Time Resolution of CMS Muon Barrel Spectrometer

Abstract

The CMS muon spectrometer can be used to search for Heavy Stable Charged Particles (HSCPs) which might signal physics beyond the Standard Model. Such particles can be distinguished from Standard Model particles by exploiting their unique signature: a low velocity, \( \beta \), associated with a high momentum of order a few hundred \( \text{GeV}/c \). It’s therefore interesting to study time and \( \beta \) resolution of CMS muon barrel spectrometer, which instead has been designed to reveal particles travelling at \( \beta \sim 1 \). To that purpose the analysis focuses on drift tubes, which are used to reveal ionizing particles and allow, knowing their relative displacement, to measure the particle’s time of flight. The problem is then modelled using a Monte Carlo simulation. This simulation allows to estimate separately the different contributions to \( \beta \) resolution: drift tubes resolution, knowledge of wire positions inside the tubes, and approximations used when modelizing the drift velocity.

1 Drift Chambers

A fast charged particle, traversing a gaseous medium, can interact with it in many ways. As far as we are interested in its detection however, only the electromagnetic interaction concerns us, being many orders of magnitude more probable than strong or weak ones and therefore leaving a signature even in thin samples of gas. For example, a charged particle travelling at \( \beta \sim 1 \) will ionize about 120 gas molecules in a cm of Argon, at normal conditions.

The very basis of drift chambers operation lies in the Coulomb interaction between the electromagnetic fields of the incoming charged particle and of the medium itself, causing both ionization and atomic excitation. The contribution of other electromagnetic processes such as bremsstrahlung, Cerenkov and transition radiation is negligible to the total energy loss, for particles heavier than electrons.

1.1 Ionization

A good expression for the average differential energy loss a charged particle suffers for coulomb interaction (loss per unit length) has been obtained by Bethe and Bloch (see [1, p. 217]):
\[-\frac{dE}{dx} = \frac{4\pi}{m_e c^2} \frac{nz^2}{\beta^2} \left( \frac{e^2}{4\pi\epsilon_0} \right)^2 \ln \left( \frac{2m_e c^2 (\beta\gamma)^2}{I} \right) - \beta^2 \] (1)

Where $E$ is the energy of the particle, $ze$ is the particle charge ($e$ charge of the electron), $m_e$ the rest mass of the electron, $n$ the electron density of the gas ($n = \frac{N_A Z \rho}{A}$) and $I$ its ionization potential.

The energy loss for coulomb interactions is therefore independent of the incident particle mass, and for ultrarelativistic particles is proportional to $\ln(\beta\gamma)$. Its trend can be summarized as follows (see Figure 1):

Figure 1: Differential energy loss as a function of the particle energy for the electron, muon, pion, proton and deuterium.

For slow projectiles there is a fast decrease as a function of velocity, dominated by the $\frac{1}{\beta^2}$ term. The energy loss reaches a constant value around $\beta \approx 0.97$, and eventually slowly increases for $\beta \rightarrow 1$ with the logarithmic term, for the "relativistic rise".

Muons produced at LHC that reach drift chambers have generally momentum greater than $\sim 500 MeV/c$, so are minimum ionizing particles. Very few have so large momentum ($\geq 200 GeV/c$) to lose energy by Bremsstrahlung.

Charges (ions and electrons) produced by the crossing of an ionizing particle quickly lose their energy in multiple collisions with the gas molecules, and assume the average thermal energy distribution of the gas $E_T = \frac{3}{2} kT \approx 0.02 \text{ eV}$ at normal conditions, corresponding to a speed $v \approx 85 \cdot 10^4 \text{ m/s}$. This swarm of charged particles will diffuse by multiple collisions following a Gaussian law:

\[ \frac{dN}{N} = \frac{1}{\sqrt{4\pi Dt}} e^{-\frac{(x^2)}{4Dt}} \] (2)

\[ ^1\text{This and many other figures are taken from [2]} \]
where \( dN/N \) is the fraction of charges found in the element \( dx \) at a distance \( x \) from the origin after a time \( t \), \( D \) denotes the diffusion coefficient. The standard deviation of this distribution is given by \( \sigma_x = \sqrt{2Dt} \) for a linear, and \( \sigma_v = \sqrt{6Dt} \) for a volume diffusion. For example, the volume spread of ions in Argon after 1 \( \mu\text{sec} \) is 5\( \mu\text{m} \).

1.2 Electron drift

When an electric field is applied across the gas volume, a net movement of the electrons along the field direction is observed. The velocity of this average motion (not to be confused with the instant velocity \( \sim 10^5 \text{ m s}^{-1} = 10 \text{ cm/\mu s} \) for electrons at normal conditions) is called drift velocity. In a simple formulation, recalling that the average displacement along the field direction between two collisions is \( \frac{e}{2m}E\tau^2 \), one can write the drift velocity as

\[
 w = \frac{e}{2m}E\tau \quad (3)
\]

where \( \tau \) is the mean time between collisions, in general a function of the electric field \( E \). It has been found that the cross section for collisions between electrons and gas molecules varies for some mixtures very strongly with \( E \), causing \( \tau \) to vary with it. This is consequence of the fact that electrons can increase their energy substantially between consecutive collisions. Their wavelength can then approach the dimensions of electron shells of the molecules, and complex quantum-mechanical effects take place (Ramsauer effect, see [4]). In Figure 2 is shown the drift velocity as a function of electric field, for the gas mixture chosen in CMS drift tubes. It is clear from the picture that a regime change in drift velocity dependance from field strenght occurs around 1 kV/cm. Between 1 and 2 kV/cm \( w \) has a plateau, being almost constant at \( \sim 5.5 \text{ cm/\mu sec} \). It is very convenient therefore to operate the chamber in this region, because drift velocity wouldn’t change with small variation of the field intensity.

