Global PID Framework Documentation

Celeste Pidcott

1 Introduction

The global PID framework is designed to use sets of PID variables to 1) use MC data to create PDFs of these variables for a range of particle hypotheses, and 2) to use the PDFs as part of a log-likelihood method to determine the PID of reconstructed global tracks from data. The framework is designed such that new PID variables can be added as they are developed. Section 1 of this document will explain how to use the PID to produce PDFs, and how to perform PID on spill data contained within a Json document. Section 2 will detail how these two actions are performed within the code, and in Section 3 the PID variables, their structure, how new ones can be added to the framework, and details of those already in place, will be discussed. This document will be updated as the PID framework and variables continue to be developed.

1.1 Using the PID scripts

1.2 Producing PDFs

Whilst the PID framework comes with PDFs provided in PIDhists.root, it is possible for a user to produce PDFs for hypotheses not included within this file. The following describes how this should be done.

- Simulation: Production of MC data for a given particle hypothesis. To produce and output the MC to a Json file, a copy of simulate_mice.py should be made, with the my_output flag set to MAUS.OutputPyJSON().
- Global Reconstruction: The MC data should then be passed through the global reconstruction. Detector information is currently added global tracks using the GlobalReconImport.py script in ${MAUS_ROOT_DIR}\bin\Global. This script calls the mapper MapCpp-GlobalReconImport, which constructs the global tracks required for the

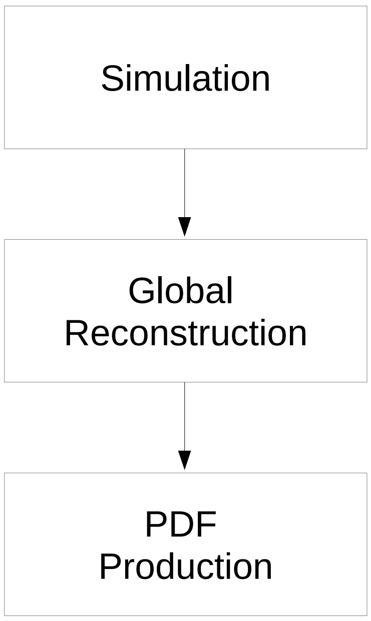

Figure 1: Steps invloved in producing a PDF from MC data

calculation of PID variables. The control variables to specify the input Json data sample, and the name of the output Json file that contains the reconstructed tracks can be set at the command line, or by using another datacard, as shown in listing 1. To run the global reconstruction with the datacard, the following should be entered at the command line:

```
${MAUS_ROOT_DIR}/bin/Global/GlobalReconImport.py
    --configuration_file <name_of_datacard>
```

which for the example in 1 would be:

```
${MAUS_ROOT_DIR}/bin/Global/GlobalReconImport.py
    --configuration_file ex_global_datacard.py
```

- PDF Production: To produce the PDFs from the reconstructed MC data, pid_pdf_production.py in ${MAUS_ROOT_DIR}\bin\Global is then used. This script calls the reducer ReduceCppGlobalPID. With this script, a datacard, such as that shown given in listing 2, that includes the input Json filename, the global_pid_hypothesis for which the PDF(s) are to be produced, and a unique_identifier (typically the time and date at which the script is run) is used by entering at the command line:

```
${MAUS_ROOT_DIR}/bin/Global/pid_pdf_generator.py
    --configuration_file example_pdf_datacard.py
```

This will create a directory within ${MAUS_ROOT_DIR}\files\PID corresponding to the hypothesis and identifier given by the datacard, which will then contain files for each PID variable, each of which will contain the PDF for that hypothesis and variable.

```python
import os

# A json document containing spills from MC data
input_json_file_name = "example_hypothesis.json"
input_json_file_type = "text"

# The json document that the global tracks will be
# written to
output_json_file_name =
        "example_hypothesis_Global_Recon.json"
output_json_file_type = "text"
```

Listing 1: An example datacard (ex_global_datacard.py) for use with GlobalReconImport.py

```python
import os
import datetime

# Use the current time and date as a unique
# identifier when creating files to contain PDFs.
# A unique_identifier is required by the reducer,
# and PDF production will fail without one.
now = datetime.datetime.now()
unique_identifier =
        now.strftime("%Y_%m_%dT%H_%M_%S_%f")

# A json document containing global tracks from MC
# data
input_json_file_name =
        "example_hypothesis_Global_Recon.json"
input_json_file_type = "text"

# The particle hypothesis that the PDF is being
# created for. A global_pid_hypothesis is required
# by the reducer, and PDF production will fail
# without one.
global_pid_hypothesis = "example"
```

Listing 2: An example datacard (example_pdf_datacard.py) for use with pid_pdf_generator.py

1.2.1 Performing PID with pre-existing hypotheses

To perform PID on data, the steps shown figure 2 should be followed.

