Alignment/MillePedeAlignmentAlgorithm/test/myMisalignmentScenario.cff"

# beamspot constraint
replace MillePedeAlignmentAlgorithm.beamspot = true
replace MillePedeAlignmentAlgorithm.aliBeamspot = false

# settings for MillePede
replace AlignmentProducer.algoConfig = { using MillePedeAlignmentAlgorithm }
replace MillePedeAlignmentAlgorithm.mode = "full"
# possibilities : full , mille, pede, pedeSteer, pedeRun, pedeRead
```

```
    # replace MillePedeAlignmentAlgorithm.pedeSteerer.pedeCommand
    #    = "/net/ruchi/export/data1/frmeier/pede/versWebEndMay2007/pede"
    replace MillePedeAlignmentAlgorithm.monitorFile = ""

    replace MillePedeAlignmentAlgorithm.pedeSteerer.method = "sparseGMRES␣4␣␣0.8"
    #                                                        <method> n(iter)  Delta(F)
    replace MillePedeAlignmentAlgorithm.pedeSteerer.options =
            { "chisqcut␣␣20.0␣␣4.5", "bandwidth␣␣6"}
    # chisqcut <first> <subseq>, outlierdownweighting <n iter>,
    # dwfractioncut <dwfractionvalue>
    # bandwidth <width>: for sparseGMRES solver method


    # *******************************************************************
    # refitting
    include "RecoTracker/TrackProducer/data/RefitterWithMaterial.cff"
    # --> module TrackRefitter

    # replace TrackRefitter.Fitter = "KFFittingSmoother"  # "KFFittingSmoother"
    # replace TrackRefitter.Propagator = "PropagatorWithMaterial" # "PropagatorWithMaterial"
    replace TrackRefitter.src = "AlignmentTracks"          # "ctfWithMaterialTracks"
    # replace TrackRefitter.producer = ""                  # ""
    # replace TrackRefitter.TTRHBuilder = "WithTrackAngle" # "WithTrackAngle"
    replace TrackRefitter.TrajectoryInEvent = true         # false

    # needed for refit of hits:
    # include "CalibTracker/Configuration/data/SiStrip_FakeLorentzAngle.cff"
    # usually without refit:
    replace TrackRefitter.TTRHBuilder = "WithoutRefit"
    # TransientTrackingRecHitBuilder: no refit of hits ...
    include "RecoTracker/TransientTrackingRecHit/data/
␣␣␣␣TransientTrackingRecHitBuilderWithoutRefit.cfi"
    # ... but matching for strip stereo should be redone:
    replace ttrhbwor.Matcher = "StandardMatcher"
    # beam halo propagation needs larger phi changes going from one TEC to another
    # replace MaterialPropagator.MaxDPhi = 1000.

    # *******************************************************************
    # Save data to DB
    #include "Alignment/CommonAlignmentProducer/data/DBConfiguration.cff"
    include "CondCore/DBCommon/data/CondDBSetup.cfi"
    service = PoolDBOutputService {
        using CondDBSetup
        string connect = "sqlite_file:testalignment.db"
        untracked string catalog = "file:testalignment.xml"
        untracked uint32 authenticationMethod = 1
        string timetype = "runnumber"
        VPSet toPut = {
            { string record = "TrackerAlignmentRcd"     string tag = "valueTag" },
            { string record = "TrackerAlignmentErrorRcd" string tag = "errorTag" }
        }
    }
    replace AlignmentProducer.saveToDB = true

    # *******************************************************************
    # input file
    # source = EmptySource {}
    source = PoolSource {
        untracked vstring fileNames = {   # file:/foo/bar or rfio:/foo/bar
        "rfio:/castor/cern.ch/cms/store/mc/2007/10/16/RelVal-DrellYan_mumu_40-
␣␣␣␣␣␣␣␣Tier0-ALCO-A1/0000/06C163C0-237C-DC11-8815-000423D6B358.root",
            # ..... abbreviated ......
            }
            untracked uint32 skipEvents = 0
      }
      untracked PSet maxEvents = {untracked int32 input = 100000 }

      # ********************************************************************
      path p = { AlignmentTracks, TrackRefitter }
}
```

Listing 5: Sample misalignment scenario file

```
// ------------------------------------
// "␣Survey&LAS␣only␣misalignment␣scenario
//␣" See CMS IN 2007/036, table 6, "Updated␣initial␣uncertainties"
// LASOnly


replace AlignmentProducer.MisalignmentScenario =
{

//using MisalignmentScenarioSettings
untracked bool saveToDbase = false

string distribution = 'gaussian'
int32 seed = 1234567
bool setError = true

bool setRotations = true
bool setTranslations = true


// TPB
PSet TPBs = {
 double scale = 1.0
 PSet Dets = { string distribution = 'gaussian'
  double dXlocal = 0.0060 double dYlocal = 0.0060 double dZlocal = 0.0060
  double phiXlocal = 0.000270 double phiYlocal = 0.000270 double phiZlocal = 0.000270}
 PSet Rods = { string distribution = 'flat'
  double dXlocal = 0.0050 double dYlocal = 0.0050 double dZlocal = 0.0050
  double phiXlocal = 0.000020 double phiYlocal = 0.000020 double phiZlocal = 0.000020}
 PSet PixelHalfBarrelLayers = { string distribution = 'flat'
  double dX = 0.0100 double dY = 0.0100 double dZ = 0.0100
  double phiXlocal = 0.000040 double phiYlocal = 0.000040 double phiZlocal = 0.000040}
}