The presence of a magnetic field at right angles with the electric one modifies the drift properties of the swarm of electrons. The Lorentz force applied to each moving charge transforms the small segment of motion between two collisions into circular trajectories, and also modifies the energy distribution. The net effect is a reduction of the drift velocity, at least at low electric field, and a movement of the swarm along a direction different from the field lines. The complete formula can be found on [1, p. 239], in our case it reduces to:

\[
 w_B = \frac{w}{\sqrt{1 + \omega^2\tau^2}}
\]

\[
 \tan(\alpha_B) = \omega\tau
\]
Figure 2: Calculated drift velocity as a function of the electric field, for the gas mixture of Ar and CO₂ (85%/15%) chosen in CMS drift tubes. Values obtained are very similar for the two different gas pressures, 730 and 770 mmHg, shown in the figure. Looking at the curve, inverting equation 3, we obtain values of $\tau$ ranging from $\sim 63$ nsec for low fields (0 to 1 kV/cm), down to $\sim 8$ nsec at $E \simeq 10$ kV/cm.

where $\omega$ is the angular velocity associated with the magnetic field, $\omega = \frac{eB}{m}$.

The angle $\omega\tau$ is the average deflection (on a plane at right angles with B) caused by the magnetic field on a particle between two collisions. Its numerical value gives an estimate of the effect the magnetic field has on drifting properties.

1.3 Avalanche production

Increasing the electric field above a few kV per cm, more and more electrons can receive enough energy between two collisions to produce inelastic phenomena, excitation of various kind, and ionization.

When the energy of an electron increases over the first ionization potential of the gas $E_i$, which is 15.6 eV for the Argon, a ion pair can be released after an impact with a gas molecule. The hitting electron continues its trip: this process is the basis of the avalanche multiplication of charges in drift chambers. Consider an electron liberated in a region of uniform (high) electric field. After a mean free path for ionization (roughly of the same order of the classical mean free path, depending on gas mixture) an
electron-ion pair will be produced, and two electrons will continue the drift to generate two other ion pairs and so on.

![Figure 3: Drop-like shape of an avalanche, showing the positive ions left behind the fast electron front. The photograph shows the actual avalanche shape, as made visible in a cloud chamber by droplets condensing around ions.](image)

Given the big difference in the drift velocity of ions and electrons, about a factor of thousand\(^2\), and the diffusion of migrating charges in the gas, the following picture of an avalanche multiplication appears (see Figure 3): at a given instant, all electrons are situated in the front of a drop-like distribution of charges, with a tail of positive ions behind, decreasing in number and lateral extension; half of the total ions are contained in the front part, since they have been produced in the last mean free path.

The multiplication factor cannot be increased at will. Secondary processes, like photon emission inducing the generation of avalanches spread over the gas volume, and deformation of the electric field (which is strongly increased near the front of the avalanche), eventually result in a spark breakdown. There is therefore a practical limit for gas chamber gain, at a value around \(10^6\), if one wants to avoid breakdowns.

### 1.4 Geometry

Keeping in mind the above considerations on drift velocity and avalanche multiplication, we consider their basic application in a drift chamber.

A minimum ionizing particle travelling through a thin layer of gas, between two flat electrodes (for example 1 cm of argon at normal conditions), will release about 120 ion pairs. If this charge is collected at one electrode,

\[^2\text{This can be seen as a consequence of the ratio between the masses of electrons and ions, using (3). In fact, the constant of proportionality } w = \mu E \text{ is called mobility, and for the gas mixture chosen in CMS is } \mu \approx 1 \div 2 \, \text{cm}^2 \, \text{V}^{-1} \, \text{sec}^{-1}.\]
the detected signal will be \( V = \frac{120\epsilon}{C} \). Recalling that \( C = \frac{4\pi}{d} \), with the geometry described above (and assuming \( \epsilon \sim \epsilon_0 \)) a 100 cm\(^2\) surface gives a typical 10 pF capacitance. This leads to a detected signal \( V_{\text{detected}} \sim 2 \mu\text{Volt} \), far below any possibility of detection.

If however a strong electric field is applied between the electrodes, avalanche multiplication can occur, boosting the signal amplitude by several orders of magnitude. This kind of parallel plane detector has, however, severe limitations, due principally to the uniform (high) value of the electric field over the volume. In fact, secondary processes like photon-induced avalanches would be far too probable. If many avalanches are produced at the same time in different places, some could connect to each other, and eventually result in a spark breakdown. This means that only moderate gains can be obtained before losing the chamber functionality.

A cylindrical coaxial geometry allows to overcome such limitation. The two electrodes are a thin metal wire stretched on the axis (positive) and, insulated from it, the conducting cylinder (negative). The electric field of the system has its maximum at the surface of the anode wire and rapidly decrease, as \( r^{-1} \), towards the cathode. Calling \( a \) and \( b \) the wire and cylinder radii, and \( V_0 \) the voltage, the field (pointing radially outward) can be expressed as:

\[
E(r) = \frac{V_0}{ln(b/a)} \frac{1}{r}
\]

Using thin wires, very high values of the field can be obtained close to the anode, while remaining moderate (i.e. not causing unwanted breakdowns) in the rest of the chamber. In most of the region where the charges are produced by the primary interaction processes, the electric field only makes electrons drift towards the anode and, of course, positive ions towards the cathode. But very close to the anode, normally at a few radii, the field gets strong enough so that multiplication starts. Because of lateral diffusion and the small radius of the anode, the avalanche surrounds the wire as shown in Figure 4; electrons are collected and positive ions (half of them produced in the last mean free path) begin to drift towards the cathode.

