T R D

Offline Software Writeup

Version 1.0, July 8, 2009

Contents

Introduction

This document is supposed to provide a description of the offline software components that are specific to the TRD. It is an attempt to collect useful informations on the design and usage of the TRD software, in order to facilitate newcomers the introduction to the code. The most important classes and procedures are described and several examples and use cases are given. However, this writeup is not meant to be a basic AliRoot introduction. For this purpose the reader is referred to the general AliRoot users guide [1].

Simulation

1.1 Geometry

Author: C. Blume (blume@ikf.uni-frankfurt.de)

The TRD geometry, as implemented in `AliTRDgeometry`, consists of several components: The readout chambers (ROC), the services, and the supermodule frame. All these parts are placed inside the TRD mother volumes, which in turn are part of the space frame geometry (`AliFRAMEv2`). Therefore, the space frame geometry has to be present to build the TRD geometry. For each of the 18 supermodules one single mothervolume is provided (BTRDxx). This allows to configure the TRD geometry in `Config.C` such that it only contains a subset of supermodules in the total ALICE geometry via `AliTRDgeometry::SetSMstatus()`. An incomplete detector setup, as it exists for first data taking, can thus be modelled. The class `AliTRDgeometry` also serves as the central place to collect all geometry relevant numbers and the definitions of various numbering schemes of detector components (e.g. sector numbers). However, all geometric parameters that refer to the pad planes are compiled in `AliTRDpadPlane`.

1.1.1 Readout Chambers

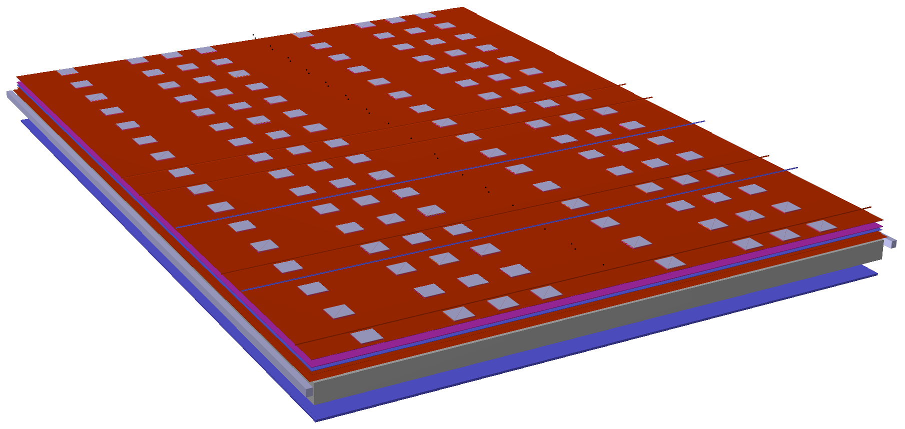


Figure 1.1: A TRD read out chamber as implemented in the AliRoot geometry. The various material layers are visible. Also, the MCMs on top of the chamber, as well as the cooling pipes are shown.

All ROCs are modelled in the same way, only their dimensions vary. They consist of an aluminum frame, which contains the material for the radiator and the gas of the drift region, a Wacosit frame (whose material is represented by carbon), that surrounds the amplification region, and the support structure, consisting of its aluminum frame, material for the read out pads, back panel, and readout boards). The material inside the active parts of the chambers (radiator, gas, wire planes, pad planes, glue, read out boards, etc.) is introduced by uniform layers of the corresponding material, whose thicknesses were chosen such to result in the correct radiation length. On top of the individual ROCs the multi chip modules (MCM) as well as the cooling pipes and cables are placed. One obvious simplification, already visible in Fig. 1.1, is that in the AliRoot geometry the pipes run straight across the chambers instead of following the meandering path as in reality.

1.1.2 Supermodules

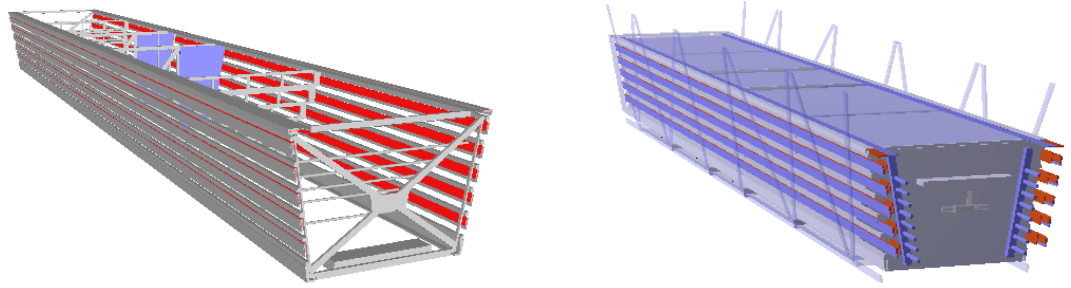


Figure 1.2: A TRD supermodule, as implemented in the AliRoot geometry. The left panel shows only the support structures of the aluminum frame, together with some service elements. The right panel shows a complete supermodule including some surrounding parts of the space frame.

The supermodule frames consist of the aluminum sheets on the sides, top, and bottom of a supermodule together with the traversing support structures. The left panel of Fig. 1.2 shows the structures that are implemented in the TRD geometry. Also, parts of the services like the LV power bus bars and cooling arteries can be seen. Additional electronics equipment (e.g. “Schütten-Box“) is represented by aluminum boxes that contain corresponding copper layers to mimic the present material. The services also include e.g. gas distribution boxes, cooling pipes, power and readout cables, and power connection panels. Part of the services extend into the baby and the back frame. Therefore, additional mother volumes have been introduced in order to accommodate this material. All supermodules have inserts of carbon fiber sheets in the bottom part of the aluminum casing, for the ones in front of the PHOS detector (sectors 11–15) also the top part includes carbon fiber inserts. The supermodules in the sectors 13–15 do not contain any ROCs in the middle stack in order to provide the holes for the PHOS detector. Instead, gas tubes of stainless steel have been built in. Generally, the TRD volumina start with the letter “**U**”. The geometry is defined by the function `AliTRDgeometry::CreateGeometry()`, which generates the TRD mother volumes (**UTI1**, **UTI2**, **UTI3**) and the volumes that constitute a single ROC. This function in turn also calls `AliTRDgeometry::CreateFrame()` to create the TRD support frame, `AliTRDgeometry::CreateServices()` to create the services, and `AliTRDgeometry::GroupChambers()` which assembles the alignable volumes for a single ROC

(**UTxx**, where **xx** is the detector number **DET-SEC**, defined inside a single super module, see below). The materials, together with their tracking parameters, that are assigned to the volumina, are defined in `AliTRD::CreateMaterials()`. In the following table the most important TRD volumina are described (**xx** = **DET-SEC** number):

Name	Description
UTR1	TRD mothervolume for default supermodules
UTR2	TRD mothervolume for supermodules in front of PHOS
UTR3	As **UTR2**, but w/o middle stack
UTxx	Top volume of a single ROC
	Defines the alignable volume for a single ROC
UAxx	Lower part of the ROCs, including drift volume and radiator
UDxx	Amplification region
UFxx	Back panel, including pad planes and PCB boards of readout electronics
UUxx	Contains services on chambers (cooling, cables, DCS boards) and MCM chips

1.1.3 Material Budget and Weight

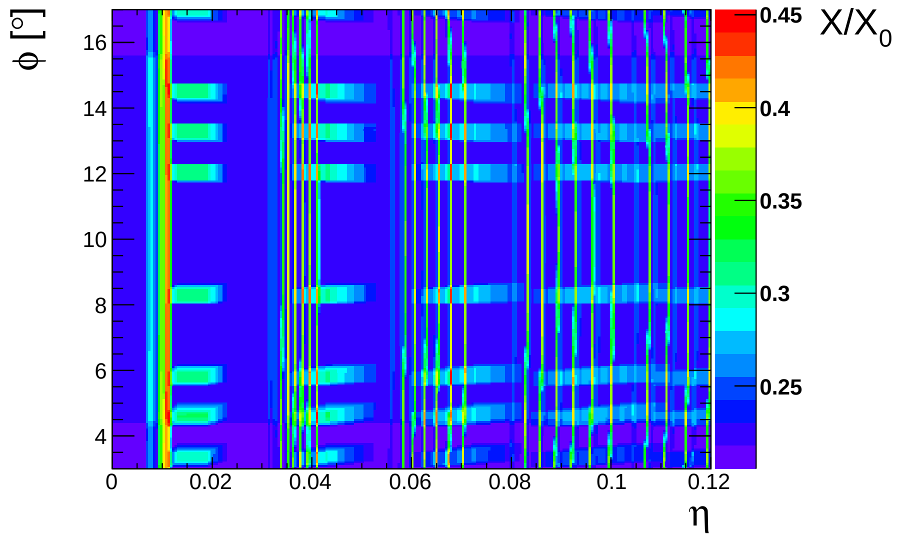


Figure 1.3: The radiation length map in units of X/X_0 in part of the active detector area of super module 0 as a function of the pseudorapidity η and the azimuth angle ϕ, calculated from the geometry in AliRoot. Visible are the positions of the MCMs and the cooling pipes as hot spots.

The volumina defining a ROC contain several layers that represent the different materials present inside a chamber and which therefore define the material budget inside the sensitive areas:

Name	Mother	Material	Description	Thickness [cm]	Density [g/cm^3]	X/X_0 [%]
URMYxx	UAxx	Mylar	Mylar layer on radiator (x2)	0.0015	1.39	0.005
URCBxx	UAxx	Carbon	Carbon fiber mats (x2)	0.0055	1.75	0.023
URGLxx	UAxx	Araldite	Glue on the fiber mats (x2)	0.0065	1.12	0.018
URRHxx	UAxx	Rohacell	Sandwich structure (x2)	0.8	0.075	0.149
URFBxx	UAxx	PP	Fiber mats inside radiator	3.186	0.068	0.490
UJxx	UAxx	Xe/CO$_2$	The drift region	3.0	0.00495	0.167
UKxx	UDxx	Xe/CO$_2$	The amplification region	0.7	0.00495	0.039
UWxx	UKxx	Copper	Wire planes (x2)	0.00011	8.96	0.008
UPPDxx	UFxx	Copper	Copper of pad plane	0.0025	8.96	0.174
UPPPxx	UFxx	G10	PCB of pad plane	0.0356	2.0	0.239
UPGLxx	UFxx	Araldite	Glue on pad plane	0.0923	1.12	0.249
		Araldite	+ additional glue (leaks)	0.0505	1.12	0.107
UPCBxx	UFxx	Carbon	Carbon fiber mats (x2)	0.019	1.75	0.078
UPHCxx	UFxx	Aramide	Honeycomb structure	2.0299	0.032	0.169
UPPCxx	UFxx	G10	PCB of readout boards	0.0486	2.0	0.326
UPRDxx	UFxx	Copper	Copper of readout boards	0.0057	8.96	0.404
UPELxx	UFxx	Copper	Electronics and cables	0.0029	8.96	0.202

This material budget has been adjusted to match the estimate given in [2], with the exception of the glue layer in the back panel (**UPGLxx**), which has been made thicker to include all the additional glue that has been applied to fix the gas leaks. Figure 1.3 shows the resulting radiation length map in the active detector area for super module 0, which has only carbon fiber inserts at the bottom and is thus one of the super modules with the largest material budget. It is clearly visible that the MCMs and the cooling pipes introduce hot spots in X/X_0. However, after averaging over the shown area, the mean value is found to be $\langle X/X_0 \rangle =$ 24.7 %. For a supermodule with carbon fiber inserts at the top and the bottom one finds $\langle X/X_0 \rangle =$ 23.8 % and in the regions of the PHOS holes (i.e. in the middle stack of supermodules 13–15) it is only $\langle X/X_0 \rangle =$ 1.9 %.

The total weight of a single TRD super module in the AliRoot geometry, including all services, is currently 1595kg, which is ca. 5% short of its real weight. A single ROC of the type L0C1 with electronics and cooling pipes weighs 21.82kg.

