USERS MANUAL

TRAGALDABAS documentation

Hardware & Software

Instituto Galego de Altas Enerxías

October 6, 2020

IGFAE
Instituto Galego de Física de Altas Enerxías

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

XUNTA DE GALICIA

Contents

Chapter 1

Summary

Chapter 2

Introduction

2.1 The Cosmic Rays

At present, cosmic rays with large energies cannot be detected directly, so that we must measure the products of the atmospheric cascades of particles initiated by the incident astroparticle. In general, this cascades of secondary particles are generated by the inelastic nuclear collision between cosmic rays –those astroparticles– and the atmospheric particles. Those secondary particles continue interacting and generating other and other secondary particles until a maximum is reached, and then the shower atenuates as far as more and more particles fall below the threshold for further particle production.

2.2 The Detector

Since 2014, in the LabCAF laboratory of the Faculty of Physics of the USC, a TRASGO type detector has been installed and taking data: TRAGALDABAS (TRAsGo for the AnaLysis of the nuclear matter Decay, the Atmosphere, the earth B-field And the Solar activity), with the intention of making a joint analysis of the data taken simultaneously with TRISTAN (TRasgo para InveSTigaciones ANtárticas), separated by a distance around 1.3×10^7 m.

This TRAGALDABAS detector is made of four planes of avalanche RCPs, but at the moment only three of them are instrumented yet. Those planes of $1.2{\times}1.5$ m^2 are placed in a range of 1.8 m high and they are made up of 120 cells each one, placed in a $30{\times}40$ array. Therefore, this device has an active area of $1.2{\times}1.5$ m^2 and covers a vertical solid angle of ~ 5 sr offering a time resolution of ~ 300 ps and track arriving angle resolution better than $3°$.

2.3 The Data Flow

The detector is taking data with coincidence trigger between planes, at a rate about 7 million of registered events per day. This analog data of coincidences is converted to digital data and it is stored, along with humidity, pressure and temperature data.

For monitoring and alerting if data it is out of expected ranges, we use a software called Nagios. It is a software that provides great versatility to consult any parameter of interest in the system. The alerts generated are received by the corresponding managers (among other means) by email, when these parameters exceed the margins defined by the network administrator.

To format the numerical data and visualize it, we use Grafana. It is a platform without ani dependency and allows creating dashboards and graphs from multiple sources.

Both applications are multi-platform open-sources, licensed under the terms of the GNU General Public License and they are accessible from the computer called Trucha

2.3.1 A PC Called Trucha

It's name cames from the trout, that is a fish. In the LabCAF, the PCs (Pe-Ce-s in spanish, fishes in english) tower computers take names of fishes.

Actually, in this PC are stored the Nagios' warnings and alerts, and defined their ranges of activation. It looks like the directory tree of the Figure 2.3.1.

To keep the code clean, readable, and manageable, each of the scripts in `/etc/nagios/scripts/` whose name begins with `sensor` parses the data from a single detector plane. Scripts that their name start with `check` call the classes defined in the previous ones for each of the functional detector planes.

Scripts in `/etc/nagios/objects/` are the configurations of the variables used for calling the later mentioned python scripts and where the limits of the alerts for Nagios are defined.

```
/etc/nagios/
├── ...
├── ...
├── scripts/
│   ├── check_allAmbient.py
│   ├── check_allPower.py
│   ├── check_all.py
│   ├── sensorHighVoltage_tragaldabas.py
│   ├── sensorHumidity_tragaldabas.py
│   ├── sensorPressure_tragaldabas.py
│   └── sensorTemperature_tragaldabas.py
└── objects/
    ├── commands.cfg
    ├── contacts.cfg
    ├── hosts.cfg
    ├── printer.cfg
    ├── services.cfg
    └── checkcommands.cfg
```

Figure 2.1: Directory tree of Trucha, where the programs for flow control are stored.

Chapter 3

Software Documentation

3.1 The soft_TT directory

Inside of soft_TT[1] directory

tenemos las listas con los archivos.hld que da Tragaldabas como output. Por ejemplo list_2019.list (que se abre con less en línea de comandos) tenemos una serie de archivos tr18234...hld

En soft_TT/testlists/ hay listas con listas en su interior guardando más nombres de archivos.hld iguales. Estos definen cuáles interesan para ejecutar en cada una.

El archivo de python multiThreadRun.py en soft_TT es el que ordena la ejecución en paralelo de los archivos.hld

3.2 Classes definitions

3.2.1 TRpcHit

TRpcHit is the class which does ... *that things* ..., and its source code is in the trpchit.cc and trpchit.h files. It has some protected variables:

- Int_t $fTrbnum$: Number of the TRB. Each plane has a TRB.
- Int_t $fCell$:
- Int_t $fCol$:
- Int_t $fRow$:
- Float_t fX: X coordinate.
- Float_t fY: Y coordinate.
- Float_t fZ: Z coordinate.
- Float_t $fTime$: Track time
- Float_t $fCharge$: **Absolute** (¿?) value of the charge.

The methods of TRpcSaeta class are:

[1]Whose full path is `/media/Datos2TB/damian/tragaldabas/soft_TT/`.

getCell

```
Int_t getCell();
```

It returns the protected variable $fCell$.

getCharge

```
Int_t getCharge();
```

It returns the protected variable $fCharge$.

getCol

```
Int_t getCol();
```

It returns the protected variable $fCol$.

getHit

```
void getHit(Int_t& trbnum, Int_t& cell,Int_t& col, Int_t& row,
        Float_t& x, Float_t& y, Float_t& z, Float_t& time,
        Float_t& charge);
```

It sets the value of the protected variables passed as parameters to the respective variables stored in memory for the detector with two planes.