This geometry has however some limitations: its external circular section doesn’t allow close packing of singular drift tubes to form more complex and precise detectors (with some exception: at ATLAS this has been the choice, because stiffness was a major concern). This can be easily resolved: the main idea behind cylindrical geometry is the anode wire. It generates in its neighborhood a field high enough to start an avalanche multiplication every time an electron drifts towards it, while in the rest of the chamber the field can be kept low at will. So the cylindrical cathode can be replaced with one of any other shape, smooth enough to leave the field unchanged locally around the wire, and such that far form the wire the field doesn’t exceed
Figure 4: Time development of an avalanche in a drift chamber. A single primary electron proceeds towards the anode, in regions of increasingly high fields, experiencing ionizing collisions. A drop-like avalanche, surrounding the wire, develops. Electrons are collected in a very short time (∼ 1 nsec) and a cloud of positive ions is left, slowly migrating towards the cathode.

a few kV/cm (to prevent secondary avalanches production, and ultimately breakdowns). The design chosen in CMS is shown in Figure 7.

1.5 Time development of the signal

As shown in Figure 4, the whole process of charge multiplication begins at a few wire radii (typically at less than 50 µm from the anode surface), and take place in about 1 nsec. We are interested in the detected signal, negative on the anode and positive on the cathode, which is consequence of the change in energy of the system due to the movement of the charges. If a charge \( Q \) is moved for \( dr \) inside a capacitor (\( V \) is its potential), the expression of the change in energy is \( dE = Q \frac{dV}{dr} dr \). So, if we want energy to be conserved, the signal \( dV \) induced on the electrodes is, calling \( C \) the total capacitance of the chamber and \( V_0 \) its voltage:

\[
dV = \frac{Q}{CV_0} \frac{dV}{dr} dr
\]

Assuming that all charges are produced at a distance \( \lambda \) from the wire (whose radius is \( a \)), in a cylindrical drift chamber of radius \( b \), the electron and ion contribution to the signal on the anode will be respectively:

\[
v^- = -\frac{Q}{CV_0} \int_a^{a+\lambda} \frac{dV}{dr} dr = -\frac{Q}{2\pi\varepsilon_0 l} \ln\left(\frac{a + \lambda}{a}\right)
\]

\[
v^+ = \frac{Q}{CV_0} \int_{a+\lambda}^{b} \frac{dV}{dr} dr = -\frac{Q}{2\pi\varepsilon_0 l} \ln\left(\frac{b}{a + \lambda}\right)
\]

The ratio of the two contributions is:

\[
\frac{v^-}{v^+} = \frac{\ln(a + \lambda) - \ln(a)}{\ln(b) - \ln(a + \lambda)}
\]
Typical values for a counter are $a = 10\ \mu m$, $\lambda = 1\ \mu m$ and $b = 10\ mm$. The previous equation then yields that the electron contribution to the signal is about 1% of the total, and can therefore be neglected. The time development of the signal (only positive ions) can then be calculated assuming that ions leaving the surface of the wire drift to the cathode with constant mobility. The time expression of the signal is:

$$v(t) = -\int_0^t dv = -\frac{Q}{2\pi\epsilon_0} \ln\left(\frac{r(t)}{a}\right)$$

Recalling that for ions $\frac{dr}{dt} = \mu E = \frac{a V_0}{ln(b/a)} t$, we have $r(t) = \left(a^2 + \frac{2a V_0}{ln(b/a)} t\right)^{1/2}$. Substituting this expression for $r(t)$ in the previous equation we obtain:

$$v(t) = -\frac{Q}{4\pi\epsilon_0} \ln\left(1 + \frac{2\mu V_0}{a^2 \ln(b/a)} t\right)$$

In practice, the growth of this signal is very fast at the beginning. It is therefore normal practice to terminate the counter with a resistance $R$, such that the signal is differentiated. Right after every avalanche a peak in the signal can then be recorded.

1.6 Space-time correlation

Electrons produced at a time $t_0$ by the incoming charged particle migrate against the electric field with velocity $w$, and reach the anode wire where avalanche multiplication occurs at time $t_1$. The coordinate of the track, with respect to the anode wire, is therefore given by:

$$x = \int_{t_0}^{t_1} w\ dt$$

which reduces to $x = w(t_1 - t_0)$ in case of a constant drift velocity. It is very convenient to have such a linear relation between time and path length, as it simplifies calculations. Constant drift velocity can be obtained combining a well shaped drift field, with small variations and appropriate direction, and the saturated drift velocity peculiarity seen in selected gas mixtures, like the Argon-Carbon Dioxide used in CMS (Figure 2). For an electric field interval between 1 and 2 kV/cm the drift velocity is roughly constant, so if the cell is designed in such a way as to avoid regions of field outside the quoted interval, the sensitivity of the response to local field variation is strongly reduced. The electric field inside CMS drift tubes (equipotential lines) is shown in Figure 8.

To measure the space-time relation in a drift chamber, we can record its drift time spectrum on a uniformly distributed beam. In fact,

$$\frac{dN}{dt} = \frac{dN}{ds} \frac{ds}{dt} \propto w(t)$$
therefore, being $\frac{dN}{ds}$ a constant, the time spectrum represents the drift velocity as a function of time of drift and its integral the space-time relation. An example of this, applied to the case of CMS drift tubes, is shown in Figure 9.