1.1.4 Naming Conventions and Numbering Schemes

The numbering schemes and the orientations of coordinate systems generally follow the official ALICE-TRD definition [3]. Therefore, the whole geometry is defined in the global ALICE coordinate system. Inside the code we use the following nomenclature (see Fig. 1.4), which should be used consistently throughout the TRD code:

Name	Definition	Range
SECTOR	TRD sector in azimuth (i.e. one supermodule)	0–17
LAYER	Layer inside a supermodule	0–5
STACK	Division of a supermodule along z-direction	0–4
DET	Single ROC in whole TRD	0–539
DET-SEC	Single ROC in one super module	0–29

Due to the holes in front of the PHOS detector, naturally not all **DET** numbers correspond to existing ROCs. A single ROC can thus be uniquely addressed by either using the three

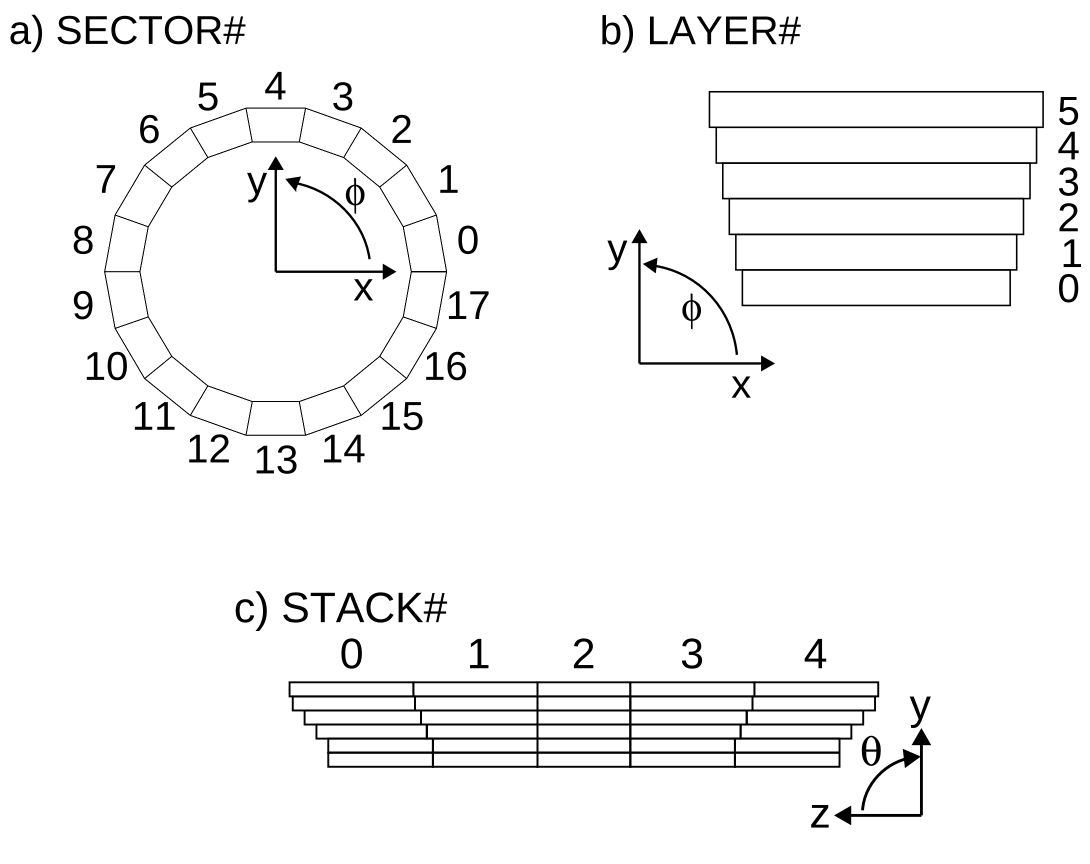


Figure 1.4: Illustration of the TRD numbering scheme for super modules, defined in the global ALICE coordinate system: a) **SECTOR** number, b) **LAYER** number, c) **STACK** number.

numbers (**LAYER**, **STACK**, **SECTOR**) or the single **DET** number. The correspondence between the two possibilities is defined as:

$$\mathbf{DET} = \mathbf{LAYER} + \mathbf{STACK}\times 5 + \mathbf{SECTOR}\times 5\times 4$$

Additionally, there is a number that is unique inside a given super module (i.e. sector) and therefore has a range of 0 – 29:

$$\mathbf{DET\text{-}SEC} = \mathbf{LAYER} + \mathbf{STACK}\times 5$$

The class `AliTRDgeometry` provides a set of functions that could/should be used to convert the one into the other:

```
AliTRDgeometry::GetDetector(layer,stack,sector)
AliTRDgeometry::GetDetectorSec(layer,stack)
AliTRDgeometry::GetLayer(det)
AliTRDgeometry::GetStack(det)
AliTRDgeometry::GetSector(det)
```

1.1.5 Pad Planes

All geometric parameters relevant to the pad planes are handled via the class `AliTRDpadPlane`. This comprises the dimensions of the pad planes and the pad themselves, the number of padrows, padcolumns, and their tilting angle. The initialization of the needed `AliTRDpadPlane` objects is done in `AliTRDgeometry::CreatePadPlaneArray()`. The number of padrows can be 12 (C0-type) or 16 (C1-type), the number of padcolumns is 144 in any case. Again, the numbering convention follows the definition given in [3]. Thus, the padrow numbers in a given pad plane increase from 0 to 11(15) with decreasing z-position, while the padcolumn numbers increase from 0 to 144 with increasing ϕ angle (i.e. counter clockwise). The tilting angle of the pads is 2 degrees, with alternating signs at different layers, beginning with +2 degrees for layer 0. The class `AliTRDpadPlane` provides a variety of functions that allow to assign a pad number (row/column) to signals generated at a given hit position and which are used during the digitization process.

1.2 Hit Generation

Author: C. Blume (blume@ikf.uni-frankfurt.de)

In the case of the TRD a single hit corresponds to a cluster of electrons resulting from the ionization of the detector gas. This ionization can be due to the normal energy loss process of a charged particle or due to the absorption of a transition radiation (TR) photon. A single TRD hit, as defined in `AliTRDhit` therefore contains the following data members:

`fTrack`	Index of MC particle in kine tree
`fX`	X-position of the hit in global coordinates
`fY`	Y-position of the hit in global coordinates
`fZ`	Z-position of the hit in global coordinates
`fDetector`	Number of the ROC (**DET** number)
`fQ`	Number of electrons created in the ionization step. Negative for TR hits
`fTime`	Absolute time of the hit in μs. Needed for pile-up events

On top of this, it is also stored in the `TObject` bit field status word whether a hit is inside the drift or the amplification region (see `AliTRDhit::FromDrift()` and `AliTRDhit::FromAmplification()`). The creation of hits is steered by `AliTRDv1::StepManager()`.

1.2.1 Energy loss

A charged particle, traversing the gas volume of the TRD chambers, will release charge proportional to its specific energy loss. In the TRD code this process is implemented in `AliTRDv1::StepManager()`. This implementation used a fixed step size. The standard value here is 0.1 cm, but other values can be set via `AliTRDv1::SetStepSize()`. The energy deposited in a given step is then calculated by the chosen MC program (typically Geant3.21), which after division by the ionization energy gives the number of electrons of the new hit. The version 2) will also work for an Ar/CO_2 mixture, which can be selected by `AliTRDSimParam::SetArgon()`.

1.2.2 Photons from transition radiation

Additionally to the hits from energy loss, also hits from the absorption of TR photons are generated. This is done in `AliTRDv1::CreateTRhit()`, which in turn is called by the chosen step manager for electrons and positrons entering the entering the drift volume. The process consists of two steps: first the number and energies of the TR photons have to be determined and then their absorption position inside the gas volume has to be calculated. The corresponding procedures, used by `AliTRDv1::CreateTRhit()`, are implemented in `AliTRDsimTR()`. This class contains a parametrization of TR photons generated by a regular foil stack radiator [4]. This parametrization has been tuned such that the resulting spectrum matches the one of the fiber radiator that used in reality. Since the TR production depends also on the momentum of the electron, the parameters have been adjusted in several momentum bins. After a TR photon has been generated and put on the particle stack, it is assumed that it follows a straight trajectory whose direction is determined by the momentum vector of the generating electron. Since the emission angle for TR photons is very small ($\sim 1/\gamma$) this is a valid approximation. The absorption length, which thus determines the TR hit position, is randomly chosen according to the absorption cross sections in the gas mixture. These energy dependent cross sections are also included in `AliTRDsimTR`.

1.2.3 Track references

The TRD simulation produces track references (`AliTrackReference`) each time a charged particle is entering the drift region and exiting the amplification region. These track references thus provide information on the position where the MC particle was entering and existing the sensitive region of a ROC, as well as on its momentum components at this positions. Also, the index to the MC particle in the kinematic tree is stored so that the full MC history can be retrieved.

1.3 Digitization

Author: C. Blume (blume@ikf.uni-frankfurt.de)

The second step in the simulation chain is the translation of the hit information, i.e. position and amount of deposited charge, into the final detector response that can be stored in digits objects (`AliTRDdigits`):

`fAmp`	Signal amplitude
`fId`	Number of the ROC (**DET** number)
`fIndexInList`	Track index
`fRow`	Pad row number
`fColumn`	Pad column number
`fTime`	Time bin number

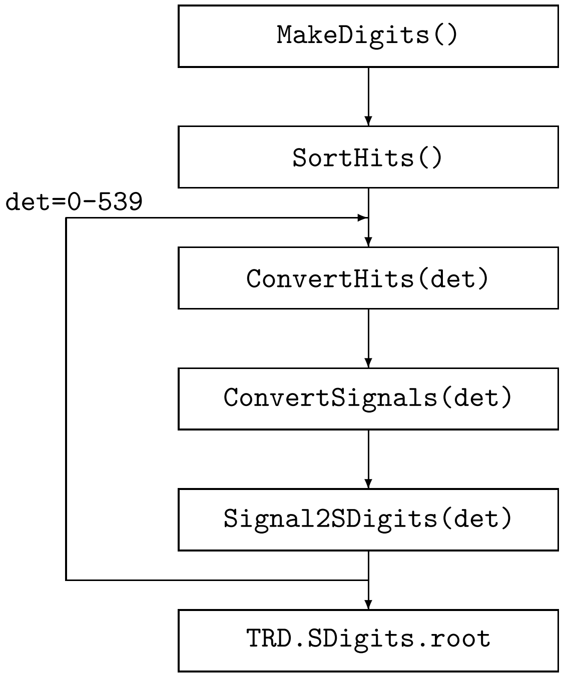
However, in practice `AliTRDdigits` is not used to store the digits information. Instead the data containers described in 1.3.4 are used for this purpose. The digitization process includes as an intermediate step between hit and digits the so-called summable digits, or sdigits:

$$\textbf{HITS} \Longrightarrow \textbf{SDIGITS} \Longrightarrow \textbf{DIGITS}$$

They sdigits contain the detector signals before discretization and the addition of noise and are used to merge several events into a single one.

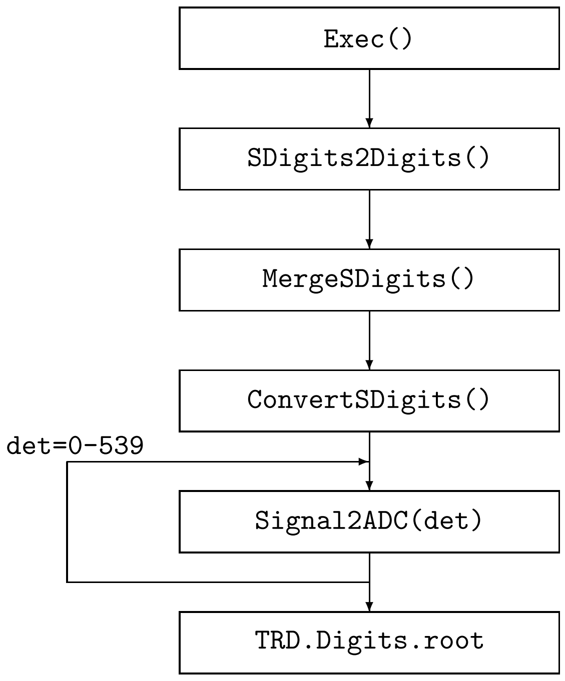
1.3.1 Digitizer

The class `AliTRDdigitizer` contains all the necessary procedures to convert hits into sdigits and subsequently sdigits into digits. The standard sequence to produce sdigits, as would be initiated by `AliSimulation`, is shown here:



The first function `SortHits()` sorts the simulated hits according to their **DET** number, so that the digitization procedures can be called for a single ROCs in the following loop. The function `ConvertHits()` does the conversion of the hit information into a detector signal. In this procedure each electron of a given hit is in principle followed along its path from the position of the primary ionization towards the anode wire. The position of this electron can be modified by diffusion in the gas (`AliTRDdigitizer::Diffusion()`), ExB effect (`AliTRDdigitizer::ExB()`), and absorption (`AliTRDdigitizer::Absorbtion()`, off per default). The drift time of the electrons is also modified according to their distance to the corresponding anode wire position (`AliTRDdigitizer::TimeStruct()`), since the electric field lines are not uniform inside the amplification region. This results in a non-isochronity of the drift time, which has been simulated with the GARFIELD program and the tabulated results of this simulation are used in the digitizing process to adjust the drift times accordingly. Once the position and the drift time of the electron at the anode wire plane are know, the signal induced on the pads can be calculated. This involves three effects: the pad response, which distributes the charge on several pads (`AliTRDdigitizer::PadResponse()`), the time response due to the slow ion drift and the PASA response function, which distributes the charge onto the following time bins, (`AliTRDdigitizer::TimeResponse()`), and the cross talk between neighboring pads `AliTRD-digitizer::CrossTalk()`). At the end of this procedure, the charge seen by each pad in each time bin is available. Also, the indices of maximally three MC particles in the kine tree contributing to a given pad signal are stored, so that in a later analysis it can be tested which particle generated what signal. As a next step the signals could either directly be converted into **DIGITS**, or, which is the default procedure, they are stored as **SDIGITIS**. The corresponding functions (`AliTRDdigitizer::Signal2SDigits()` and `AliTRDdigitizer::Signal2ADC()`) are called from `AliTRDdigitizer::ConvertSignals()`, depending on the configuration. The function `AliTRDdigitizer::Signal2SDigits()` stores the signals as **SDIGITS** in data structures of the type `AliTRDarraySignal` (see section 1.3.4).

If desired, the **SDIGITS** can now be added to the **SDIGITS** from other simulated events, e.g. in order to embed a specific signal into a background event (`AliTRDdigitizer::MergeSDigits()`). After this optional step, the **SDIGITS** are finally being converted into **DIGITS**. This process is steered by the function (`AliTRDdigitizer::ConvertSDigits()`).



The essential step in the final **SDIGITS** $\Longrightarrow$ **DIGITS** conversion is performed by the function `AliTRDdigitizer::Signal2ADC()`. Here pad signals, that are stored as floats, are finally translated into integer ADC values. This conversion involves a number of parameters: the pad coupling and time coupling factors, the gain of the PASA and of the amplification at the anode wire, and the input range and baseline of the ADCs. The coupling factors take into account that only a fraction of the incoming signal is sampled in the digitization process. At this point also the relative gain factors derived from the calibration procedures for a given dataset will be used to distort the simulated data correspondingly. The noise is generated according to a Gaussian distribution of a given width and added to the output. Finally, the converted signals are discretize into the ADC values of the defined resolution. At this stage also the zero suppression mechanism is applied to the simulated ADC values (`AliTRDdigitizer::ZS()`), in order to reduce the output volume (see section 1.3.5). These **DIGITS** can then serve as input to the raw data simulation (see section 1.4).

