Development and Application of a Novel Graphical Environment for Physics Data Analysis with the CMS Experiment

von

Gero Müller

Diplomarbeit in Physik

vorgelegt der

Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen

im September 2008

angefertigt am

III. Physikalischen Institut A

Professor Dr. Martin Erdmann

Contents

1 Introduction

In 2008 the worlds largest proton-proton collider, the Large Hadron Collider (LHC), will start operations and produce about a billion collisions per second. As it is not possible to process all of these events only the most interesting ones are selected by hard- and software triggers, still resulting in millions of events after a short time of data acquisition.

After many steps of reconstruction the raw detector readouts are transformed into a list of identified particles and jets for each event. In a typical analysis the physics process of there-constructed events needs to be identified. Therefore a carefully chosen set of variables can be extracted from each list which are then compared with the help of statistical methods.

When performing high energy physics analyses a huge amount of data (Monte Carlo or real) need to be processed using a large number of different algorithms. One step to better manage-ability of an analysis is to divided into smaller and manageable parts, so-called modules. But as the module count and their complexity increase, the need to simplify even the module steering arises.

In this thesis a graphical analysis software is presented which helps the physicist to design, manage and execute his analysis.

A short description of the LHC and the CMS detector is given in Chapter 2. In Chapter 3 the Standard Model of particle physics and the Top Quark Physics are briefly introduced. The next Chapter describes the underlying software toolkit used for the analysis software developed in this thesis. Chapter 5 introduces the modular design approach while in Chapter 6 examples for more sophisticated modules are presented. A demontration analysis which was designed and executed using the techniques introduced earlier is described in Chapter 7. And finally in Chapter 8 a summary is given.

2 The LHC Accelerator and the CMS Experiment

2.1 The Large Hadron Collider

The Large Hadron Collider (LHC) is a proton-proton collider with a center-of-mass energy of up to $\sqrt{s} = 14$ TeV. The protons are accelerated in two beams with an energy of 7 GeV each. It is installed in the tunnel of the former Large Electron-Positron Collider (LEP) with a circumference of 27 km at the European Organisation for Nuclear Research (CERN) [1,2]. Its design luminosity is $\mathcal{L} = 10^{34}$ cm^{-2}s^{-1}. In addition to proton-proton collisions the LHC is also designed for heavy ion collisions (e.g. Pb-Pb) with $\sqrt{s} = 5.5$ TeV and $\mathcal{L} = 10^{27}$ cm^{-2}s^{-1}. The beam pipes have crossing points at the four detector sites where the roughly 1 billion collisions per second are examined by the following experiments.

ALICE - A Large Ion Collider Experiment, which is especially constructed for the investigation of heavy ion collisions [3]

ATLAS - A Toroidal LHC ApparatuS, a general-purpose detector which covers a wide range of physics analyses [4]

CMS - Compact Muon Solenoid, also a general-purpose detector [5]

LHC-b - Large Hadron Collider beauty Experiment, which aims to analyze phenomena in decays of B-mesons with high precision and high statistics [6]

Figure 2.1 shows the accelerator and detector facilities at CERN. The particle beams are kept on track by superconducting dipole magnets which can produce magnetic fields of up to 8.3 T at 1.9 K. The beam is divided into 2808 bunches of about 10^{11} protons each. The Protons are created in a hydrogen source and later formed to bunches in the Proton Synchrotron, already with the final timing of 25 ns. These bunches are accelerated to 450 GeV in the SPS (Super Proton Synchrotron) and then injected into the LHC where they are finally accelerated by cavities to 7 TeV.

2.2 The CMS Detector

The Compact Muon Solenoid is a general purpose particle detector [1]. It fulfills the LHC physics programme requirements like good muon identification, good charged particle momentum resolution and reconstruction efficiency, good electromagnetic energy resolution, and a

