ESIPAP European School of Instrumentation in Particle and Astroparticle Physics

2020 edition

Supervisers: Eric CHABERT, Eric CONTE

Computing session 4

Data analysis using the framework ROOT

Abstract:
This computing session is dedicated to the so-called framework ROOT. Born at CERN, this framework is used every day by thousands of physicists to analyze data and to perform simulations. This session is an opportunity to have a quick overview of basics but most used functionalities provided by ROOT. More than a simple tutorial, the session also aims to use ROOT for analysis purpose in the context of the SenseHat data.

Pedagogical goals:

Software handling

- Browsing efficiently software documentation
- Understanding the description of ROOT classes
- Reading and understanding code examples. Extracting relevant information useful for the development of your programs

ROOT skills

- Reading data file (csv and ROOT format)
- Reading and writing trees
- Creating and customizing graphics
- Fitting data distributions according to model functions
- Writing results in several formats

Compiling/linking

- Creating an executable file from a simple source file using ROOT classes
- Using a Makefile with links to the ROOT libraries

Contents

1 Creating histograms **3**

2 Data handling **4**
2.1 Simple CSV-ROOT converter 4
2.2 Configurable CSV-ROOT converter [**optional**] 5

3 Data analysis **6**
3.1 Reading a TTree 6
3.2 Producing 1D Histograms 8
3.2.1 Trend plots 8
3.3 Fitting the histograms 10
3.4 Producing 2D Histograms 12

4 SenseHat: intercalibration and resolution **13**

5 Going further **14**

1 Creating histograms

Among all functionalities available in `ROOT`, producing histograms is certainly the most intensively used. In that section, we will learn how to create, fill and manipulate histograms. We will use the class TH1F. Tutorials can be found here. A good way to start, would be to use the relevant information described in the user guide.

As a startup, on can use the following instructions to create and draw a histogram with a single value. Copy the following instruction in a file `SimpleHisto.cpp`

Listing 1: Creating a simple histogram

```
void SimpleHisto(){
 int nbins = 10;
 float min = 0;
 float max = 20;
 TH1F* h = new TH1F("myHisto","Its title",nbins,min,max); //instantiation
 h->Fill(5.6); //fill histo with a single value
 TCanvas* c = new TCanvas("c");
 //create a canvas. If not specify a default
 //canvas c1 would have been created
 c->cd(); //specify where to display the graphs
 h->Draw(); // draw it
 c->Print("histo.jpg"); //save the histo as an image
 c->SaveAs("histo.root"); // save it in a root-file
 c->SaveAs("histo.C"); // could also be save in a .C file
 //with all corresponding instruction
}
```

You can the launch it from the terminal:

```
bash$root -l SimpleHisto.cpp
```

- Modify the program `SimpleHisto.C` to fill the histogram based on the data stored in the file http://echabert.web.cern.ch/echabert/ESIPAP/ROOT/data.txt.
- Adapt the range and the binning.
- Add x and y-axis titles (respectively "Energy [GeV]" and "Nb electrons").
- Increase the size of the histogram line and change its color to be filled in blue.

2 Data handling

The data produced by the acquisition program developed during the Computing Session 1 are stored in a CSV format. The first step will be to read the files and convert them into a binary format structured by ROOT (.root file).

2.1 Simple CSV-ROOT converter

- **Goal:** Write a program `CVS2ROOT.cpp` that must read a CSV file, create a TTree from the data it contains and store the TTree in a TFile.
- **Remarks**: in this first simple program, the name of the input data file can be hard-coded and there is no need to use function (beyond the main).
- **Compile** the program with g++ using the appropriate commands to indicate the location of `ROOT` headers and perform a proper linking with `ROOT` librairies using the command:

```
bash$ g++ `root-config --glibs --cflags ` -o CSV2ROOT CSV2ROOT.cpp
```

- Test the program
- Check the content of the output root-file (using TBrowser)

- Use the method ReadFile of the class TTree. More information can be found here. This method of TTree needs to know how are structured the data: name and type of variables, refered as *branch descriptor*. It can be specified in the first line of the CSV file or as argument of the method.

