1 Alice TPC calibration using tracks

All calibration classes (components) using the tracks and ESDs derives from the AliTPCcalibBase. Following classes implemented:

- **AliTPCcalibCalib.** Re-apply calibration on the cluster level, refit the track. To be (optionally) used as prefilter before calibration.
- **AliTPCcalibTracks.** Cluster shape and error parameterization.
- **AliTPCcalibTracksGain.** Gain calibration
- **AliTPCcalibAlign.** Internal alignment of the TPC chambers.
- **AliTPCcalibLaser.** Calibration using laser tracks. Drift vleloctity and unlinearities. ExB effect.
- **AliTPCcalibTime.** Time dependent calibration of drift velocity and the gain.
- **AliTPCcalibCosmic.** Currently only for performance study.
- **AliTPCcalibUnlinearity.** Unlinearity calibration.

Additional class **AliTPCAnalysisTaskcalib** was implemented as a steering class to invoke the calibration tasks inside the AliROOT Analysis framework.

The base functionality to be optionally implemented by the derived classes.

- **Process(AliESDEvent *event)** To implement if full even necessary: AliTPCcalibLaser, AliTPCcalibCosmic, AliTPCcalibTime.
- **Process(AliTPCseed *track)** All clusters needed: AliTPCcalibTracks, AliTPCcalibTracksGain
- **Process(AliESDtrack *track)**
-
- **Merge(TCollection)** All components are able to merge results. (Histograms, AliTPCCalPads, Fitters)
- **Terminate()** Function to be called before saving results. (Usually update of fitters)
- **Bool AcceptTrigger()** Don't process event with non appropriate trigger mask

- **Analyze()** Analyze raw data (histograms, calPads fitters). Extract parameterizations fits.

  Using the components inside of the Analysis framework, optionaly streamers (Trees) can be used. By default the usage of the streamers is disabled. They have to be explicitly enabled in the task configuration macro.

  The usage of stremers is important during the development phase (see example 1):

  - 1. Possible to debug derivation of the intermediate results. Important when the derivation is not straightforward.
  - 2. Possible to tune selection criteria
  - 3. If the variable of interest is function of many other variables

---

```
{
//
// 0. Enabling stremer - to be done in the ConfigurationMacro: e.g $ALICE_RO
//
  AliTPCcalibTime *calibTime = new AliTPCcalibTime("calibTime","calibTime", s
  calibTime->SetDebugLevel(20);
  calibTime->SetStreamLevel(10);
}

{
//
// Example filling of debug streamer  in the AliTPCcalibTime.cxx component
//
if (fStreamLevel>0){
    TTreeSRedirector *cstream = GetDebugStreamer();
    if (cstream){
      (*cstream)<<"dz"<<
        "run="<<fRun<<              //  run number
        "event="<<fEvent<<          //  event number
        "time="<<fTime<<            //  time stamp of event
        "trigger="<<fTrigger<<      //  trigger
        "mag="<<fMagF<<             //  magnetic field
        // dump the info which we fill in histogram
        "dz="<<fDz<<
        "psurface="<<psurface<<
        "ppit="<<ppit<<
        "tsidea="<<tsidea<<
        "tsidec="<<tsidec<<
        "viroc="<<viroc<<
        "voroc="<<voroc<<
        "\n";
        }
     }
    }
    //
    // Using debug stremers - fitting visualization
    //
    {
      TFile f("calibTime.root");
      TTree * tree = (TTree*)f.Get("dz");
      tree->Draw("dz:time");  // visualize the delta z as function of time
      tree->Draw("dz:time","abs(dz)<20");    // apply delta z cut
      tree->Draw("dz:time","abs(dz)<20&&trigger==16");  // apply trigger type se
      tree->Draw("dz:time","abs(dz)<20&&trigger==16&&viroc==1400");
    //select only data with nominal gain
    }
```

Listing 1: StreamerExample.C

1.1 Usage of the calibration components in Analysis framework

The calibration components were designed in the way that they can be used stand-alone or inside of the AliRoot Analysis framework. Possibility to use components inside of the Analysis framework is important mainly during the development phase.

