

TRD

Offline Software Writeup

Version 1.0, May 27, 2009

Contents

Introduction	2
Simulation	3
1.1 Geometry	3
1.1.1 Readout Chambers	3
1.1.2 Supermodules	4
1.1.3 Material Budget and Weight	5
1.1.4 Naming Conventions and Numbering Schemes	6
1.1.5 Pad Planes	7
1.2 Hit Generation	8
1.2.1 Energy loss	8
1.2.2 Photons from transition radiation	8
1.2.3 Track references	9
1.3 Digitization	9
1.3.1 Digitizer	9
1.3.2 Simulation parameter	11
1.3.3 Digits manager	12
1.3.4 Data containers	12
1.3.5 Zero suppression	13
1.4 Raw Data Simulation	13
1.5 Trigger Simulation	13
1.5.1 MCM simulation	14
1.5.2 TRAP configuration	15
1.5.3 Tracklet classes	15
1.5.4 GTU simulation	16
1.5.5 CTP interface	16
Reconstruction	17
2.1 Raw Data Reading	17
2.2 Cluster Finding	17
2.2.1 Cluster position reconstruction in the xy plane	17
2.3 Tracking	17
2.3.1 title	17
2.3.2 title	17
2.3.3 title	19
2.3.4 title	20
2.3.5 title	20
Calibration	21
3.1 Database Entries	21
3.2 DAQ Calibration	22
3.2.1 Pedestal algorithm	23
3.2.2 Drift velocity and timeoffset algorithm	24
3.3 HLT Calibration	24
3.4 Preprocessor	26
3.5 Offline Calibration	27

3.5.1	AliTRDCalibraVector container	27
3.5.2	Additional method to calibrate the drift velocity	28
3.5.3	The calibration AliAnalysisTask	29
Alignment		30
4.1	???	30
Quality Assurance (QA)		31
5.1	???	31
High Level Trigger (HLT)		32
6.1	???	32
References		33
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 und 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 mother volume 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

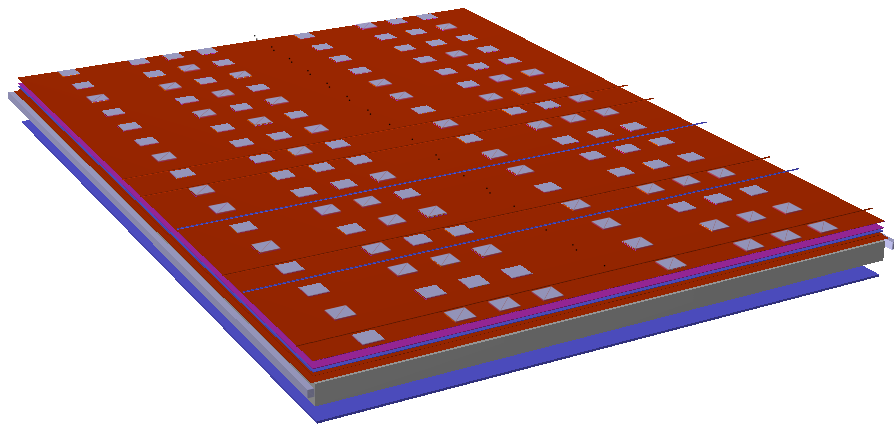


Figure 1.1: A TRD read out chamber as implemented in the AliRoot geometry. The various material layers are visible. Also, the MCMs on top of the chamber, as well as the cooling pipes are shown.