1.3.2 Simulation parameter

The parameters that are needed to configure the digitization, are either read from the OCDB (e.g. calibration gain factors) or are taken from the parameter class `AliTRDSimParam`. This class contains the default values of these parameters, but it can be configured in order to test different scenarios. The following table lists the available parameters:

Parameter	Description	Default value
`fGasGain`	Gas gain at the anode wire	4000
`fNoise`	Noise of the chamber readout	1250
`fChipGain`	Gain of the PASA	12.4
`fADCoutRange`	ADC output range (number of ADC channels)	1023 (10bit)
`fADCinRange`	ADC input range (input charge)	2000 (2V)
`fADCbaseline`	ADC intrinsic baseline in ADC channels	0
`fElAttachProb`	Probability for electron attachment per meter	0
`fPadCoupling`	Pad coupling factor	0.46
`fTimeCoupling`	Time coupling factor	0.4
`fDiffusionOn`	Switch for diffusion	kTRUE
`fElAttachOn`	Switch for electron attachment	kFALSE
`fTRFOn`	Switch for time response	kTRUE
`fCTOn`	Switch for cross talk	kTRUE
`fTimeStructOn`	Switch for time structure	kTRUE
`fPRFOn`	Switch for pad response	kTRUE
`fGasMixture`	Switch for gas mixture (0: Xe/CO2, 1: Ar/CO2)	0

1.3.3 Digits manager

Author: H. Leon Vargas (hleon@ikf.uni-frankfurt.de)

The class `AliTRDdigitsManager` handles arrays of data container objects in the form of ROOT's `TObjArray`. Its main functionality is that it provides setters and getters for the information of each chamber.

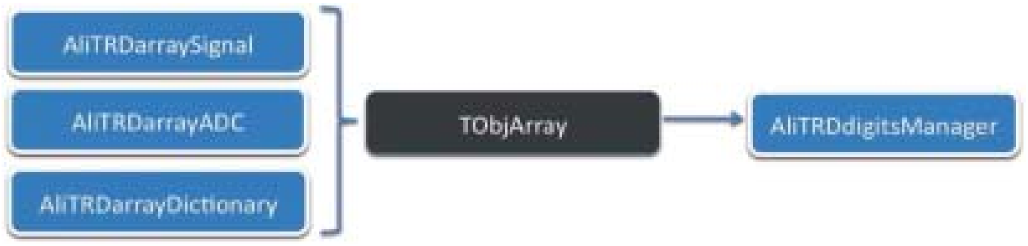


Figure 1.5: Data containers used in the class `AliTRDdigitsManager`.

1.3.4 Data containers

During simulation different kinds of information are created and stored in various data containers depending on their characteristics. These containers were designed with the idea of keeping the code as simple as possible and to ease its maintenance. The simulated signals or sdigits for a given row, column and time bin of each detector, as generated by `AliTRDdigitizer::ConvertHits()`, are stored in an object of the class `AliTRDarraySignal`. This class stores the data in an array of floating point values. In this case, the compression method takes as an argument a threshold. All the values equal or below that threshold will be set to zero during compression. The threshold can take any value greater or equal to zero. The sdigits data is used during event merging.

In the simulation the information about the particles that generated the hits (index in kine tree) is stored for each detector in an object of the class `AliTRDarrayDictionary`. In this case the information is stored in an array of integer values, which is initialized to -1.

In the digitizer, the signals stored in the sdigits are converted afterwards into ADC values and kept in objects of the class `AliTRDarrayADC`. This class saves the ADC values in an array of short values. The ADC range uses only the first 9 bits, bits 10 to 12 are used to set the pad status. An uncompressed object of the class `AliTRDarrayADC` should only contain values that are equal or greater than -1, because the compression algorithm of this class uses all the other negative values in the range of the short data type. The value -1 in the data array is used in the simulation to indicate where an ADC value was “zero suppressed”. This is done in this way so we are be able to discriminate between real zeroes and suppressed zeroes. For the details of the use of pad status refer to the method `AliTRDarrayADC::SetPadStatus()` in the implementation file of this class.

1.3.5 Zero suppression

Author: H. Leon Vargas (hleon@ikf.uni-frankfurt.de)

The zero suppression algorithm was applied at the end of digitization in order to decrease the size of the digits file. The code is implemented in the class `AliTRDmcmSim`. This algorithm is based on testing three conditions on the ADC values of three neighboring pads as seen in Fig. 1.6 (for more information see the Data Indication subsection in the TRAP User Manual). The conditions are the following:

1) Peak center detection:

ADC-1(t) $\leq$ ADC(t) $\geq$ ADC+1(t)

2) Cluster:

ADC-1(t)+ADC(t)+ADC+1(t) $>$ Threshold

3) Absolute Large Peak:

ADC(t) $>$ Threshold

If a given combination of these conditions is not fulfilled, the value ADC(t) is suppressed. The algorithm runs over all ADC values.

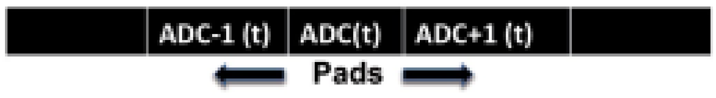


Figure 1.6: Zero suppression code.

1.4 Raw Data Simulation

1.5 Trigger Simulation

Author: J. Klein (jklein@physi.uni-heidelberg.de)

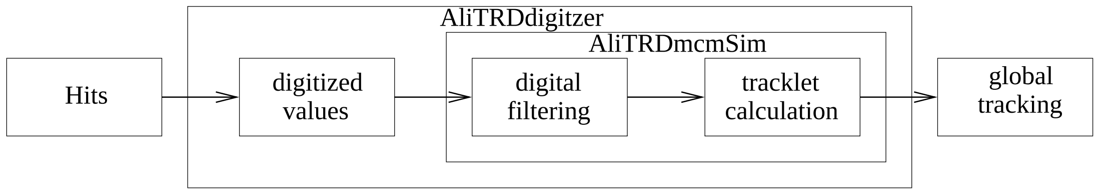


Figure 1.7: Overview of the trigger simulation

The trigger generation chain of the TRD can be simulated within AliRoot as well. It contains several stages as in the real hardware (s. Fig. 1.7).

For each event the hits in the active volume are converted to digitized signals in the AliTRD-digitizer. The digital processing as done in the TRAP is simulated in its method `RunDigitalProcessing()` calling the MCM simulation (in `AliTRDmcmSim`) which implements the filters, zero-suppression and tracklet calculation. Here the same integer arithmetics is used as in the real TRAP. The trigger-relevant preprocessed data, i.e. the tracklets, are stored using a dedicated loader. From there they are accessed by the GTU simulation which runs the stack-wise tracking. The individual stages are discussed in more detail in the following sections.

1.5.1 MCM simulation

The MCM simulation is contained in `AliTRDmcmSim`. This class mimics the digital part of an MCM. It can be used for the simulation after digitization has been performed.

Internally, an object of `AliTRDmcmSim` can hold the data of 21 ADC channels both raw and filtered. After the instantiation `Init()` has to be called to define the position of the MCM. Then, the data can be fed using either of the following methods:

`SetData(Int_t iadc, Int_t *adc)`
: Set the data for the given ADC channel *iadc* from an array *adc* containing the data for all timebins.

`SetData(Int_t iadc, Int_t it, Int_t adc)`
: Set the data for the given ADC channel *iadc* and timebin *it* to the value *adc*.

`SetData(AliTRDarrayADC *adcArray)`
: Set the data for the whole MCM from the digits array pointed to by *adcArray*.

`LoadMCM(AliRunLoader *rl, Int_t det, Int_t rob, Int_t mcm)`
: This method automatically initializes the MCM for the specified location and loads the relevant data via the runloader pointed by *rl*.

After loading of the data the processing stages can be run individually:

`Filter()`
: The pedestal, gain and tail cancellation filters are run on the currently loaded raw data. The filter settings (including bypasses) are used as configured in the TRAP (s. 1.5.2). The unfiltered raw data is kept such that it is possible to rerun Filter(), e.g. with different settings.

`Tracklet()`
: The tracklet calculation operates on the filtered data (which is identical to the unfiltered

data if Filter() was not called). First, the hits are calculated and the fit registers filled. Subsequently, the straight line fits for the four most promising tracklets are calculated.

`ZSMapping()`
: This methods performs the zero-suppression which can be based on different criteria (to be configured in the TRAP).

The results of the MCM simulation can be accessed in different ways:

`WriteData(AliTRDarrayADC *digits)`
: Hereby, the data are written to the pointed digits array. It is part of the TRAP configuration whether raw or filtered data is written (EBSF).

`ProduceRawStream(UInt_t *buf, Int_t bufsize, UInt_t iEv)`
: Produce the raw data stream for this MCM as it will appear in the raw data of the half-chamber.

`ProduceTrackletStream(UInt_t *buf, Int_t bufsize)`
: Produce the raw stream of tracklets as they appear in raw data.

`StoreTracklets()`
: The tracklets are stored via the runloader. This has to be called explicitly, otherwise the tracklets will not be written.

1.5.2 TRAP configuration

The TRAP configuration is kept in `AliTRDtrapConfig` which is implemented as singleton. After obtaining a pointer to the class by a call to `AliTRDtrapConfig::Instance()` values can be changed and read by:

`SetTrapReg(TrapReg_t reg, Int_t value, Int_t det, Int_t rob, Int_t mcm)`
: This sets the given TRAP register given as the abbreviation from the TRAP manual with preceding 'k' (enum) to the given value. If you specify *det*, *rob* or *mcm* the values are changed for individual MCMs. Not specified the setting is applied globally.

`GetTrapReg(TrapReg_t reg, Int_t det, Int_t rob, Int_t mcm)`
: This method gets the current value of the given TRAP registers. If the values are set individually for different MCMs you have to pass *det*, *rob* and *mcm*. Otherwise, these parameters can be omitted.

`PrintTrapReg(TrapReg_t reg, Int_t det, Int_t rob, Int_t mcm)`
: It is similar to the preceding method but prints the information to stdout.

The calculated tracklets can be stored by a call to `AliTRDmcmSim::StoreTracklets()`.

1.5.3 Tracklet classes

In order to unify the different sources of tracklets, e.g. real data or simulation, all implementations of tracklets derive from the abstract base class `AliTRDtrackletBase`. The following implementations are currently in use:

`AliTRDtrackletWord`
: This class is meant to represent the information as really available from the FEE, i.e. only a 32-bit word and the information on the detector it was produced on.

`AliTRDtrackletMCM`
: Tracklets of this type are produced in the MCM simulation and contain additional MC information.

`AliTRDtrackletGTU` This class is used during the GTU tracking and contains a pointer to a tracklet and information assigned to it during the global tracking.

1.5.4 GTU simulation

The simulation of the TRD global tracking on tracklets is steered by AliTRDgtuSim. This class provides all the interface. The following classes are involved:

`AliTRDgtuParam`
: This class contains or generates the relevant parameters used for the GTU tracking.

`AliTRDgtuTMU`
: This class holds the actual tracking algorithm as it runs in one Track Matching Unit (TMU) which corresponds to one stack.

The GTU simulation can be run by calling `AliTRDgtuSim::RunGTU(AliLoader *loader, AliESDEvent *esd)` where *loader* points to the TRD loader and *esd* to an ESD event. The latter can be omitted in which case the output is not written to the ESD. The tracklets are automatically retrieved via the loader and the found tracks of type `AliTRDtrackGTU` are internally stored in a tree for which a getter exists to access. If a pointer to an `AliESDEvent` is given, the tracks are also written to the ESD (as `AliESDTrdTrack`). For this the method `AliTRDtrackGTU::CreateTrdTrack()` is used which creates the `AliESDTrdTrack` (with reduced information compared to `AliTRDtrackGTU`).

1.5.5 CTP interface

The interface to the central trigger is defined in `AliTRDTrigger`. This class is called automatically during simulation and produces the trigger inputs for TRD (in `CreateInputs()`). They are only considered if they are part of the used trigger configuration (e.g. GRP/CTP/p-p.cfg).

The actual trigger generation has to be contained in `Trigger()`. Currently, the GTU simulation is run from here using the previously calculated tracklets. The generated tracks are stored and the trigger inputs are propagated to CTP. Which trigger classes make use of the TRD inputs has to be defined in the trigger configuration.

Reconstruction

Author: A. Bercuci (A.Bercuci@gsi.de)

2.1 Raw Data Reading

2.2 Cluster Finding

2.2.1 Cluster position reconstruction [1]

Author: A. Bercuci (A.Bercuci@gsi.de)

Calculation of cluster position in the radial direction in local chamber coordinates (with respect to the anode wire position) is using the following parameters:
t_0 - calibration aware trigger delay $[\mu s]$
v_d - drift velocity in the detector region of the cluster $[cm/\mu s]$
z - distance to the anode wire [cm]. By default average over the drift cell width
q & x_q - array of charges and cluster positions from previous clusters in the tracklet [a.u.]

The estimation of the radial position is based on calculating the drift time and the drift velocity at the point of estimation. The drift time can be estimated according to the expression:

$$t_{drift} = t_{bin} - t_0 - t_{cause}(x) - t_{TC}(q_{i-1}, q_{i-2}, ...) \tag{2.1}$$

where t_0 is the delay of the trigger signal. t_{cause} is the causality delay between ionization electrons hitting the anode and the registration of the mean signal by the electronics - due to the rising time of the TRF. A second order correction here comes from the fact that the time spreading of charge at anode is the convolution of TRF with the diffusion and thus cross-talk between clusters before and after local clusters changes with drift length. t_{TC} is the residual charge from previous (in time) clusters due to residual tails after tail cancellation. This tends to push cluster forward and depends on the magnitude of their charge.