// TPE
// new hierarchy for TPE used
PSet TPEs = {
 double scale = 1.0
 string distribution = 'gaussian'

 PSet Dets = {
  double dXlocal = 0.0005 double dYlocal = 0.0005 double dZlocal = 0.0005
  double phiXlocal = 0.000100 double phiYlocal = 0.000100 double phiZlocal = 0.000100}
 PSet Panels = {
  double dXlocal = 0.0010 double dYlocal = 0.0010 double dZlocal = 0.0010
  double phiXlocal = 0.000200 double phiYlocal = 0.000200 double phiZlocal = 0.000200}
 PSet Blades = {
  double dXlocal = 0.0010 double dYlocal = 0.0010 double dZlocal = 0.0010
  double phiXlocal = 0.000200 double phiYlocal = 0.000200 double phiZlocal = 0.000200}
 PSet HalfDisks = {
  double dXlocal = 0.0050 double dYlocal = 0.0050 double dZlocal = 0.0050
  double phiXlocal = 0.001000 double phiYlocal = 0.001000 double phiZlocal = 0.001000}
// HalfCylinders do not work yet!
//PSet HalfCylinders =
// {
//  double dXlocal = 0.0050 double dYlocal = 0.0050 double dZlocal = 0.0050
// }
}

// TIB
PSet TIBs = {
 string distribution  = 'gaussian'
 double scale = 1.0
 PSet Dets = {
  double dXlocal = 0.0180 double dYlocal = 0.0180 double dZlocal = 0.0180
  double phiXlocal = 0.000412 double phiYlocal = 0.000412 double phiZlocal = 0.000412}
 PSet Rods = {
  double dXlocal = 0.0450 double dYlocal = 0.0450 double dZlocal = 0.0450
  double phiXlocal = 0.000293 double phiYlocal = 0.000293 double phiZlocal = 0.000293}
 PSet BarrelLayers = {
  double dXlocal = 0.0750 double dYlocal = 0.0750 double dZlocal = 0.0750
  double phiXlocal = 0.000488 double phiYlocal = 0.000488 double phiZlocal = 0.000488}
}

// TID
PSet TIDs = {
 double scale = 1.0
 string distribution  = 'gaussian'
 PSet Dets = {
  double dXlocal = 0.0054 double dYlocal = 0.0054 double dZlocal = 0.0054
  double phiXlocal = 0.000250 double phiYlocal = 0.000250 double phiZlocal = 0.000250}
 PSet TIDRings = {
  double dXlocal = 0.0185 double dYlocal = 0.0185 double dZlocal = 0.0185
  double phiXlocal = 0.000850 double phiYlocal = 0.000850 double phiZlocal = 0.000850}
 PSet TIDLayers = {
  double dXlocal = 0.0350 double dYlocal = 0.0350 double dZlocal = 0.0350
  double phiXlocal = 0.000532 double phiYlocal = 0.000532 double phiZlocal = 0.000532}
}

// TOB
PSet TOBs = {
 double scale = 1.0
 string distribution  = 'gaussian'
 PSet Dets = {
  double dXlocal = 0.0032 double dYlocal = 0.0032 double dZlocal = 0.0032
  double phiXlocal = 0.000075 double phiYlocal = 0.000075 double phiZlocal = 0.000075}
 PSet Rods = {
  double dXlocal = 0.0100 double dYlocal = 0.0100 double dZlocal = 0.0100
  double phiXlocal = 0.000040 double phiYlocal = 0.000040 double phiZlocal = 0.000040}
 PSet BarrelLayers = { // Not used since TOB layers do not make sense!
  double dXlocal = 0.0000 double dYlocal = 0.0000 double dZlocal = 0.0000 }
 PSet HalfBarrels = {
  double dXlocal = 0.0060 double dYlocal = 0.0060 double dZlocal = 0.0500
  double phiXlocal = 0.000010 double phiYlocal = 0.000010 double phiZlocal = 0.000010}
}

// TEC
PSet TECs = {
 double scale = 1.0
 string distribution  = 'gaussian'
 PSet Dets = {
  double dXlocal = 0.0022 double dYlocal = 0.0022 double dZlocal = 0.0022
  double phiXlocal = 0.000050 double phiYlocal = 0.000050 double phiZlocal = 0.000050}
 PSet Petals = {
  double dXlocal = 0.0070 double dYlocal = 0.0070 double dZlocal = 0.0070
  double phiXlocal = 0.000030 double phiYlocal = 0.000030 double phiZlocal = 0.000030}
 PSet EndcapLayers = {
  double dXlocal = 0.0060 double dYlocal = 0.0060 double dZlocal = 0.0150
  double phiXlocal = 0.000015 double phiYlocal = 0.000015 double phiZlocal = 0.000015}
}