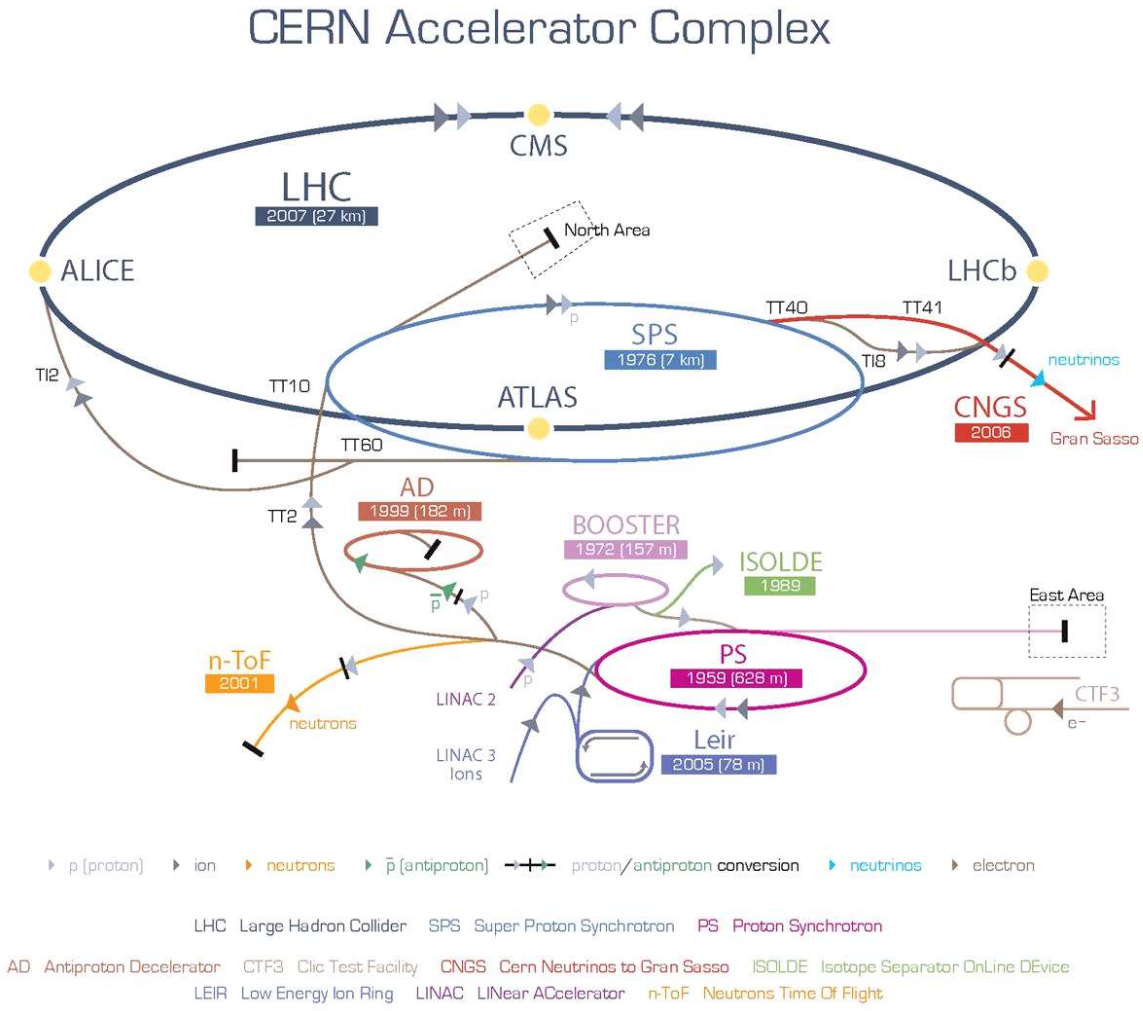


Figure 2.1: The accelerators and detectors at CERN [3].

good missing transverse energy resolution. What distinguishes the CMS detector from others are its high-field solenoid, a full silicon-based inner tracking system, and a fully active scintillating crystal-based electromagnetic calorimeter. In figure 2.2 the overall layout is shown. The detector dimensions are a length of 21.6 m, a diameter of 14.6 m, and total weight of 12500 t.

2.2.1 Magnet

For a good momentum resolution a high bending power is needed. Therefore CMS chose a large superconducting solenoid with a field of 4 T. It carries a current of 19.5 kA and stores 2.7 GJ of energy.

2.2.2 The Muon System

The muon system uses 3 types of gaseous detectors. In the barrel region drift tubes (DT) are used. Here the background produced by neutrons is small and the residual magnetic field as

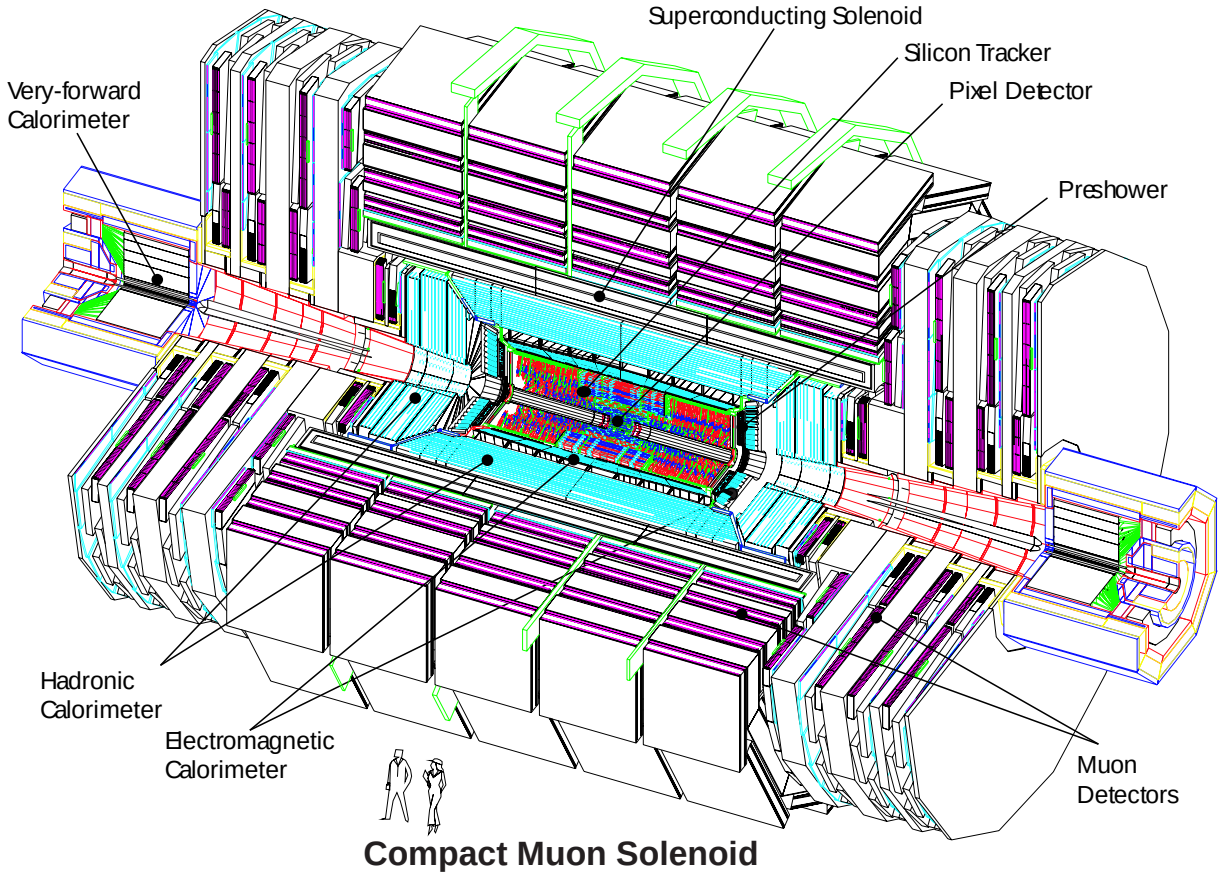


Figure 2.2: Cutaway view of the CMS detector and its compounds [4].

well as the muon rate are low. In the two endcaps where the background, magnetic field and muon rate are high, cathode strip chambers (CSC) are deployed. The barrel region is defined by $|\eta| < 1.2$ and the endcap region by $1.2 < |\eta| < 2.4$ (see figure 2.3). Additionally resistive plate chambers (RPC) are installed in both the barrel and the endcap regions. The drift tubes have a ϕ precision better than 100 μm in position and about 1 mrad in direction. The cathode strip chambers have a typical spatial resolution of 200 μm and an angular resolution in ϕ of the order of 10 mrad. The Muon System contains over 25000 m^2 of active detection planes, and close to 1 million electronic channels.