- Int_t $fTrbnum$ → Int_t *trbnum*
- Int_t $fCell$ → Int_t *cell*
- Int_t $fCol$ → Int_t *col*
- Int_t $fRow$ → Int_t *row*
- Float_t fX → Float_t x
- Float_t fY → Float_t y
- Float_t fZ → Float_t z
- Float_t $fTime$ → Float_t *time*
- Float_t $fCharge$ → Float_t *charge*

getRow

```
Int_t getRow();
```

It returns the protected variable $fRow$.

getTime

```
Int_t getTime();
```

It returns the protected variable $fTime$.

getTrbnum

```
Int_t getTrbnum();
```

It returns the protected variable $fTrbnum$.

getX

```
Int_t getX();
```

It returns the protected variable fX.

getY

```
Int_t getY();
```

It returns the protected variable fY.

getZ

```
Int_t getZ();
```

It returns the protected variable fZ.

setCell

```
void setCell(Int_t num );
```

It sets the value passed to the *num* parameter to the protected variable $fCell$.

setCharge

```
void setCharge(Float_t val );
```

It sets the value passed to the *val* parameter to the protected variable *fCharge*.

setCol

```
void setCol(Int_t num );
```

It sets the value passed to the *num* parameter to the protected variable *fCol*.

setHit

```
void setHit(Int_t trbnum,Int_t  cell,Int_t  col,Int_t  row,
        Float_t x, Float_t y, Float_t z, Float_t time,
        Float_t charge);
```

It sets the value of the parameters to their respective protected variables.

- Int_t *trbnum* → Int_t *fTrbnum*
- Int_t *cell* → Int_t *fCell*
- Int_t *col* → Int_t *fCol*
- Int_t *row* → Int_t *fRow*
- Float_t *x* → Float_t *fX*
- Float_t *y* → Float_t *fY*
- Float_t *z* → Float_t *fZ*
- Float_t *time* → Float_t *fTime*
- Float_t *charge* → Float_t *fCharge*

setRow

```
void setRow(Int_t num );
```

It sets the value passed to the *num* parameter to the protected variable *fRow*.

setTime

```
void setTime(Float_t val );
```

It sets the value passed to the *val* parameter to the protected variable *fTime*.

setTrbnum

```
void setTrbnum(Int_t num );
```

It sets the value passed to the *num* parameter to the protected variable $fTrbnum$.

setX

```
void setX(Float_t val );
```

It sets the value passed to the *val* parameter to the protected variable fX.

setY

```
void setY(Float_t val );
```

It sets the value passed to the *val* parameter to the protected variable fY.

setZ

```
void setZ(Float_t val );
```

It sets the value passed to the *val* parameter to the protected variable fZ.

3.2.2 TRpcHitF

TRpcHitF is the class which does ... *that things* ..., and its source code is in the trpchitf.cc and trpchitf.h files. It has some private variables:

- TRpcCalPar *$fPar$:
- TClonesArray *$fRpcRawHits$:
- TClonesArray *$fRpcHitHits$:
- TActiveCells *$fActiveCells$:
- Int_t $totalNHits$:

The methods of TRpcHitF class are:

addRpcHit

```
TRpcHit* addRpcHit();
```

It returns the pointer to a new `(hits[totalNHits++]) TRpcHit()`.

execute

IDK [...]

init

```
Int_t init(TString filename, TString filenameActive);
```

It creates a new TRpcCalPar object called $fPar$ with name $filename$ and another new TActiveCells object called $fActiveCells$ with name $filenameActive$. Then takes `gEvent -> getRpcRawHits()` and stores it in the private variable $fRpcRawHits$.

getRpcHits

```
TClonesArray* getRpcHits();
```

It returns the private variable $fRpcHitHits$.

getRpcRawHits

```
TClonesArray* getRpcRawHits();
```

It returns `gEvent->getRpcRawHits()`.

3.2.3 TRpcSaeta

TRpcSaeta is the class which does ... *that things* ..., and its source code is in the trpcsaeta.cc and trpcsaeta.h files. All its initial values are set as -1 with the constructor. It has some protected variables:

- Float_t fX: X coordinate.
- Float_t fXP: X slope.
- Float_t fY: Y coordinate.
- Float_t fYP: Y slope.
- Float_t fZ: Z coordinate.
- Float_t $fTime$: Track time.
- Float_t $fSlow$: Slowness.
- Float_t fAl: Alpha angle.
- Float_t fBe: Beta angle.
- Float_t fGa: Gamma angle.
- Int_t $fSaN$: Saeta order.
- Int_t $find0$: Hit index.
- Int_t $find1$: Hit index.

- Int_t $find2$: Hit index.
- Float_t $fChi2$: Chi-square.