}
```

C Example for geometric distortion

The following three figures show full plots of geometric distortions from the study presented in section 3.5.2 ($Z \to \mu\mu$ events, LASOnly scenario).

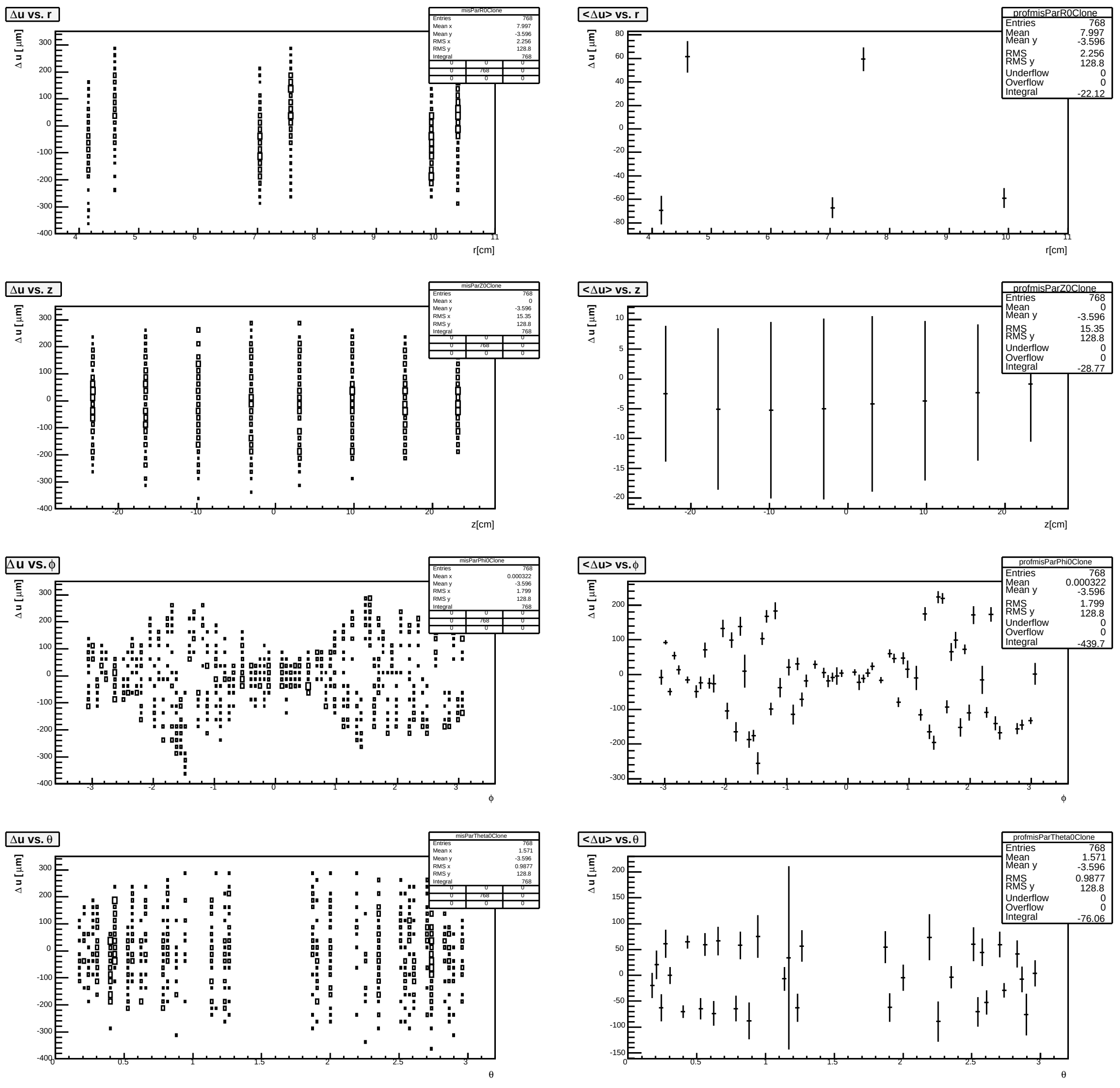


Figure 23: Geometric distortions after alignment – u-direction The left columns shows two-dimensional histograms of the misalignment distribution in the local frame (values are relative to the true positions) vs. some global coordinates. The right column are the corresponding profile plots showing the mean and RMS for each group of alignables at the same value given in the abscissa.

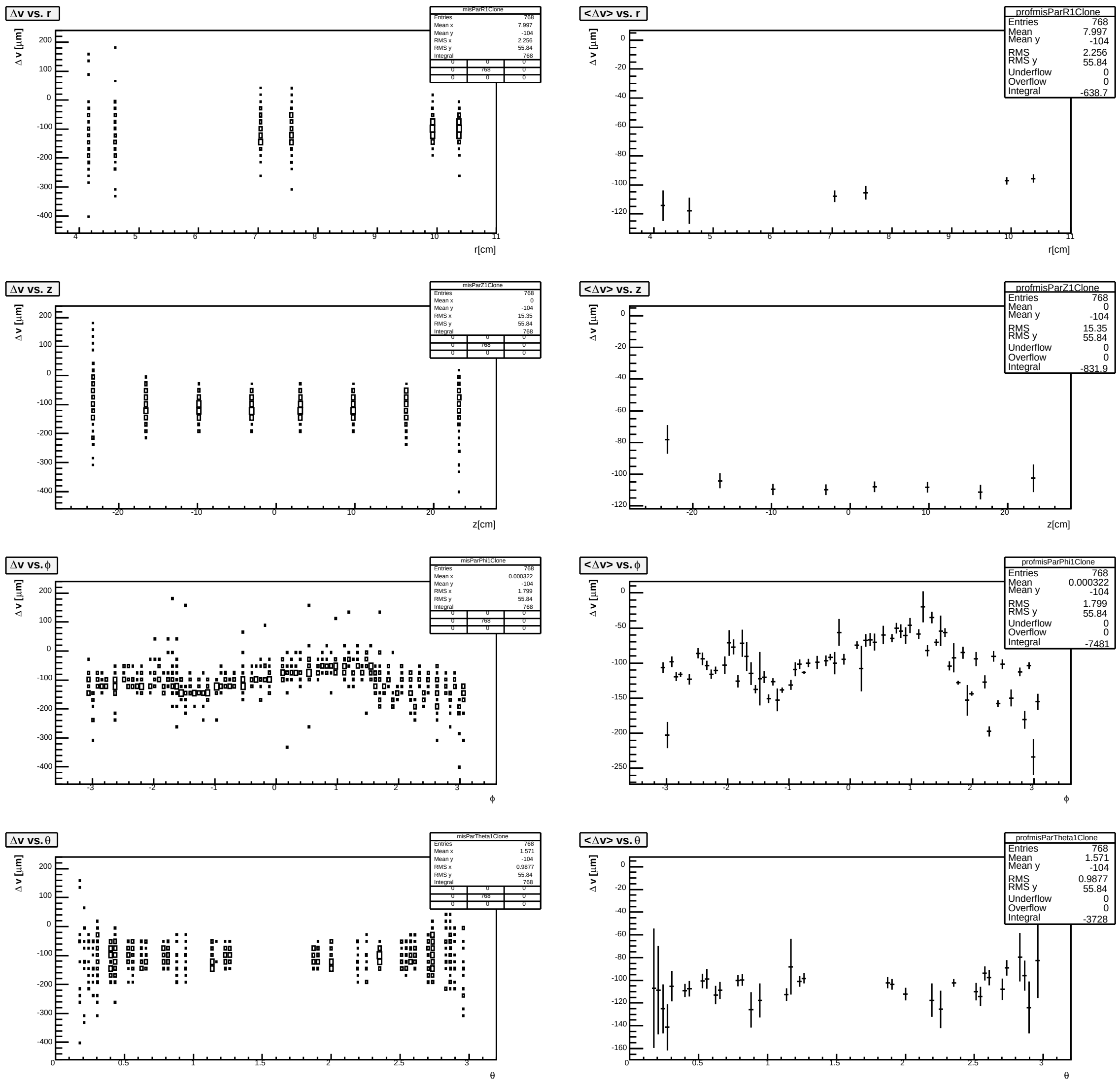


Figure 24: Geometric distortions after alignment − ***w*-direction** See figure 23 for details.

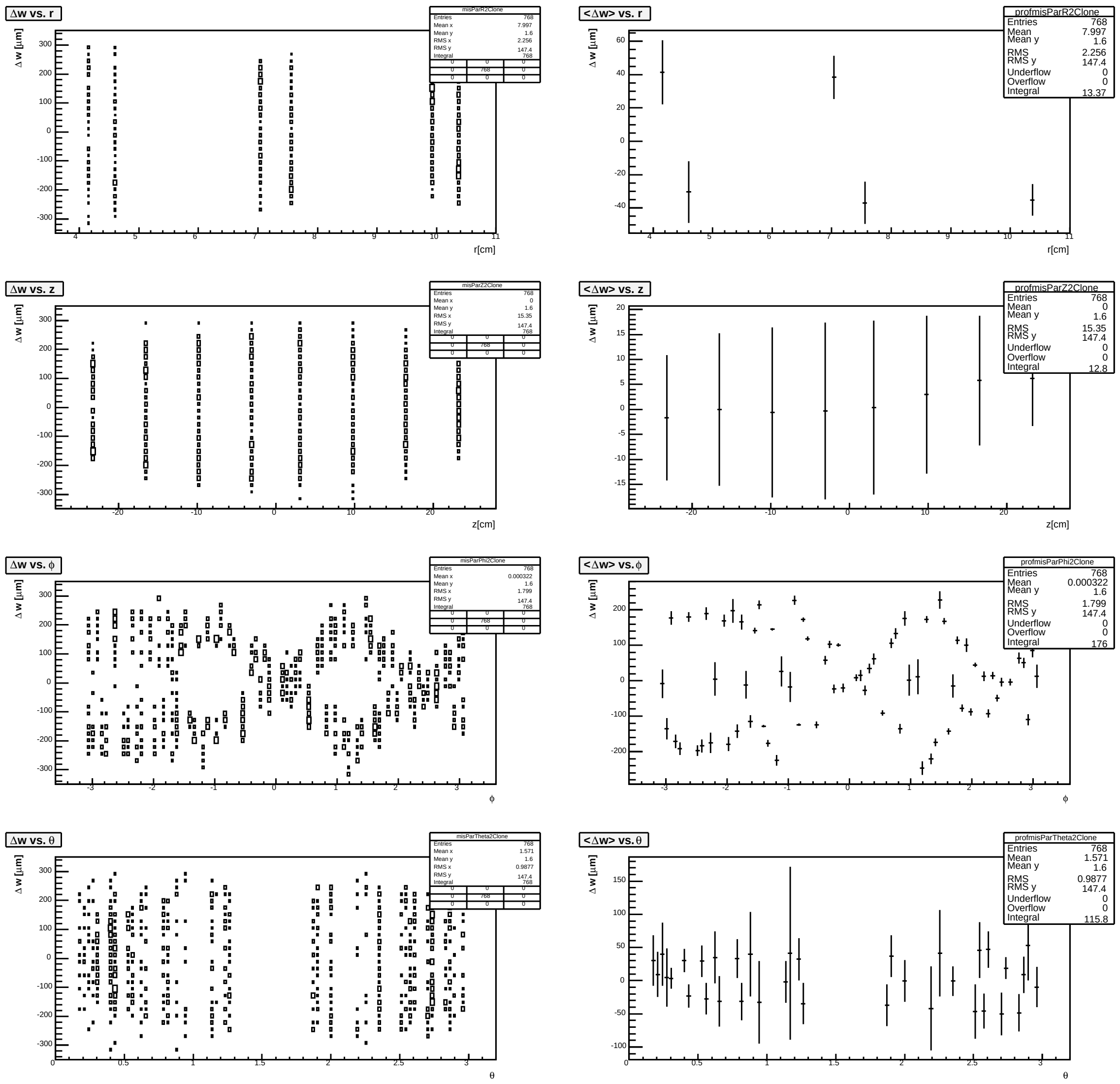


Figure 25: Geometric distortions after alignment – w-direction See figure 23 for details.

D Removal of global shifts

This is an example output of a root script written by Hans-Christian Kästli. It corrects for global shifts by calculating the center of mass of the pixel barrel and fits the orientation in space to an optimum position with respect of the alignment output.

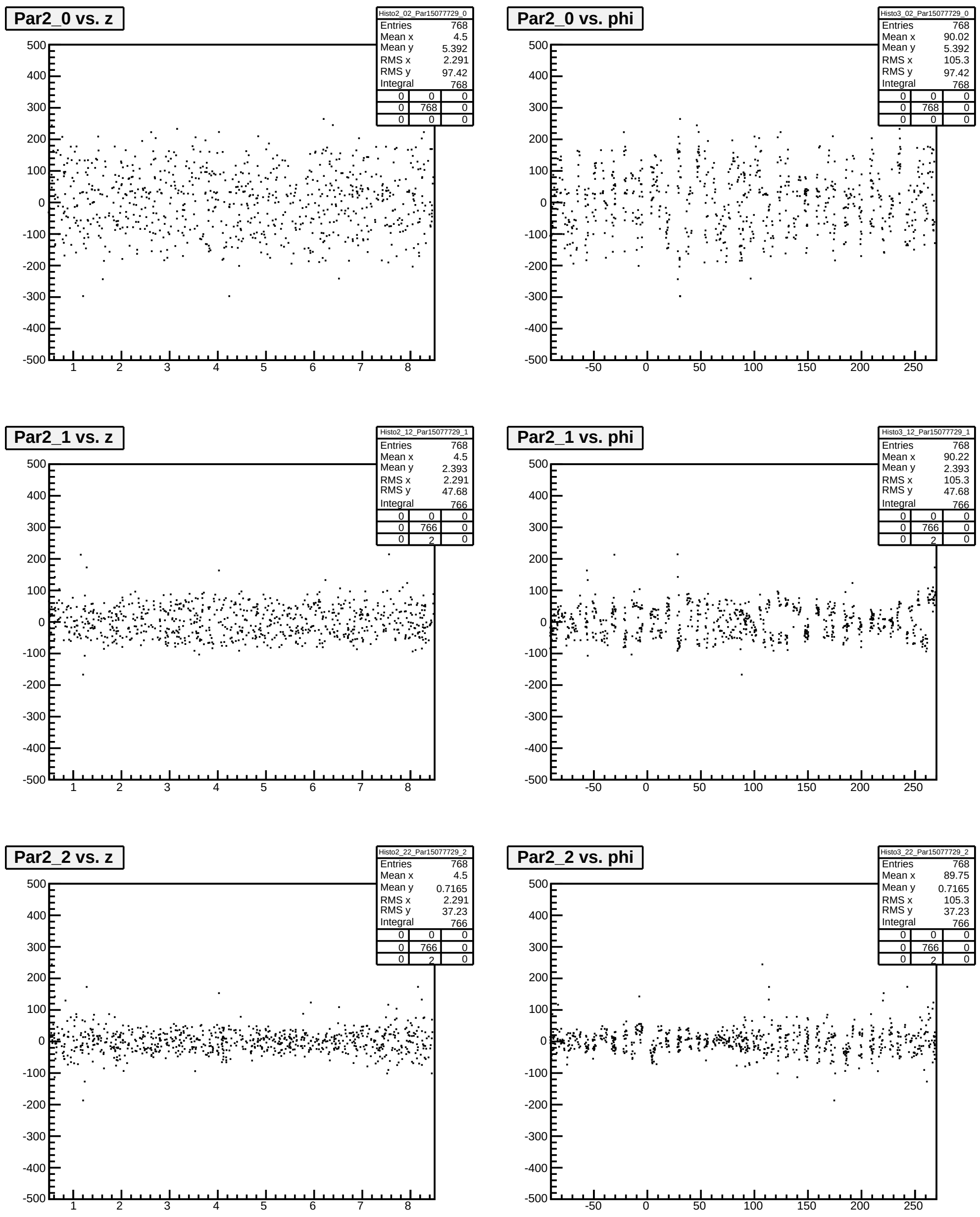