2.2.3 Electromagnetic calorimeter

The Electromagnetic calorimeter (ECAL) is constructed from 61200 lead tungstate crystals which are mounted in the central barrel and 7324 crystals in each of the endcaps. There are additional preshower devices placed in front of the endcap crystals. In the barrel region silicon avalanche photodiodes (APD) are used as photodetectors. In the endcaps vacuum phototriodes are used instead. The calorimeters in the barrel section (EB) are called “supermodules” and cover a pseudorapidity interval of $0 < |\eta| < 1.479$. Those in the endcaps (EE) are located at a distance of 314 cm from the primary interaction point and cover the pseudorapidity interval of

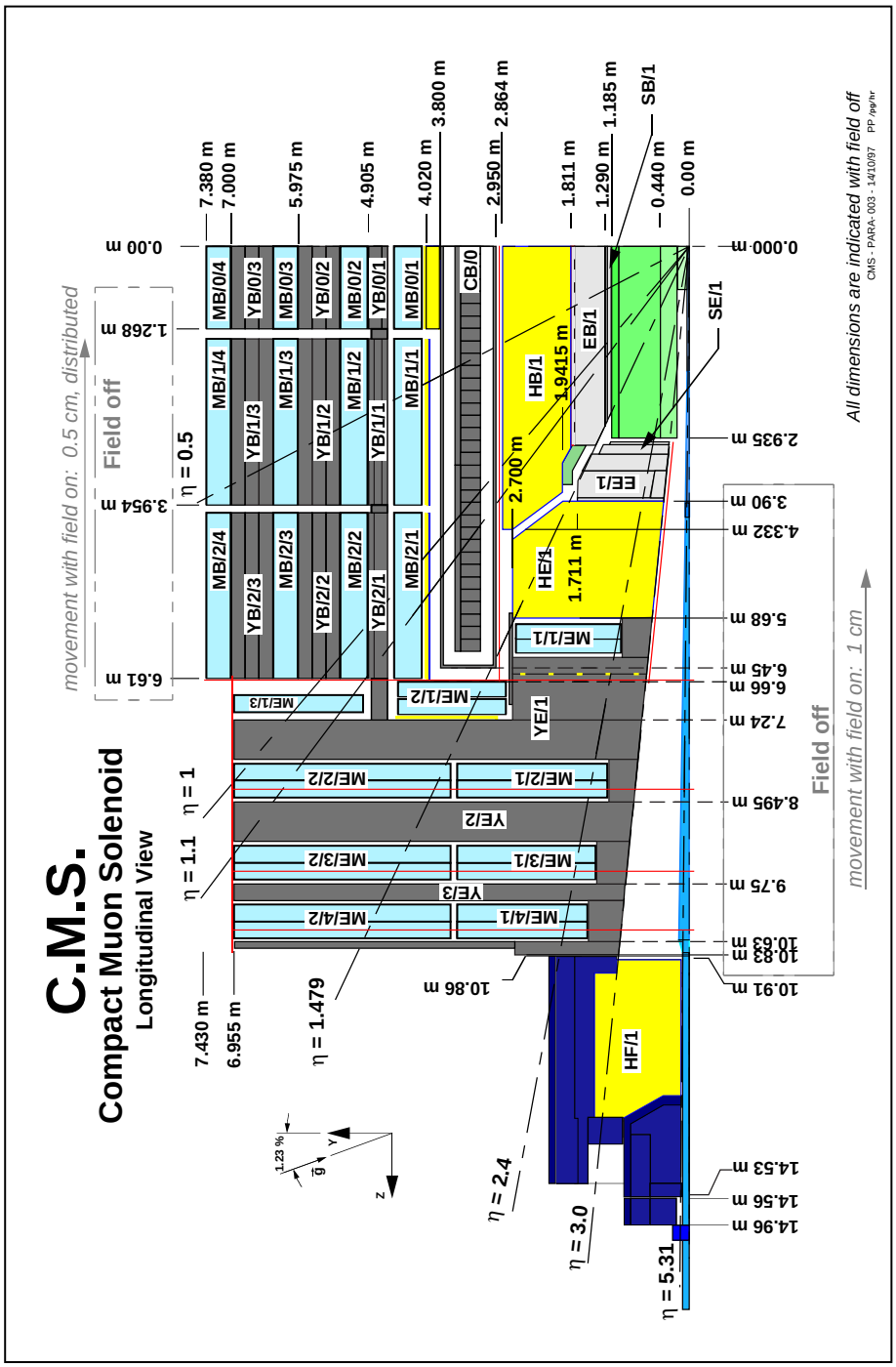


Fig. 1.1.2(color): Longitudinal view of one quadrant of the CMS detector.

Figure 2.3: Longitudinal view of one quadrant of the CMS detector. [5].

$1.479 < |\eta| < 3.0$. The energy resolution can be parameterized as

$$\left(\frac{\sigma}{E}\right)^2 = \left(\frac{S}{\sqrt{E}}\right)^2 + \left(\frac{N}{E}\right)^2 + C^2 \tag{2.1}$$

where S is the stochastic term and has measured values of $3.63 \pm 0.1\ \sqrt{GeV}$ for 20 x 20 mm^2 triggers and 2.83 ± 0.3 for 4 x 4 mm^2 triggers. N is the noise term and was determined to be 124 MeV. C is a constant term of 0.26 ± 0.04. These values are calculated from energy distributions acquired in an array of 3x3 pixels.

2.2.4 Hadron calorimeter

The Hadron calorimeter surrounds the ECAL and lies mostly inside the coil and was designed to minimize the non-Gaussian tails in the energy resolutions and to provide good containment for the missing energy measurement. The calorimeter has 4 parts:

Hadron barrel (HB), they consist of 32 towers each, are located in the barrel region and cover the pseudorapidity $-1.4 < \eta < 1.4$.

Hadron outer (HO), they contain scintillators and are located outside of the coil. They increase the effective thickness of the calorimetry to over 10 interaction length which leads to decreased tails in the energy resolutions. They cover $-1.26 < \eta < 1.26$.

Hadron endcap (HE) do consist of 14 towers each and cover the pseudorapidity $1.3 < |\eta| < 3.0$.

Hadron forward (HF) detectors are composed of steel/quartz fibre and cover the pseudorapidity $3.0 < \eta < 5.0$.