The methods of TRpcSaeta class are:

getAl

```
Float_t getAl();
```

It returns the alpha α angle in radians fAl. This is the angle between the trayectory of the incident particle and the x-axis.

getBe

```
Float_t getBe();
```

It returns the beta β angle in radians fBe. This is the angle between the trayectory of the incident particle and the y-axis.

getChi2

```
Float_t getChi2();
```

It returns the chi squared χ^2 value $fChi2$.

getGa

```
Float_t getGa();
```

It returns the gamma γ angle in radians fGaw. This is the angle between the trayectory of the incident particle and the z-axis.

getInd

```
Int_t getInd(Int_t n);
```

It returns the hit index for the n parameter:

- $n = 0 \rightarrow find0$
- $n = 1 \rightarrow find1$
- $n = 2 \rightarrow find2$

If $n \neq 0, 1, 2$, it returns -1.

getPhi

```
Float_t getPhi();
```

It returns the phi ϕ angle. This is the principal value of the arc tangent of fAl/fBe, expressed in radians.

getPhiDeg

```
Float_t getPhiDeg();
```

It returns the phi ϕ angle in degrees. This is the principal value of the arc tangent of fAl/fBe.

getRpcSaeta2Planes

```
void getRpcSaeta2Planes(Float_t& x0, Float_t& y0, Float_t& t0,
        Float_t& al, Float_t& be, Float_t& ga, Int_t& ind0,
        Int_t& ind1);
```

It sets the value of the protected variables passed as parameters to the respective variables stored in memory for the detector with two planes.

- Float_t $fX0 \rightarrow$ Float_t $x0$
- Float_t $fY0 \rightarrow$ Float_t $y0$
- Float_t $fTime \rightarrow$ Float_t $t0$
- Float_t $fAl \rightarrow$ Float_t al
- Float_t $fBe \rightarrow$ Float_t be
- Float_t $fGa \rightarrow$ Float_t ga
- Int_t $find0 \rightarrow$ Int_t $ind0$
- Int_t $find1 \rightarrow$ Int_t $ind1$

getRpcSaeta3Planes

```
void getRpcSaeta3Planes(Float_t& x0, Float_t& xP, Float_t& y0,
        Float_t& yP, Float_t& z, Float_t& t0, Float_t& sl,
        Float_t& al, Float_t& be, Float_t& ga, Int_t& san,
        Int_t& ind0, Int_t& ind1, Int_t& ind2, Float_t& chi2);
```

It sets the value of the protected variables passed as parameters to the respective variables stored in memory for the detector with three planes.

- Float_t $fX0 \rightarrow$ Float_t $x0$
- Float_t $fXP \rightarrow$ Float_t xP
- Float_t $fY0 \rightarrow$ Float_t $y0$
- Float_t $fYP \rightarrow$ Float_t yP

- Float_t fZ → Float_t z
- Float_t $fTime$ → Float_t $t0$
- Float_t $fSlow$ → Float_t sl
- Float_t fAl → Float_t al
- Float_t fBe → Float_t b
- Float_t fGa → Float_t ga
- Float_t $fSaN$ → Float_t san
- Int_t $find0$ → Int_t $ind0$
- Int_t $find1$ → Int_t $ind1$
- Int_t $find2$ → Int_t $ind2$
- Int_t $fChi2$ → Int_t $chi2$

getSaetaN

```
Float_t getSaetaN();
```

It returns the value of the saeta order $fSaN$.

getSlow

```
Float_t getX0();
```

It returns the slownes of the particle $fSlow$, which is the inverse of velocity $1/v$ (Units?).

getTheta

```
Float_t getTheta();
```

It returns the theta θ angle. This is the principal value of the arc cosine of fGa, expressed in radians.

getThetaDeg

```
Float_t getThetaDeg();
```

It returns the theta θ angle in degrees. This is the principal value of the arc cosine of fGa.

getTime

```
Float_t getTime();
```

It returns the time of the hit measured since the tracking started (Units?) $fTime$.

getX0

```
Float_t getX0();
```

It returns the X coorinate, fX (Units?).

getXP

```
Float_t getXP();
```

It returns the X slope, fXP (Units?).

getY0

```
Float_t getY0();
```

It returns the Y coorinate, fY (Units?).

getYP

```
Float_t getYP();
```

It returns the Y slope, fYP (Units?).

getZ0

```
Float_t getZ0();
```

It returns the Z coorinate, fZ (Units?).

setRpcSaeta2Planes

```
void setRpcSaeta2Planes(Float_t x0,Float_t y0,Float_t t0,
        Float_t al,Float_t be,Float_t ga,Int_t ind0,Int_t ind1);
```

It sets the value of the parameters to their respective protected variables for the detector with only two planes.

- Float_t $x0 \rightarrow$ Float_t $fX0$
- Float_t $y0 \rightarrow$ Float_t $fY0$
- Float_t $t0 \rightarrow$ Float_t $fTime$

- Float_t $al \rightarrow$ Float_t fAl
- Float_t $be \rightarrow$ Float_t fBe
- Float_t $ga \rightarrow$ Float_t fGa
- Int_t $ind0 \rightarrow$ Int_t $find0$
- Int_t $ind1 \rightarrow$ Int_t $find1$

setRpcSaeta3Planes

```
void setRpcSaeta3Planes(Float_t x0,Float_t xP,Float_t y0,
        Float_t yP,Float_t z, Float_t t0,Float_t sl, Float_t al
        Float_t be,Float_t ga, Int_t san,Int_t ind0,Int_t ind1
        Int_t ind2, Float_t chi2);
```

It sets the value of the parameters to their respective protected variables for the detector with three planes.