2 CMS Muon System

The Compact Muon Solenoid experiment has been designed to meet the goals of LHC physics programme, with a very precise and high acceptance muon system. In order to achieve good momentum resolution, without making stringent demands on muon chamber resolution and alignment, a high magnetic field (4 T) has been chosen. The field is generated by a superconducting solenoid positioned at the heart of CMS, centered at the center of collision and with the axis along the beam direction.

For future reference, we define CMS coordinate system (see Figure 5). The origin is centered at the collision point, the $y$-axis points vertically upward, the $x$-axis radially inward toward LHC center. The $z$-axis is along the beam direction, and $\phi$ angle is measured from the $x$-axis in the $x$-$y$ plane. The polar angle $\theta$ is measured from the $z$-axis and pseudorapidity is defined as $\eta = -\ln(tan^{2}\frac{\theta}{2})$. Transverse momentum $p_T$ is defined as the three-momentum projection on $x$-$y$ plane, and therefore is a two components vector. Muon apparatus in CMS has been devised to perform tracking job better in transverse plane than in $r$-$z$ plane. This is because CMS has chosen to adopt a magnetic field parallel to beam line, so that charged particles bend in the transverse plane.

2.1 Barrel Detector

We are interested in the barrel region ($|\eta| < 1.2$), which is equipped with drift tubes hosted inside the iron magnet return yoke, divided in 5 wheels along the $z$-axis. Every wheel is in turn divided into 12 sectors, each covering a 30° azimuthal angle. Referring again to Figure 5, we see that each sector contains 4 chambers (light gray in the figure). The 3 innermost chambers consist of 12 layers of drift tubes divided into 3 groups, called SuperLayers, of 4 consecutive layers (see Figure 6). The tubes inside each SuperLyer are staggered by half a tube. Two SuperLayers measure the $r$-$\phi$ coordinate in the bending plane (their tubes have wires parallel to the beam line), and the third measures the $z$-coordinate running parallel to the beam. A honeycomb structure separates an $r$-$\phi$ SuperLayer from the other two. This gives a lever arm length of about 28 cm for the measurement of the track direction inside each chamber in the bending plane. The outermost chamber instead has only 2 SuperLayers that measure the $r$-$\phi$ coordinate, its eventual contribution to the muon $\eta$ measurement was found to be marginal.
2.2 CMS Drift Tubes

A sketch of CMS drift tubes design can be seen in Figure 7. All aluminium components (I-beams separating cells in the same layer, and plates separat-
ing layers) are grounded. Insulated from them, there is a pair of cathode strips glued inside the I-beam: together with the anode wire they shape the field horizontally inside the cell. Another pair of electrode strips are stucked over and below the wire, to improve shaping of the electric field and the linearity of space-time relation (most noticeably in presence of magnetic field).

Figure 7: Cross sectional view of a drift tube, showing drift lines and isochrones. It can be clearly seen that, at least in case of a vertically incident particle, ionization electrons produced in the middle horizontal plane are the first to reach the wire.

Figure 8: Voltages and equipotential lines: the cathode strips, electrode strips and anode wire are set respectively to $-1.2, 1.8$ and $3.6$ kV. The distance between the cathode strips and the wire is $21$ mm so that the average horizontal drift field is around $1.7$ kV/cm, inside the gas drift velocity saturation interval (see Figure 2).

The tubes are operated with an $Ar – CO_2$ (85% / 15%) mixture in order to avoid dealing with organic flammable gases and to have a small Lorenz
angle in presence of stray magnetic fields. This gas mixture and the drift cell design described above allow to get a linear relation between time and drift path (see Figures 2 and 8, it can be seen that drift velocity saturation occurs between 1 and 2 kV/cm). A gas gain around $10^5$ was chosen to operate the drift cells.

Figure 9: Number of recorded hits as a function on drift times, for a uniformly distributed beam irradiating a CMS drift tube. As discussed in section 1.6, this is proportional to the drift velocity as a function of time.

2.3 Meantimer algorithm

A simple method to obtain useful informations about drift chambers behaviour is provided by Meantimer algorithms. These simple formulas (a full account is given in [5]) are mainly used in the calibration procedure, to estimate the maximum drift time and therefore the average drift velocity in the cell. Moreover, they also measure the cell resolution, which can be used as an estimate of the uncertainties associated to each measurement.\(^3\)

The meantimer formulas are relations among the drift times produced by a track in consecutive layers of a superlayer ($t_i$, where \(i\) stands for the particular layer) and the maximum drift time ($T_{\text{max}}$) in a semi-cell (i.e. half

\(^3\)In the approximation of constant drift velocity is important to consider the stray magnetic field and the track impact angle: these parameters will vary substantially, on average, moving from chamber to chamber and also from superlayer to superlayer due to the different positions within the return yoke and to the different pseudorapidities of the impact angles in the \(r - z\) cells. For this reason, the reconstruction algorithm based on a constant drift velocity requires a calibration procedure that allows the average velocity to be found separately for different groups of cells.
cell), under the assumption of a constant drift velocity. The mathematical expression of the meantimer relation depends on the track angle and on the pattern of cells hit by the track. The simplest case is the one shown in Figure 10.

\[ T_{\text{max}} = t_1 + t_3 + t_2 \]

Figure 10: Simple tracks for which meantimer formula \( T_{\text{max}} = t_1 + t_3 + t_2 \) is valid. It is clearly visible that \( T_{\text{max}} \) as calculated above is the time needed to drift across half-cell, in the assumption of constant drift velocity.

