Pellet tracking simulations software

Marek Jacewicz

January 10, 2011

Contents

1 Recommended computer environment 2
2 Setup simulation 2
3 Simulations 3
  3.1 Processing steps ........................................ 3
  3.2 Description of classes .................................. 4
  3.3 Compilation and executable ............................. 9
4 How to start the simulation? 9
5 Appendix 1. PTSMain.cpp 11
6 Appendix 2. PTS.config 12
7 Appendix 3. script.csh 15
8 Appendix 4. prepare.csh 16
1 Recommended computer environment

The code is written in C++ and relies on ROOT libraries and Standard Template Library (STL) for container classes. Version 5.26 of ROOT was used during development of the code. The code should be fairly portable, but as a benchmark the linux environment at the Department was used where version 5.26 of ROOT has been compiled and tested on institute’s computers (should work for all computers in the cluster)

This ROOT version is available under the directory

```
/home/tsl3/wasa/jacewicz/root
```

This path should be set to ROOTSYS system-variable in your logging script.

For bash shells you can write:

```
export ROOTSYS=/home/tsl3/wasa/jacewicz/root
export PATH=$ROOTSYS/bin:"$PATH"
export LD_LIBRARY_PATH=$ROOTSYS/lib:"$LD_LIBRARY_PATH"
```

For csh shells something like:

```
setenv ROOTSYS /home/tsl3/wasa/jacewicz/root
set path=($path $ROOTSYS/bin)
setenv LD_LIBRARY_PATH $ROOTSYS/lib
```

The simulation package should work also with the older version of the ROOT framework but this has not been tested.

2 Setup simulation

Simulation is kept in the SUBVERSION repository available from for example

```
svn co file:///home/jacewicz/svnrepo/PelletSim
```

A directory PelletSim will be created and latest version of the simulation uploaded. (Treat this as an experimental version as it could contain untested code/settings).

Stable release of the simulation package can be found as a compressed tar-ball at (choose the latest):

```
/home/jacewicz/PelSim_vXXXXXXX.tgz
```

Upon uncompressing the archive a dedicated directory PelSim will be created containing all the software. New release is only provided after a new milestone is reached in the simulation.
3 Simulations

Previous version of the code was split into two steps with two programs run one after the other separately (or together with a help of simple unix shellscript). Currently there is only one main program where both steps of the simulation are contained: Pellet generation and Analysis.

3.1 Processing steps

Step 1 Generate true and measured information for the individual pellets in a sample.

- generate pellet (time, position, direction, speed)
- propagate to eventual skimmer, skim i.e. remove if outside opening (not implemented)
- propagate to measurement positions (1,2,3 etc), determine true time and position
- simulate light signal (time, duration, strength) and camera (field of vision, exposure, efficiency) at measurement positions. Determine measured information (time, position)
- propagate to the interaction region, determine the time interval the pellet is in the region
- generate time and position for the interaction taking into account the accelerator beam profile

Step 2 Correlate information from the measurement positions. Take into consideration information from other pellets.

Example 1: Correlate information from measurement position 1 and 2, by using the nominal speed $v$.

- sort the records in the sample in true time order of arrival to measurement position 1
- note if several pellets can have given signal in same exposure
- sort the records in the sample in true time order of arrival to measurement position 2
- check which pellet has given the signal that best fits together with $v$. Note which pellet it is and use its measurement information.
- note if several pellets can have given signal in same exposure
Example 2: Correlate information from measurement position 3 and 1-2 by using the speed calculated from the measurements at 1-2. Take into consideration information from other pellets.

- sort the records in the sample in true time order of arrival to measurement position 3
- check which pellet has given the signal that best fits \( v_{1-2} \). Use its measurement information.
- note if several pellets can have given signal in same exposure

For each pellet there is now a record with all measurement information that is necessary for making tracking and evaluate the tracking procedure. The records can be analyzed in different ways depending on the special issues e.g. number of pellets in the interaction region at the same time and accuracy and efficiency of the tracking information.