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

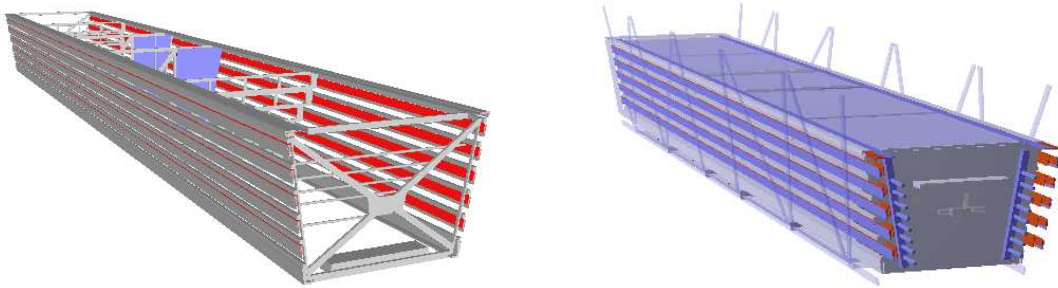


Figure 1.2: A TRD supermodule, as implemented in the AliRoot geometry. The left panel shows only the support structures of the aluminum frame, together with some service elements. The right panel shows a complete supermodule including some surrounding parts of the space frame.

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 arterias 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 accomodate 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

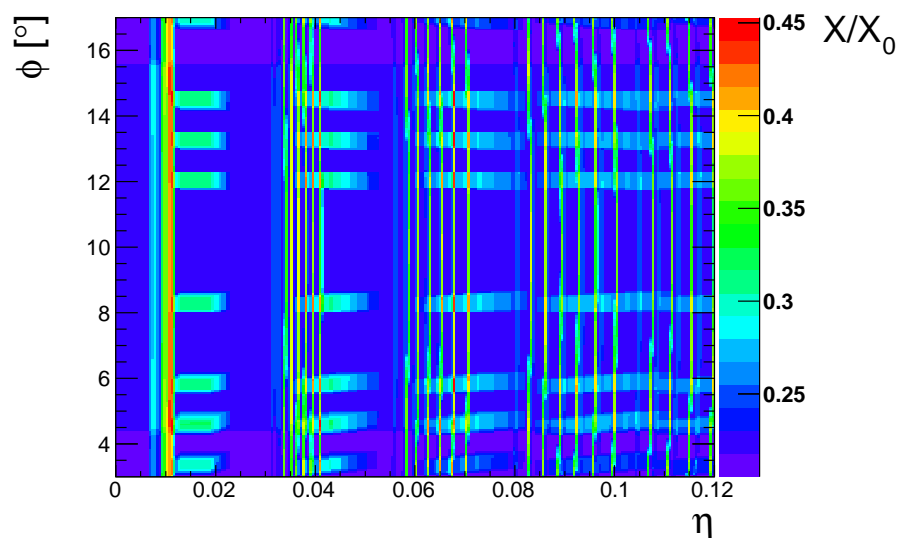


Figure 1.3: The radiation length map in units of X/X_0 in part of the active detector area of super module 0 as a function of the pseudorapidity η and the azimuth angle ϕ , calculated from the geometry in AliRoot. Visible are the positions of the MCMs and the cooling pipes as hot spots.

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 LOC1 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