The drift velocity varies with the drift length (and distance to anode wire) as described by cell structure simulation. Thus one, in principle, can calculate iteratively the drift length from the expression:

$$x = t_{drift}(x) * v_{drift}(x) \tag{2.2}$$

In practice we use a numerical approach (see `AliTRDcluster::GetXcorr()` and Figure 2.8 left) to correct for anisochronity obtained from a MC comparison (see `AliTRDclusterResolution-::ProcessSigma()` for the implementation). Also the calibration of the 0th approximation (no x dependence) for t_{cause} is obtained from MC comparisons and is impossible to disentangle in real life from trigger delay.

For **the calculation of the $r - \phi$ offset** of the cluster from the middle of the center pad three methods are implemented:

- Center of Gravity (COG) see `AliTRDcluster::GetDYcog()`
- Look-up Table (LUT) see `AliTRDcluster::GetDYlut()`
- Gaussian shape (GAUS) see `AliTRDcluster::GetDYgauss()`

[1]The procedures described in this section are implemented in the functions `AliTRDcluster::GetXloc()`, `AliTRDcluster::GetYloc()`, `AliTRDcluster::GetSX()` and `AliTRDcluster::GetSY()`.

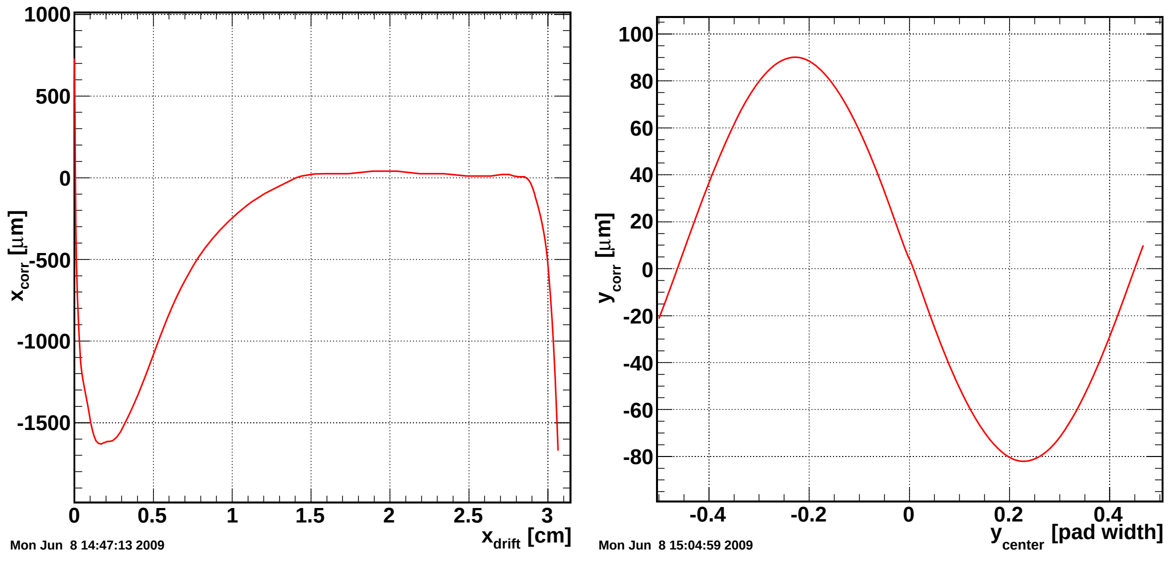


Figure 2.8: Correction of the radial and $r - \phi$ position of the TRD cluster.

In addition for the case of LUT method position corrections are also applied (see `AliTRDcluster-::GetYcorr()` and Figure 2.8 right).

One may calculate the $r - \phi$ offset, based on the Gaussian approximation of the PRF, from the signals q_{i-1}, q_i and q_{i+1} in the 3 adjacent pads by:

$$y = \frac{1}{w_1 + w_2}\left[w_1\left(y_0 - \frac{W}{2} + \frac{s^2}{W}\ln\frac{q_i}{q_{i-1}}\right) + w_2\left(y_0 + \frac{W}{2} + \frac{s^2}{W}\ln\frac{q_{i+1}}{q_i}\right)\right] \tag{2.3}$$

where W is the pad width, y_0 is the position of the middle of the center pad and s^2 is given by

$$s^2 = s_0^2 + s_{diff}^2(x,B) + \frac{\tan^2(\phi - \alpha_L) * l^2}{12} \tag{2.4}$$

with s_0 being the PRF for 0 drift and track incidence ϕ equal to the Lorentz angle α_L and the diffusion term being described by:

$$s_{diff}(x,B) = \frac{D_L\sqrt{x}}{1 + (\omega\tau^2)} \tag{2.5}$$

with x being the drift length. The weights w_1 and w_2 are taken to be q_{i-1}^2 and q_{i+1}^2 respectively.

Determination of shifts by comparing with MC

The resolution of the cluster corrected for pad tilt with respect to MC in the $r-\phi$ (measuring) plane can be expressed by:

$$\begin{aligned}
\Delta y &= w - y_{MC}(x_{cl}) && (2.6)\\
w &= y_{cl}^{'} + h * (z_{MC}(x_{cl}) - z_{cl}) && (2.7)\\
y_{MC}(x_{cl}) &= y_0 - dy/dx * x_{cl} && (2.8)\\
z_{MC}(x_{cl}) &= z_0 - dz/dx * x_{cl} && (2.9)\\
y_{cl}^{'} &= y_{cl} - x_{cl} * \tan(\alpha_L) && (2.10)
\end{aligned}$$

where x_{cl} is the drift length attached to a cluster, y_{cl} is the $r-\phi$ coordinate of the cluster measured by charge sharing on adjacent pads and y_0 and z_0 are MC reference points (as example the track references at entrance/exit of a chamber). If we suppose that both $r-\phi$ (y) and radial (x) coordinate of the clusters are affected by errors we can write

$$x_{cl} = x^*_{cl} + \delta x \quad (2.11)$$
$$y_{cl} = y^*_{cl} + \delta y \quad (2.12)$$

where the starred components are the corrected values. Thus by definition the following quantity

$$\Delta y^* = w^* - y_{MC}(x^*_{cl}) \quad (2.13)$$

has 0 average over all dependency. Using this decomposition we can write:

$$< \Delta y >=< \Delta y^* > + < \delta x * (dy/dx - h * dz/dx) + \delta y - \delta x * \tan(\alpha_L) > \quad (2.14)$$

which can be transformed to the following linear dependence:

$$< \Delta y >=< \delta x > *(dy/dx - h * dz/dx)+ < \delta y - \delta x * \tan(\alpha_L) > \quad (2.15)$$

if expressed as function of $dy/dx - h * dz/dx$. Furtheremore this expression can be plotted for various clusters i.e. we can explicitely introduce the diffusion (x_{cl}) and drift cell - anisochronity (z_{cl}) dependences. From plotting this dependence and linear fitting it with:

$$< \Delta y >= a(x_{cl}, z_{cl}) * (dy/dx - h * dz/dx) + b(x_{cl}, z_{cl}) \quad (2.16)$$

the systematic shifts will be given by:

$$\delta x(x_{cl}, z_{cl}) = a(x_{cl}, z_{cl}) \quad (2.17)$$
$$\delta y(x_{cl}, z_{cl}) = b(x_{cl}, z_{cl}) + a(x_{cl}, z_{cl}) * \tan(\alpha_L) \quad (2.18)$$

In Figure 2.9 left there is an example of such dependency.

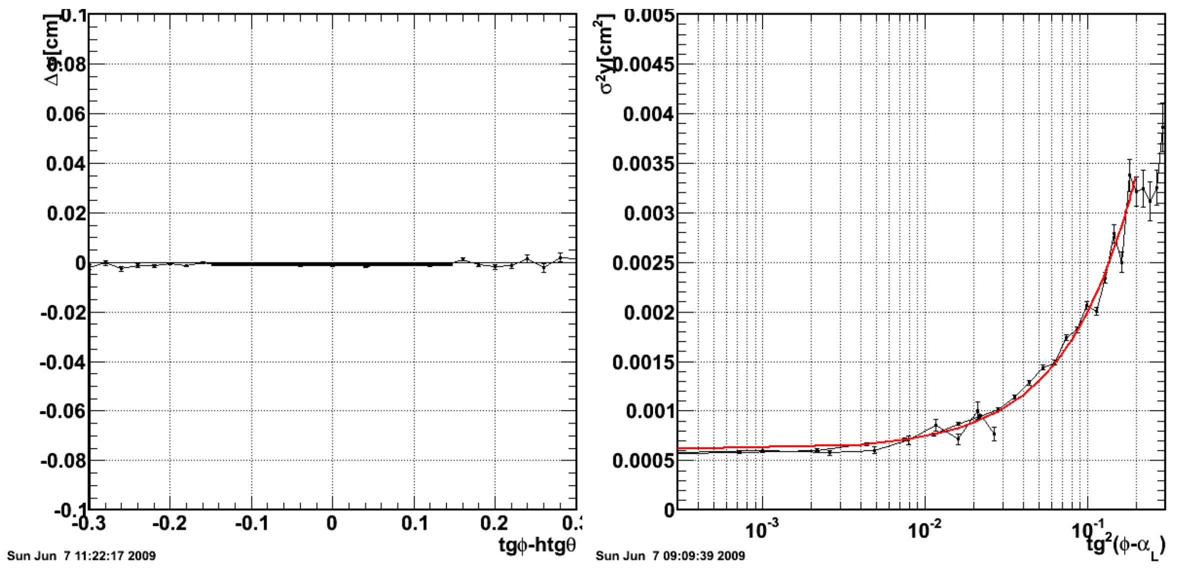


Figure 2.9: Linear relation to estimate radial and $r-\phi$ cluster shifts and error.

The occurance of the radial shift is due to the following conditions:

- The approximation of a constant drift velocity over the drift length (larger drift velocities close to the cathode wire plane).
- The superposition of charge tails in the amplification region (first clusters appear to be located at the anode wire).
- The superposition of charge tails in the drift region (shift towards anode wire).
- Diffusion effects which convolute with the TRF thus enlarging it.
- Approximate knowledge of the TRF (approximate measuring in test beam conditions).

The numerical results for ideal simulations for the radial are displayed in Figure 2.8.

The representation of $dy = f(y_cen, x_drift|layer, \phi = \tan(\alpha_L))$ can be also used to estimate the systematic shift in the $r - \phi$ coordinate resulting from an imperfection in the cluster shape parameterization. From Eq. 2.14 with $\phi = \tan(\alpha_L)$ one gets:

$$
\begin{aligned}
< \Delta y > &= < \delta x > *(\tan(\alpha_L) - h * dz/dx) + < \delta y - \delta x * \tan(\alpha_L) > && (2.19)\\
< \Delta y > (y_{cen}) &= -h* < \delta x > (x_{drift}, q_{cl}) * dz/dx + \delta y(y_{cen}, ...) && (2.20)
\end{aligned}
$$

where all dependences are made explicit. This last expression can be used in two ways:

- By average on the dz/dx we can determine directly dy (the method implemented here - see Figure 2.8 right).
- By plotting as a function of $dzdx$ one can determine both dx and dy components in an independent method.

The occurance of the $r - \phi$ shift is due to the following conditions:

- Approximate model for cluster shape (LUT).
- Rounding-up problems.

2.2.2 Cluster error parametrization [2]

Author: A. Bercuci (A.Bercuci@gsi.de)

The error of TRD cluster is represented by the variance in the $r - \phi$ and radial direction. For the z direction the error is simply given by:

$$\sigma_z^2 = L_{pad}^2/12 \tag{2.21}$$

The parameters on which the $r - \phi$ **error parameterization** depends are:

- s^2 - variance due to PRF width for the case of Gauss model. Replaced by parameterization in case of LUT.
- dt - transversal diffusion coefficient.
- $e \times B$ - tangens of Lorentz angle.

[2]The procedures described in this section are implemented in the functions `AliTRDcluster::SetSigmaY2()`, `AliTRDclusterResolution::ProcessCharge()`, `AliTRDclusterResolution::ProcessCenterPad()`, `AliTRDclusterResolution::ProcessSigma()` and `AliTRDclusterResolution::ProcessMean()`.

- x-drift length - with respect to the anode wire.
- z-offset from the anode wire.
- $\tan(p)$ - local tangens of the track momentum azimuthal angle.

The ingredients from which the error is computed are:

- PRF (charge sharing on adjacent pads) - see `AliTRDcluster::GetSYprf()`.
- Diffusion (dependence with drift length and [2nd order] distance to anode wire) (see `AliTRDcluster::GetSYdrift()`).
- Charge of the cluster (complex dependence on gain and tail cancellation) - see (`AliTRD-cluster::GetSYcharge()`).
- Lorentz angle (dependence on the drift length and [2nd order] distance to anode wire) - see `AliTRDcluster::GetSX()`.
- Track angle (superposition of charges on the anode wire) (see `AliTRDseedV1::Fit()`).
- Projection of radial (x) error on $r-\phi$ due to fixed value assumed in tracking for x - see `AliTRDseedV1::Fit()`.

The last 2 contributions to cluster error can be estimated only during tracking when the track angle is known ($\tan(p)$). For this reason the errors (and optional position) of TRD clusters are recalculated during tracking and thus clusters attached to tracks might differ from bare clusters.

Taking into account all contributions one can write the the TRD cluster error parameterization as:

$$\sigma_y^2 = (\sigma_{diff} * \mathrm{Gauss}(0, \mathrm{s_{ly}}) + \delta_\sigma(\mathrm{q}))^2 + \tan^2(\alpha_\mathrm{L}) * \sigma_\mathrm{x}^2 + \tan^2(\phi - \alpha_\mathrm{L}) * \sigma_\mathrm{x}^2 + [\tan(\phi - \alpha_\mathrm{L}) * \tan(\alpha_\mathrm{L}) * \mathrm{x}]^2/12 \tag{2.22}$$

From this formula one can deduce that the simplest calibration method for PRF and diffusion contributions is by measuring resolution at $B = 0$ T and $\phi = 0$. To disentangle further the two remaining contributions one has to represent s^2 as a function of drift length.