### 3.2 Description of classes

The first step in the simulation is a generation of a single pellet, followed from generation point to the measuring points and beam crossing.

The following classes/files are used inside the first simulation package:

- **PTSMain - main program**, (see Section 5 for simplified example code).
- **PTSPosition - class for pellets true information**

```cpp
class PTSPosition {
  Int_t iIndex; // event number
  TVector3 vPosition; // 3D position
  Double_t fTime; // time at generation
  Double_t fVelocity; // velocity
  Double_t fDirectionZ; // dx/dy direction
  Double_t fDirectionX; // dz/dy direction
  vector<PTSTrack> vTracks;
}
```

- **PTSMeasurement - class for pellets measured information** (derived from PTSPosition)

```cpp
class PTSMeasurement : public PTSPosition {
  TVector3 vCameraPosition; // camera position(filled at initialization)
  Double_t fExposureX; // from overlap with camera exposure
  Double_t fPlaceInCycle; // where in cycle arrived the hit
  Double_t CoordinateXZ; // line orientation X or Z
  Double_t ValueX; // cluster position in X from camera
  Double_t ValueZ; // cluster position in Z from camera
  Double_t Direction; //
```
• PTSBeam - contains information about pellets interaction with beam (derived from PTSPosition)

```cpp
class PTSBeam : public PTSPosition {
    Double_t fMeasTime; // measured time "real time"
    Double_t fMeasTimeTop;
    Double_t fMeasTimeBottom;
    Double_t fBeamPositionY; // from generation
}
```

• PTSPellet - "container" class for pellets, contains all information about the pellet obtained during simulation

```cpp
class PTSPellet {
    PTSEnv *pEnv; // access to simulation parameters
    TF1 *pgenPosFunction; // functions used to generate info
    TF1 *pgenTimeFunction;
    TF1 *pgenDirFunction;
    TF1 *pgenVelFunction;
    TF1 *pgenBeamFunction;
    // Measurements
    vector<PTSMeasurement> vMeasurements; //keeps info from measurement points
    PTSBeam vBeam; // keeps info about ion beam crossing
    PTSPosition vGenerationPoint; // keeps info about pellet generation
}
```

Each pellet at the generation has a unique velocity, direction, time etc. These parameters are controlled by variables in the configuration file and can be adjusted to match a chosen distribution. At the moment all pre-defined distribution in ROOT package can be used. Below you can see examples of how to generate time with: Gaussian, Landau, Laplace and flat (rectangular) shape.

- **TimeSmearingFunction: gaus**
  - **TimeSmearingParams: 1.0 0.0 .000001**
  - **TimeSmearingRange: -0.0001 0.0001**

- **TimeSmearingFunction: TMath::Landau(x,[0],[1],0)**
  - **TimeSmearingParams: 0.5e-7 1.75e-7**
  - **TimeSmearingRange: -0.00001 0.00001**

- **TimeSmearingFunction: TMath::LaplaceDist(x,[0],[1])**
  - **TimeSmearingParams: 0. 1.0e-6**
  - **TimeSmearingRange: -0.000001 0.000001**

- **TimeSmearingFunction: pol0**
  - **TimeSmearingParams: 1.0**
  - **TimeSmearingRange: -0.0000025 0.0000025**
Figure 1: Example of the different smearing of the generation time, from left: Gaussian, Landau, Laplace and flat distribution.