- Float_t $x0 \rightarrow$ Float_t $fX0$
- Float_t $xP \rightarrow$ Float_t fXP
- Float_t $y0 \rightarrow$ Float_t $fY0$
- Float_t $yP \rightarrow$ Float_t fYP
- Float_t $z \rightarrow$ Float_t fZ
- Float_t $t0 \rightarrow$ Float_t $fTime$
- Float_t $sl \rightarrow$ Float_t $fSlow$
- Float_t $al \rightarrow$ Float_t fAl
- Float_t $be \rightarrow$ Float_t fBe
- Float_t $ga \rightarrow$ Float_t fGa
- Float_t $san \rightarrow$ Float_t $fSaN$
- Int_t $ind0 \rightarrow$ Int_t $find0$
- Int_t $ind1 \rightarrow$ Int_t $find1$
- Int_t $ind2 \rightarrow$ Int_t $find2$
- Int_t $chi2 \rightarrow$ Int_t $fChi2$

setTime

```
void setTime(Float_t val);
```

It sets the value of the parameter *val* to the protected variable $fTime$.

3.2.4 TRpcSaetaF

TRpcSaetaF is the class which does ... *that things* ..., and its source code is in the trpcsaetaf.cc and trpcsaetaf.h files. It has some protected variables:

- TClonesArray *$fRpcHitHits$:
- TClonesArray *$fRpcHitCorr$:
- TClonesArray *$fRpcSaeta2Planes$:
- TClonesArray *$fRpcSaeta3Planes$:
- Int_t *totalNHits*:
- Int_t *totalNHits2Planes*:
- Int_t *totalNHits3Planes*:
- Int_t *totalNHitsCorr*:

And it also has other public variables:

- TMatrixF *InputSaeta2Planes*: the saeta vector with next values as arguments

  ```
  TMatrixF InputSaeta2Planes(Float_t x1, Float_t y1,
  Float_t t1, Float_t z1, Float_t x2, Float_t y2, Float_t t2,
  Float_t z2);
  ```

  where (x_1, y_1, z_1, t_1) are the coordinates in the first plane and (x_2, y_2, z_2, t_2) in the second one

  $$InputSaeta2Planes = \begin{pmatrix} \frac{x_2 z_1 - x_1 z_2}{Dz} \\ \frac{x_1 - x_2}{Dz} \\ \frac{y_2 z_1 - y_1 z_2}{Dz} \\ \frac{y_1 - y_2}{Dz} \\ \frac{t_2 z_1 - t_1 z_2}{Dz} \\ \frac{t_1 - t_2}{Dz} \end{pmatrix},$$

  and where Float_t $Dz = z1 - z2$. So that,

  $$InputSaeta2Planes \equiv \mathbf{S} = (x_0, x_p, y_0, y_p, t_0, s_0)$$

  is the saeta vector in the parameters space represented with respect to the origin of coordinates.

- TMatrixF $KMatrix$: the K matrix with next values as arguments

```
TMatrixF KMatrix(TMatrixF SIn, Float_t z);
```

where SIn is the input saeta vector

$$SIn = \begin{pmatrix} x_0 \\ x_p \\ y_0 \\ y_p \\ t_0 \\ s_0 \end{pmatrix},$$

and z the height of the current plane. The k value is

$$k = \sqrt{1 + x_p^2 + y_p^2},$$

so that, $KMatrix$ is

$$\begin{pmatrix} w_x & w_x z & 0 & 0 & 0 & 0 \\ w_x z & w_x z^2 + s_0^2 w_t x_p^2 z^2/k^2 & 0 & s_0^2 w_t x_p^2 z^2/k^2 & s_0 w_t x_p z/k & s_0 w_t y_p z^2 \\ 0 & 0 & w_y & w_y z & 0 & 0 \\ 0 & s_0^2 w_t x_p^2 z^2/k^2 & w_y z & w_y z^2 + s_0^2 w_t x_p^2 z^2/k^2 & s_0 w_t y_p z/k & s_0 w_t y_p z^2 \\ 0 & s_0 w_t x_p z/k & 0 & s_0 w_t y_p z/k & w_t & w_t k z \\ 0 & s_0 w_t y_p z^2 & 0 & s_0 w_t y_p z^2 & w_t k z & w_t k^2 z^2 \end{pmatrix}$$

- TMatrixF $AVector$: constructs the **measurement** (¿?) vector for a non-linear model with next values as arguments

```
TMatrixF AVector(TMatrixF SIn, Float_t x, Float_t y,
Float_t t, Float_t z);
```

where (x, y, z, t) are the measured coordinates and SIn and k have the same form as in TMatrixF $KMatrix$. So $AVector$ is

$$AVector = \begin{pmatrix} w_x x \\ z\dfrac{w_x x k^2 + s_0 w_t x_p (tk + s_0(x_p^2 + y_p^2)z)}{k^2} \\ w_y y \\ z\dfrac{w_y y k^2 + s_0 w_t y_p (tk + s_0(x_p^2 + y_p^2)z)}{k^2} \\ w_t \dfrac{tk + s_0(x_p^2 + y_p^2)z}{k} \\ w_t(tk + s_0(x_p^2 + y_p^2)z)z \end{pmatrix}.$$

The methods of TRpcSaetaF class are:

addRpcHit

```
TRpcHit* addRpcHit();
```

It returns the pointer *rpchit*, which points to a new `(hits[totalNHitsCorr++]) TRpcHit()`, the next hit.

addRpcSaeta2Planes

```
TRpcSaeta* addRpcSaeta2Planes();
```

It returns the pointer *RpcSaeta*, which points to a new `(RpcSaeta2Planes [totalNHits2Planes++]) TRpcSaeta()`, the next [...].