In this simple case the correspondent meantimer relation is:

\[ T_{\text{max}} = \frac{t_1 + t_3}{2} + t_2 \]

The average of the meantimer \( T_{\text{max}} \), computed for several tracks, contains information about the average drift velocity in different regions of the cell, since it is computed using drift times produced by hits all over the gas volume. An example of such a distribution, calculated with test beams on an actual drift chamber, is in Figure 11. There we see also a gaussian fit, providing a peak value around 380 ns and a sigma of 3.8 ns. These two values can be used as a first esteem of the DTs mean drift velocity and of their time resolution (\( v_{\text{drift}} \approx \frac{T_{\text{max}}}{L/2} = 54.4 \mu m/\text{ns}, \sigma = 3.8 \text{ ns} \)).

3 Montecarlo simulation

I have developed a simplified simulation of the spectrometer behaviour in order to understand which are the factors that affect \( \beta \) resolution. Time is taken relative to the bunch crossing: I assume that the first layer of (hardware) data acquisition devices have already performed the bunch recognition, so we know precisely when the track was produced. I assume that the energy loss of the projectile is negligible with respect to its velocity, so its \( \beta \) doesn’t change through the detector. This assumption is true either for minimum ionizing muons as produced normally in CMS (\( \beta \approx 1 \)), either for the hypothetical HSCPs generated with \( \beta \) in the range \([0.5, 1]\) but with high momentum (a few hundreds GeV/c).
Figure 11: Sample meantimer distribution as calculated on an actual test beam. The gaussian fit provides useful values, which can be used to estimate the mean drift velocity in the group of cells considered ($\frac{L}{2T_{\text{max}}}$) and their time resolution ($\sigma$).

Multiple scattering is ignored, this is a good approximation because we assume that the HSCP is much heavier than the nuclei. I also assume that the particles do not decay when traversing the detector. For muons this is easily met, because even at $\beta = 0.5$ they have an effective mean life $\gamma \tau \approx 2.53 \, \mu s$, while the time needed to cross the detector (its ray is $\sim 7.5 \, m$) at $\beta = 0.5$ is about $50 \, ns$.

The program is divided in two parts. The first one only generates a track: it returns an array of all the wires and times they recorded a signal (i.e. and avalanche), the second takes for input the array generated as above, and tries to fit the correct $\phi$, $\beta$ and $x_0$ parameters. It uses a minimum $\chi^2$ approach.

3.1 Generating tracks

Let’s briefly summarize the main aspects of this ”generator”:

- It generates tracks in the $z = 0$ plane ($\eta = 0$), so it only uses the $r - \phi$ superlayers.

- Magnetic field is not considered, nor other causes of deflection of the track (like multiple scattering when traversing iron), so tracks go in straight line through all the detector.
• Tracks start at the center of CMS with vertical direction: $\phi$ lies in the interval $[\frac{\pi}{2} - \frac{\pi}{12}, \frac{\pi}{2} + \frac{\pi}{12}]$. This means that only the upper vertical column of drift chambers is crossed, as it covers an angle of $\frac{\pi}{6}$ in the $r - \phi$ plane.

• All aluminium separators of the drift tubes (I-beams, floors and ceilings) are assumed to be very thin, so are neglected.

• Length and positions of the drift tubes inside the detector have been taken from [6], the Technical Design Report of CMS, so drift tubes in consecutive layers are correctly staggered and layers are organised in SuperLayers with the right spacing.

• Tracks have an additional $x_0$ free parameter, the starting coordinate along the $\hat{x}$ axis.

• Time recorded by each DT consists of two components: the time of flight and the time of drift. Time of flight is approximated to the time needed to reach the wire with $\beta$ velocity starting from CMS center. Time of drift can be calculated in two ways: assuming constant drift velocity or with a non-linear space-time relation (like Figure 9) resembling more closely the actual behaviour of the DT. This has been implemented in a simplified manner, dividing the cell length in three sections, each with different constant drift velocity. In both cases drift isochrones are assumed to be perfect circles (instead of ellipses, see Figure 7).

• To account for errors on time measurement due to many factors (propagation of signals through wires and electronic readout, starting point of the avalanche, cell non-linearities and stray magnetic field lines...) generated times gets an additional random gaussian term, with $\sigma = 4\,\text{ns}$. This esteem is taken from the meantimer distribution, Figure 11, who has a gaussian width of 3.8\,\text{ns}. But this is $(t_1 + t_3)/2 - t_2$: assuming equal resolutions on each $t_i$, the gaussian width of the meantimer distribution is $\sigma^2_{\text{meantimer}} = (1/4 + 1/4 + 1)\sigma^2_{\text{time}}$, so there is a small factor $\sqrt{1.5}$.

• There are 32 layers (4 chambers, each with two $r - \phi$ SuperLayers), so the output array will have 32 entries: for each we record an integer, and a float, representing which wire hit and when.

• Each track is therefore identified by three parameters: its angle $\phi$, velocity $\beta$ and initial $x$ coordinate $x_0$. 

15
3.2 Fitting tracks

- Initially, the fitting routine guesses the value of $\phi$ by looking only at which wires are hit and approximating the particle trajectory as it passed exactly on them. Providing good starting points speeds up the minimizing procedure.