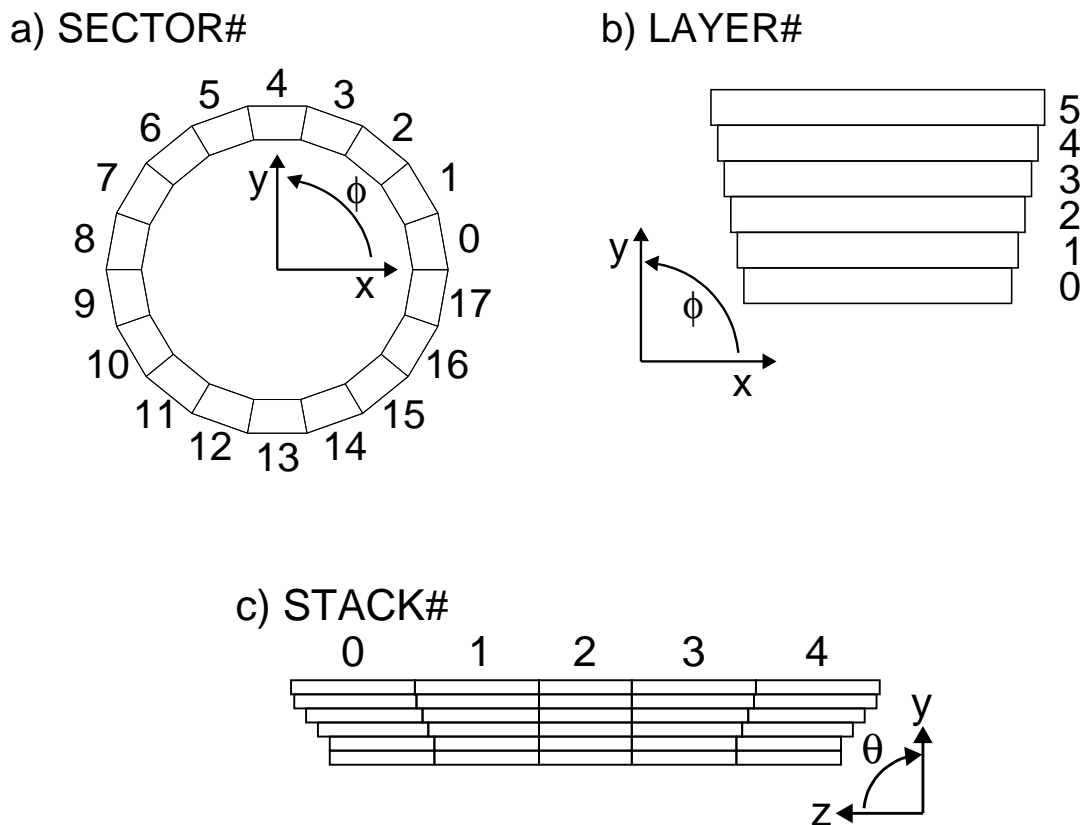


Figure 1.4: Illustration of the TRD numbering scheme for super modules, defined in the global ALICE coordinate system: a) **SECTOR** number, b) **LAYER** number, c) **STACK** number.

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

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

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

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

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:

<code>fTrack</code>	Index of MC particle in kine tree
<code>fX</code>	X-position of the hit in global coordinates
<code>fY</code>	Y-position of the hit in global coordinates
<code>fZ</code>	Z-position of the hit in global coordinates
<code>fDetector</code>	Number of the ROC (DET number)
<code>fQ</code>	Number of electrons created in the ionization step. Negative for TR hits
<code>fTime</code>	Absolute time of the hit in μ s. 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/CO₂ 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 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`):

<code>fAmp</code>	Signal amplitude
<code>fId</code>	Number of the ROC (DET number)
<code>fIndexInList</code>	Track index
<code>fRow</code>	Pad row number
<code>fColumn</code>	Pad column number
<code>fTime</code>	Time bin number

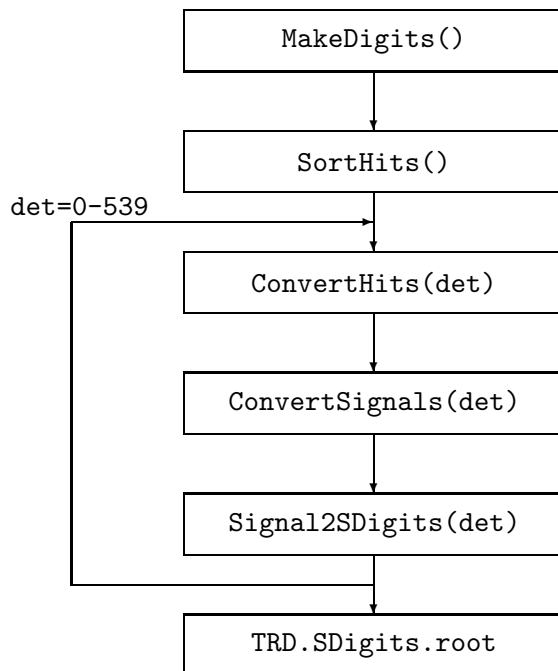
However, in practise `AliTRDdigits` is not used to store the digits information. Instead the data containers described in 1.3.4 are used for this purpose. The digitization process includes as an intermediate step between hit and digits the so-called summable digits, or `sdigits`:

HITS \implies SDIGITS \implies 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.