2.2.5 The Tracking System

Due to the decreasing particle flux at higher radii different detector types are used. Close to the interaction point pixel detectors with a total area of about 1 m^2 are used. The 3 barrel layers are located at radii of about 4.4 cm, 7.3 cm and 10.2 cm and have a length of 53 cm. The two endcaps on each side are assembled in a turbine-like geometry with blades rotated by 20^o and extend from 6 to 15 cm in radius. They are placed about 10 and 20 cm next to the barrel layers. The layout of the pixel detector is depicted in figure 2.4. The size of the pixels is about 100 x 150 μm^2 resulting in an occupancy of 0.1% per pixel per crossing and a total of 66 million pixels. At a distance of 20 cm to 55 cm silicon microstrip detectors with cell sizes starting at 10 cm x 80 μm are installed. They have an occupancy of 2-3% per crossing. Beyond 55 cm larger-pitch silicon microstrips are used. The low flux allows to use cells with a maximum size of 25 cm x 180 μm to keep the occupancy at one percent. Together the 9.6 million silicon strip detectors have an area of 200 m^2. The strip tracker consists of 4 parts, figure 2.5:

Tracker Inner Barrel (TIB), consists of 4 layers of silicon sensors covering $|z| < 65cm$.

Tracker Outer Barrel (TOB), is made of 6 layers and covers $|z| < 110$ cm

Tracker End Cap (TEC), consists of 9 discs ranging from $|z| = 120$ cm to $|z| = 280$ cm

Tracker Inner Discs (TID), comprises 3 smaller discs that fill the gap between the TIP and the TEC.

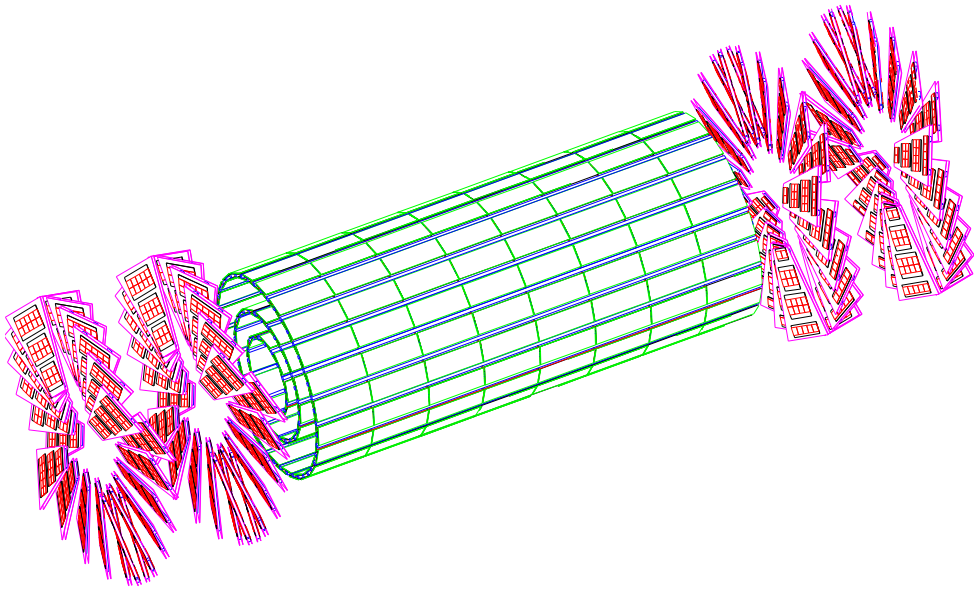


Figure 2.4: Layout of the CMS pixel detector [6].

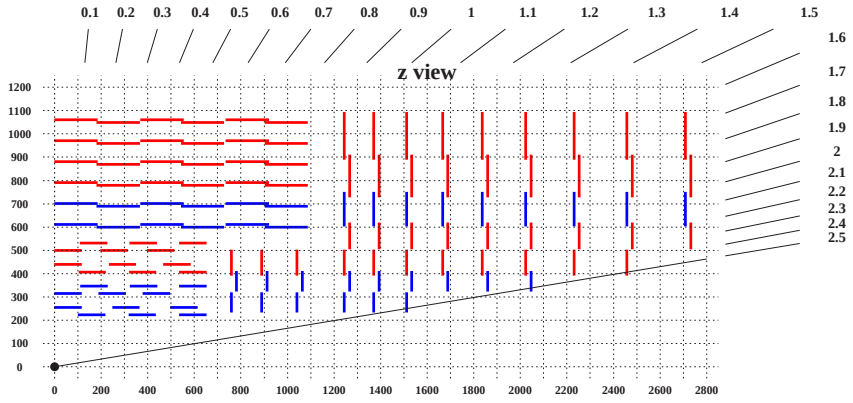


Figure 2.5: Layout of strip tracker of the CMS experiment (1/4 of the z view) [7].

3 Single Top and the Standard Model

According to the Standard Model (SM) of particle physics matter consists of elementary particles which interact by exchanging field quanta which are described by quantized fields [8]. This chapter first gives a rough overview of elementary particles and the forces acting between them. In the second part the physics of electroweak single top decays is briefly introduced.

3.1 Elementary Particles of the Standard Model

Elementary particles are point-like and structureless. They carry spin 1/2 and therefore are fermions. As listed in table 3.1 there are quarks and leptons. There are six quarks of different flavour, the up and down quarks, the charm and strange quarks, and the top and bottom quarks. There are six leptons, namely the electron, the electron neutrino, the muon and the muon neutrino, and the tau together with the tau neutrino. All of them have associated antiparticles with the same mass but opposite charge. The electron has a charge of one while neutrinos have no and quarks a fractional charge of one-third or two-third the charge of the electron. All particles carry some kind of quantum numbers. The quarks carry a flavour quantum number and the leptons carry electron, muon and tau quantum numbers. Quarks and leptons are also grouped into 3 generations. The up and down quarks as well as the electron and the electron neutrino belong to the first generation. Charm quark, strange quark, muon and muon neutrino belong to the second generation while the top quark, the bottom quark, the tau and the tau neutrino belong to the third generation.