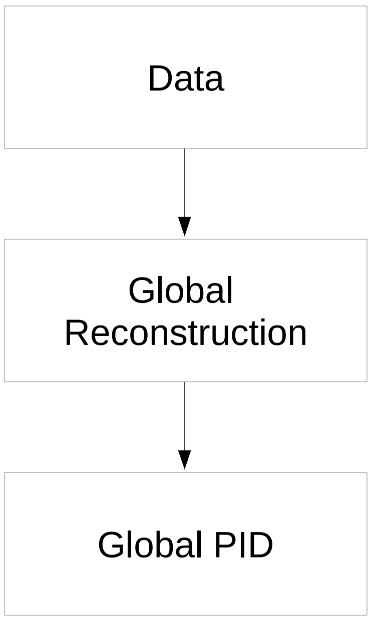

Figure 2: Steps invloved in performing the PID for a data sample

- Data: This can be experimental or MC data, however the spill data must be passed to the PID in a Json document.

- Global Reconstruction: In the same way as described above, the data should then be passed through the global reconstruction, currently using the GlobalReconImport.py script in ${MAUS_ROOT_DIR}\bin\Global, with a corresponding datacard containing the name of the input Json file and the name of the output file.

- Global PID: To perform the PID on the reconstructed data, GlobalPID.py in ${MAUS_ROOT_DIR}\bin\Global is then used. This script calls the MapCppGlobalPID mapper. With this script, a datacard, such as that shown given in listing 3, that includes the input and output Json filenames, is used, by entering the following at the command line:

  ```
  ${MAUS_ROOT_DIR}/bin/Global/GlobalPID.py
          --configuration_file
                  example_pid_datacard.py
  ```

```
import os

# A json document containing spills from data
input_json_file_name =
        "example_hypothesis_Global_Recon.json"
input_json_file_type = "text"

# The json document that the global tracks will be
# written to
output_json_file_name =
        "example_hypothesis_Global_PID.json"
output_json_file_type = "text"
```

Listing 3: An example datacard (example_pid_datacard.py) for use with GlobalPID.py

As the framework currently stands, the output document would now contain the global tracks with the PID set (where it has been possible to do so) to whichever particle hypothesis had the highest log-likelihood. For tracks where the PID could not be determined, the track PID will be left as 0.

2 MapCppGlobalPID and ReduceCppGlobalPID

2.1 MapCppGlobaPID

The steps taken in MapCppGlobalPID for a single track are shown in figure 3. To express this more fully, the data, having passed through the global reconstruction, is then passed to the PID. For each track, the values of each PID variable are calculated. Each of these values is then compared to the corresponding PDFs for all particle hypotheses, the number of entries in the corresponding bin providing the probability from which the log-likelihood is calculated. For each particle hypothesis, the log-likelihoods of all of the PID variables are summed to give a log-likelihood for that hypothesis. The PID of the track is then obtained by comparing the log-likelihoods of the hypotheses.

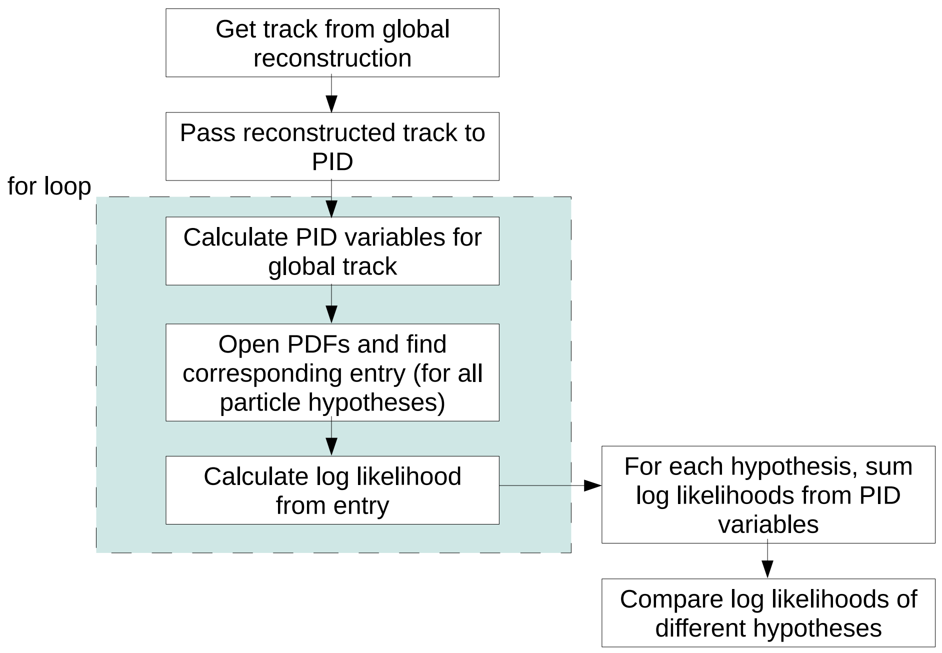
Get track from global reconstruction
Pass reconstructed track to PID
for loop
Calculate PID variables for global track
Open PDFs and find corresponding entry (for all particle hypotheses)
Calculate log likelihood from entry
For each hypothesis, sum log likelihoods from PID variables
Compare log likelihoods of different hypotheses