- PTSEnv - class for design/control input. Responsible for the read-out of the configuration file.
- PTSHistoManager - histogram/tree manager
  Handles all booking and filling of the graphic objects (at the moment it is limited to TH1D-, TH2D-histograms and TTrees). Histograms (and trees) are not filled immediately but “on demand” allowing to save the same histograms under different conditions resulting e.g. from the result of the analysis. At the moment this feature is not used extensively and the histograms are only filled at the end of analysis.
- PTSDataObject - ”container” class for data used for histograming with different condition
- PTSTreeObject - This is object saved in the output tree. Contains generation, measurement points and beam crossing information.

```cpp
class PTSTreeObject {
  PTSPosition genInfo;
  vector< PTSMeasurement > measInfo;
  PTSBeam beamInfo;
}
```

- PTSPixelCluster - cluster class. Simple clustering of pixels in the line-scan camera

```cpp
class PTSPixelCluster {
  vector<Double_t> pixels; // pixels with geometrical "hit"
  vector<Double_t> ratio; // how much pellet was seen by this pixel in %
  Int_t NumberOfPixels;
}
```

- PTSTrack - contains information after each step during tracking (3D position and time)
The second step in the simulation is an analysis of a collection of pellets obtained in the first step. They are saved in the ROOT TTree structure. There is one class called **PelStudies** which is analyzing the tree with several available functions.

These analysis subroutines simulate the behavior of a pellet stream. Relation between pellets are studied at different points in the target setup. The idea is to turn on and off each of the subroutines in the main program (**PTSMain.cpp** at the end of file) in order to study a particular aspect of the pellet tracking. Due to the processing time of some of the routines it is not recommended to include all of them at once unless the number of the events is reduced (1000 is a safe limit for quick checks - full processing takes approximately 10 seconds, with 10000 events the processing time increases to 1000 seconds). Simple timing is provided in the main program (**PTSMain.cpp**)

Short information about technical aspect of each subroutine is given below

**PelletOccupancy()** We calculate the expected time of flight to the next measurement point using true velocity and then search for the correct pellet at the second measuring point. Events are classified to different cases and the shortest time difference between expected arrival time and time of nearest found pellet is saved. The search for the pellets is limited to a time window related to the velocity spread with a factor given by an internal variable “nsigma”, which is by default set to 2.

**BeamOccupancy()** Here we investigate behavior of the pellets inside the accelerator beam. The main parameters of interest are the multiplicity of the pellets, the time difference between pellets and the fraction of time with/without any pellets in the beam.

**TimeOccupancy()** Checking number of pellets during beam crossing. The snapshots of the number of pellets in the ion beam (recorded during short time interval set by a control variable in the configuration file “TimeOccupancyShift”) are saved. The beam profile can be assumed rectangular or Gaussian, however the change has to be done by commenting/uncommenting a small fraction of the code.

By default also text information are written to a file named “time_scan.data” in the following format:

```
Vel dVel Dist Period Expos Thr Freq Avg1 Avg2 Sigma1 Sigma2
```
“Avg1” and “Sigma1” are calculated for all the bins, “Avg2” and “Sigma2” only for the filled ones. **Note:** This information is appended to the “time_scan.data” file and not overwritten. In this way one can run simulation scanning several parameters and saving result from the analysis of the ion beam crossing in one file. Additional scripts “time_plot.C”, “time_tables.C” and “scan_to_table.C” can be helpful to present the information collected in the data and the ROOT file.

**PositionStudy()** First attempt at study of the behavior of the pellets inside a pellet stream. Deprecated, becoming obsolete soon.

**TrackingStudy()** Study of the intra-beam interactions in the pellet stream. For this analysis, a special tracking is done at the generation stage of the simulation. That is why this analysis is controlled from the configuration file (a flag called: “UseTracking” expects a “yes/no” answer). Information for each pellet is saved during the propagation from the generation point to the first measurement point. The other controlling variable is “UseFixedStepSize” (answer “yes” with divide the distance using “FixedStepSize” size steps, answer “no” will use a step size related to the direction smearing spread and pellet size: $2 \times \phi_{pellet}/\sigma_{direction}$. By default the fixed tracking step is small (10 mm) and the distance will decide the number of points saved for each pellet and therefore the amount of a required disk-space. User should be aware that putting first measuring point at the distance of e.g. 4 meters from the generation point would result in >500 MB output file (for one set of parameters).