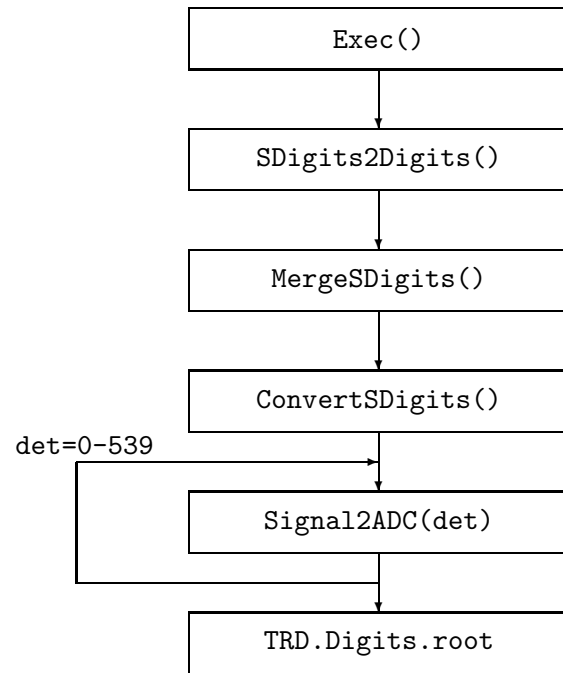
1.3.1 Digitizer

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-isochrony 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 reponse 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 neighbouring pads (`AliTRDdigitizer::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 **SDIGITS**. The correponding 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** \Rightarrow **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 discretized 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/CO ₂ , 1: Ar/CO ₂)	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 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:

$$\text{ADC-1}(t) \leq \text{ADC}(t) \geq \text{ADC+1}(t)$$

2) Cluster:

$$\text{ADC-1}(t) + \text{ADC}(t) + \text{ADC+1}(t) > \text{Threshold}$$

3) Absolute Large Peak:

$$\text{ADC}(t) > \text{Threshold}$$

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)

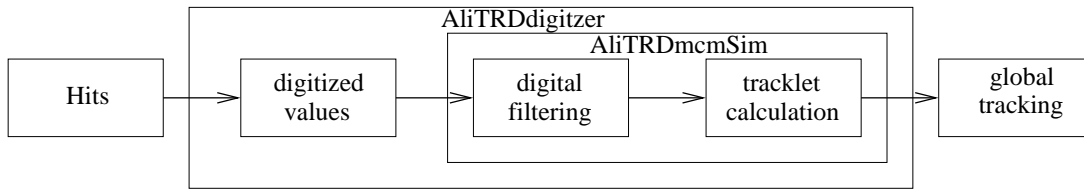


Figure 1.7: Overview of the trigger simulation

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 mimicks 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 in the xy plane

Author: A. Bercuci (A.Bercuci@gsi.de) The procedures described in this section are implemented in the functions `AliTRDcluster::GetXloc()`, `AliTRDcluster::GetYloc()`, `AliTRDcluster::GetSX()` and `AliTRDcluster::GetSY()`.

2.3 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.

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 2.3.4) and stand alone tracking (section 2.3.5).