3.2 Forces of the Standard Model

The Standard Model incorporates 3 fundamental forces, see table 3.2. The electromagnetic force, the strong and the weak force. The electromagnetic force is responsible for the emission of light, e.g. Nuclei are kept stable by the strong force and the nuclear decay is explained by the weak force. All three forces are quantized fields, formally described by gauge theories. The quanta of the fields are: the massless photon (γ) of the electromagnetic force, 8 massless gluons of the strong force, and the $W^{\pm}$ and Z^0 bosons of the weak force. All of them are spin 1 gauge bosons.

While the quarks take part in all 3 types of interactions, the leptons only interact via the electromagnetic (when they carry a charge) and the weak force.

Particle	Generation	Mass [MeV/c^2]	Charge (Q)	3$^{\text{rd}}$ component of weak-isospin (T_3)	Color
Up quark (u)	1	1.5-3	+2/3	-1/2	rgb
Down quark (d)	1	3-7	-1/3	+1/2	rgb
Charm quark (c)	2	$(1.25 \pm 0.09) \times 10^3$	+2/3	-1/2	rgb
Strange quark (s)	2	70-120	-1/3	+1/2	rgb
Top quark (t)	3	$(170.9 \pm 1.8) \times 10^3$ [9]	+2/3	-1/2	rgb
Bottom quark (b)	3	$(4.20 \pm 0.07) \times 10^3$	-1/3	+1/2	rgb
Electron (e^-)	1	0.511	-1	+1/2	
Electron neutrino (ν_e)	1	$< 2 \times 10^{-6}$	0	-1/2	
Muon (μ^-)	2	106	-1	+1/2	
Muon neutrino (ν_μ)	2	< 0.19	0	-1/2	
Tau (τ^-)	3	1777	-1	+1/2	
Tau neutrino (ν_τ)	3	< 18.2	0	-1/2	

Table 3.1: Leptons and quarks and their properties. Masses are taken from Ref. [10] if not stated differently. The third component of the weak-isospin is stated for left-handed particles only. 'rgb' in the last column shall indicate that the quarks exist in three different color states 'red', 'green' and 'blue'.

Force	Range [m]	Relative strength	Force carrier	Mass [GeV/c^2]	'Charge'
Strong force	10^{-15}	1	8 gluons (g)	massless	color
Electromagnetic force	inf	10^{-2}	photon (γ)	massless	charge
Weak force	10^{-13}	10^{-2}	$W^\pm$	80.403 ± 0.029	weak-isospin
			Z^0	91.1876 ± 0.0021	

Table 3.2: The three forces described by the Standard Model. The masses correspond to the force carrying bosons and are taken from Ref. [10].

3.2.1 Electroweak forces

The electroweak theory is, as the name suggests, a unification of the electromagnetic and the weak interactions into one framework. It was developed by Glashow,Weinberg and Salam in the 1960s.

Starting with the free particle Lagrangian density for massless fermions and bosons:

$$\mathcal{L}_0 = \mathrm{i}\bar{\Psi}\gamma_\mu\delta_\mu\Psi$$

Now $\mathcal{L}_0$ shall be invariant under local phase transformations:

$$\Psi_L \rightarrow e^{ig\vec{\alpha}(x)\cdot\vec{T}+\mathrm{i}g\prime\beta(x)Y}\Psi_L \qquad \text{and} \qquad \Psi_R \rightarrow e^{ig\prime\beta(x)Y}\Psi_R$$

Where $\vec{\alpha}(x)$ is an arbitrary three-component vector and $\vec{T}$ is the weak-isospin operator whose components T_i are the generators of $SU(2)_L$. They are represented by $T_i = 1/2\sigma_i$ with the Pauli-Matrices:

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_2 = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \quad \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

$\beta(x)$ is an arbitrary one-dimensional function, and Y is the weak hypercharge with

$$Y = 2(Q - T_3)$$

and being the generator of the symmetry group $U(1)_\gamma$. The gauge invariance is achieved by replacing δ^μ by the covariant derivate

$$D_\mu = \partial_\mu + \mathrm{i}g\vec{W}_\mu\vec{T} + \mathrm{i}g'Y\frac{1}{2}B_\mu$$

and adding the kinetic energy terms

$$-\frac{1}{4}\vec{W}_{\mu\nu}\vec{W}^{\mu\nu} - \frac{1}{4}B_{\mu\nu}B^{\mu\nu}$$

resulting in

$$\mathcal{L}_{EW} = \sum_{fermions} (i\bar{\psi}_L\gamma^\mu[\partial_\mu + ig\vec{W}_\mu\vec{T} + ig'Y_L\frac{1}{2}B_\mu]\psi_L + i\bar{\psi}_R\gamma^\mu[\partial_\mu - ig'Y_R\frac{1}{2}B_\mu]\psi_R) - \frac{1}{4}\vec{W}_{\mu\nu}\vec{W}^{\mu\nu} - \frac{1}{4}B_{\mu\nu}B^{\mu\nu} \quad (3.1)$$

Where $\vec{W}_\mu = (W_{1\mu}, W_{2\mu}, W_{3\mu})^T$ is the isotriplet for $SU(2)_L$ and B_μ is the singlet for $U(1)_Y$.

In quantum field theory quarks and leptons are represented by the spinor fields Ψ where the weak interaction only couples to the left-handed particles. Therefore left- and right-handed fields $\Psi_L = 1/2(1-\gamma_5)\Psi$ and $\Psi_R = 1/2(1+\gamma_5)\Psi$ are introduced. The left-handed states of one generation are grouped into weak-isospin doublets while the right-handed states form singlets: The up-type quarks (u, c, t) and the neutrinos carry the isospin $T_3 = -1/2$ and down-type

$$\begin{array}{cccccc} \begin{pmatrix} u \\ d \end{pmatrix}_L & \begin{pmatrix} c \\ s \end{pmatrix}_L & \begin{pmatrix} t \\ b \end{pmatrix}_L & \begin{pmatrix} \nu_e \\ e \end{pmatrix}_L & \begin{pmatrix} \nu_\mu \\ \mu \end{pmatrix}_L & \begin{pmatrix} \nu_\tau \\ \tau \end{pmatrix}_L \\ u_R & c_R & t_R & & & \\ d_R & s_R & b_R & e_R & \mu_R & \tau_R \end{array}$$

quarks (d, s, b), electron, muon and tau leptons carry $T_3 = 1/2$.