- Then, it generates times recorded by each wire exactly the same way as the track generator did, given a set of (guessed) values for $\phi$, $\beta$ and $x_0$. With the so obtained fitted times it evaluates the function:

$$f(\phi, \beta, x_0) = \sum_{i=1}^{32} \left( \frac{t_{i\text{track}} - t_{i\text{fitted}}}{\sigma_i} \right)^2$$

where the sum runs over the wires that hit. This is exactly the so-called $\chi^2$ test of the fit. Actually, $\sigma$ is constant and independent of the wire in our simplification.

- The problem of determining which values of $\phi$, $\beta$ and $x_0$ do actually minimize $\chi^2$ is left to an optimization algorithm, (L-BFGS-B algorithm\(^4\)) which basically reiterates $f$ in a smart way, evaluating it until it finds a minimum.

3.3 Implementation

The logic of my simulation has been explained, now I spend a few words on its implementation. I decided not to use standard CERN software and libraries, like ROOT, because I wanted total control over the behaviour of my program, even if this would have slowed down development a little bit. So I chose Python as programming language, and started writing from scratch. The only external library I used is the numerical minimization algorithm (from the SciPy package). All code of the simulation can be read in the appendix.

3.4 Results

To check that my simulation was working as it should, I ran a few tests.

3.4.1 Drift times distributions

One of the first was about the space-time relation. I have implemented two functions to get the drift time knowing the distance of the track from the

\(^4\)L-BFGS is an algorithm for quasi-Newton optimization. It uses the Broyden–Fletcher–Goldfarb–Shanno update to approximate the Hessian matrix (L-BFGS stands for "limited memory BFGS").
wire: the first approximating constant drift velocity throughout all the cell volume, the second resembling more closely the shape of Figure 9. So I generated 1 million tracks hitting the cell at random positions, and plotted the distribution of the calculated drift times (the same procedure described in section 1.6). The results, for both linear and non-linear space-time relation, are plotted in Figure 12.

![Figure 12: The total drift times spectrum on uniformly distributed beams, for the two functions to get drift times. The non-linear approximation is obtained dividing the cell in three sections, each with different (but constant) drift velocity. It can be seen that it resembles the shape of Figure 9. The edges of both distributions are not sharp, because of the gaussian error added to drift times. To plot the two histograms the same number of iterations (1000000) was used, so the two graphs have the same scale of proportionality between number of entries and drift velocity.](image)

So these two functions work well, they are correctly tuned one with the other because the endpoints of the distributions ($T_{max}$) coincide. It will be interesting to see what happens when fitting tracks generated with the non-linear function, using the simplified linear relation (i.e. approximating constant drift velocity to fit "real" tracks).

### 3.4.2 Fitting $\phi$ and $x_0$

To check that the fitting routine worked, I started with a simplified situation. I used the linear approximated space-time relation (for both generating and fitting the track), $\phi$ and $x_0$ were the only free parameters ($\beta = 1$ was fixed). I generated and fitted some 3000 tracks, the results are good: the relative errors on the two fit are quite small, as shown in Figure 13.
Figure 13: The two histograms represent the distributions of errors on fit of \( \phi \) and \( x_0 \), with \( \beta \) kept fixed and drift velocity approximated constant. I didn’t compute relative errors like \( \frac{\phi_{\text{generated}} - \phi_{\text{fitted}}}{\phi_{\text{generated}}} \) because it would have had a singularity for \( \phi = 0 \). Sigmas of the two gaussians are \( 3 \cdot 10^{-5} \) rad for \( \phi \) and 0.02 cm for \( x_0 \).

3.4.3 Adding \( \beta \) to the fit

The next step is to add the \( \beta \) variable to the fit. For the moment, I maintained the linear space-time relation approximation (constant drift-velocity). The result of the fit of \( \beta \), relative errors distribution for 3000 tracks, is in Figure 14.

Figure 14: Relative errors on fit of \( \beta \), with constant drift velocity approximation. Here the error is of the order of \( 2 \div 3\% \).
3.4.4 Using the non linear space time relation for drifting electrons

Tracks are now generated using the non-linear space time relation (Figure 13 right), and I compare the efficiency of the fit of $\beta$ for the two possibility:
a) the approximated linear space-time relation of Figure 13 (left) is used in the fit routine, saving processing power and time;
b) the same non-linear relation used when generating the track is applied in the fit;

Figure 15: The two histograms represents the relative error distributions for $\beta$ in two different cases. In the first, tracks are generated with non-linear relation for drifting electrons, and fitted with the linear approximation. In the second, the same non-linear relation is used in both routines.

The results are shown in Figure 15. Some remarks here are necessary: the center of the first gaussian (in which linear relation is used for fitting) is not on 0. Computing the average of the distribution I found that the center is at $-0.068$, which means that $\beta$ is overestimated of 6.8%, on average. This could be due to a non optimal calibration of mean drift velocity (the one used in the constant drift vel. approximation).
Secondly, the two graph in Figure 15 has the same scale on x axis, so it is evident that the first is broader than the second. In fact, computed sigmas are $\sigma_a = 0.035$ and $\sigma_b = 0.025$, so relative errors are 3.5% and 2.5% respectively.
3.4.5 Dependance from errors on time measurement

At last, I tried to check how errors on time measurement do affect precision in the fit of $\beta$. Until now, all generated tracks had gaussian errors on measured times with mean 0 and $\sigma = 4$ ns. This value come from the meantimer distribution in Figure 11, computed with test beam on a CMS drift chamber at CERN (see [7]).

I ran two other sessions (3000 tracks generated and fitted), all with the non linear space-time relation, with different sigmas for errors on time: $\sigma_{\text{time}}^a = 8$ ns and $\sigma_{\text{time}}^b = 12$ ns respectively, shown in Figure 16.