addRpcSaeta3Planes

```
TRpcSaeta* addRpcSaeta3Planes()
```

It returns the pointer *RpcSaeta*, which points to a new `(RpcSaeta3Planes [totalNHits3Planes++]) TRpcSaeta()`, the next [...].

execute

```
Int_t execute();
```

The main function of the class. It makes the loops for finding hits in planes and stores them on Ttrees.

getRpcHits

```
TClonesArray* getRpcHits() {return gEvent->getRpcHits(); }
```

[...]

getRpcHitsCorr

```
ClonesArray* getRpcHitCorr() {return fRpcHitCorr; }
```

[...]

getRpcSaeta2Planes

```
ClonesArray* getRpcSaeta2Planes() {return fRpcSaeta2Planes; }
```

It returns the private $fRpcSaeta2Planes$ pointer.

getRpcSaeta3Planes

```
ClonesArray* getRpcSaeta3Planes() {return fRpcSaeta3Planes; }
```

It returns the private $fRpcSaeta3Planes$ pointer.

init

```
Int_t init();
```

It initializes the variables $fRpcHitCorr$, $fRpcSaeta2Planes$ and $fRpcSaeta3Planes$ if they are not defined by TClonesArray, creating the new ones.

3.2.5 TTMatrix

TTMatrix is the class which does ... *that things* ..., and its source code is in the ttmatrix.cc and ttmatrix.h files. It has some protected variables:

- Float_t $fX1$: X coordinate in the first plane
- Float_t $fY1$: Y coordinate in the first plane
- Float_t $fT1$: time in the first plane
- Float_t $fZ1$: Z coordinate in the first plane
- Float_t $fX2$: X coordinate in the second plane
- Float_t $fY2$: Y coordinate in the second plane
- Float_t $fT2$: time in the second plane
- Float_t $fZ2$: Z coordinate in the second plane
- Float_t fwx: deviation of cell shape in x direction
- Float_t fwy: deviation of cell shape in y direction
- Float_t fwz: ¿?

And it also has other public variables:

- TMatrixF $AVector$: is the vector which ... *do things* ... ¿Is it the **measurement** vector?

  ```
  TMatrixF AVector(TMatrixF SIn, Float_t x, Float_t y,
          Float_t t, Float_t z);
  ```

where SIn is the input saeta vector

$$SIn = \begin{pmatrix} x_0 \\ x_p \\ y_0 \\ y_p \\ t_0 \\ s_0 \end{pmatrix},$$

and (x, y, z, t) the measured coordinates and time. The k value is

$$k = \sqrt{1 + x_p^2 + y_p^2},$$

so that, $AVector$ is

$$AVector = \begin{pmatrix} w_x x \\ z\dfrac{w_x x k^2 + s_0 w_t x_p(tk + s_0(x_p^2 + y_p^2)z)}{k^2} \\ w_y y \\ z\dfrac{w_y y k^2 + s_0 w_t y_p(tk + s_0(x_p + y_p)z)}{k^2} \\ w_t\dfrac{tk + s_0(x_p + y_p)z}{k} \\ w_t z(tk + s_0(x_p + y_p)z) \end{pmatrix}.$$

Here the w_x, w_y, w_z, w_t variables are initializated to zero all of them when $AVector$ is created.

- TMatrixF $KMatrix$: is the vector which ... *do things* ... ¿Is it the **measurement** vector?

```
TMatrixF KMatrix(TMatrixF SIn, Float_t z);
```

where SIn is the input saeta vector

$$SIn = \begin{pmatrix} x_0 \\ x_p \\ y_0 \\ y_p \\ t_0 \\ s_0 \end{pmatrix},$$

and (x, y, z, t) the measured coordinates and time. The k value has the same shape as in the $AVector$. So, the $KMatrix$ is

$$\begin{pmatrix} w_x & w_x z & 0 & 0 & 0 & 0 \\ w_x z & w_x z^2 + s_0^2 w_t x_p^2 z^2/k^2 & 0 & s_0^2 w_t x_p^2 z^2/k^2 & s_0 w_t x_p z/k & s_0 w_t y_p z^2 \\ 0 & 0 & w_y & w_y z & 0 & 0 \\ 0 & s_0^2 w_t x_p^2 z^2/k^2 & w_y z & w_y z^2 + s_0^2 w_t x_p^2 z^2/k^2 & s_0 w_t y_p z/k & s_0 w_t y_p z^2 \\ 0 & s_0 w_t x_p z/k & 0 & s_0 w_t y_p z/k & w_t & w_t k z \\ 0 & s_0 w_t y_p z^2 & 0 & s_0 w_t y_p z^2 & w_t k z & w_t k^2 z^2 \end{pmatrix},$$

with w_x, w_y, w_z, w_t initializated as zero.