3.2.2 Strong interactions

The strong force is described by the quantum chromodynamics (QCD) and the underlying SU(3)$_c$ gauge symmetry. The charge assigned to the particles by this theory is called colour.

There are the colours red, green and blue and their anticolours. Combining all three or a colour and its anticolour yields a neutral colourcharge. No free quarks have been observed in experiments only in pairs. A quark and antiquark form mesons while baryons are built of three quarks. Both therefore have a neutral colourcharge. The strong force is mediated by 8 baryons, called gluons. They carry one colour and one anticolour charge and therefore couple to themselves. Quarks are the only fermions carrying colourcharge, so leptons do not take part in this interaction.

3.3 Single Top Quark Production

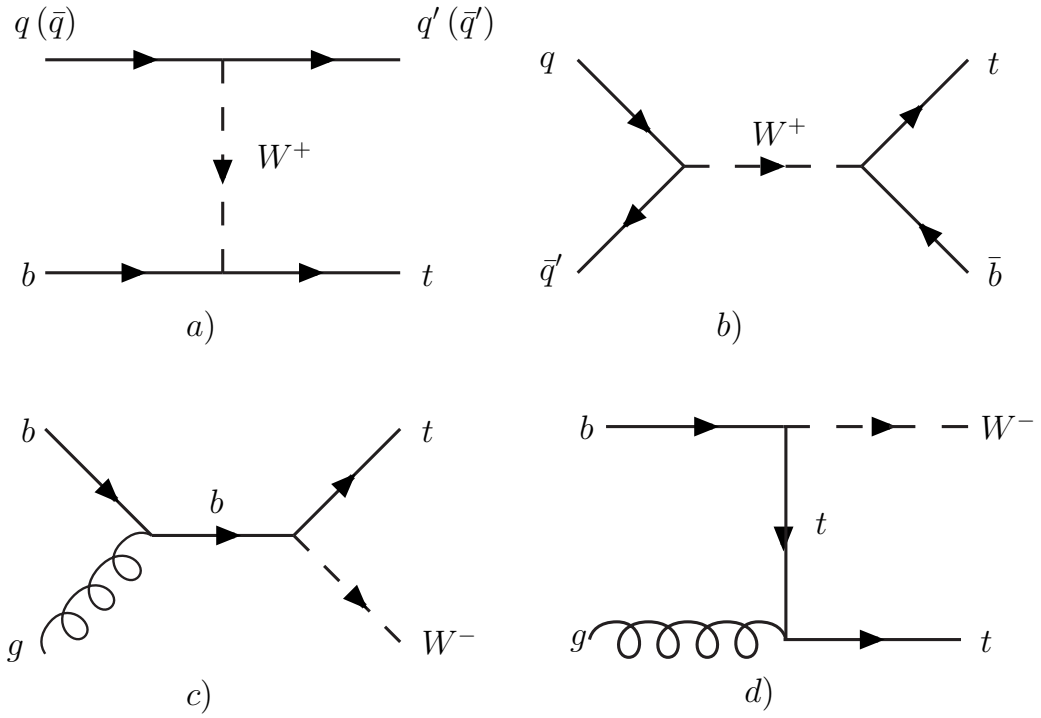


Figure 3.1: Lowest order Feynman diagrams for single-top-quark production processes: t channel (a), s channel (b), and associated tW production (c,d). [11]

Top quarks are produced at the LHC by two types of processes: QCD and electroweak production. In QCD production top quarks are created in $t\bar{t}$ pairs via the processes $q\bar{q} \rightarrow t\bar{t}$ and $g\bar{g} \rightarrow t\bar{t}$. The electroweak single-top quark production is composed of three channels: the s-channel, the t-channel and the associated Wt production. All three are produced via the Wtb vertex and are distinguished by the virtuality Q^2 of the W boson. Due do their small CKM matrix element the production channels using a Wtd or Wts vertex are suppressed. Table 3.3 lists the different cross sections expected at the LHC.

3.3.1 t-channel

This channel is also called t-channel as the virtuality of the W boson is equal to $Q^2 = -q^2 = -\hat{t}$. A virtual W hits a sea quark (b) inside the proton. For high energies the mass can be neglected,

Process	σ (t-channel) (pb)	σ (s-channel) (pb)	σ (Wt) (pb)
pp $\rightarrow$ t	(156 $\pm$ 8)	(6.6 $\pm$0.6)	$14.0^{+3.8}_{-2.8}$
pp $\rightarrow \bar{t}$	(91 $\pm$ 5)	(4.1 $\pm$0.4)	$14.0^{+3.8}_{-2.8}$

Table 3.3: Predicted total cross sections for single top quark production processes. They are given for pp collisions at $\sqrt{s}$ = 14 TeV and a top quark mass of 175 GeV/c^2 [8].

$\hat{t} \approx -p_1 \cdot p_3 = q^2$, the W is spacelike. The leading order differential cross section is given by

$$\frac{\mathrm{d}\hat{\sigma}_{ij}}{\mathrm{d}\hat{t}} = \frac{\pi\alpha_w^2}{4} \cdot |V_{ij}|^2 |V_{tb}|^2 \cdot \frac{\hat{s} - M_{top}^2}{(\hat{s} - m_b^2)(\hat{t} - M_W^2)^2}$$

for quark-quark or antiquark-antiquark collisions and

$$\frac{\mathrm{d}\hat{\sigma}_{ij}}{\mathrm{d}\hat{t}} = \frac{\pi\alpha_w^2}{4} \cdot |V_{ij}|^2 |V_{tb}|^2 \cdot \frac{\hat{u} - b_b^2}{(\hat{s} - m_b^2)^2} \frac{\hat{u} - M_{top}^2}{(\hat{t} - m_W^2)^2} \tag{3.2}$$

for quark-antiquark collisions. Where $\hat{t}$ is one of the three Mandelstam variables for which

$$\hat{s} + \hat{t} + \hat{u} = m_b^2 + M_{top}^2$$

is true.

$$\hat{t} = -\frac{\hat{s}}{2}\left(1 - \frac{m_b^2}{\hat{s}}\right)\left(1 - \frac{m_{top}^2}{\hat{s}}\right) \cdot (1 - \cos\hat{\theta})$$

3.3.2 s-channel

This processes is mediated by a timelike W boson and involves quark and a antiquark. The virtuality of the W, $q^2 = \hat{s} \geq (M_{top} + m_b)^2$, is positive. Its leading order parton cross section is

$$\hat{\sigma}_{ij}(\hat{s}) = \frac{\pi\alpha_w^2}{2} \cdot |V_{ij}|^2 |V_{tb}|^2 \tag{3.3}$$

$$\text{with} \quad q_0 = \frac{\hat{s} + M_{top}^2 - m_b^2}{2\sqrt{\hat{s}}}$$

The ratio between the t-channel and the s-channel will be 23 at the LHC. As this is a quark-antiquark reaction it will be much harder to measure than processes with quark-quark interaction like the gluon initiated process, $t\bar{t}$ and W-gluon-fusion processes.