In the Gaussian model the diffusion contribution can be expressed as:

$$\sigma_y^2 = \sigma_{PRF}^2 + \frac{x\delta_t^2}{(1+\tan(\alpha_L))^2} \tag{2.23}$$

thus resulting the PRF contribution. For the case of the LUT model both contributions have to be determined from the fit (see `AliTRDclusterResolution::ProcessCenterPad()` for details).

Parameterization with respect to the distance to the middle of the center pad

If $\phi = \alpha_L$ in Eq. 2.22 one gets the following expression:

$$\sigma_y^2 = \sigma_y^2|_{B=0} + \tan^2(\alpha_L) * \sigma_x^2 \tag{2.24}$$

where we have explicitely marked the remaining term in case of absence of magnetic field. Thus one can use the previous equation to estimate s_y for B = 0 and than by comparing in magnetic field conditions one can get the s_x. This is a simplified method to determine the error parameterization for s_x and s_y as compared to the one implemented in `ProcessSigma()`. For more details on cluster error parameterization please see also `AliTRDcluster::SetSigmaY2()`.

Parameterization with respect to drift length and distance to the anode wire

As the $r - \phi$ coordinate is the only one which is measured by the TRD detector we have to rely on it to estimate both the radial (x) and $r - \phi$ (y) errors. This method is based on the following assumptions. The measured error in the y direction is the sum of the intrinsic contribution of the $r-\phi$ measurement with the contribution of the radial measurement - because x is not a parameter of Alice track model (Kalman).

$$\sigma^2|_y = \sigma^2_{y*} + \sigma^2_{x*} \tag{2.25}$$

In the general case

$$\begin{aligned} \sigma^2_{y*} &= \sigma^2_y + \tan^2(\alpha_L)\sigma^2_{x_{drift}} &(2.26)\\ \sigma^2_{x*} &= \tan^2(\phi - \alpha_L) * (\sigma^2_{x_{drift}} + \sigma^2_{x_0} + \tan^2(\alpha_L) * x^2/12) &(2.27)\end{aligned}$$

where we have explicitely show the Lorentz angle correction on y and the projection of radial component on the y direction through the track angle in the bending plane (ϕ). Also we have shown that the radial component in the last equation has two terms, the drift and the misalignment (x_0). For ideal geometry or known misalignment one can solve the equation

$$\sigma^2|_y = \tan^2(\phi - \alpha_L) * (\sigma^2_x + \tan^2(\alpha_L) * x^2/12) + [\sigma^2_y + \tan^2(\alpha_L)\sigma^2_x] \tag{2.28}$$

by fitting a straight line:

$$\sigma^2|_y = a(x_{cl}, z_{cl}) * \tan^2(\phi - \alpha_L) + b(x_{cl}, z_{cl}) \tag{2.29}$$

the error parameterization will be given by:

$$\begin{aligned} \sigma_x(x_{cl}, z_{cl}) &= \sqrt{a(x_{cl}, z_{cl}) - \tan^2(\alpha_L) * x^2/12} &(2.30)\\ \sigma_y(x_{cl}, z_{cl}) &= \sqrt{b(x_{cl}, z_{cl}) - \sigma^2_x(x_{cl}, z_{cl}) * \tan^2(\alpha_L)} &(2.31)\end{aligned}$$

In Figure 2.9 left, there is an example of such dependency.

The error parameterization obtained by this method are implemented in the functions `AliTRDcluster::GetSX()` and `AliTRDcluster::GetSYdrift()`.

An independent method to determine s_y as a function of drift length (see `AliTRDclusterResolution::ProcessCenterPad()`) is to plot cluster resolution as a function of drift length at $\phi = \alpha_L$ as seen in Eq. 2.24. Thus one can use directly the previous equation to estimate s_y for $B = 0$ and than by comparing in magnetic field conditions one can get the s_x.

One has to keep in mind that while the first method returns the mean s_y over the distance to the middle of center pad (y_{center}) distribution the second method returns the *STANDARD* value at $y_{center} = 0$ (maximum). To recover the standard value one has to solve the obvious equation:

$$\sigma_y^{STANDARD} = \frac{<\sigma_y>}{\int s exp(s^2/\sigma) ds} \tag{2.32}$$

with "$< s_y >$" being the value calculated in first method and "sigma" the width of the s_y distribution calculated in the second.

Parameterization with respect to cluster charge

In Eq. 2.24 one can explicitely write:

$$\sigma_y|_{B=0} = \sigma_{diff} * Gauss(0, s_{ly}) + \delta_\sigma(q) \tag{2.33}$$

which further can be simplified to:

$$< \sigma_y|_{B=0} > (q) \quad = \quad < \sigma_y > + \delta_\sigma(q) \tag{2.34}$$

$$< \sigma_y > \quad = \quad \int f(q)\sigma_y dq \tag{2.35}$$

The results for s_y and $f(q)$ are displayed in Fig. 2.10: The function has to extended to accom-

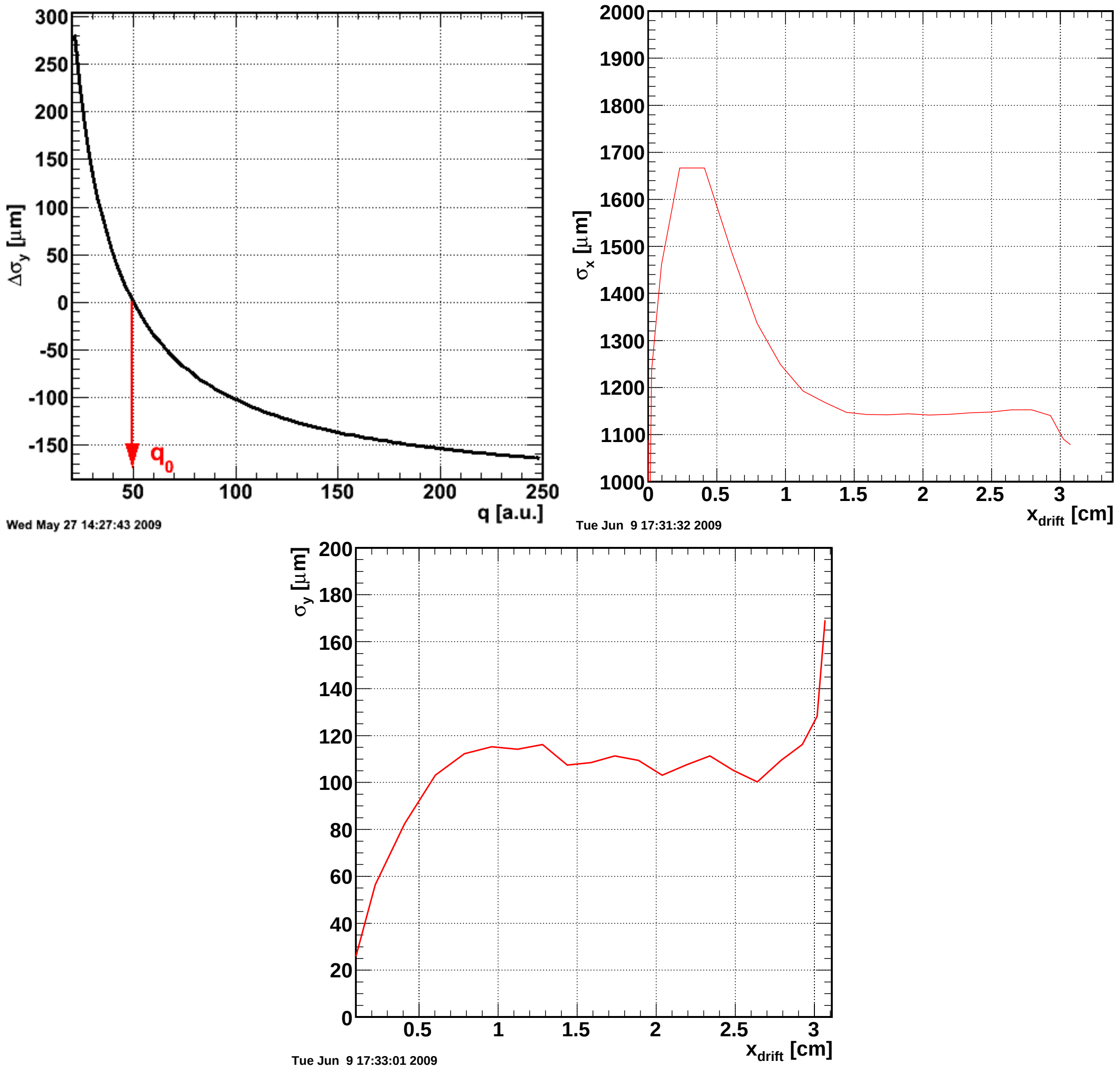


Figure 2.10: Cluster error parameterization for different components.

modate gain calibration scaling and errors.

2.3 The TRD tracklet

Author: A. Bercuci (A.Bercuci@gsi.de)

The tracking in TRD can be done in two major ways:

- Track prolongation from TPC.

- Stand alone track finding.

The first mode is the main tracking mode for all barrel tracks while the second is used to peak-up track segments fully contained in the TRD fiducial volume like conversions. Another feature of the TRD tracking besides the relative high thickness (conversions) is the spatial correlation of the signals in the radial direction due to residual tails in the cluster signals. This feature asked for an intermediate step between clusters and tracks, the tracklets. The TRD tracklets are linear fits of the clusters from one chamber. They are implemented in the class `AliTRDseedV1` and they represent the core of the TRD offline reconstruction. In the following the tracklets will be described independently of the framework in which they are living (tracking) in the sections 2.3.1, 2.3.2 and 2.3.3 and than their usage will be outlined in the barrel (section 2.4.1) and stand alone tracking (section 2.4.2).

2.3.1 Tracklet building - Attaching clusters to tracklet [1]

Projective algorithm to attach clusters to seeding tracks. The following steps are performed:

- Collapse x coordinate for the full detector along track direction $dydx$.
- Truncated mean on y ($r - \phi$) direction.
- Purge clusters.
- Truncated mean on z direction.
- Purge clusters.

Optionally one can use the z, dz/dx information from the seeding track to correct for tilting.

We start up by defining the track direction in the xy plane and roads. The roads are calculated based on tracking information (variance in the $r - \phi$ direction) and estimated variance of the standard clusters (see `AliTRDcluster::SetSigmaY2()`) corrected for tilt (see `GetCovAt()`). From this the road is:

$$r_y = 3 * \sqrt{12 * (\sigma^2_{Trk}(y) + \frac{\sigma^2_{cl}(y) + \tan^2(\alpha_L)\sigma^2_{cl}(z)}{1 + \tan^2(\alpha_L)})} \quad (2.36)$$

$$r_z = 1.5 * L_{pad} \quad (2.37)$$

2.3.2 Tracklet fitting[2]

Fit in the xy plane

The fit is performed to estimate the y position of the tracklet and the track angle in the bending plane. The clusters are represented in the chamber coordinate system (with respect to the anode wire - see `AliTRDtrackerV1::FollowBackProlongation()` on how this is set). The x and y position of the cluster and also their variances are known from clusterizer level (see `AliTRDcluster::GetXloc()`, `AliTRDcluster::GetYloc()`, `AliTRDcluster::GetSX()` and `AliTRDcluster::GetSY()`). If a Gaussian approximation is used to calculate y coordinate of the cluster the position is recalculated taking into account the track angle.

Since errors are calculated only in the y directions, radial errors (x direction) are mapped to y by projection i.e.

$$\sigma_{x|y} = \tan(\phi)\sigma_x \quad (2.38)$$

[1]The procedures described in this section are implemented in the function `AliTRDseedV1::AttachClusters()`.

[2]The procedures described in this section are implemented in the function `AliTRDseedV1::Fit()`.

and also by the Lorentz angle correction.

Fit in the xz plane

The ”fit” is performed to estimate the radial position (x direction) where pad row cross happens. If no pad row crossing the z position is taken from geometry and radial position is taken from the xy fit (see below).

There are two methods to estimate the radial position of the pad row cross:
1. leading cluster radial position: Here the lower part of the tracklet is considered and the last cluster registered (at radial x_0) on this segment is chosen to mark the pad row crossing. The error of the z estimate is given by :

$$\sigma_z = \tan(\theta)\Delta x_{x_0}/\sqrt{12} \tag{2.39}$$

The systematic errors for this estimation are generated by the following sources: - no charge sharing between pad rows is considered (sharp cross) - missing cluster at row cross (noise peak-up, under-threshold signal etc.).
2. charge fit over the crossing point: Here the full energy deposit along the tracklet is considered to estimate the position of the crossing by a fit in the qx plane. The errors in the q directions are parameterized as $\sigma_q = q^2$. The systematic errors for this estimation are generated by the following sources:

- No general model for the qx dependence.
- Physical fluctuations of the charge deposit.
- Gain calibration dependence.

Estimation of the radial position of the tracklet

For pad row cross the radial position is taken from the xz fit (see above). Otherwise it is taken as the interpolation point of the tracklet i.e. the point where the error in y of the fit is minimum. The error in the y direction of the tracklet is (see `AliTRDseedV1::GetCovAt()`):

$$\sigma_y = \sigma^2_{y_0} + 2x\, cov(y_0, dy/dx) + \sigma^2_{dy/dx} \tag{2.40}$$

and thus the radial position is:

$$x = -cov(y_0, dy/dx)/\sigma^2_{dy/dx} \tag{2.41}$$

Estimation of tracklet position error

The error in y direction is the error of the linear fit at the radial position of the tracklet while in the z direction is given by the cluster error or pad row cross error. In case of no pad row cross this is given by:

$$\sigma_y = \sigma^2_{y_0} - 2cov^2(y_0, dy/dx)/\sigma^2_{dy/dx} + \sigma^2_{dy/dx} \tag{2.42}$$

$$\sigma_z = L_{pad}/\sqrt{12} \tag{2.43}$$

For pad row cross the full error is calculated at the radial position of the crossing (see above) and the error in z by the width of the crossing region - being a matter of parameterization.

$$\sigma_z = \tan(\theta)\Delta x_{x_0}/\sqrt{12} \tag{2.44}$$

In case of no tilt correction (default in the barrel tracking) the tilt is taken into account by the rotation of the covariance matrix. See `AliTRDseedV1::GetCovAt()` or 2.3.3 for details.