Figure 26: Example for removal of global shifts The upper row shows the distribution of module position in w (= Par2) vs. z and ϕ. The middle row shows the direct result from the alignment and the last row the result after correction for global shifts. The remaining misalignment RMS improves from 98 μm for the initial misalignment over 48 μm after the alignment to 37 μm after removal of global shifts. The data sample was $Z \to \mu\mu$, LASOnly, FPix aligned on all six degrees of freedom and using the momentum constraint. Plot courtesy of Hans-Christian Kästli.

E Optimization using the simplex algorithm

The parameters for the error enlargement were scaled to the same values for all studies presented in this thesis. But this must not be the optimum. To find it, the error scaling values were minimized using the *simplex algorithm* as described for example in [6]. Four values were optimized, including the forward pixel. The results are shown in table 5 and are far better in all four scenarios than using the same values. In the 100 and 1000 pb^{-1} scenarios, the remaining misalignment after the alignment is still worse than the initial misalignment.

The optimized error scaling parameters refer to an older version of the error enlargement code when independent scaling of the subdetectors was not a requirement. *Endcap* covers all endcap structures, *HalfBarrel* all barrel structures except the pixel barrel. As this study was performed to show that optimization is possible, it is still of value.

Table 5: Remaining misalignment after optimization. The table shows the optimum settings for the error enlargement as found using a simplex optimization after several steps. The procedure was stopped when the simplices became sufficiently small and no improvement was observed between three steps.
Observe that in the 100 and 1000 pb^{-1} scenarios the remaining misalignment in v is still larger than the initial misalignment, as listed in table 2.

Scenario	error scaling				RMS remaining misalignment		
	PxEndcap	TID	Endcap	HalfBarrel	u	v	w
LASOnly	14.25	128.5	1.00	126	50.1	30.9	42.4
10 pb^{-1}	8.19	52.0	0.88	24.4	12.0	20.7	15.0
100 pb^{-1}	4.63	3.15	0.86	4.02	10.1	16.1	12.95
1000 pb^{-1}	3.18	2.00	0.90	2.48	6.41	13.2	7.78

F Latest results

Some additional studies were performed but the computing resources were not sufficient to finish them on time to be included in the main text. These results are now presented here.