3.3.3 Associated Production

In this process a quark is produced in association with an on-shell (real) or close to on-shell W boson ($q^2 \approx m_W^2$). The initial b quark is a sea quark inside the proton.

4 Software for HEP Event Analysis and Visualisation

In this chapter the tools used as basis for the graphical analysis design are presented.

4.1 PXL

The Physics eXtension Library (PXL) is a class collection for the advanced analysis in a high energy physics experiment. PXL is written in C++ and intensively uses the Standard Template Library (STL). The syntax of PXL is designed to support experienced users as well as newcomers in developing their own analysis, reaching from simple selection algorithms to complex multi-process analyses. PXL allows the physicist to store all event information in PXL objects such as particles, vertices or collisions. Users can inherit from PXL classes, and use their fea-

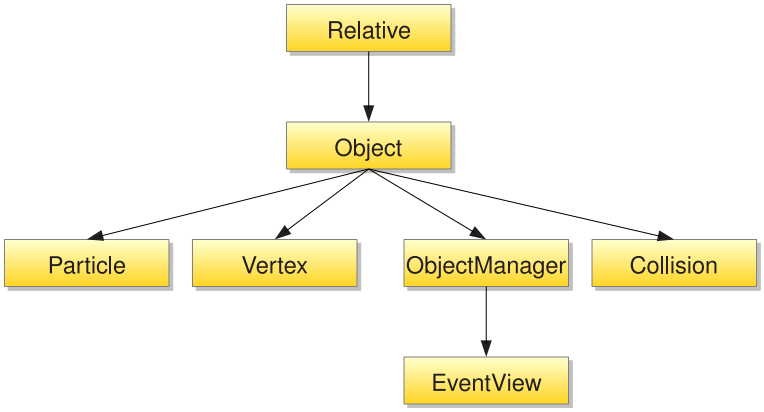


Figure 4.1: Excerpt of the PXL class structure.

tures, in order to maintain the structure of their analysis specific event information. PXL objects contain single aspects of an individual event. In order to maintain information about a whole process, PXL provides the event view (pxl::EventView). The latter is a generalized object container where beyond the aforementioned physics objects user information can be stored as well. Providing the relation management for building up the decay trees, PXL enables efficient duplication of an already filled event container with all particles, relations and user information. This allow users, e.g., to comfortably build all ambiguous reconstruction versions in the reconstruction procedure of a decay tree from final state particles. A list of EventViews are combined in an Event (pxl::Event) (fig. 4.2), which holds all EventViews the user wishes to keep for a high energy collision event.

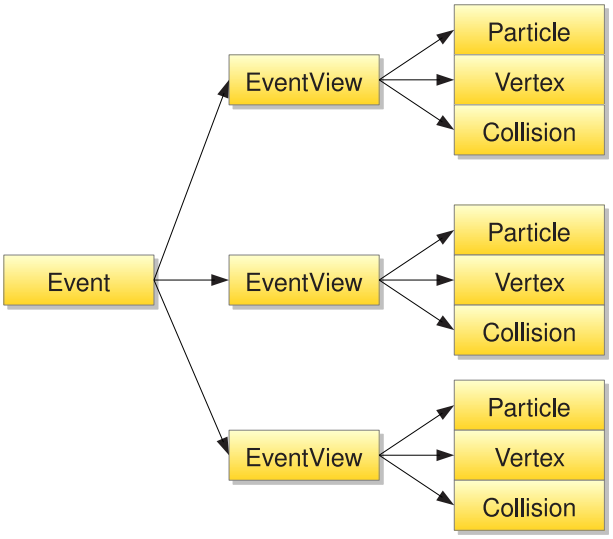


Figure 4.2: Example of a typical object hierarchy.

4.1.1 Universally Unique IDentifier

Each object has a universally unique identifier (UUID) which is created by pseudo random numbers and formatted according to the RFC4122 standard [12]. It is 16 bytes long and thus has $2^{128} \approx 3.4 \cdot 10^{38}$ possible values, what makes it practically impossible to create duplicates. The Mersenne Twister [13] random number generator is used to create the four 32 bit values for the UUID. In its canonical form it looks like this:

```
550e8400-e29b-41d4-a716-446655440000
```

UUIDs are also used for the type information in the serialisation code.

4.1.2 Relations

One of the key features of PXL is the relation management. Between all objects, PXL objects or user defined objects ultimately inheriting from pxl::Relative, relations can be established. There are two kinds of relations, “hard” and “soft” relations. Hard relations are only allowed between objects belonging to the same object owner, e.g. EventView, but are therefore always accessible via their pointer. They are used for mother-daughter relations and flat relations. The “top” particle would have a “W” particle and a “b” particle as daughters. Soft relations have no such restrictions, they are simple UUID pairs. This allows to have relations between any object ever created. They do not have to be in memory at the same time, they do not even need to be stored in the same file.

4.1.3 UserRecord

All Objects have a so called UserRecord, where the user can store arbitrary data. Each record is identified by a std::string and has a pxl::Variant as value. This Variant can store the following

types: bool, (unsigned) char, (unsigned) short, (unsigned) int, (unsigned) long, float, double, string and pointer. UserRecords are commonly used to store additional information like isolation values, b-tagging and so on. They are also used to steer analysis modules and algorithms.

4.1.4 Serialization and I/O

PXL provides methods for binary serialization. Every class that should be stored needs to inherit from pxl::Serializable and needs to implement the serialize (deserialize) method. They accept an pxl::InputStream (pxl::OutputStream) to (from) which they stream all information about the object. Additionally every class needs a unique identifier (UUID) for the deserialization process. Using a UUID instead of a string makes it much easier for developers to add new object types as they do not have to worry about how to pick a type name that was not already taken. In PXL all classes provide complete serialization methods allowing the user to store a whole event with just one command.