(see example macros 2). New calibration entries created by calibration can be stored at differnt locations. In order to verify calibration the OCDB has to be setup. See example macro 3. In order to reaply the new calibration from specific storage the calibration component for track refitting has to be activated, the component **AliTPCcalibCalib** has to added to the task in calibration configuration macro.

```
//
// Config calibration
// example macro in $ALICE_ROOT/TPC/macros/CalibrateTPC.C
// To be modified according studies

AliAnalysisTask * SetupCalibTask(){

  //1.  Make calibration task
  AliTPCAnalysisTaskcalib *task1=new AliTPCAnalysisTaskcalib("TPC␣calibration␣tas
  //
  //2.  Make calibration components
  //
  AliTPCcalibTracks *calibTracks =
    new AliTPCcalibTracks("calibTracks", "Resolution␣calibration␣object␣for␣track
  AliTPCcalibTracksGain *calibTracksGain =
    new AliTPCcalibTracksGain("calibTracksGain","Gain␣calibration␣using␣tracks",c
  AliTPCcalibAlign *calibAlign = new AliTPCcalibAlign("alignTPC","Alignment␣of␣th
  AliTPCcalibAlign *calibAlignAll = new AliTPCcalibAlign("alignTPCAll","Alignment
  AliTPCcalibLaser *calibLaser = new AliTPCcalibLaser("laserTPC","laserTPC");
  AliTPCcalibCosmic *calibCosmic = new AliTPCcalibCosmic("cosmicTPC","cosmicTPC")
  AliTPCcalibCalib *calibCalib = new AliTPCcalibCalib("calibTPC","calibTPC");
  //
  //3. Setup calibration components - Example
  //
  calibTime->SetDebugLevel(20);
  calibTime->SetStreamLevel(10);
  calibUnlinearity->SetDebugLevel(20);
  calibUnlinearity->SetStreamLevel(10);
  calibUnlinearityAll->SetDebugLevel(20);
  calibUnlinearityAll->SetStreamLevel(10);
  calibCalib->SetTriggerMask(-1,-1,kFALSE);
  calibTracks->SetTriggerMask(-1,16,kTRUE);
  //
  //4. Plugin the calibration components to tasks
  //
  task1->AddJob(calibCalib);
  task1->AddJob(calibAlign);
  task1->AddJob(calibAlignAll);
  task1->AddJob(calibLaser);
  task1->AddJob(calibCosmic);
  task1->AddJob(calibTime);
  task1->AddJob(calibTracksGain);
  task1->AddJob(calibTracks);
  task1->AddJob(calibUnlinearity);
  task1->AddJob(calibUnlinearityAll);
  //
  //
  return task1;
}
```

Listing 2: CalibrateTPC.C

```
//
// Config OCDB for calibration
// example macro in $ALICE_ROOT/TPC/macros/ConfigOCDB.C
//


void ConfigOCDB(Float_t bfield, Int_t runNumber){
  //
  // import geometry
  //
  printf("SETUP␣OCBD␣for␣PROOF\n");
  TGeoManager::Import("/u/miranov/proof/geometry.root");
  AliGeomManager::LoadGeometry("/u/miranov/proof/geometry.root");
  //
  // Setup magnetic field
  //
  AliMagF* field = new AliMagFMaps("Maps","Maps", 2, 1., 10., AliMagFMaps::k5kG);
  AliTracker::SetFieldMap(field,0);
  //
  // Setup calibration entries
  //
  AliCDBManager::Instance()->SetDefaultStorage("local://$ALICE_ROOT");
  AliCDBManager::Instance()->SetSpecificStorage("TPC/Calib/Parameters","local://$
  AliCDBManager::Instance()->SetSpecificStorage("TPC/Calib/ClusterParam","local:/
  //  AliCDBManager::Instance()->SetSpecificStorage("TPC/Calib/PadTime0","local:/
  AliCDBManager::Instance()->SetSpecificStorage("TPC/Calib/PadTime0","local:///u/
  //
  AliCDBManager::Instance()->SetRun(runNumber);
}
```

Listing 3: ConfigOCDB.C

The calibration components are self containing, they can be optionally stored in the reference OCDB. For real correction, values extracted from the components are used to fill OCDB entries.