**ScanPelletVelocity()** Subroutine is looping over different velocity hypotheses with given accuracy. The scanning window should include the “true” generated velocity and the parameters should be adjusted inside the subroutine to scan the region of interest with good accuracy. In the analysis of real events similar subroutine would have to be run iteratively, scanning first roughly wide range of velocities and later zooming into the discovered velocity with higher precision.

**FindPelletVelocity()** We calculate the time difference between two measuring points and with the knowledge of that distance we calculate the velocity. All combinations between pellets are tested. This is more direct method, however the obtain spectra are suffering from combinatorial background.
3.3 Compilation and executable

Compilation is done by use of Makefile with command 'make'. The resulting executable is called PTSMain and takes the following command line parameters:

```
./PTSMain config_file root_file
```

**config_file** - configuration file-name (default is PTS.config) - this is a text file with all controlling parameters (example is given in the Section 6)

**root_file** - ROOT file-name - optional parameter - file with output information. If no name is given, a name based on the simulation parameters is created. In fact two root files will be created. First one with the name given by the user (or auto-generated) (e.g. 'myhistos.root' or 'F10t20e16Thr0.3pel0.03pix0.04b2.root') will contain histograms from the simulation and the second one with prefix 'tree_' (e.g. 'tree_myhistos.root' or 'tree_F10t20e16Thr0.3pel0.03pix0.04b2.root') with the results from the simulation collected in the ROOT **TTree** for the second stage of processing.

4 How to start the simulation?

1. Single run

   Prepare configuration file and run as explained above. Resulted histograms are inside ROOT file under directories **GENERATION** and **ANALYSIS**.

   With one of the parameters in the configuration file ("OverwriteFiles" with a "yes/no" answer) we can decide to overwrite any existing ROOT file ("yes" answer) or to use an existing ROOT file containing the output tree during the second stage analysis ("no" answer). This feature can be used if we decide to generate big, multi-pellet files and want to save time on the reprocessing. Bear in mind that generation of 10000 pellets takes only approximately 3-5 seconds so the default setting is to overwrite the existing file to avoid confusion.

2. Scan run

   To facilitate the start of the simulation scanning variables on input, 2 additional scripts are available: **script.csh** and **prepare.csh**.

   The purpose of **script.csh** is to generate automatically new configuration file that can be immediately use in the simulation. It can take
several parameters from the command line, for example the default script (see the Section 7) takes 4 parameters:

```
csh script.csh freq vel_spread pos1 pos2
```

- **freq** - frequency in Hz
- **vel_spread** - velocity spread in millimeter/s
- **pos1, pos2** - positions of the first/second measurement points in millimeters (default generation point is at 3000 mm)

Example, the following call:

```
csh script.csh 5000 3000 2500 2400
```

will produce the following output files:

- `freq5000_3000_2500_2400.config` - full config file,
- `freq5000_3000_2500_2400.root` - root file with histograms from single pellet simulation
- `tree_freq5000_3000_2500_2400.root` - root file with tree with pellets

The script.csh is kept very simple and allows for easy change/addition of command line parameters.

An additional script **prepare.csh** allows for an automated scan of several parameters. Please edit the file if you need to change the values included in the scan or to change the positions of the measuring points. Please remember that by default the pellet generation point is at 3000 mm i.e. the given values must be smaller than this. At the moment the scan is performed in frequency and velocity spread but the script can be easily extended (see the Section 8).