![Figure 16: The two histograms represents the relative error distributions for $\beta$ for two different sigmas for errors on measured times.](image)

Computation of the sigmas of these two distribution resulted in relative errors on $\beta$ of 5.1% for $\sigma_{\text{time}}^a = 8$ ns, and 7.3% for $\sigma_{\text{time}}^b = 12$ ns. Recalling that we had 2.5% error on $\beta$ with the original time sigma, we have an approximatelly linear relation between errors on time measurement and errors on $\beta$ estimation.
References


Appendix

#!/usr/bin/env python
#-*- coding:utf-8 -*-

#Iniziato il 29 Aprile 2009, Enzo Busseti

#gli import che servono
import random
import math
from scipy import optimize

def trova_tempo_non_linear(distanza_filone, sigma, drift_velocity_seg1 = 0.00694):
    """Prendo come input la distanza dal filo a cui sono passato,
e restituisco il tempo di drift sporcato da una
gaussiana di sigma scelta""
    err = 0
    if (sigma != 0):
        err = random.gauss(0, sigma)
    drift_time = 0
    drift_velocity_seg2 = drift_velocity_seg1 / 1.212
    drift_velocity_seg3 = drift_velocity_seg1 / 1.333
    if (distanza_filone < 0.1):
        drift_time = distanza_filone / drift_velocity_seg1
    if (distanza_filone > 0.1 and distanza_filone < 0.5):
        drift_time = 0.1 / drift_velocity_seg1 +
        (distanza_filone - 0.1) / drift_velocity_seg2
    if (distanza_filone > 0.5):
        drift_time = 0.1 / drift_velocity_seg1 + 0.4 /
        drift_velocity_seg2 + (distanza_filone - 0.5) /
        drift_velocity_seg3
    return drift_time + err

def trova_tempo_linear(distanza_filone, sigma, mean_drift_velocity = 0.00544):
    """Prendo come input la distanza dal filo a cui sono passato,
e restituisco il tempo di drift sporcato da una
gaussiana di sigma scelta""
    err = 0
    if (sigma != 0):
        err = random.gauss(0, sigma)
    return distanza_filone/mean_drift_velocity + err

def funzione_da_minimizzare(parametri, y_layers, traccia, space_time, semilarghezza_cella = 2.1):
    """Parametri[] contiene le cose da fit pare: lo 0 e' phi, l'1 e' beta, il 2 e' x0""
    chis = 0
    pendenza = math.tan(parametri[0])
    coseno_phi = math.cos(parametri[0])
    seno_phi = math.sin(parametri[0])
    for layer in range(32):

        22
lunghezza_flight = math.sqrt( y_layers[layer]**2 + (semilarghezza_cella * traccia[layer][0] - parametri[2])**2 )
time_of_flight = lunghezza_flight / (parametri[1] * 30.0)

lunghezza_drift = abs(traccia[layer][0] * semilarghezza_cella - pendenza * y_layers[layer] - parametri[2]) / coseno_phi
if (space_time == 'linear'):
time_of_drift = trova_tempo_linear(lunghezza_drift, 0)
elif (space_time == 'non_linear'):
time_of_drift = trova_tempo_non_linear(lunghezza_drift, 0)
residual = traccia[layer][1] - time_of_flight - time_of_drift
chis += residual * residual
return chis