- TMatrixF *InputSaeta2Planes*: the saeta vector with next values as arguments

```
TMatrixF InputSaeta2Planes(Float_t x1, Float_t y1,
Float_t t1, Float_t z1, Float_t x2, Float_t y2, Float_t t2,
Float_t z2);
```

where x_1, y_1, z_1, t_1 are the coordinates in the first plane and x_2, y_2, z_2, t_2 in the second one. This is a function that returns

$$InputSaeta2Planes = \begin{pmatrix} \dfrac{x_2 z_1 - x_1 z_2}{z_1 - z_2} \\ \dfrac{x_1 - x_2}{z_1 - z_2} \\ \dfrac{y_2 z_1 - y_1 z_2}{z_1 - z_2} \\ \dfrac{y_1 - y_2}{z_1 - z_2} \\ \dfrac{t_2 z_1 - t_1 z_2}{z_1 - z_2} \\ \dfrac{t_1 - t_2}{z_1 - z_2} \end{pmatrix},$$

and assigns the parameters $x_1, y_1, z_1, t_1, x_2, y_2, z_2, t_2$ to the protected variables of the calss $fX1, fY1, fZ1, fT1, fX2, fY2, fZ2, fT2$ respectively.

3.2.6 Unpacker

Unpacker is the class which does ... *that things* ..., and its source code is in the trpcunpacker.cc and trpcunpacker.h files.

Unpacker has some protected variables:

- HldEvent* *pEvent*: current event read from file.
- Int_t *EventNr*: event Counter.
- Int_t *EventLimit*: maximum event number per file.
- Int_t $subEvtId$:
- TFile* *pRootFile*: pointer to TFile with the output tree.
- std::string $inputFile$: wk 28.05
- std::string $outputFile$: wk 28.05
- Int_t *fpga_code*: address of the data source (e.g. given fpga) decoded from hld file.
- Int_t $refCh$:

*** The Constructors**

```
Unpacker(const char* dir,const char* name, const char* odir,
        Int_t nEvt, TString luptab, TString calpar);
```

Using this construtor of unpacker we have access the data output of the tracking code. By executing this code into the ROOT command line we can check whether or not what we rebuild is compatible with the current methods.
Parameters:

- const char* *dir*: Directory of input data, which is the output of Tragaldabas.
- const char* *name*: Name of the file to unpack, (i. e. `tr18249041152.hld`.)
- const char* *odir*: Output directory, here will appear the unpacked files.
- Int_t *nEvt*: Number of events (i. e. 1000).
- TString *luptab*: Name of the luptable txt file (i. e. `luptable_corr_20180423.txt`).
- TString *calpar*: Name of the CalPar txt file (i. e. `2018_day_203_CalPars.txt`).

The methods of the Unpacker class are:

eventLoop

```
Bool_t eventLoop(Int_t NbEvt=50000,Int_t startEvt=0);
```

Loop over all events, data is written to the root tree.

- Int_t $NbEvt$:
- Int_t $startEvt$:

eventLoopFillCal

```
Bool_t eventLoopFillCal(Int_t nbEvt, Int_t startEv, TH1D** hq,
        TH1D** hdt, TH1D** hdt2);
Bool_t eventLoopFillCal(Int_t nbEvt, Int_t startEv, TH1D** h1D,
        TH2D** h2D, TH3D** h3D);
```

Loop over all events, data written to the root tree

- Int_t $NbEvt$:
- Int_t $startEvt$:
- TH1D** hq:
- TH1D** $h1D$:
- TH1D** hdt:
- TH2D** $h2D$:
- TH1D** $hdt2$:
- TH3D** $h3D$:

eventLoopSyncCheck

```
Int_t eventLoopSyncCheck(Int_t nbEvt,Int_t startEv);
```

Loop over all events, data is written to the root tree.

- Int_t $NbEvt$:
- Int_t $startEvt$:

fillCalibration

```
void Unpacker::fillCalibration(const char* dir, TString list,
        const char* odir,const char* ofile, Int_t nEvt,Int_t n);
```

- const char* dir: path to input file.
- TString $list$: list of `files.hld`.
- const char* $odir$: path to output directory.
- const char* $ofile$: path to output file.
- Int_t $nEvt$: number of events.
- Int_t n
  - $n = 0$ (standard mode): just calculate the pedestals and exchande in the parameter file which is previously declared.
  - $n = 1$ (special mode): create pedestals and set time offsets to 0.

fillHistograms

```
void fillHistograms(const char* dir, TString list,
        const char* odir, const char* ofile, Int_t nEvt,
        Int_t n,Int_t n2);
```

- const char* dir
- TString $list$
- const char* $odir$
- const char* $ofile$
- Int_t $nEnv$
- Int_t n
- Int_t $n2$

getFileHitFinderPar

```
void getFileHitFinderPar( void );
```

It gets the variable TString *fileHitFinderPar*, the name of the file with hit finder params.

getFileHitFinderParOut

```
void getFileHitFinderParOut( void );
```

It gets the variable TString *fileHitFinderParOut*, the name of the file with hit finder params.

getFileLookupPar

```
void getFileLookupPar( void );
```

It gets the variable TString *fileLookupPar*, the name of the file with lookup params.

getFileTrackFinderPar

```
void getFileTrackFinderPar( void );
```

It gets the variable TString *fileTrackFinderPar*, the name of the file with track finder params.

getpEvent

```
HldEvent* getpEvent(void);
```

It returns the pointer HldEvent* *pEvent*.

getEventLimit

```
Int_t getEventLimit()
```

It returns the Int_t *EventLimit*.

getEventNr

```
Int_t getEventNr()
```

It returns the Int_t $EventNr$.