2.3.1 Tracklet building - Attaching clusters to tracklet¹

2.3.2 Tracklet fitting²

Fit in the xy plane

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

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

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.

$$\sigma_{x|y} = tg(\phi)\sigma_x \quad (2.1)$$

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 :

$$\sigma_z = tg(\theta)\Delta x_{x_0}/12 \quad (2.2)$$

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()`):

$$\sigma_y = \sigma_{y_0}^2 + 2x cov(y_0, dy/dx) + \sigma_{dy/dx}^2 \quad (2.3)$$

and thus the radial position is:

$$x = -cov(y_0, dy/dx)/\sigma_{dy/dx}^2 \quad (2.4)$$

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:

$$\sigma_y = \sigma_{y_0}^2 - 2cov^2(y_0, dy/dx)/\sigma_{dy/dx}^2 + \sigma_{dy/dx}^2 \quad (2.5)$$

$$\sigma_z = Pad_{length}/12 \quad (2.6)$$

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.

$$\sigma_z = tg(\theta)\Delta x_{x_0}/12 \quad (2.7)$$

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 errors³

Computes covariance in the y - z plane at radial point x (in tracking coordinates) and returns the results as:

`cov[0] = Var(y)`
`cov[1] = Cov(yz)`
`cov[2] = Var(z)`

Details

For the linear transformation

$$Y = T_x X^T \quad (2.8)$$

The error propagation has the general form

$$C_Y = T_x C_X T_x^T \quad (2.9)$$

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

$$T_x = (1 \ x) \quad (2.10)$$

$$X = (y_0 \ dy/dx) \quad (2.11)$$

$$C_X = \begin{pmatrix} Var(y_0) & Cov(y_0, dy/dx) \\ Cov(y_0, dy/dx) & Var(dy/dx) \end{pmatrix} \quad (2.12)$$

and secondly to take into account the tilt angle

$$T_\alpha = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \quad (2.13)$$

$$X = (y \ z) \quad (2.14)$$

$$C_X = \begin{pmatrix} Var(y) & 0 \\ 0 & Var(z) \end{pmatrix} \quad (2.15)$$

using simple trigonometrics one can write for this last case

$$C_Y = \frac{1}{1 + tg^2\alpha} \begin{pmatrix} \sigma_y^2 + tg^2(\alpha)\sigma_z^2 & tg(\alpha)(\sigma_z^2 - \sigma_y^2) \\ tg(\alpha)(\sigma_z^2 - \sigma_y^2) & \sigma_z^2 + tg^2(\alpha)\sigma_y^2 \end{pmatrix} \quad (2.16)$$

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

$$C_Y = \begin{pmatrix} \sigma_y^2 & (\sigma_z^2 - \sigma_y^2)tg(\alpha) \\ ((\sigma_z^2 - \sigma_y^2)tg(\alpha) & \sigma_z^2 \end{pmatrix} \quad (2.17)$$

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.

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

2.3.4 Track propagation in barrel tracking⁴

TRD Tracklet initialization and Kalman fit⁵

2.3.5 Track finding stack wise⁶

TRD track finding⁷

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

⁵The procedures described in this section are implemented in the function `AliTRDtrackerV1::FollowBackProlongation()`.

⁶The procedures described in this section are implemented in the function `AliTRDtrackerV1::Clusters2TracksStack()`.

⁷The procedures described in this section are implemented in the function `AliTRDtrackerV1::MakeSeeds()`.

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/<year>/<LHCPeriod>/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	UShort	–	1.0
		1181952	UShort	–	1.0
ChamberVdrift	Mean drift velocity per chamber	540	Float	cm/ μ s	1.5
		540	Float	cm/ μ s	1.5
LocalVdrift	Drift velocity per pad	1181952	UShort	–	1.0
		1181952	UShort	–	1.0
ChamberT0	Minimum timeoffset in the chamber	540	Float	timebin	0.0
		540	Float	timebin	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.