One major disadvantage of the studies presented in section 3.5 was that they were performed with only one seed for the random number generator. This has been improved by using 9 other seeds. The results are shown in figure 27 for $Z \to \mu\mu$ and in figure 28 for minimum bias samples. The error bars show the standard deviation out of at least eight measurements[14], being an estimate for 1σ. For each event type, three cases were studied. The first case is the same as used for the results already presented. The two other cases use a beamspot constraint as described in 3.5.6, which was meanwhile successfully backported to our software version. Only the case where the beamspot is treated as a constraint has been used, as treating it as an alignable lead to very bad results, which are not yet understood.

As can be seen in the plots, the alignment benefits a lot from a beamspot constraint. For example, the best performance for LASOnly using $Z \to \mu\mu$ is around 20, 30 and $50\,\mu$m for the directions u, v and w respectively. The results using minimum bias are not as good as with $Z \to \mu\mu$. As this is latest data, the results were not analyzed in detail. Especially some prominent outliers were not investigated. In all cases with beamspot constraint on, the beamspot has been assumed to be at its nominal position at the origin $(0, 0, 0)$ and with its nominal shape. For minimum bias MC-samples we do not know if this assumption is true, as the configuration files used for their generation are no longer accessible. Nevertheless, it will be necessary in the future to first calculate the beamspot and its orientation before using a beamspot constraint – the assumption used in this study is far from the reality when it comes to real data.

Figure 29 shows results from a study where the number of events has been varied. For $Z \to \mu\mu$, an increase in integrated luminosity shows clearly an improvement in all directions, most prominently in v and w. The optimum for u is reached already around 10 000 events, which is still quite an amount of integrated luminosity (46 days at start-up conditions, half a day at design luminosity) and therefore not very suitable. The behaviour with minimum bias follows the same pattern but with results far worse. The point using 500 000 needs further investigation and is questionable so far.

F.1 Conclusion

The introduction of a beamspot improves the results but is not sufficient. The simple approach used here needs improvements, as it relies on a nominal beamspot position, shape and orientation.