HexStrToInt

```
UInt_t HexStrToInt(const char* str) {
        UInt_t t;
        std::stringstream s;
        s << std::hex << str;
        s >> t;
        return t;
}
```

It is easy to see that `HexStrToInt` takes a const char* *str* as argument and returns its integer value as UInt_t *t*.

setInputFile

```
std::string setInputFile(const char* filename);
std::string setInputFile(const char* dir,const char* filename);
```

It sets $filename$ as name of the **`filename.hld`** input file and returns a std::string *inputFile* (*wk 28.05*). If a directory *dir* is passed as parameter, the location for **`filename.hld`** is changed to *dir*.

setFileHitFinderPar

```
void setFileHitFinderPar( TString fileName);
```

It sets $fileName$ to the variable TString $fileHitFinderPar$, the name of the file with hit finder params.

setFileHitFinderParOut

```
void setFileHitFinderParOut( TString fileName);
```

It sets $fileName$ to the variable TString $fileHitFinderParOut$, the name of the file with hit finder params.

setFileLookupPar

```
void setFileLookupPar( TString fileName);
```

It sets $fileName$ to the variable TString $fileLookupPar$, the name of the file with lookup params.

setFileTrackFinderPar

```
void setFileTrackFinderPar( TString fileName);
```

It sets *fileName* to the variable TString *fileTrackFinderPar*, the name of the file with track finder params.

setpEvent

```
Bool_t setpEvent(Int_t subId);
void setpEvent(HldEvent* evt);
```

It sets *pEvent* (the current event read from file) by reading hld file, where *subId* is the subevent id and returns *kTRUE*. If parameter is a pointer, pEvent is set as *evt* and doesn't returns anything.

setRootFile

```
Bool_t setRootFile(const char* filename);
```

It sets *filename* as name to a new root output file and returns the ROOT specific constant *kTRUE*.

syncCheck

```
Int_t syncCheck(const char* dir, TString file,
        Int_t nEvt, Int_t n);
```

- const char* *dir*
- TString *file*
- Int_t *nEnv*
- Int_t *n*
  - $n = 0$ (standard mode): just calculate the pedestals and exchande in the parameter file which is previously declared.
  - $n = 1$ (special mode): create pedestals and set time offsets to 0.

Chapter 4

Kalman Filter

The Kalman filter method is intended for finding the optimum estimation $\mathbf{r}$ of the unknown vector $\mathbf{r}^t$, which describes the SAETA[1], according to the measurements $\mathbf{m}_k$, $k = 1...n$ of the vector $\mathbf{r}^t$.

The Kalman filter starts with a certain initial approximation $\mathbf{r} = \mathbf{r}_0$ and refines the vector $\mathbf{r}$, consecutively adding one measurement ater the ohter. The optimum value is attained after the addition of the last measurement.

Like it is seen in table 4.1, the upper plane T1 has first index and its height is 1.8 m, while lower plane T4 has the latest and it is at ground. So that, since we are starting Kalman filter by the lowest plane, $\mathbf{r}_0 \equiv \mathbf{r}_4$, and k indices go from $k = 4$ to $k = 1$

Name	Height / mm	Index
T1	1800	0
T2	900	1
T3	600	2
T4	0	3

Table 4.1: The four planes of TRAGALDABAS.

The vector $\mathbf{r}^t$ can change from one measurement to the next

$$\mathbf{r}^t = F_k \mathbf{r}^t_{k+1} + \boldsymbol{\nu}_k \tag{4.1}$$

where F_k is a linear operator, $\boldsymbol{\nu}_k$ is a process noise between $(k-1)$-th and k-th measurements

The measurement $\mathbf{m}_k$ linearly depends won $\mathbf{r}^t_k$:

$$\mathbf{m}_k = H_k \mathbf{r}^t_k + \boldsymbol{\eta}_k, \tag{4.2}$$

where $\boldsymbol{\eta}_k$ is an error of the k-th measurement.

It is assumed that measurement errors $\boldsymbol{\eta}_i$ and the process noise $\boldsymbol{\nu}_i$ are uncorrelated, inbiased ($\langle \boldsymbol{\eta}_i \rangle = \langle \boldsymbol{\nu}_i \rangle = \mathbf{0}$) and those covariance matrices V_k, Q_k are

[1]Te particle is defined completely by a set of parameters $\mathbf{r}_k = (x_k, x'_k, y_k, y'_k, t_k, 1/v)^T$. We call it **SAETA** (SmAllest sET of pArameters).

knownw:

$$\langle \boldsymbol{\eta}_i \cdot \boldsymbol{\eta}_i^T \rangle \equiv V_i,$$
$$\langle \boldsymbol{\nu}_j \cdot \boldsymbol{\nu}_j^T \rangle \equiv Q_j. \tag{4.3}$$

The Kalman filter starts with an initial hypothetical vector $\mathbf{r_0}$, then for each measurement $\mathbf{m}_k$ a vector $\mathbf{r}_k$ is calculated, wwhich is the optimum estimation of the vector $\mathbf{r}^t$ according to the first k measurements.

The conventional Kalman filter algorithm consists of four stages:

1. Initialization step: Choose an appropieate value of the vector $\mathbf{r}_0$. We use an hypothetical normal one

$$\mathbf{r}_0 = (x_0, 0, y_0, 0, t_0, sc)^T, \tag{4.4}$$

where x_0, y_0, t_0 are the coordinates of hit in the T4 plane, $sc = 1/c$ the slowness (inverse of light celerity), and $x' = y' = 0$ the projections on x and y axes respectively:

$$x' = \frac{cx}{cz} = \frac{\sin\theta\cos\phi}{\cos\theta},$$
$$y' = \frac{cy}{cz} = \frac{\sin\theta\sin\phi}{\cos\theta}, \tag{4.5}$$

in spherical coordinates.