2.3.3 Tracklet errors[3]

In general, for the linear transformation

$$Y = T_x X^T \tag{2.45}$$

the error propagation has the general form

$$C_Y = T_x C_X T_x^T \tag{2.46}$$

We apply this formula 2 times. First to calculate the covariance of the tracklet at point x we consider:

$$\begin{aligned} T_x &= (1\ x) && (2.47)\\ X &= (y0\ dy/dx) && (2.48)\\ C_X &= \begin{pmatrix} Var(y0) & Cov(y0, dy/dx) \\ Cov(y0, dy/dx) & Var(dy/dx) \end{pmatrix} && (2.49) \end{aligned}$$

and secondly to take into account the tilt angle

$$\begin{aligned} T_\alpha &= \begin{pmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{pmatrix} && (2.50)\\ X &= (y\ z) && (2.51)\\ C_X &= \begin{pmatrix} Var(y) & 0 \\ 0 & Var(z) \end{pmatrix} && (2.52) \end{aligned}$$

using simple trigonometric one can write for this last case

$$C_Y = \frac{1}{1+\tan^2\alpha} \begin{pmatrix} \sigma_y^2 + \tan^2(\alpha)\sigma_z^2 & \tan(\alpha)(\sigma_z^2 - \sigma_y^2) \\ \tan(\alpha)(\sigma_z^2 - \sigma_y^2) & \sigma_z^2 + \tan^2(\alpha)\sigma_y^2 \end{pmatrix} \tag{2.53}$$

which can be approximated for small alphas (2 deg) with

$$C_Y = \begin{pmatrix} \sigma_y^2 & (\sigma_z^2 - \sigma_y^2)\tan(\alpha) \\ ((\sigma_z^2 - \sigma_y^2)\tan(\alpha) & \sigma_z^2 \end{pmatrix} \tag{2.54}$$

before applying the tilt rotation we also apply systematic uncertainties to the tracklet position which can be tuned from outside via the `AliTRDrecoParam::SetSysCovMatrix()`. They might account for extra misalignment/miscalibration uncertainties.

2.3.4 Energy loss calculations[4]

Using the linear approximation of the track inside one TRD chamber (TRD tracklet) the charge per unit length can be written as:

$$\frac{dq}{dl}(x) = \frac{q_c}{dx(x) * \sqrt{1 + (\frac{dy}{dx})^2_{fit} + (\frac{dz}{dx})^2_{ref}}} \tag{2.55}$$

where q_c is the total charge collected in the current time bin and dx is the length of the time bin (see Fig. 2.11 right). The representation of charge deposit used for PID differs thus in principle from the measured dQ/dt distribution (see Fig. 2.11 left) The following correction are applied:

[3]The procedures described in this section are implemented in the function `AliTRDseedV1::GetCovAt()`.

[4]The procedures described in this section are implemented in the function `AliTRDseedV1::CookdEdx()` and `AliTRDseedV1::GetdQdl()`.

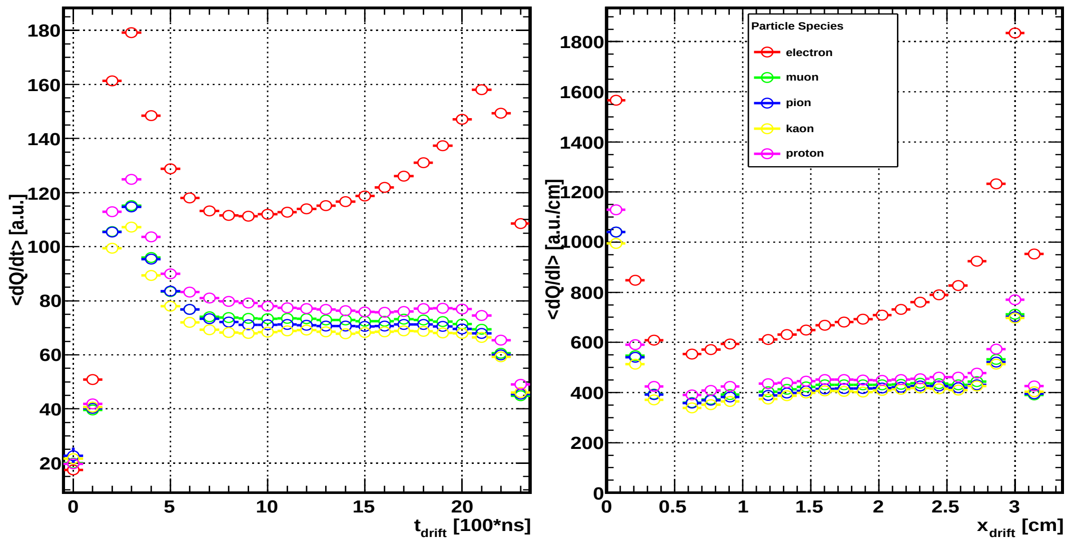


Figure 2.11: Energy loss measurement on the tracklet as a function of drift time [left] and respectively drift length [right] for different particle species.

- Charge: pad row cross corrections [diffusion and TRF asymmetry] TODO.
- dx: anisochronity.

Due to the anisochronity of the TRD detector drift velocity varies as function of drift length and distance to the anode wire. Thus

$$dx(x) = dx(\inf) + \delta_x(x,z) \tag{2.56}$$

$$= dt * v_d^{\inf} + \delta_x(x,z) \tag{2.57}$$

the dependence of δ_x can be found in Fig. 2.8.

2.3.5 Particle identification[5]

Retrieve the PID probabilities for $e^\pm$, $\mu^\pm$, $K^\pm$, $\pi^\pm$ and $p^\pm$ from the OCDB according to tracklet information:

- Estimated momentum at tracklet reference point.
- dE/dx measurements.
- Tracklet length.
- TRD layer.

According to the steering settings specified in the reconstruction one of the following methods are used.

- Neural Network [default] - option ”nn”.
- 2D Likelihood - option ”!nn”.

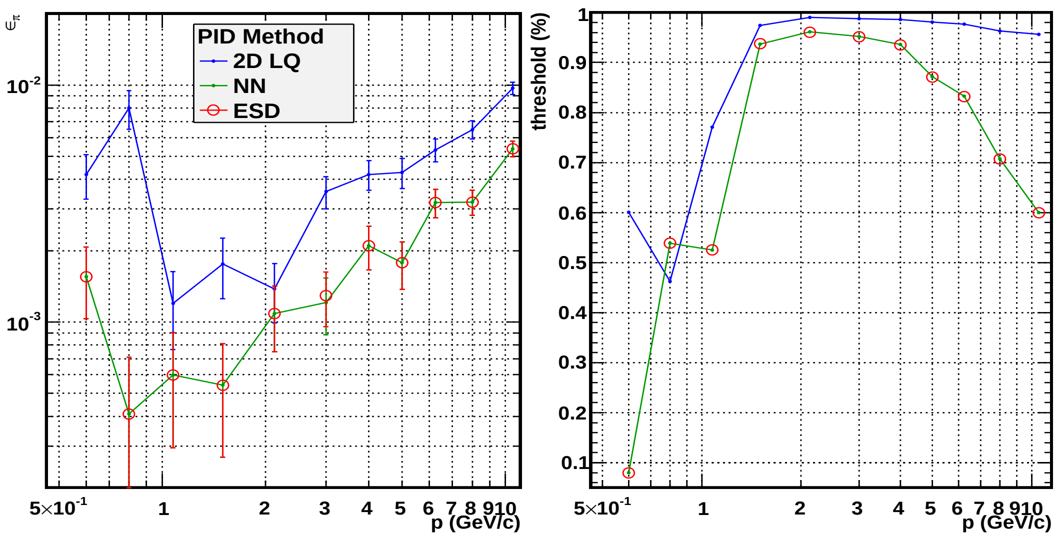


Figure 2.12: Pion efficiency capability of the TRD for tracks with 6 tracklets as a function of momentum [left] and the corresponding threshold value for selecting 90% of electrons [right] for the two methods used.

At track level the PID information is calculated by delegating the function of the tracklets. The number of tracklets used is also computed. The tracklet information are considered independent. For the moment no global track measurement of PID is performed as for example to estimate bremsstrahlung probability based on global χ^2 of the track. The status bit `AliESDtrack::kTRDpid` is set during the call of `AliTRDtrackV1::UpdateESDtrack()`. The PID performance of the TRD for tracks with 6 tacklets is displayed in Fig. 2.12.

2.4 Tracking

Author: A. Bercuci (A.Bercuci@gsi.de)

The tracking procedures in TRD are responsible to attach clusters to tracks and to estimate/update the track parameters accordingly. The main class involved in this procedure is `AliTRDtrackerV1` and the helper classes `AliTRDcluster`, `AliTRDseedV1` and `AliTRDtrackV1`. Additionally, information from `AliTRDrecoParam` is mandatory to select the proper setup of the reconstruction.

2.4.1 Track propagation in barrel tracking[1]

Propagate the ESD tracks from TPC to TOF detectors and building of the TRD track. For building a TRD track an ESD track is used as seed. The informations obtained on the TRD track (measured points, covariance, PID, etc.) are than used to update the corresponding ESD track.

[5]The procedures described in this section are implemented in the function `AliTRDtrackV1::CookPID()` and `AliTRDseedV1::CookPID()`.

[1]The procedures described in this section are implemented in the function `AliTRDtrackerV1::PropagateBack()`.

Each track seed is first propagated to the geometrical limit of the TRD detector. Its prolongation is searched in the TRD and if corresponding clusters are found tracklets are constructed out of them (see `AliTRDseedV1::AttachClusters()`) and the track is updated. Otherwise the ESD track is left unchanged.

The following steps are performed:

1. Selection of tracks based on the variance in the $y - z$ plane.

2. Propagation to the geometrical limit of the TRD volume. If track propagation fails the `AliESDtrack::kTRDStop` is set.

3. Prolongation inside the fiducial volume (see `AliTRDtrackerV1::FollowBackProlonga-tion()`) and marking the following status bits:

`AliESDtrack::kTRDin`	Tracks enters the TRD fiducial volume.
`AliESDtrack::kTRDStop`	Tracks fails propagation.
`AliESDtrack::kTRDbackup`	Tracks fulfills the χ^2 conditions and qualifies for refitting.

4. Writting to friends, PID, MC label, quality etc. Setting the status bit `AliESDtrack::kTRDout`.

5. Propagation to TOF. If track propagation fails the `AliESDtrack::kTRDStop` is set.

TRD Tracklet initialization and Kalman fit[2]

Starting from the arbitrary radial position of the track this is extrapolated through the 6 TRD layers. The following steps are being performed for each layer:

1. Propagate track to the entrance of the next chamber:
   - Get chamber limits in the radial direction.
   - Check crossing sectors.
   - Check track inclination.
   - Check track prolongation against boundary conditions (see exclusion boundaries on `AliTRDgeometry::IsOnBoundary()`).

2. Build tracklet (see `AliTRDseed::AttachClusters()` for details) for this layer if needed. If only the Kalman filter is needed and tracklets are already linked to the track this step is skipped.

3. Fit tracklet using the information from the Kalman filter.

4. Propagate and update track at reference radial position of the tracklet.

5. Register tracklet with the tracker and track. Update pulls monitoring.

During the propagation a bit map is filled detailing the status of the track in each TRD chamber.

[2]The procedures described in this section are implemented in the function `AliTRDtrackerV1::FollowBackProlongation()`.

`AliTRDtrackV1::kProlongation`	Track prolongation failed.
`AliTRDtrackV1::kPropagation`	Track prolongation failed.
`AliTRDtrackV1::kAdjustSector`	Failed during sector crossing.
`AliTRDtrackV1::kSnp`	Too large bending.
`AliTRDtrackV1::kTrackletInit`	Fail to initialize tracklet.
`AliTRDtrackV1::kUpdate`	Fail to attach clusters or fit the tracklet.
`AliTRDtrackV1::kUnknown`	Anything which is not covered before.

By default the status of the track before first TRD update is saved.

2.4.2 Stand alone track finding[3]

Seeding tracklets and build candidate TRD tracks. The procedure is used during barrel tracking to account for tracks which are either missed by TPC prolongation or are conversions inside the TRD volume. For stand alone tracking the procedure is used to estimate all tracks measured by TRD.

TRD track finding[4]

The following steps are performed:

1. Build seeding layers by collapsing all time bins from each of the four seeding chambers along the radial coordinate. See `AliTRDtrackingChamber::GetSeedingLayer()` for details. The chambers selection for seeding is described in `AliTRDtrackerV1::Clusters2-TracksStack()`.

2. By using the seeding clusters from the seeding layer (step 1) build combinatorics using the following algorithm:

   - For each seeding cluster in the lower seeding layer find.
   - All seeding clusters in the upper seeding layer inside a road defined by a given ϕ angle. The angle is calculated on the minimum p_{t} of tracks from the main vertex, accessible by the stand alone tracker.
   - For each pair of two extreme seeding clusters select middle upper cluster using roads defined externally by the reco params.
   - Select last seeding cluster as the nearest to the linear approximation of the track described by the first three seeding clusters. The implementation of the road calculation and cluster selection can be found in the functions `AliTRDchamberTimeBin::Build-Cond()` and `AliTRDchamberTimeBin::GetClusters()`.

3. Helix fit to the set of eeding clusters (see `AliTRDtrackerFitter::FitRieman(AliTRD-cluster**)`). No tilt correction is performed at this level

4. Initialize seeding tracklets in the seeding chambers.

5. **Filter 0:** χ^2 cut on the y and z directions. The threshold is set externally by the reco params.

[3]The procedures described in this section are implemented in the function `AliTRDtrackerV1::Clusters2TracksStack()`.