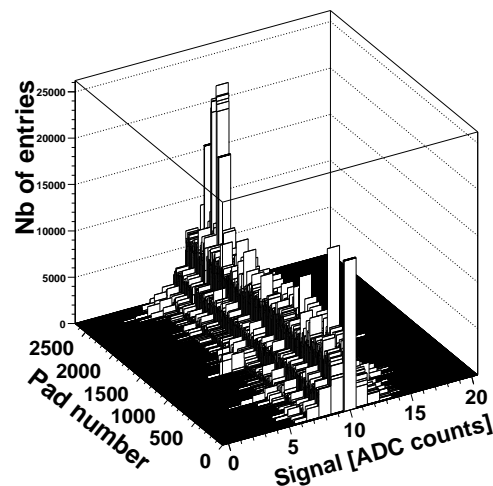


Figure 3.8: 2D histogram of the detector 0 (SM 0, S0, L0) with the ADC value distributions around the baseline (10 ADC counts) for each pad (PEDESTAL run 34510).

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.3.8 for chamber 0 (SM 0 Stack 0 Layer 0). The chambers should

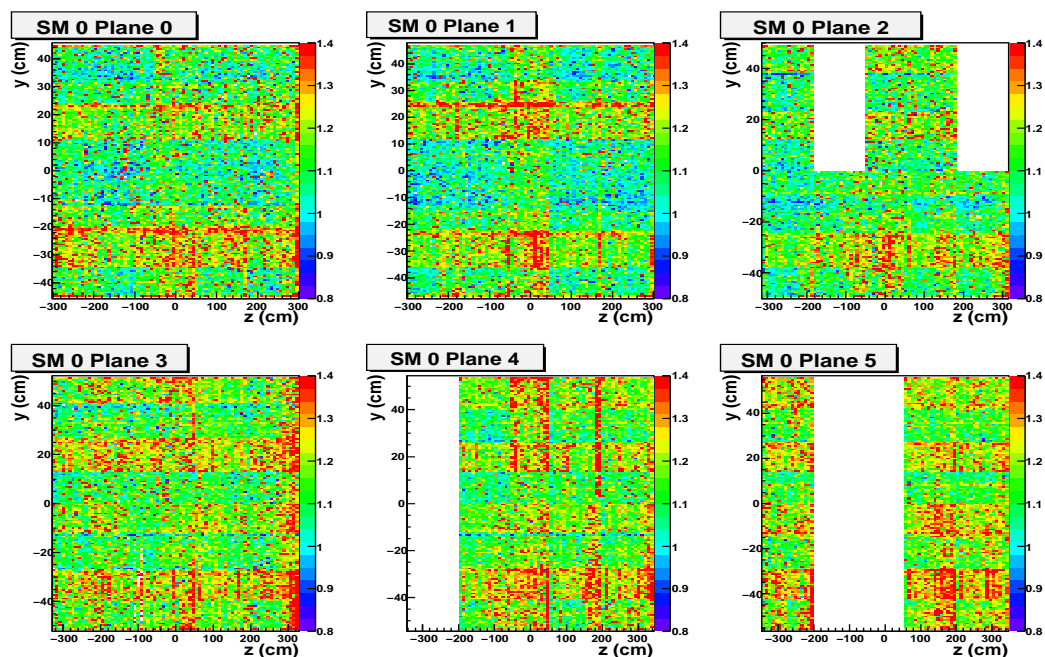


Figure 3.9: Noise in the six planes of SM 0 (PEDESTAL run 38125). The five stacks in each layer are in the z direction.

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.3.9 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.

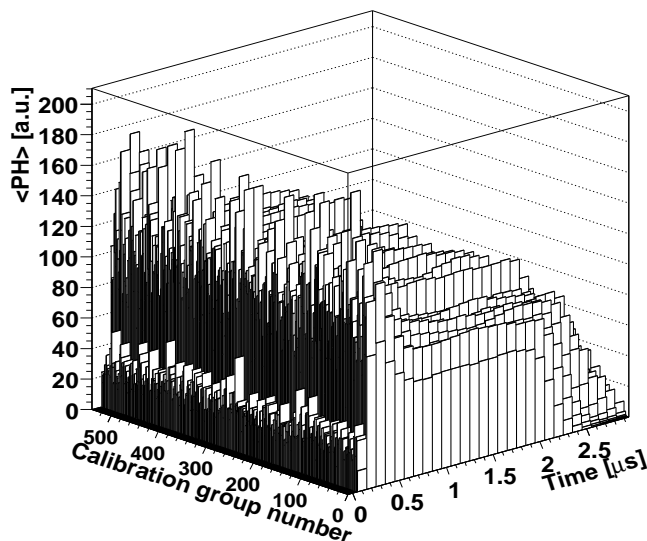


Figure 3.10: 2D histogram containing the average pulse height distributions of each calibration group (here detector), produced with decalibrated simulated pp events.