Its covariance matrix is set to $C_0 = I \cdot \mathrm{inf}^2$, where inf denotes a large positive number. We used:

$$C_0 = \begin{pmatrix} \mathrm{SIGX}^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathrm{VSLP} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathrm{SIGY}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathrm{VSLP} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathrm{SIGT}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathrm{VSLN} \end{pmatrix}, \tag{4.6}$$

where VSLP $= 0.1^2$ and VSLN $= 0.01^2$ are the variances for slope and slowness respectively, and

$$\mathrm{SIGX} = \frac{1}{\sqrt{12}}\mathrm{WCX},$$
$$\mathrm{SIGY} = \frac{1}{\sqrt{12}}\mathrm{WCY},$$
$$\mathrm{SIGT} = 300 \text{ ps}, \tag{4.7}$$

the variances for cell and time dimensions, with WCX = 125 mm and WCY = 120 mm the width of cells on x and y axes respectively.

2. Prediction step: Propagate the vector from $(k+1)$-th to k-th plane by propagation matrix

$$F_k = \begin{pmatrix} 1 & dz_k & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & dz_k & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & ks_k dz_k \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.8}$$

where dz_k is the distance between k-th and the plane $(k+1)$-th below

$$dz_k = z_k - z_{k+1}, \tag{4.9}$$

and ks_k is

$$ks_k = \sqrt{1 + x_k'^2 + y_k'^2}, \tag{4.10}$$

so that,

$$\begin{aligned} \tilde{\mathbf{r}}_k &= F_k \mathbf{r}_{k-1}, \\ \tilde{C}_k &= F_k C_{k-1} F_k^T. \end{aligned} \tag{4.11}$$

3. Process noise: In contrast to the prediction step, describing deterministic changes of the vector $\mathbf{r}^t$ in time, the process noise describes probabilistic deviations of the vector $\mathbf{r}^t$.

$$\begin{aligned} \hat{\mathbf{r}}_k &= \tilde{\mathbf{r}}_k, \\ \hat{C}_k &= \tilde{C}_k + Q_k. \end{aligned} \tag{4.12}$$

4. Filtration step: At this step the state vector $\hat{\mathbf{r}}_k$ is updated with the new mweasurement $\mathbf{m}_k$ to get the optimal estimate of $\mathbf{r}_k$ and its covariance matrix C_k:

$$\begin{aligned} K_k &= \hat{C}_k H_k^T (V_k + H_k \hat{C}_k H_k^T)^{-1}, \\ \mathbf{r}_k &= \hat{\mathbf{r}}_k + K_k(\mathbf{m}_k - H_k \hat{\mathbf{r}}_k), \\ C_k &= \hat{C}_k - K_k H_k \hat{C}_k, \\ \chi_k^2 &= \chi_{k+1}^2 + (\mathbf{m}_k - H_k \hat{\mathbf{r}}_k)^T (V_k + H_k \hat{C}_k H_k^T)^{-1} (\mathbf{m}_k - H_k \hat{\mathbf{r}}_k). \end{aligned} \tag{4.13}$$

Here, the k-th measurement is

$$\mathbf{m}_k = (x_k, y_k, t_k)^T, \tag{4.14}$$

and its covariance matrix

$$V_k = \begin{pmatrix} \text{SIGX}^2 & 0 & 0 \\ 0 & \text{SIGY}^2 & 0 \\ 0 & 0 & \text{SIGT}^2 \end{pmatrix}. \tag{4.15}$$

The matrix H_k of the model is simply the identity matrix

$$H_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \tag{4.16}$$

which converts every $\hat{\mathbf{r}}_k = (x_k, x_k', y_k, y_k', t_k, sc)^T$ into $\mathbf{m}_{\mathbf{r}k} = (x_k, y_k, t_k)^T$ for comparing $\delta \mathbf{m}_k = \mathbf{m}_k - \mathbf{m}_{\mathbf{r}k}$ in (4.13).

The following designations are used in Eqs. (4.11)-(4.13): $\mathbf{r}_{k+1}$, C_{k+1} are the optimum estimation, obtained at the previous step and the error covariance matrix; the matrix F_k relates the state at step $k+1$ to the state at step k; [2] $\tilde{\mathbf{r}}_k$, $\tilde{C}_k$ are predicted estimation of $\mathbf{r}_k^t$ before the process noise; $\hat{\mathbf{r}}_k$, $\hat{C}_k$ are predicted estimation of $\mathbf{r}_k^t$ after the process noise; $\mathbf{m}_k$, V_k are the k-th measurement and its covariance matrix; the matrix H_k is the model of measurement; the matrix K_k is the so-called gain matrix; the value χ_k^2 is the total χ^2-deviation of the obtained estimation $\mathbf{r}_k$ from the measurements $\mathbf{m}_1, ...\mathbf{m}_k$.

The vector $\mathbf{r}_n$ obtained after the filtration of the last measurement is the desired optimal estimation of the $\mathbf{r}_n^t$ with the covariance matrix C_n.

In track fitting applications, the state vector $\mathbf{r}_k$ is vector of the track parameters, the prediction matrix F_k dewscribes extrapolation of the track in the magnetic field from one detector to another, and the matrix of noise Q_k describes the effect of multiple scattering in the material.

[2] Remember that we start from the lowest plane to de highest.