class generatore:
    def __init__(self, x_0 = 0, phi = 0, beta = 1, space_time = "linear", sigma = 3.8):
        #inizializzo un po' di variabili
        #x del punto da cui parte la traccia
        self.campo_variazione_x_0 = (-10, 10)
        self.x_0 = x_0
        self.x_0_random = True
        #inizializzazione di phi, Angolo con la verticale a cui esce la traccia, in [-math.pi/12, math.pi/12] gradi
        self.phi = phi
        self.phi_random = True
        self.campo_variazione_phi = (-math.pi/12, math.pi/12)
        #inizializzazione di beta
        self.beta_random = True
        self.campo_variazione_beta = (0.5, 1.0)
        self.beta = beta
        self.space_time = space_time
        #risoluzione della singola camera (in nanosecondi), cioe' sigma della gaussiana con cui sporco le tracce, 0 se non voglio errore
        self.sigma = 3.8
        #queste sono costanti, le unita' sono cm e nanosecondi
        self.larghezza_cella = 4.2
        self.semilarghezza_cella = 2.1
        #coordinate y dei piani dei fili per ogni layer in r-phi
        self.y_layers = {
            401.0, 402.4, 403.8, 405.2,
            447.5, 448.9, 450.3, 451.7,
            482.8, 484.2, 485.6, 487.0,
            529.3, 530.7, 532.1, 533.5,
            589.8, 591.2, 592.6, 594.0,
            636.3, 637.7, 639.1, 640.5,
            692.3, 693.7, 695.1, 696.5,
            738.8, 740.2, 741.6, 743.0
        }
    def __repr__(self):
        return str(self)
print "Beta =", self.beta, 
if (self.beta_random): print "(random)"
else: print "(non random)"
print "Phi =", self.phi, "rad ", 
if (self.phi_random): print "(random)"
else: print "(non random)"
print "X_0 =", self.x_0, 
if (self.x_0_random): print "(random)"
else: print "(non random)"
print "Relazione spazio−tempo =", self.space_time
return "Sigma = " + str(self.sigma) + " ns"
def randomizza(self):  
if (self.phi_random): self.randomizza_phi() 
if (self.beta_random): self.randomizza_beta() 
if (self.x_0_random): self.randomizza_x_0() 
def randomizza_phi(self):  
self.phi = random.uniform(self.campo_variazione_phi[0], self.campo_variazione_phi[1])
def randomizza_beta(self):  
self.beta = random.uniform(self.campo_variazione_beta[0], self.campo_variazione_beta[1])
def randomizza_x_0(self):  
self.x_0 = random.uniform(self.campo_variazione_x_0[0], self.campo_variazione_x_0[1])
def trova_fil(self, layer, x_traccia):  
#con queste due righe riconosco qual ' e ' il filo buono  
filo = x_traccia // self.semilarghezza_cella  
if (((filo + layer) % 2 == 1): filo += 1 
return int(filo) 
def genera_traccia(self):  
traccia = [] 
antipendenza = math.tan(self.phi)  
for layer in range(32):  
x_traccia = self.x_0 + antipendenza * self.y_layers[layer]  
t_traccia = math.sqrt(x_traccia*x_traccia + self.y_layers[layer] * self.y_layers[layer]) / (self.beta * 30.0)  
filo = self.trova_fil(layer, x_traccia)  
offset = (x_traccia − filo − self.semilarghezza_cella)  
lunghezza_drift = abs(offset / math.cos(self.phi))  
if self.space_time == 'linear':  
traccia.append( (filo, (t_traccia + trova_tempo_linear(lunghezza_drift, self.sigma)) ) 
else if self.space_time == 'non_linear':  
traccia.append( (filo, (t_traccia + trova_tempo_non_linear(lunghezza_drift, self.sigma)) ) 
return traccia
class fittatore:
    def __init__(self, x_0 = 0, phi = 0, beta = 1, space_time = "linear", sigma = 3.8):
        self.x_0 = x_0
        self.phi = phi
        self.beta = beta
        self.space_time = space_time
        self.sigma = sigma
        self.beta_fit = True
        self.phi_fit = True
        self.x_0_fit = True
        #queste sono costanti, le unità sono cm e nanosecondi
        self.larghezza_cell = 4.2
        self.semilarghezza_cell = 2.1
        self.drift_velocity = 0.00544
        #coordinate y dei piani dei fili, per ogni layer in r-phi
        self.y_layers = [
        401.0, 402.4, 403.8, 405.2,
        447.5, 448.9, 450.3, 451.7,
        482.8, 484.2, 485.6, 487.0,
        529.3, 530.7, 532.1, 533.5,
        589.8, 591.2, 592.6, 594.0,
        636.3, 637.7, 639.1, 640.5,
        692.3, 693.7, 695.1, 696.5,
        738.8, 740.2, 741.6, 743.0]
    def __repr__(self):
        res = ''
        print
        print "Sto fittando le variabili:",
        if self.phi_fit: print "phi ",
        else: res += "phi = " + str(self.phi)
        if self.beta_fit: print "beta ",
        else: res += "beta = " + str(self.beta)
        if self.x_0_fit: print "x_0 ",
        else: res += "x_0 = " + str(self.x_0)
        print
        print "Sto usando la rel. spazio−tempo: " +
        self.space_time
        return "Variabili tenute fisse: " + res
    def inizializza_fit(self, traccia):
        self.traccia = traccia
        self.x_0 = 0
        phi = 0
        pesi = 0
        self.lunghezza_flight = []
        for layer in range(32):
            sigma = 1 /
            math.sqrt(self.y_layers[layer]*self.y_layers[layer] +
                       self.traccia[layer][0]*4.41 *
                       self.traccia[layer][0])
            print "#4.41 sarebbe 2.1 * 2.1"
phi += math.atan((self.traccia[layer][0] * 
    self.semlarghezza_cella) / 
    self.y_layers[layer]) / (sigma*sigma)

pesi += 1 / (sigma*sigma)
self.phi = phi / pesi
self.beta = 1

# if self.beta_fit: self.beta = 0.75
def calcola_chiq (self):
    """Calcolo il chi quadro della funzione fittata, uso 
    quindi i valori fittati di phi, beta e x_0""
    chis = funzione_da_minimizzare([self.phi, self.beta, 
        self.x_0], self.y_layers,
        self.traccia,
        self.space_time,
        semilarghezza_cella =
        self.semlarghezza_cella)

    chis = chis / (self.sigma * self.sigma)
    print("Chi quadro ottenuto = " + str (chis))
    self.chiq = chis

def fit(self, t):
    self.inizializza_fit(t)
    self.minimizzatore = optimize.fmin_l_bfgs_b
    guess = (self.phi, self.beta, self.x_0)
    bounds = []
    if self.phi_fit: bounds.append((-0.26179938779914941,
        0.26179938779914941))
    else: bounds.append((self.phi, self.phi))
    if self.beta_fit: bounds.append((0.5, 1))
    else: bounds.append((self.beta, self.beta))
    if self.x_0_fit: bounds.append((-10, 10))
    else: bounds.append((self.x_0, self.x_0))
    args = (self.y_layers, self.traccia, self.space_time)
    parametri, fval, self.m = 
        self.minimizzatore(funzione_da_minimizzare, guess, 
        args=args, approx_grad=True, bounds=bounds,
        imprint=-1, factr = 10.0)
    self.phi = parametri[0]
    self.beta = parametri[1]
    self.x_0 = parametri[2]
    print("Parametri ottenuti:")
    print("Beta =", self.beta)
    print("Phi =", self.phi)
    print("X_0 =", self.x_0)
    self.calcola_chiq()