To start write:

```
csh prepare.csh
```

and be patient.
 Appendix 1. *PTSMain.cpp*

```cpp
int main(int argc, char *argv[]) {
    // some initialization ...
    PTSEnv *fEnv = new PTSEnv(filename);
    PTSHistoManager *hm = new PTSHistoManager(fileHist);
    PTSDataObject *fData = 0;
    fData = hm->CreateDataObject("GENERATION", "generation of pellets");
    hm->CreateTree();
    // prepare Pellet class
    PTSPellet *pellet = new PTSPellet(fEnv);
    for (Int_t nev = 0; nev < PTSEnv::NumberOfEvents; nev++) {
        // calculate information for each pellet
        pellet->Generate(RunningTime, nev);
        pellet->Measure();
        pellet->Tracking();
        pellet->CalculateBeam(PTSEnv::BeamCenter);
        // access info for each pellet
        PTSPosition gPoint = pellet->GetGenerationPoint();
        PTSBeam beam = pellet->GetBeamInfo();
        vector<PTSMeasurement> vec;
        Int_t iout = pellet->GetMeasPoints(vec);
        // some histograming ... like :
        fData->Save1D("TotalNPixels","Number of pixels found per event (all layers)",
                       10, pair<Float_t, Float_t> (-0.5, 9.5), npix);
        // fill histograms and tree
        hm->Fill(fData, "_all");
        PTSTreeObject *tobj = hm->GetTree();
        (tobj->genInfo) = gPoint;
        (tobj->measInfo) = vec;
        (tobj->beamInfo) = beam;
        // some cleaning before next event ....
    } // some cleaning after all events ....
} // generation is finished, start with analysis
PelStudies *pelstud = new PelStudies(tree, fEnv, hm);
pelstud->Setup();
pelstud->Sort();
pelstud->PelletOccupancy();
pelstud->BeamOccupancy();
pelstud->TimeOccupancy();
pelstud->PositionStudy();
pelstud->TrackingStudy();
pelstud->ScanPelletVelocity();
pelstud->FindPelletVelocity();
// some cleaning after analysis }
```
6 Appendix 2. *PTS.config*

# GENERAL
# FOR UNIFORM DISTRIBUTION USE POL0 AND PARAMETER SET TO 1
# THE WIDTH HAS TO BE SET AS 2 ADDITIONAL PARAMETERS E.G
# X AND Z POSITION SMEARING ARE SHARING SMEARING FUNCTION AND PARAMETERS
# FOR GAUS: IF EQUAL VALUE ARE GIVEN IN RANGE FIELD
# 3 SIGMA RANGE WILL BE ASSUMED
# TIME VARIABLES SHOULD BE GIVEN IN SECONDS
# AND DISTANCES IN MILLIMETERS
#________________________________________________________________________
#
# NUMBER OF EVENTS TO PROCESS
#_____________________________________
NumberOfEvents: 1000
#
# FREQUENCY IN Hz
#_____________________________________
Frequency: 10000.
#
# SYNCHRONIZATION DRIFT IN SECONDS/1S
#_____________________________________
FrequencyDrift: 0.000020
#
# TIME SMEARING IN S
#_____________________________________
TimeSmearingFunction: gaus
TimeSmearingParams: 1.0 0.0 .000005
TimeSmearingRange: -0.0001 0.0001
#
# POSITION SMEARING
#_____________________________________
PositionSmearingFunction: pol0
PositionSmearingParams: 1.0
PositionSmearingRange: -0.025 0.025
#
# DIRECTION SMEARING
#_____________________________________
DirectionSmearingFunction: gaus
DirectionSmearingParams: 1.0 0.0 0.0001
DirectionSmearingRange: 0.01 0.01
#
# VELOCITY SMEARING IN MM/S
#_____________________________________
VelocitySmearingFunction: gaus
VelocitySmearingParams: 1.0 60000.0 600.
VelocitySmearingRange: 50000. 50000.
#
# BEAM PROFILE
#_________________________
BeamSmearingFunction: gaus
BeamSmearingParams: 1.0 0.0 1.0
BeamSmearingRange: -2.0 2.0
#
# FIXED Y POSITION IN MM
#_________________________
YPosition: 3000.
#
# MEASUREMENT POINTS IN MM
#_________________________
NumberOfMeasPoints: 4
MeasurementPoint1: 2350.
MeasurementPoint2: 2250.
MeasurementPoint4: -2500.
#
# CAMERA RELATED SPECIFICATIONS
#_________________________
CameraPeriod: 0.00002
CameraExposure: 0.000016
ExposureThreshold: 0.3
PelletSize: 0.03
PixelSize: 0.04
#
# BEAM
# SHOULD WE USE BEAM PROFILE? YES/NO
#_________________________
BeamProfileOn: yes
#
# BEAM CENTER AND SIZE IN Y DIRECTION
# USED WHEN BeamProfileOn IS SET TO 'NO'
#_________________________
BeamCenter: 0.
BeamSize: 2.0
#
# OVERWRITING FILES?
#_________________________
OverwriteFiles: yes
#
# USING FIXED STEP SIZE IN TRACKING?
# STEP SIZE SHOULD BE GIVEN IN MM
#
UseTracking: yes
UseFixedStepSize: yes
FixedStepSize: 10.0
#