Fig.3.10 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.

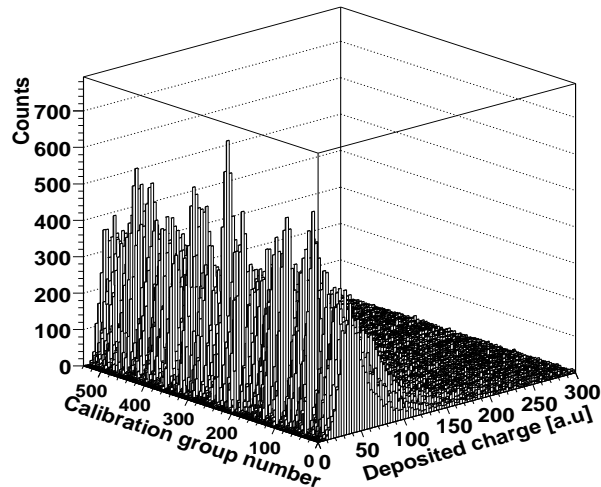


Figure 3.11: A 2D histogram containing the dE/dx distributions of each detector. These were produced with decalibrated simulated pp events.

Fig.3.11 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 uniformly 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 performed 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 AliHLTRDCalibrationComponent, whose members are an AliCDBManager together with the path for the current database used, and an AliTRDCalibraFillHisto object. The main functions are:

- AliHLTRDCalibrationComponent::InitCalibration, where the TH2I and TProfile2Ds are created.
- AliHLTRDCalibrationComponent::ProcessCalibration, where the function AliTRDCalibraFillHisto::UpdateHistogramsV1(AliTRDtrackV1 *t) is called to fill the histograms.
- AliHLTRDCalibrationComponent::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 developed 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 algorithms 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 `AliTRDCalibraFit::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 preprocessor 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 . . .	DCS FXS electronic configuration	DAQ FXS calibration DA noise/ (v_{dE}/t_0)	HLT FXS calibration DA $g/(v_{dE}/t_0)/\sigma_{PRF}$
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 . . .) 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 developed. 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.

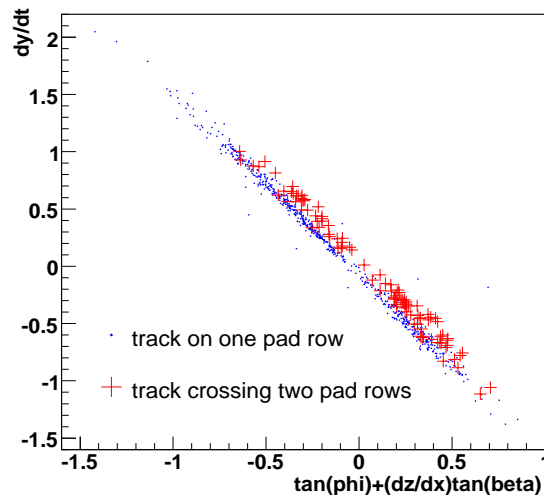


Figure 3.12: The correlation between dy/dt and $\tan(\phi) + (dz/dx) \tan(\beta_{tilt})$ for the reconstructed track in one chamber. The tracks crossing at least two pad rows are in red crosses and those crossing one pad row in blue points.

Fig.3.12 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 `AliTRDCalibraFit::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 too 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/~demscher/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. Schossmailer 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.