2.2 Configurable CSV-ROOT converter [optional]

- **Goal:** The program `CVS2ROOTv2.cpp` must take as argument in the command line, the name of CSV files (*prefix.csv*) and produce a root-file with the same prefix (*prefix.root*).
- **Constraints:** Create a function which does the conversion from CSV to root-file. Manipulate the filenames using std::string in order to achieve the above mentioned goal.
- Compile and test the program
- Check the content of the output root-file.
- Run our program on all the CSV files you've obtained.

- Create a table in which you could simultaneously write the size of the CSV and root-files.
- How can your interpret our observations ?

3 Data analysis

In that section, we will read the data stored in the root-files obtained previously and perform a basic data analysis using `ROOT` functionalities of drawing and fitting.

3.1 Reading a TTree

As a first step, you need to be able to write a simple program to read the TTree you have saved previously and access its content.

- **Goal:** The program must open a TFile, access the TTree it contains and loop over its content. For test purpose, use std::cout command to display the values of the variables stored. Once it works, you can comment the corresponding line.
- **Saving the results:** The histograms have to be saved in two ways:
  - all TH1D stored in a new root-file. The advantage is that you can always change the layout later on.
  - each TH1D drawn in a given image file (possible extensions: jpg, png, pdf, ...) using the method Print() of the class TCanvas. Choose one of those formats.
  - Report: write a report (on the terminal or in a text file) indicating for each variable the mean value and its error, the standard deviation and its error.
- **Results:** Run the program on two kinds of runs: one where the conditions were stables and one where they haver changed.

- If needed, you can read a dedicated section in the `HowTo` part of the `ROOT` website on how-to-read-tree
- TH1D has method GetMean(), GetMeanError(), GetStdDev(), GetStdDevError()
- The output of our programs can be saved on a dedicated directory called by example `results`.
- You may need to adapt the range and the binning of the differnt histograms.
- You can start with a very fine binning, smaller that the announced resolution of the sensors.
- The choice of the binning can be a compromise between known resolution and statistics.
- Using variables to define the histograms rather than hard-coded values can be helpful for later developments.

.

- What did you observed in case of very fine binning ? How do you interpert that ?
- What kind of models could describe the distributions ?

3.2 Producing 1D Histograms

3.2.1 Trend plots

It is usual to monitor the evolution of observables with time. In that section, you will develop a program in two steps in order to firstly produce and save histograms and then overlay some of them.

- **Goal:** The program must create a graphic for each of the input observables (all temperatures, pressure and relative humidity) as function of time. An option could have been to use the class TGraph but in our case, we will use the class TH1D where each of the bin (in the x-axis) will correspond to the individual measurements ordered in time.
- **Saving the results:** The histograms have to be saved in two ways:
  - all TH1D stored in a new root-file. The advantage is that you can always change the layout later on.
  - each TH1D drawn in a given image file (possible extensions: jpg, png, pdf, ...) using the method Print() of the class TCanvas. Choose one of those formats.
- **Results:** Run the program on two kinds of runs: on where the conditions were stables and one where they changed.

- **Layout:** Are you satisfied by the layout ? You can add axis titles using the method SetTitle() of TAxis objects which can be accessed through GetXaxis() and GetYaxis() methods of TH1D.

- **Analysis:** Compare the results with our expectations and try to understand unexpected features if any.

Overlaying histograms

As we have several sensors measuring temperature, it could be great to overlay the histograms in a same canvas. Extend the previous program. It would be convenient to write a function with takes as arguments two pointers of TH1D (plus the relevant variables) in order to avoid code duplication.

- Write appropriate titles to the x- and y-axis (if not yet done)
- Overlay the TH1D using the option "same" in the method Draw().
- Add an appropriate legend using the class TLegend and different colors (markers) for the TH1D histograms.
- Make sure that the range on the y-axis covers the maximum of all displayed curves.
- Store the result using methods presented previously.

