A tutorial for NA62Analysis: creating the VertexCDA and Pi0Reconstruction analyzers.

Nicolas Lurkin

December 12, 2013

This document will describe the process of creating the VertexCDA and Pi0Reconstruction analyzers within the NA62Analysis framework. It is intended to guide future analyzer authors by describing the complete procedure. Assumption is done that the environment is already configured, NA62Analysis is installed and the User directory has been created. The installation and configuration procedure is described at `http://sergiant.web.cern.ch/sergiant/NA62FW/html/analysis.html`. Before starting, the framework environment should be set by sourcing the env.(c)sh file in the **scripts/** folder of the user directory. Every shell command given in this document is to be executed from the top of the user directory.

1 Pi0Reconstruction

This analyzer is implementing the reconstruction of a π^0 candidate from two photon candidates in the Liquid Krypton calorimeter. The eventual reconstructed candidate is then made available to further analyzer as a **KinePart** object.

xxxDescription of the algorithmxxx

1.1 Pi0Reconstruction implementation

The first step is to create the skeleton of the new analyzer. This is automatically done with the framework python script:

```
NA62AnalysisBuilder.py new Pi0Reconstruction
```

The source code of the newly created analyzer can be found in **Examples/include/Pi0Reconstruction.hh** and **Examples/src/Pi0Reconstruction.cc**. Every standard method of the analyzer and each section of code will be described thereafter.

1.1.1 Constructor

As this analyzer only needs information from the LKr, this is the only requested TTree. The analyzer will run some acceptance checks as well and need access to the global instance of the **DetectorAcceptance** object:

```
RequestTree("LKr", new TRecoLKrEvent);
fDetectorAcceptanceInstance = GetDetectorAcceptanceInstance();
```

Without forgetting to include the header for the LKr reconstructed events

```
#include "TRecoLKrEvent.hh"
```

1.1.2 InitHist

In this method all the histograms that will be needed during the processing are created and registered to the framework:

- Histograms for the photon candidates reconstructed energy

```
BookHisto("g1Energy", new TH1I("G1Energy", "Energy of g1", 100, 0, 75000));
BookHisto("g2Energy", new TH1I("G2Energy", "Energy of g2", 100, 0, 75000));
```

- 2D histograms to compare reconstructed photon candidates energy with the true Monte Carlo energy

```
BookHisto("g1Reco", new TH2I("g1Reco", "g1 Reco vs. Real", 100, 0, 75000, 100,
    0, 75000));
BookHisto("g2Reco", new TH2I("g2Reco", "g2 Reco vs. Real", 100, 0, 75000, 100,
    0, 75000));
```

- 2D histograms to compare reconstructed photon candidates momentum with the true Monte Carlo momentum

```
BookHisto("g1px", new TH2I("g1px", "g1 px Reco vs. Real", 200, 0, 2000, 200,
    0, 2000));
BookHisto("g2px", new TH2I("g2px", "g2 px Reco vs. Real", 200, 0, 2000, 200,
    0, 2000));
BookHisto("g1py", new TH2I("g1py", "g1 py Reco vs. Real", 200, 0, 2000, 200,
    0, 2000));
BookHisto("g2py", new TH2I("g2py", "g2 py Reco vs. Real", 200, 0, 2000, 200,
    0, 2000));
BookHisto("g1pz", new TH2I("g1pz", "g1 pz Reco vs. Real", 10, 240000, 250000,
    10, 240000, 250000));
BookHisto("g2pz", new TH2I("g2pz", "g2 pz Reco vs. Real", 10, 240000, 250000,
    10, 240000, 250000));
```

- Histograms for the reconstructed π^0 candidate properties

```
BookHisto("pi0Energy", new TH1I("pi0Energy", "Energy of pi0", 100, 0, 75000));
BookHisto("pi0Mass", new TH1I("pi0Mass", "Reconstructed mass of pi0", 200, 0,
    200));
BookHisto("pi0MCMass", new TH1I("pi0MCMass", "MC mass of pi0", 200, 0, 200));
```

- Histograms for LKr Monitoring

```
BookHisto("clusterPosition", new TH2I("clusterPosition", "Cluster position on
    LKr", 500, -2000, 2000, 500, -2000, 2000));
BookHisto("photonsNbr", new TH1I("photonsNbr", "Photons number/event", 10, 0,
    10));
BookHisto("energyCalib", new TGraph());
BookHisto("g1EnergyFraction", new TH1I("g1EnergyFraction", "Fraction between
    real energy and reco energy", 1000, 0, 100));
BookHisto("g2EnergyFraction", new TH1I("g2EnergyFraction", "Fraction between
    real energy and reco energy", 1000, 0, 100));
```

- Histograms for the pair selection algorithm

```
BookHisto("gPairSelected", new TH1I("gPairSelected", "Pair of gamma selected
    for Pi0", 10, 0, 10));
```

- Histograms specific to Monte Carlo events

```
BookHisto("g1FirstVol", new TH1I("g1FirstVol", "First touched volume for g1",
     15, 0, 15));
BookHisto("g2FirstVol", new TH1I("g2FirstVol", "First touched volume for g2",
     15, 0, 15));
BookHisto("pdgID", new TH1I("pdgID", "Non complete events : pdgID", 0, 0, 0));
```

1.1.3 InitOutput

This analyzer should provide further analyzers with a π^0 candidate if any is found. The output object is first declared in the header:

```
KinePart pi0;
```

And then registered in the framework under the name "*pi0*" in the **InitOutput** method. It should be noted that to avoid collisions between independent analyzers, this name is automatically prepended with the name of the analyzer and a dot. In this case, to access this object from another analyzer, one will have to request "*Pi0Reconstruction.pi0*"

```
RegisterOutput("pi0", &pi0);
```

1.1.4 DefineMCSimple

In this method, the specific event signature $K^+ \rightarrow \pi^+ X \pi^0 \rightarrow \gamma\gamma X$ is defined where X can be any kind or any number (including 0) of additional particle. This will allow to do extra-processing to assess the performances of the analyzer when running on on this kind of simulated events.

```
int kID = fMCSimple->AddParticle(0, 321); //ask for beam Kaon
fMCSimple->AddParticle(kID, 211); //ask for positive pion from initial kaon decay
int pi0ID = fMCSimple->AddParticle(kID, 111); //ask for positive pion from initial
     kaon decay
fMCSimple->AddParticle(pi0ID, 22); //ask for positive pion from initial kaon decay
fMCSimple->AddParticle(pi0ID, 22); //ask for positive pion from initial kaon decay
```

1.1.5 Process

1.1.6 ExportPlot

All the histograms previously booked with **BookHisto** are saved in the output ROOT file with

```
SaveAllPlots();
```

1.1.7 DrawPlot

Similarly if the analysis is running in graphical mode, all the histograms previously booked with **BookHisto** should be displayed on screen:

```
DrawAllPlots();
```

1.2 Validation