Table of Contents

Instructions on Preparing PIDCalib Samples

This page provides information on how to create PIDCalib samples. It is for experts only.

Latest PIDCalib setup instructions

Please follow the instructions on PIDCalib Package webpage. Besides the above, you also need to

```
getpack PIDCalib/CalibDataSel head (do inside a DaVinci version, same version as makePIDCalibNtup
```

Package: Produce tuples from DST

The files you need now boil down to: Src/TupleToolPIDCalib.cpp TupleToolPIDCalib.h EvtTupleToolPIDCalib.cpp EvtTupleToolPIDCalib.h

These are places to put Tuple variables.

And dev/makePIDCalibNtuples.ganga.py makePIDCalibNtuples_Run2.py makePIDCalibNtuples.py

makePIDCalibNtuples.py is for Run 1 where the input is various stripping lines

makePIDCalibNtuples_Run2.py is for Run2 where the input is various trigger lines and this matching needs to be done too.

makePIDCalibNtuples.ganga.py simply runs the jobs.

In dev/makePIDCalibNtuples.ganga.py, add

```
S5TeVdn = PIDCalibJob (
                    year           = "2015"
                  , stripVersion   = "5TeV"
                  , magPol         = "MagDown"
                  , maxFiles       = -1
                  , filesPerJob    = 1
                  , simulation     = False
                  , EvtMax         = -1
                  , bkkQuery       ="LHCb/Collision15/Beam2510GeV-VeloClosed-MagDown/Real Data/Re
                  , bkkFlag        = "OK"
                  , stream         = "Turbo"
                  , backend        = Dirac()
  )
```

Then execute the file inside ganga:

```
In [5]:execfile('makePIDCalibNtuples.ganga.py')
Preconfigured jobs you can just submit:
. PIDCalib.up11.submit()
. PIDCalib.validation2015.submit()
. PIDCalib.up12.submit()
. PIDCalib.down11.submit()
. PIDCalib.S23r1Up.submit()
. PIDCalib.down12.submit()
. PIDCalib.S5TeVdn.submit()
. PIDCalib.test.submit()
. PIDCalib.S23r1Dn.submit()
---------------------

In [6]:PIDCalib.S5TeVdn.submit()
```

After all jobs finished, you may need to download them to local directory.

```
In [6]:for js in j.subjobs:
   $ ...: if js.status == 'completed':
   $ ...: js.backend.getOutputData()
```

: Produce tuples for each particles

The RICH performance changes as a function of time (depends on conditions and alignment changes). A RooDataSet can only hold so many events and variables before it becomes too large and won t save correctly. Both of these facts leads us to have a) more than one file per decay channel and b) the numerical index of each file ascends with run number. This is useful so that if someone wants to run over a specific run period they can just select the few relevant files.

Hence this means that the workflow goes like this:

Ntuples finish making --> Run ranges are defined -- > Data split into those specific run ranges -- > any additional selection applied -- >mass fit performed in each run range for each charge -- > data is sWeighted --> spectator variables are added to the data set --> both charge datasets are merged . The same set of steps is repeated for each decay channel. For the protons, since there is more than one momentum range the fit is done separately in each range and then merged. For D*->D(Kpi)pi the data the definition of the run range and the data splitting is a common step for both K and pi, however the mass fits and are done twice (even though it is the same fit).

First, get the package:

```
getpack PIDCalib/CalibDataScripts head
```

Inside, There are 3 src directories Src for S20/S0r1 data Src_S21 for S21 Src_Run2 for S22/23

The reason for different directories is due to changes in the ntuple format/naming conventions and changes in stripping cuts, which changed the selection cuts subsequently applied. Also the variables stored in the calibration datasets has also changed as a function of time. E.g for Run 2 we save online and offline variables.

In cmt/requirements The first step is to choose the correct src directory to compile. This is just done by changing src directory to your corresponding one (except that of SetSpectatorVars.cpp) Then the usual

```
cmt br cmt make
```

Then go to jobs/Stripping23 and modify configureGangaJobs.sh

Before submitting jobs to PBS, you need to do the following to make it recognize you: add the following lines in ~/.gangarc

```
preexecute =
  import os
  env = os.environ
  jobid = env["PBS_JOBID"]
  tmpdir = None
  if "TMPDIR" in env: tmpdir = env["TMPDIR"].rstrip("/")
  else: tmpdir = "/scratch/{0}".format(jobid)
  os.chdir(tmpdir)
  os.environ["PATH"]+=":{0}".format(tmpdir)
postexecute =
  import os
  env = os.environ
  jobid = env["PBS_JOBID"]
  tmpdir = None
```

```
if "TMPDIR" in env: tmpdir = env["TMPDIR"].rstrip("/")
else: tmpdir = "/scratch/{0}".format(jobid)
os.chdir(tmpdir)
```

make sure you have the above lines everytime you run jobs.

and then go to GetRunRanges /

Here you will see a set of scripts, one for each polarity, one for each particle species. You shouldn t need to change anything it will look inside your .gangarc file to find your gangadir location etc etc. the output of these jobs gets sent to the jobs/Stripping23/ChopTrees directory as a .pkl file. This file contains the run number ranges that the script which this script calls defines. The file that is actually run by the ganga job is $CALIBDATASCRIPTSROOT/scripts/sh/GetRunRanges.sh which in turn calls $CALIBDATASCRIPTSROOT/scripts/python/getRunRanges.py for Dst and Jpsi. All this script does is look at your tuples, see how the candidates are distributed by runnumber and then split into an number of ranges such that each range contains about a million candidates but avoids the last dataset having too few.

do

```
ganga ganga_gp_getRunRanges_Dst_MagDown.py
```

Please change Stripping version accordingly.