3.3 Fitting the histograms

A frequent treatment of the data consist in fitting them with a predefined model. This allows to firstly test the data/model compatibility but also to derive parameters. We will reuse previously obtained histograms to firstly check if the observables were stable with time and secondly to determine the SenseHat temperature sensor resolution as well as to verify their intercalibration.

Fitting trend plots

- **Goal:** The program will use trend histograms produced earlier and will fit them with two models: a constant and a first order polynom. To do so, you can use the method Fit of TH1D to do that. Read the related documentation to choose the argument(s).
- The result from the fit can be accessed through a variable of type TFitResultsPtr from which it is possible to retrieve the parameters and their uncertainties as well as a χ^2.

- **Generalization:** In order to avoid code duplication, write the above-mentioned code in a function that will take as argument a TH1D, will perform both fits and will write a formatted report which may help the user to determine if the temperature was stable or not. Write a program which read a root-file, check the existence of predefined histogram names and them perform the above-mentioned fits.
- You can read the related document here.
- Think about code protection to handle errors.

Fitting temperature distributions

- **Goal:** The program must read the histograms of the temperature measured by the pressure and humidity sensors in a run where the conditions were stable (cf previous tasks). Those histograms can be fitted by a gaussian function. Retrieve the parameters and the function and from them, determine firstly the detector resolution, secondly compare the mean parameters (and their uncertainties) to determine if the sensors need to be inter-calibrated.
- **Generalization:** The above task can be achieved by developing a function that will take as argument a TH1D. A formatted report from the fit result should be printed at the end of the function.

- **Results:** Compare the resolution you've obtained with the ones indicated on the datasheet. Are they compatible ?
- **Going further:** The previous study can be repeated by analyzing data where the temperature was stable but at different values.

3.4 Producing 2D Histograms

It is often convenient to work with 2D histograms in order to visually check how data are distributed and if the observables are correlated.

- **Correlation of temperature measurements on the Sense-Hat board**
  - **Goal:** The program must create a TH2D where the x- and y-axis correspond to the temperature measured by the pressure and humidity sensors. Use appropriate ranges and binning as well as axis titles. Several drawing options are available. Use the one called "COLZ" as a first trial but you can then extend our tests with additional options. Extract the linear correlation factor between the observables from the TH2D. Write the histogram on a output file.
  - **Generalization:** Write the previous code using a function. Find a solution to change the fitting range via function argument(s).
  - **Results:** Study the results for several incoming data files. What do you conclude ?
- **Correlation between the temperature measured on the SenseHat and the raspberry activity**
  - **Goal:** The program must create TH2D histograms to evaluate the correlation between the temperature measured on the SenseHat and the raspberry activity which are monitored through a ARM and GPU temperatures. Moreover a TH2D of the latters should be created.
  - **Generalization:** Use appropriate ranges and binning as well as axis titles. Extract the linear correlation factor between the observables from the TH2D histograms. All this should be embedded in a function.
  - **Results:** Study the results you've obtained and try to draw conclusions. You can study data taken in different conditions in term of distance between the raspberry and the SenseHat as well as the raspberry activity.

4 SenseHat: intercalibration and resolution

From the conclusions you've drawn, propose improvement on the setup. As guideline, think about the following aspects:

- **Calibration:** you can inject an intercalibration factor of the temperature sensors. Modify the code on the raspberry and check the results you will obtain.
- **Resolution:** improve the temperature measurement accuracy by averaging several consecutive measurements. One can determine in advance the number of measurements to achieve the desired accuracy. Modify the code on the raspberry to obtain temperature measurement.
- **Temperature correlation:** influence of the raspberry activity on the SenseHat temperatures. Runs with different distance between the raspberry and the SenseHat can be fone.

5 Going further

Configurable plots

- **Goals:** Several options could be chosen by the user through a configuration file (a simple text file):
  - the range and binning
  - the color/marker of the histograms
  - the format(s) of the output file
  - the name of the directory in which all results will be stored
- **Automatization:** An other possibility to avoid issues with the ranges is to evaluate the min and max values of input data. Try this only if you have enough time at the end.

Linearity

You can check if the intercalibration factor between the two temperature sensors is constant or depends on the temperature.