Figure 3: Flow chart detailing steps taken in MapCppGlobaPID

3 ReduceCppGlobalPID

The steps taken in ReduceCppGlobalPID are shown in figure 4. MC data for a given particle hypothesis, having passed through the global reconstruction, is then passed to the PID. For each track, the values of each PID variable are calculated. A histogram is filled with these values. If the behaviour has been turned on in the PID variable class, then a single event is spread over all bins in the histogram, to ensure that when the PDF is used by the PID, there will no empty bins, thus avoiding cases where the log-likelihood takes the log of zero. The histogram is then normalised to create the PDF, which is then written and saved to file. If a MC track returns a variable value outside of the allowed range of the histogram (as defined within the variable class) then the value for that track is not included.

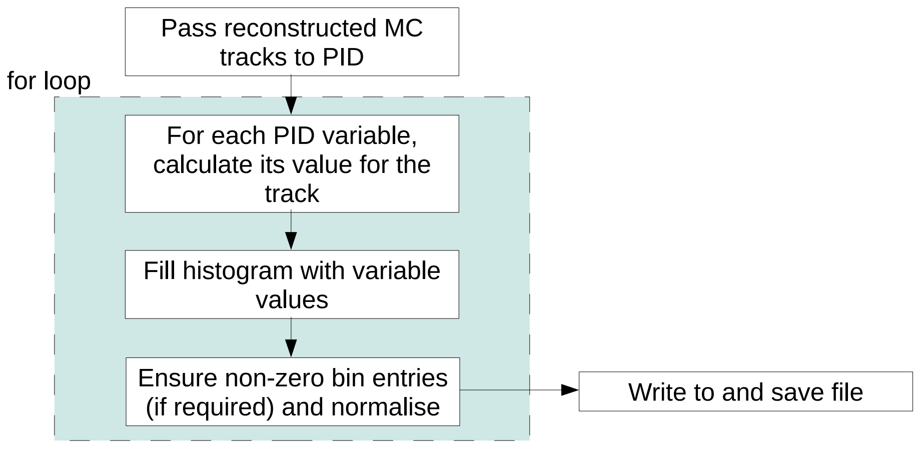
Pass reconstructed MC tracks to PID
for loop
For each PID variable, calculate its value for the track
Fill histogram with variable values
Ensure non-zero bin entries (if required) and normalise
Write to and save file

Figure 4: Flow chart detailing steps taken in ReduceCppGlobaPID

4 PID Variables

Information from the MICE detectors will be incorporated into a set of PID variables that can be used to distinguish between particle hypotheses. The Global PID framework has been written such that any number of PID variables can be developed and added as necessary, all represented by their own class, derived from a base class.

4.1 PID Base Class

The base PID class (PIDBase.hh and .cc) contains the functions to:

- Create the PDFs (and the files that contain them)
- Use the PDFs with globally reconstructed tracks
- Populate the PDFs with variable values (after checking that value is valid)
- Perform the log-likelihood for an incoming globally reconstructed track (after checking that value of variable for track falls within range of PDF).
- Calculate the value of the PID variable (this is a virtual function to be defined in the derived classes)

4.2 PID Variable Classes

Each PID variable will be implemented in a derived class of the base PID class. Because of how the framework is designed, new variables can be added as they are developed.

4.2.1 Adding PID Variables

In each derived variable class, the following should be included:

- The variable name should be set
- The function to calculate the PID variable should be defined.
- The minimum, maximum, and number of bins for PDFs created using the variable should be set. The values of the minimum and maximum define the allowed range of values that the PID variable can take.
- In some cases it may be necessary to ensure that all bins in a PDF return non zero entries, and so by setting the variable _nonZeroHistEntries to true, a single event spread accross all bins will be added

4.2.2 PIDVarA

There is currently a single PID variable defined within the framework, PIDVarA (see PIDVarA.hh and .cc), which uses the difference between the times measured at TOF1 and TOF0 as its variable. Only for tracks where there is a single TOF0 and a single TOF1 time measurement, and for which the time difference between the detectors falls within the minimum and maximum set within the class, will a valid value of the variable be returned. Otherwise, the value of the variable is set to -1, such that it falls outside of the allowed range for the variable, and so variable for the track is not used in PDF production, or in the PID.