This is just a one-subjob ganga job.

The ganga version works is v600r44, please check if there are other versions.

Don't forget to do it for all the files, like mu, MagUp etc.

after that, go to ../ChopTrees and do

```
ganga ganga_gp_chopTrees_Dst_MagDown.py
```

Please also change stripping version if needed.

This will create a list of jobs on batch and could be viewed by qstat

All this script does is call $CALIBDATASCRIPTSROOT/scripts/sh/ChopTrees.sh which in turn calls $CALIBDATASCRIPTSROOT/scripts/python/ChopTrees.py. All this does is look at the .pkl file in ChopTrees from the previous step. It then goes into your gangadir/jobdir. It loops over all the subjobs. In each subjob it creates a different file for each run range. So for example before you do this the only root file in directory would be PIDCalib.root. Once you ve finished this stage it will look more like this: PID_0_dst_k_and_pi.root etc.

You ll notice that most of the files are empty since that ganga sub jobs didn t contain any runs that fall into run range x etc. That s not a problem, but this is why you need to run the job at oxford since there is a lot of diskspace.

Fit, sWeight and final tuples

In CalibDataScripts /jobs/Stripping5TeV/Dst, Lam0, Jpsi, so run corresponding codes to get calibration samples.

The directory PhysFit /RooPhysFitter should be obtained to benefit from the new functions inside.

The produced file is in the location setting in configureGangaJobs.sh and it looks like: CalibData _2015/MagDown/K, pi, Mum p etc.

User scripts

The corresponding PID production should be added into user scripts so that they can be recognized.

1. Change python/PIDPerfScripts/Definitions.py (probably also in the install area)

2. Copy pklfiles into PIDCalib/PIDPerfScripts/pklfiles

The location of accessed files can be changed in PIDCalib/PIDPerfScripts/cmt (remember to change back)

Upload files

A simple code to change pi to Pi:

```
void ChangeName (){
for(int i=0;i&lt;36;i++){
    std::cout&lt;&lt;"mv DSt_pi_MagDown_Strip21r1_"&lt;&lt;i&lt;&lt;".root DSt_Pi_MagDown_Strip21
  }
}
```

The file used is $CALIBDATASCRIPTSROOT/scripts/python/uploadData.py

First copy these files in /data/lhcb/users/CalibData/ (oxford cluster).

After that, you do:

```
SetupProject LHCbDirac
lhcb-proxy-init -g lhcb_calib
python uploadData.py 15a 5TeV
```

For lhcb-proxy-init -g lhcb_calib, you need to ask Joel for lhcb_calib permit.

Test plots

Plots to compare with can be found in RICH performance paper: http://arxiv.org/pdf/1211.6759v2.pdf

To test the produced files: scripts in

/home/qian/cmtuser/Urania_v4r0/PIDCalib/PIDPerfScripts/scripts/python/Plots

and

/home/qian/cmtuser/Urania_v4r0/PIDCalib/PIDPerfScripts/scripts/python/PubPlots

are useful.

For example:

First, you could run Plots/PlotCalibEfficiency.py

Then PubPlots /NicePlots/MakeHistNice.C

You have to edit MakeHistNice to point to the correct root files and histograms that were made in step one

The MakeHistNice essentially calls one function (PlotFourTH1FwErrors)

and then adds a bunch of formatting. you just have to prepare which 4 histograms you want drawn and what the labels should read etc

if you do python PlotCalibrationEfficiency.py it should be obvious what to type to make the raw curves

Another example: scripts/python/PubPlots/MakeIDvsMisIDCurveRunRange.py

python MakeIDvsMisIDCurveRunRange.py 26 MagUp K Pi DLLK> -15 -13 -11 -9 -7 -5 -3 -1 1 3 5 7 9 11 13 15

(When running it, you may probably encounter memory leak problem, my solution is

Urania_v5r0/PIDCalib/PIDPerfTools/PIDPerfTools/PerfCalculator.h: virtual ~PerfCalculator( ) {delete m_Data; }

You may also change other parts correspondingly (remove Dataset.Delete() etc).

I do not commit it to svn to avoid problems in other parts, and it is up to you to see how to act.)

then in scripts/python/PubPlots/NicePlots/MakeGraphNice.C

Commit and release

The last step is to commit to LHCb software and make a release:

In practice you only need to tag PIDPerfScripts.

First make sure everything is committed (changes in Definitions.py and that the new pkl files are committed).

Then in cmt/requriments change the locations back to the grid locations and change the version number to the new one.

in doc/release.notes make the version notes (open the file and you will see what i mean) and write in any changes you did.

also in CMakeLists.txt, change the version to your tagged version

then from PIDPerfScripts

svn commit -m Making Version 10rX cmt/requirements doc/release.notes then

svn commit -m v10rX (in PIDCalib/PIDPerfScripts directory)

tag_package PIDCalib/PIDPerfScripts (in Urania directory)

then check that it has indeed made so something like get pack PIDCalib/PIDPerfScripts list-versions and you should see your new version available.

-- WenbinQian - 2016-03-17

This topic: LHCb > PIDSamplePrepare
Topic revision: r23 - 2016-11-08 - DonalHill