[4]The procedures described in this section are implemented in the function `AliTRDtrackerV1::MakeSeeds()`.

6. Attach (true) clusters to seeding tracklets (see `AliTRDseedV1::AttachClusters()`) and fit tracklet (see `AliTRDseedV1::Fit()`). The number of used clusters used by current seeds should not exceed ... (25).

7. **Filter 1:** Check if all 4 seeding tracklets are correctly constructed.

8. Helix fit to the clusters from the seeding tracklets with tilt correction. Refit tracklets using the new approximation of the track. The model of the Riemann tilt fit is based on solving simultaneously the equations:

$$R^2 = (x - x_0)^2 + (y^* - y_0)^2 \tag{2.58}$$
$$y^* = y - \tan(h)(z - z_t) \tag{2.59}$$
$$z_t = z_0 + dzdx * (x - x_r) \tag{2.60}$$

with (x, y, z) the coordinate of the cluster, (x_0, y_0, z_0) the coordinate of the center of the Riemann circle, R its radius, x_r a constant reference radial position in the middle of the TRD stack and $dzdx$ the slope of the track in the $x - z$ plane. Using the following transformations

$$t = 1/(x^2 + y^2) \tag{2.61}$$
$$u = 2 * x * t \tag{2.62}$$
$$v = 2 * \tan(h) * t \tag{2.63}$$
$$w = 2 * \tan(h) * (x - x_r) * t \tag{2.64}$$

one gets the following linear equation

$$a + b * u + c * t + d * v + e * w = 2 * (y + \tan(h) * z) * t \tag{2.65}$$

where the coefficients have the following meaning

$$a = -1/y_0 \tag{2.66}$$
$$b = x_0/y_0 \tag{2.67}$$
$$c = (R^2 - x_0^2 - y_0^2)/y_0 \tag{2.68}$$
$$d = z_0 \tag{2.69}$$
$$e = dz/dx \tag{2.70}$$

The error calculation for the free term is thus

$$\sigma = 2 * \sqrt{\sigma_y^2(tilt\ corr...) + \tan^2(h) * \sigma_z^2} * t \tag{2.71}$$

From this simple model one can compute χ^2 estimates and a rough approximation of $1/p_t$ from the curvature according to the formula:

$$C = 1/R = a/(1 + b^2 + c * a) \tag{2.72}$$

9. **Filter 2:** Calculate likelihood of the track (see `AliTRDtrackerV1::CookLikelihood()`). The following quantities are checked against the Riemann fit:

   - Position resolution in y.
   - Angular resolution in the bending plane.
   - Likelihood of the number of clusters attached to the tracklet.

10. Extrapolation of the helix fit to the other 2 chambers *non seeding* chambers:
    - Initialization of extrapolation tracklets with the fit parameters.
    - Attach clusters to extrapolated tracklets.
    - Helix fit of tracklets
11. Improve seeding tracklets quality by reassigning clusters based on the last parameters of the track (see `AliTRDtrackerV1::ImproveSeedQuality()` for details).
12. Helix fit of all 6 seeding tracklets and χ^2 calculation
13. Hyperplane fit and track quality calculation (see `AliTRDtrackerFitter::FitHyperplane()` for details.
14. Cooking labels for tracklets. Should be done only for MC.
15. Register seeds.

Calibration

Author: R. Bailhache (rbailhache@ikf.uni-frankfurt.de)

3.1 Database Entries

A local database with default parameters can be found in the AliRoot installation directory. The official database is in Alien under the directory `/alice/data/`$\langle$`year`$\rangle$`/`$\langle$`LHCPeriod`$\rangle$`/OCDB`. The calibration objects are stored in root files named according to their run validity range, their version and subversion number. For the TRD they are in the subdirectory `$AliRoot/OCDB/TRD/Calib` and correspond to a perfect TRD detector. The parameters are listed in Tab.3.1.
They are related to the calibration of:

Parameter	Description	Number of channels	Data type	Unit	Default value
ChamberGainFactor	Mean gas gain per chamber	540	Float	–	1.0
LocalGainFactor	Gas gain per pad	1181952 1181952	UShort UShort	– –	1.0 1.0
ChamberVdrift	Mean drift velocity per chamber	540 540	Float Float	cm/μs cm/μs	1.5 1.5
LocalVdrift	Drift velocity per pad	1181952 1181952	UShort UShort	– –	1.0 1.0
ChamberT0	Minimum timeoffset in the chamber	540 540	Float Float	timebin timebin	0.0 0.0
LocalT0	Timeoffset per pad	1181952	UShort	timebin	0.0
PRFWidth	Width of the PRF per pad	1181952	UShort	pad width	0.515 (layer 0) 0.502 (layer 1) 0.491 (layer 2) 0.481 (layer 3) 0.471 (layer 4) 0.463 (layer 5)
DetNoise	Scale factor	540	Float	–	0.1
PadNoise	Noise per pad	1181952	UShort	ADC counts	12
PadStatus	Status per pad	1181952	char	–	–

Table 3.1: Entries in the database

- the gas gain: `ChamberGainFactor` and `LocalGainFactor`
- the electron drift velocity: `ChamberVdrift` and `LocalVdrift`

- the timeoffset: `ChamberT0` and `LocalT0`
- the width of the Pad Response Function: `PRFWidth`
- the noise per channel: `DetNoise`, `PadNoise` and `PadStatus`.

To save disk space the values per pad are stored in UShort (2 Bytes) format in AliTRDCalROC objects, one per chamber, that are members of a general `AliTRDCalPad` object. The final constants have a numerical precision of 10^{-4}. They are computed by multiplication (gain, drift velocity and noise) or addition (timeoffset) of the detector and pad coefficients. From the pad noise level a status is determined for each pad (masked, bridgedleft, bridgedright, read by the second MCM, not connected). One example macro (`AliTRDCreate.C`) to produce a local database is given in the `$AliRoot/TRD/Macros` directory.
During the simulation of the detector response and the reconstruction of the events the parameters are used to compute the amplitude of the signal and its position inside the detector. The database has to be first choosen with the help of the `AliCDBManager`. The parameters are then called by an `AliTRDcalibDB` instance. The macro `$AliRoot/TRD/Macros/ReadCDD.C` shows how to read a local database and plot the gas gain or drift velocity as function of the detector number or pad number.

3.2 DAQ Calibration

Calibration procedures are performed online during data-taking on different systems. The principal role of the Data AcQuisition System is to build the events and archive the data to permanent storage tapes. In addition it also provides an efficient access to the data. Nevertheless the complete reconstruction of the events with tracks is not available. Two algorithms are executed on DAQ for the TRD: a pedestal algorithm and an algorithm for the drift velocity and timeoffset. They are implemented as rpm packages, that can be easily built inside AliRoot compiled with the DATE software [5]. The outputs of the algorithms are stored in root files and put on the DAQ File Exchange Server (FXS). At the end of the run they are picked up by the so called SHUTTLE and further processed in the Preprocessor to fill finally the OCDB.

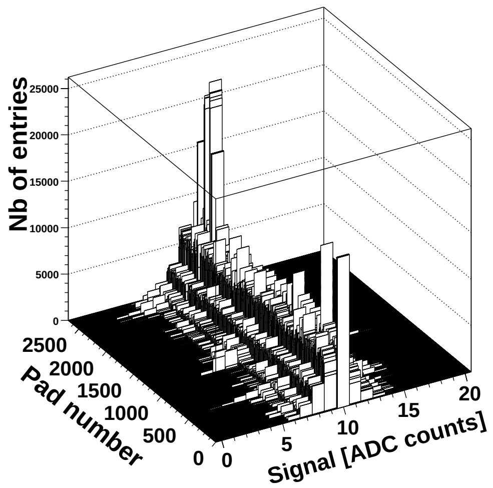


Figure 3.13: 2D histogram of the detector 0 (SM 0, S0, L0) with the ADC value distributions around the baseline (10 ADC counts) for each pad (PEDESTAL run 34510).

3.2.1 Pedestal algorithm

During a pedestal run empty events without zero suppression are taken with the TRD alone and a random trigger. They are used to determine the noise in ADC counts of each pad. The algorithm can be found in the `TRDPEDESTALda.cxx` file of the AliRoot TRD directory. It is executed on the Local Data Concentrators (LDCs), which are part of the dataflow and gives access to sub-events. The TRD has three LDCs corresponding to the following blocks of supermodules (SMs):

- 0-1-2-9-10-11
- 3-4-5-12-13-14
- 6-7-8-15-16-17

Three algorithms are therefore executed in parallel during a PEDESTAL run for a full installed TRD. After about 100 events, the data-taking stops automatically and a 2D histogram is filled for each chamber with the ADC amplitude distributions around the baseline for each pad. Such a histogram is shown in Fig.3.13 for chamber 0 (SM 0 Stack 0 Layer 0). The chambers should

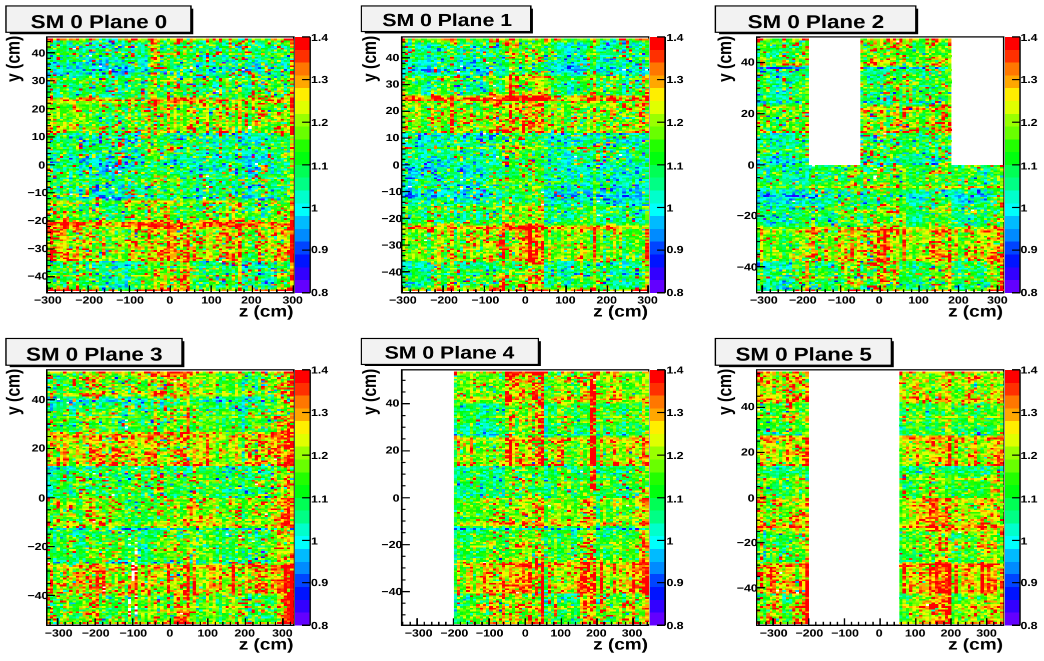


Figure 3.14: Noise in the six planes of SM 0 (PEDESTAL run 38125). The five stacks in each layer are in the *z* direction.

be so configured that the data is without zero suppression otherwise an error message appears on the DAQ online Logbook. The container class is called `AliTRDCalibPadStatus` and allows to further fit the distributions with a Gaussian to determine the baseline and noise of each pad. The function is called `AliTRDCalibPadStatus::AnalyseHisto()`. In Fig.3.14 the noise in SM 0 is plotted for the PEDESTAL run 38125. It shows stripe patterns of higher noise in the z-direction (beam direction) correlated to the static pad capacitance of the pad plane. The noise distributions has to be first corrected for the expected noise variations induced by the pad capacitance before a status can be given to each pad. This is not done on the DAQ but just before storing the parameters inside the Offline Condition Database (OCDB) in the Preprocessor.

3.2.2 Drift velocity and timeoffset algorithm

The drift velocity and timeoffset are calibrated with physics events, pp or $PbPb$ collisions. The algorithm is called `TRDVDRIFTda.cxx` and can be found in the AliRoot TRD directory. It is executed on a dedicated monitoring server, which is not part of the dataflow and gives access to full events of the TRD. The physics events are used to fill continuously during data-taking an average pulse height for each detector. They are stored in a `TProfile2D`, which is a member of a `AliTRDCalibraFillHisto` object. The `TProfile2D` is written at the end of the run in a root file put on the DAQ FXS.

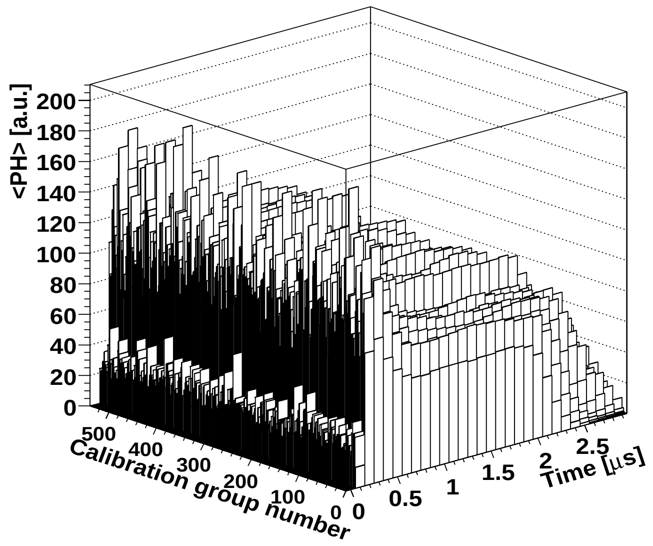


Figure 3.15: 2D histogram containing the average pulse height distributions of each calibration group (here detector), produced with decalibrated simulated pp events.