[14] A minor fraction of the runs experienced problems and were rescheduled, but their results came in too late. The mass of point ran without problems and were calculated using ten values.

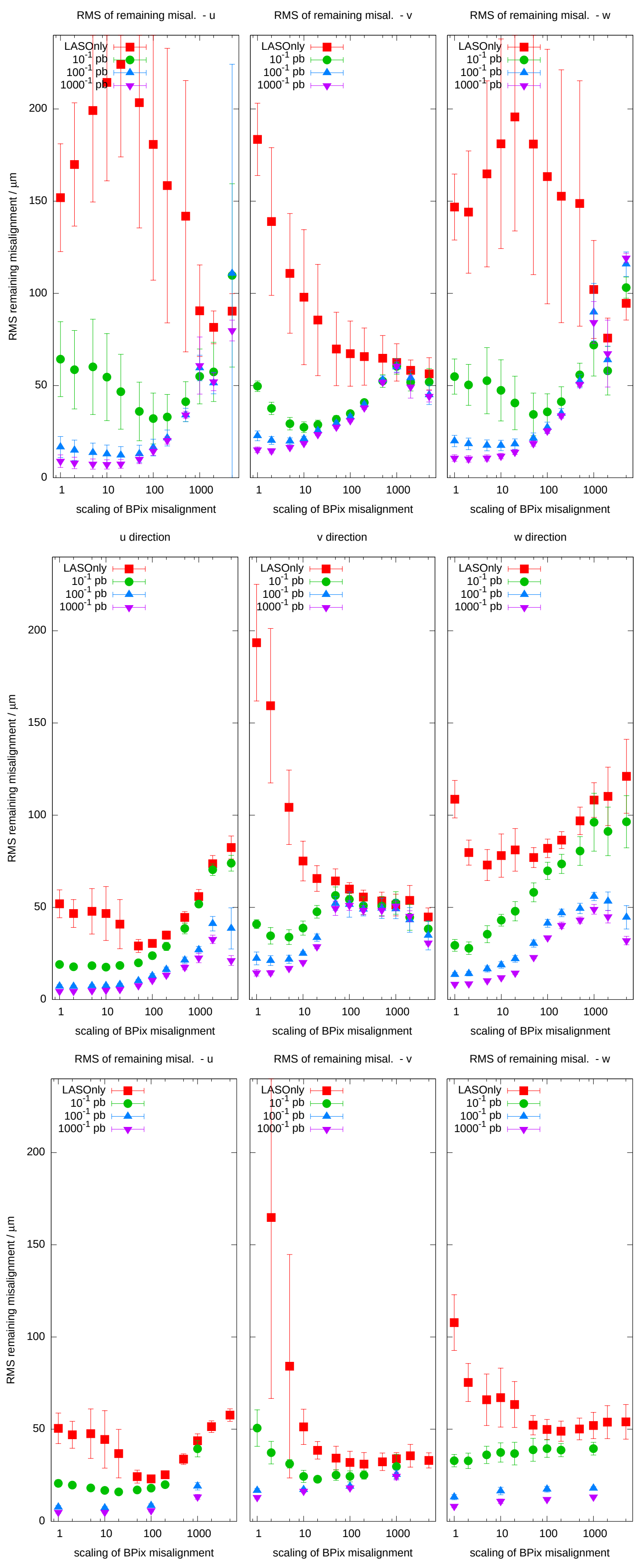


Figure 27: $Z \to \mu\mu$. Top row: no beamspot constraint, middle row: beamspot constraint, bottom row: beamspot constraint + FPix aligned in $uvw\alpha\beta\gamma$. Error bars show standard deviation out of 10 different seeds. 100 000 events used. Details see text.

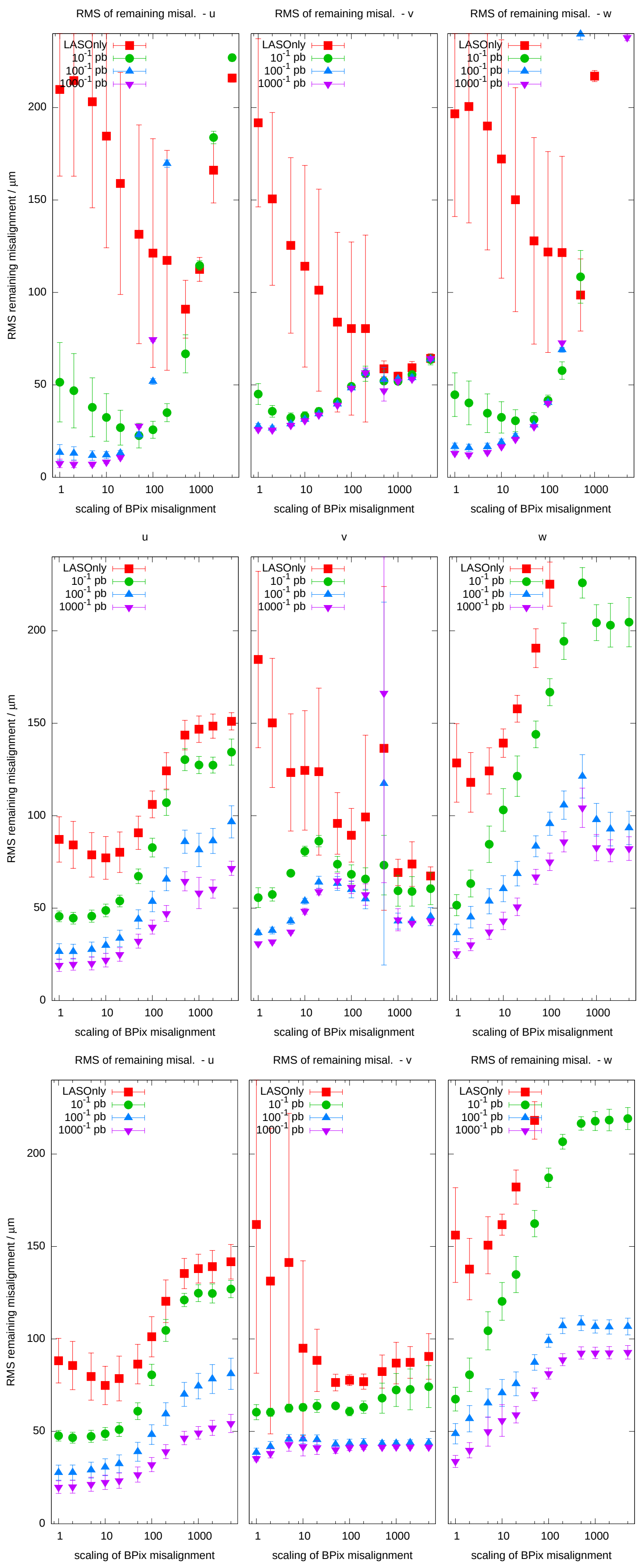


Figure 28: MinimumBias. Top row: no beamspot constraint, middle row: beamspot constraint, bottom row: beamspot constraint + FPix aligned in $uvw\alpha\beta\gamma$. Error bars show standard deviation out of 10 different seeds. 60 000 events used. Details see text.

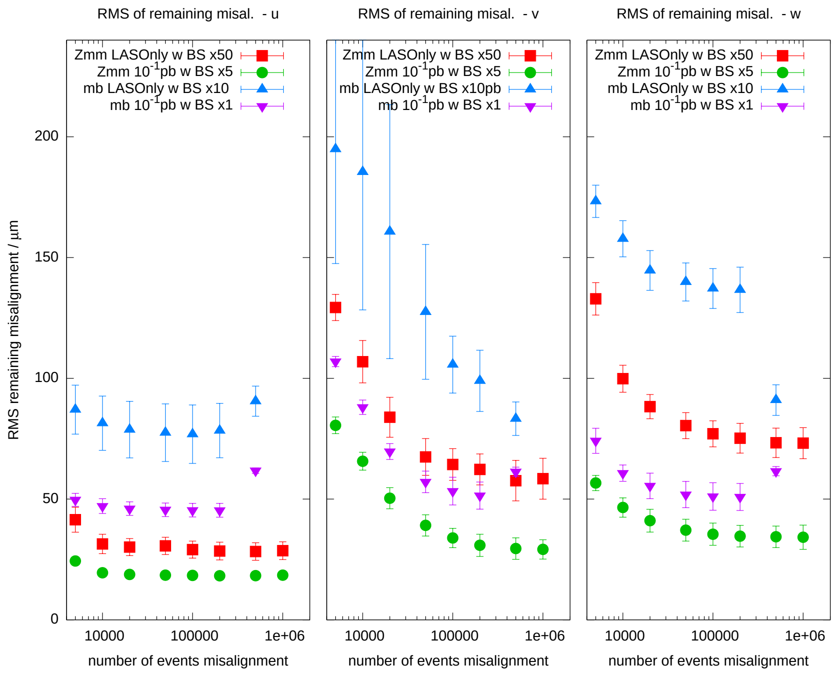


Figure 29: Luminosity for alignment. Shown are four different cases, each of the using a beamspot constraint. The error scaling values used were the estimated optima from the studies in figures 27 and 28. 100 000 $Z \to \mu\mu$-events correspond to about 460 full days at start-up luminosity and about 4.6 days at design luminosity. 100 000 minimum bias events correspond to about 1 000 seconds of data taking at 100 Hz.