The usage cluster error and shape parameterization and the gain parameterization (position- due electron attachementand diffusion, and the angular) is implemented in the **AliTPCclusterParam** class.

AliTPCcalibTracks and AliTPCcalibTracksGain component will provide the functionality to update current AliTPCclusterParam. Currently it is done in the macros. Calibration sequence:

- Reconstruction with current AliTPCclusterParam
- Calibration, filling the fitters and histograms
- Analyze fits and histogram. Visual inspection.
- Update AliTPCclusterParam

1.2 Cluster error and shape parameterization - AliTPCcalibTracks

The main purpose, get the cluster error and shape parameterization as function of the cluster position, inclination angle and amplitude.

The mean width of the cluster distribution is given by:

$$\sigma_{\mathrm{t}}^2 = \sigma_{\mathrm{preamp}}^2 + D_{\mathrm{L}}^2 L_{\mathrm{drift}} + \frac{\tan^2\alpha\ L_{\mathrm{pad}}^2}{12}, \tag{1}$$

$$\sigma_{\mathrm{p}}^2 = \sigma_{\mathrm{PRF}}^2 + D_{\mathrm{T}}^2 L_{\mathrm{drift}} + \frac{\tan^2\beta\ L_{\mathrm{pad}}^2}{12}, \tag{2}$$

where σ_{preamp} and σ_{PRF} are the r.m.s. of the time response function and pad response function, respectively.

The measured width is binned in the L and in the inclination angle $\tan_{\mathrm{PRF}}$. Following parameters are fitted:

$$\sigma_y^2 = k_0 + k_1 L_{\mathrm{drift}} + k_2 \tan^2\alpha \tag{3}$$

The agreement between the data and the theoretical formula is on the level of 5% over the full drift length, and inclination angle tan from 0 up to 1.4. Such precision is sufficient to signed overlapped tracks. In order to estimate the drift length, more precise formula should be used. The second order Taylor approximation can be used:

$$\sigma_y^2 = k_0 + k_1 L_{\mathrm{drift}} + k_2 \tan^2\alpha + k_3 L_{\mathrm{drift}}^2 + k_4 \tan^4\alpha + k_5 L_{\mathrm{drift}} \tan^2\alpha \tag{4}$$

The measured width of the cluster distribution depends also on the deposited charge, this dependence we consider as second order correction. (To be implemented in the AliTPCClusterparam)

$$\sigma_y^2 = k_0 + k_1 L_{\mathrm{drift}} + k_2 \tan^2\alpha + k_3 L_{\mathrm{drift}}^2 + k_4 \tan^4\alpha + k_5 L_{\mathrm{drift}} \tan^2\alpha + k_6/Q_{max} + k_7 L_{\mathrm{drift}}/Q_{max} + k_8 \tan^2\alpha/Q \tag{5}$$

Pictures to be added: First approximation, second order approximation, comparison of MC and real data. The track mean shape, real track, overlapped events.

1.3 Laser calibration, drift velocity, ExB - AliTPCcalibLaser

The resolution of the v drift velocity determination using laser tracks is on the level of 10^-4 for one event 250 microns for full drift length. This number critically depends on how many laser tracks are we using. If the laser intensity is low, we have smaller number of tracks and the resolution is worse. This number is already on the edge of our knowledge of the temperature condition (map) in the TPC. Therefore one event is sufficient to determine the drift velocity. Some rough cut on the chi2 of fit should be applied.

In AliTPCcalibLaser for drift velocity we fit 3 parameters: $P_0 = t_0$ offset $P_1 = v_d$ correction $P_2 = v_d$ global y correction event by event. I prefer to store the graph of the fitted parameters. P_0, $P1_1$ and P_2 as function of time stamps.