Fig.3.15 shows an output `TProfile2D` obtained from simulated decalibrated pp collisions at 14 TeV. The first peak in time corresponds to the amplification region, where the contributions of ionization electrons, which come from both sides of the anode wire plane, are overlapping. The flat plateau results from the electrons in the drift region. The tail is caused by the Time Response Function. From this average signal as function of time the drift velocity and timeoffset can be extracted by fit procedures. This last step is performed in the Preprocessor.
Since no tracking is available on DAQ, a simple tracklet finder is used. It was optimized for a low charged particle multiplicity environment. The algorithm looks for a maximum of the signal amplitudes in the chamber after integration over all timebins. The average pulse height is then filled for a spot of two pad rows (z direction) and four pad columns ($r\phi$ direction) around the maximum. Further details can be found in the function `AliTRDCalibraFillHisto::ProcessEventDAQ`.

3.3 HLT Calibration

The High Level Trigger has the big advantage to provide an online reconstruction of the events. The idea is then to run the calibration procedures in a transparent way, independent whether online or offline. The same function
`AliTRDCalibraFillHisto::UpdateHistogramsV1(AliTRDtrackV1 *t)` is used to fill the dE/dx

distributions (gain), the average pulse height (drift velocity and timeoffset) and the Pad Response Function for each detector in respectively one `TH2I` and two `TProfile2Ds`. The calibration is nevertheless done per chamber, whereas by integrating statistics it will be possible to get the gain, drift velocity and timeoffset distributions inside the chambers offline. Therefore the class `AliTRDCalibraFillHisto` contains a flag (`fIsHLT`) to avoid extra calculations not needed at the detector level.

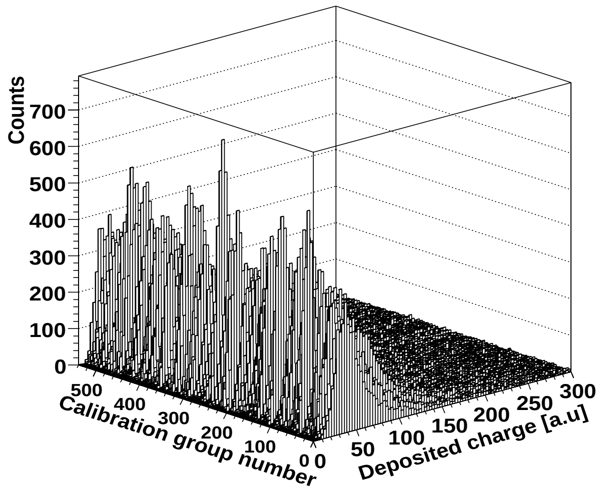


Figure 3.16: A 2D histogram containing the dE/dx distributions of each detector. These were produced with decalibrated simulated pp events.

Fig.3.16 shows one example of a `TH2I` histogram, where the dE/dx distributions of each detector is stored for pp collisions at 14 TeV. No minimal p_T cut was applied on the TRD tracks. Assuming that the charged particles are uniformy distributed over the TRD chambers, the position of the Most Probable Value of the dE/dx distribution is used to calibrate the gain.
At the beginning of each run, a local copy of the OCDB is updated on the HLT cluster: the HCDB (HLT Condition Database). The last set of calibration objects are used to reconstruct the events. The gain correction preformed during the tracking has to be taken into account when filling the dE/dx distributions. That is why the calibration algorithm has to know which database was used during the reconstruction. The TRD HLT code can be found in the `HLT/TRD` subdirectory of the AliRoot installation. The calibration is implemented as an `AliHLTTRDCalibrationComponent`, whose members are an `AliCDBManager` together with the path for the current database used, and an `AliTRDCalibraFillHisto` object. The main functions are:

- `AliHLTCalibrationComponent::InitCalibration`, where the `TH2I` and `TProfile2Ds` are created.
- `AliHLTCalibrationComponent::ProcessCalibration`, where the function `AliTRDCalibraFillHisto::UpdateHistogramsV1(AliTRDtrackV1 *t)` is called to fill the histograms.
- `AliHLTCalibrationComponent::FormOutput`, which returns a `TObjArray` with the histograms.

The histograms are shipped at the end of each run to the HLT File Exchange Server to be picked up by the SHUTTLE and further processed by the Preprocessor, exactly as the data from the calibration on DAQ.

3.4 Preprocessor

The online systems, like the Detector Control System (DCS), the DAQ and the HLT, are protected from outside by a firewall. A special framework, called the SHUTTLE, has been developped to retrieve offline data in the online systems or store relevant information from the online systems in the OCDB. The SHUTTLE has access to the DCS, DAQ and HLT FXS. At the end of each run the reference data, outputs of the calibration algorithms on DAQ and HLT, are retrieved and further processed to determine the calibration constants (gain, drift velocity, timeoffset and width of the Pad Response Function). The reference data are finally stored in the Grid reference Data Base, whereas the results of the fit procedures are stored in the OCDB. The code is contained in the `AliTRDPreprocessor` class. The Process function is executed for the run types: PEDESTAL, STANDALONE, DAQ and PHYSICS.

- The PEDESTAL run are dedicated to the calibration of the noise on DAQ. Only the output of the DAQ pedestal algorithm is retrieved at the SHUTTLE. From the noise and baseline of each pad, a pad status is determined. Disconnected pads are recognizable by a small noise. Bridged pads have the same noise and baseline. The noise and padstatus of the previous pedestal run in the OCDB are taken for half chambers, which were not On. Finally the database entries `DetNoise`, `PadNoise` and `PadStatus` are populated in the OCDB. More informations can be found in the function `AliTRDPreprocessor::ExtractPedestals`.

- The STANDALONE runs are used to check the data integrity or the correlated noise. The data are taken with the TRD alone and a random trigger. Only the DCS data are retrieved.

- The DAQ run are test runs for the DAQ people. Only the DCS data are retrieved.

- The PHYSICS run are global runs including more than one detector and different trigger clusters. They are used for the calibration of the gain, driftvelocity and timeoffset, and width of the PRF. Therefore the output of the calibration algorihms running on HLT are retrieved. If the procedure is not successful the output of the driftvelocity/timeoffset algorithm on DAQ is also retrieved. The reference data, the histograms, are fitted using an `ALiTRDCalibraFit` instance:

  – `AliTRDCalibraFit::AnalyseCH(const TH2I *ch)` determines the MPVs of the dE/dx distributions and compares them to a reference value.
  – `AliTRDCalibraFit::AnalysePH(const TProfile2D *ph)` fits the average pulse height and determines the position of the amplification region peak and the end of the drift region for each chamber. Knowing the length of the drift region one can deduce the drift velocity. The amplification peak gives also information on the timeoffset.
  – `AliTRDCalibraFit::AnalysePRFMarianFit(const TProfile2D *prf)` determines the spread of the clusters as function of azimuthal angle of the track. The minimum gives the width of the PRF.

  The results of each fit procedure are stored in a `TObjArray` of `AliTRDCalibraFit::AliTRDFitInfo` objects, one per chamber, which is a member of the `AliTRDCalibraFit` instance. The functions `AliTRDCalibratFit::CreateDetObject*` and `::CreatePadObject*` allow to create from the `TObjArray` the final calibration objects, that have to be put in the OCDB.

Tab.3.2 summarizes the tasks executed by the prepocessor for each run type. The DCS data points are measurements of the currents, voltages, temperatures and other variables of the

run type	DCS data points temperatures voltages, etc $\cdots$	DCS FXS electronic configuration	DAQ FXS calibration DA noise/(v_{dE}/t_0)	HLT FXS calibration DA g/(v_{dE}/t_0)/σ_{PRF}
DAQ	yes	yes	no	no
PEDESTAL	no	no	yes (noise)	no
STANDALONE	yes	yes	no	no
PHYSICS	yes	yes	yes (v_{dE}/t_0)	yes

Table 3.2: Tasks performed by the TRD preprocessor for every run type.

chambers as function of time. They are saved in the DCS Archive DB during the run and made available at the SHUTTLE by AMANDA.

3.5 Offline Calibration

The offline calibration of the gain, driftvelocity/timeoffset and width of the PRF is meant to improve the first calibration online. It follows the following steps:

- Fill reference data (the dE/dx distributions, the average pulse heights $\cdots$) during the reconstruction of the events offline.
- Store the reference data in root files in AliEn.
- Merge the reference data of different runs and/or calibration groups.
- Fit the reference data to extract the calibration constants and create the calibration objects.
- Store the calibration objects according to their run validity in the OCDB.

The calibration procedure is not performed per detector anymore but per pad, at least for the first step, the filling of the reference data. Depending on the available statics the reference data of different pads (calibration groups) can be merged together to determine a mean calibration coefficient over these pads.

3.5.1 AliTRDCalibraVector container

The high granularity of the calibration, with a total number of 1181952 pads, implies that the size of the reference data has to be reduced to the strict minimum needed.

reference data for	Number of calibration groups	size in MB
gain	1181952	225
driftvelocity/timeoffset	1181952	271
PRF	131328	200
All together		696

Table 3.3: Size of the `AliTRDCalibraVector` object for a given granularity.

The `TH2I` and `TProfile2D` objects are not a good option anymore. Therefore a container class, `AliTRDCalibraVector`, was developped. The `TH2I` corresponds to an array of UShort (2 Bytes) for the number of entries in each bin, the `TProfile2D` to an array of UShort for the number of entries in each bin and two arrays of Float for the sum of the weights and the sum of the squared weights in each bin. The mean value and its error are computed per hand in the functions `AliTRDCalibraVector::UpdateVector*`, where the object is filled with new data. The size of the `AliTRDCalibraVector` object is summarized in Tab.3.3.

3.5.2 Additional method to calibrate the drift velocity

In addition an other method is available for the calibration of the drift velocity. It is based on the comparison of the slope of the TRD tracklet in the azimuthal plane xy with the ϕ angle of the global track. It can be shown that the slope dy/dt of a TRD tracklet depends linearly on its global track parameters, $\tan(\phi) + (dz/dx)\tan(\beta_{tilt})$ [6]. The slope parameter is the drift velocity in the electric field direction, whereas the constant gives the tangent of the Lorentz angle. If the TRD tracklet crosses two different pads in the z direction (the beam direction), the relation is not true anymore. Therefore such tracklets are rejected in the calibration procedure. The reference data are a `TObjArray` of one `TH2F` histogram for each detector.

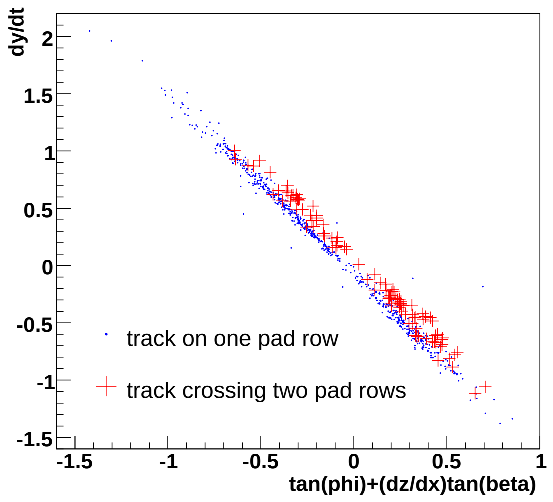


Figure 3.17: The correlation between dy/dt and $\tan(\phi) + (dz/dx)\tan(\beta_{tilt})$ for the reconstructed track in one chamber. The tracks crossing at least two pad rows are in red crosses and those crossing one pad row in blue points.

Fig.3.17 shows one example of such a histogram. They are filled in the function `AliTRDCalibraFillHisto: :UpdateHistogramsV1(AliTRDtrackV1 *t)`, like the reference data for other calibration constants, if the flag `fLinearFitterDebugOn` is true.
The histograms are stored in the container class, `AliTRDCalibraVdriftLinearFit`, for which a `Merge` and `Add` function have been implemented. In a second step, the `AliTRDCalibraVdriftLinearFit` objects can be merged together for different runs. In a third step, the `TH2F` histograms are fitted in the function `AliTRDCalibraVdriftLinearFit::FillPEArray`. The result parameters are members of the `AliTRDCalibraVdriftLinearFit` object, as well as their error coming from the fit procedures.

Finally the `AliTRDCalibraVdriftLinearFit` object is passed to an `AliTRDCalibraFit` instance through the function `AliTRDCalibraFit::AnalyseLinearFitters`, in which the Lorentz angle is computed from the fit parameters and stored together with the drift velocity in a `TObjArray`, member of the `AliTRDCalibraFit` instance. As for the other calibration constants the functions `AliTRDCalibratFit::CreateDetObject*` and `::CreatePadObject*` allows to create the final calibration objects, that have to be put in the OCDB. Since the Lorentz angle is not a OCDB entries, it is only used for debugging.

3.5.3 The calibration AliAnalysisTask

The reference data of the calibration are filled in an AliAnalysisTask during the reconstruction or after the reconstruction. Since it needs some informations only stored in the AliESDfriends, they have to be written if one wants to run the calibration. This will be the case only for TRD track above a given p_T since the size of the events is otherwise to big.

Alignment

4.1 ???

Quality Assurance (QA)

5.1 ???

High Level Trigger (HLT)

6.1 ???

References

[1] *The ALICE Offline Bible* http://aliceinfo.cern.ch/export/sites/AlicePortal/Offline/galleries/Download/OfflineDownload/OfflineBible.pdf.

[2] C. Adler, *Radiation length of the ALICE TRD*

[3] D. Emschermann, *Numbering Convention for the ALICE TRD Detector.*, http://www.physi.uni-heidelberg.de/d̃emscher/alice/numbering/more/TRD_numbering_v04.pdf.

[4] M. Castellano et al., Comp. Phys. Comm. **55**, 431 (1988), Comp. Phys. Comm. **61**, 395 (1990),

[5] K. Schossmaier et al., *The Alice Data Acquisition and Test Environment DATE V5*, CHEP06.

[6] R. Bailhache, *Calibration of the ALICE Transition Radiation Detector and a study of Z^0 and heavy quark production in pp collisions at the LHC*, PhD thesis, University of Darmstadt (Germany), 2009.