# CONTROLLING VARIABLES FOR ANALYSIS SUBROUTINES
#
TimeOccupancyShift: 2e-6
7 Appendix 3. script.csh

#!/bin/csh
#
set freq = $1
@ dvel = $2
@ pos1 = $3
@ pos2 = $4
#
set name = 'freq'$freq'_'$dvel'_'$pos1'_'$pos2
set outputfile = $name'.config'
set rootname = $name'.root'
rm -f $outputfile
touch $outputfile

echo 'NumberOfEvents: 10000' >> $outputfile
echo 'Frequency: '$freq'.' >> $outputfile
echo 'FrequencyDrift: 0.00002' >> $outputfile
echo 'TimeSmearingFunction: gaus' >> $outputfile
echo 'TimeSmearingParams: 1.0 0.0 5e-8' >> $outputfile
echo 'TimeSmearingRange: -100e-8 100e-8' >> $outputfile
echo 'PositionSmearingFunction: pol0' >> $outputfile
echo 'PositionSmearingParams: 1.0' >> $outputfile
echo 'PositionSmearingRange: -0.025 0.025' >> $outputfile
echo 'DirectionSmearingFunction: gaus' >> $outputfile
echo 'DirectionSmearingParams: 1.0 0.0 0.001' >> $outputfile
echo 'DirectionSmearingRange: -0.01 0.01' >> $outputfile
echo 'VelocitySmearingFunction: gaus' >> $outputfile
echo 'VelocitySmearingParams: 1.0 60000.0 '$dvel'.' >> $outputfile
echo 'VelocitySmearingRange: 50000. 50000.' >> $outputfile
echo 'BeamSmearingFunction: gaus' >> $outputfile
echo 'BeamSmearingParams: 1.0 0.0 1.0' >> $outputfile
echo 'BeamSmearingRange: -2.0 2.0' >> $outputfile
echo 'YPosition: 3000.' >> $outputfile
echo 'NumberOfMeasPoints: 2' >> $outputfile
echo 'MeasurementPoint1: '$pos1'.' >> $outputfile
echo 'MeasurementPoint2: '$pos2'.' >> $outputfile
echo 'CameraPeriod: 0.000014' >> $outputfile
echo 'CameraExposure: 0.000010' >> $outputfile
echo 'ExposureThreshold: 0.3' >> $outputfile
echo 'PelletSize: 0.03' >> $outputfile
echo 'PixelSize: 0.04' >> $outputfile
echo 'BeamProfileOn: 0' >> $outputfile
echo 'BeamCenter: 0.' >> $outputfile
echo 'BeamSize: 1.4' >> $outputfile
echo 'OverwriteFiles: yes' >> $outputfile
echo 'UseFixedStepSize: yes' >> $outputfile
echo 'TimeOccupancyShift: 20e-6' >> $outputfile
echo '' >> $outputfile

../PTSMain $outputfile $rootname

8 Appendix 4. prepare.csh

#!/bin/csh
#
set freqs = (5000 10000 20000)
set dvels = (6 60 600)
@ pos1 = 2350
set pos2 = (2300 2250 -350)
#
foreach f ($freqs)
foreach s ($dvels)
foreach p1 ($pos1)
foreach p2 ($pos2)
csh script.csh $f $s $p1 $p2
end
end
end
end