The PXL I/O files consist of isolated chunks. There are two kind of chucks, data chunks and information chunks. Information chunks can be used to separate different types of events inside a file. Data chunks contain one or more objects derived from pxl::Serializable. Chunks are normally compressed using the ZLIB[1] compression library, but it is also possible to not use compression. This design of isolated chunks was chosen to improve file robustness. PXL I/O files are still valid even if an analysis quits unexpected, as there is no global header or index that needs to be written or updated at the end.

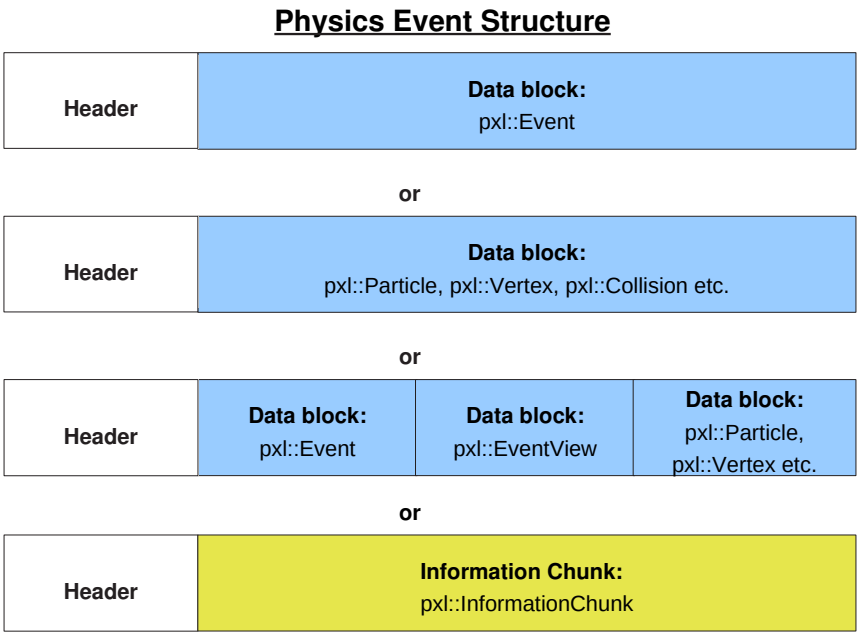


Figure 4.3: Possible chunk structures in PXL I/O files.

[1] http://http://www.zlib.net/

4.1.5 Physics Objects

Based on the infrastructure presented above, PXL provides easy-to-use physics classes. All have a name, a unique identifier, a UserRecord and are capable of having relations.

Events are highest in the class hierarchy and represent the physical event. Each event can be viewed from different perspectives though, like from the generator level or the reconstructed level. These different views are represented by EventViews. The Event is typically a collection of EventViews.

EventViews are classes that contain all information about a specific view of a physical event. The generator view for example contains the decay tree created with Monte Carlo methods while the reconstructed view contains the particles which are gained from the reconstruction software.

Particle This class represents any particle in the event. It additionally has a four-vector, particle id and a charge.

Vertex This class represents vertices and therefore has a three-vector.

Collision This is a simple helper class for multi-collision events.

4.1.6 Weak Pointer

Keeping pointers to objects owned by object owners, like the Event or EventView, in th euser code is not safe, as the pointer gets invalid as soon as the owner of the object pointed at is deletedand with it the object itself. To prevent segmentation faults, or even worse, pointers to wrong objects located at the same address when the memory is used for another object of the same type, the weak pointer is introduced. It is a helper class that acts like a normal pointer but in fact keeps track of the validity of the pointer. When the object the weak pointer points at is deleted, the weak pointer is invalidated. The user is then able to check if the weak pointer is valid and take appropriate actions. All classes in PXL can and should be handled with the help of the provided weak pointer class.

4.1.7 Python Interface

In high energy physics experiments the number and size of events that need to be processed can be very high, what requires the usage of a programming language allowing high performance computations, in the case of PXL C++ is used. This language on the other hand is hard to learn and handle. That is one of the reasons to make PXL available in the wide spread and easy-to-use and learn scripting language Python [14]. In order to make C++ classes available in Python, a lot of wrapper code is needed. And as Python is much slower than C++ this binding is mainly useful to steer the high performance C++ machine.

As there are no templates in Python all templated methods from PXL are only available for the types known to PXL. The name of the type is then appended to the function name:

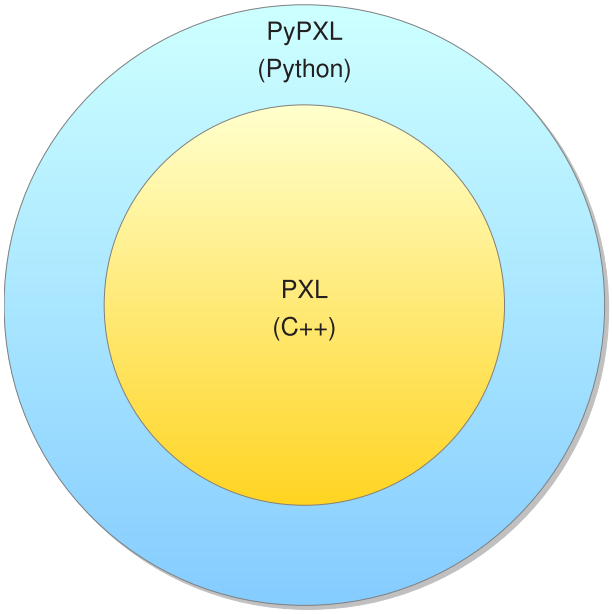


Figure 4.4: Python interface wrapped around PXL

```
findUserRecord<double>("key")
```

gets

```
findUserRecordDouble("key")
```

Additionally all getObjectsOfType template functions are specialized for all PXL classes:

```
getObjectsOfType<EventView>()
```

gets

```
getEventViews()
```

4.2 VisualPXL

VisualPXL was in the first place developed to ease the inspection of PXL I/O data files. At the time only console text output of the decay chain was possible. This output was hard to read and there was no easy way to get more details on the physics objects. Though the decay was visible through indentation it was not at all intuitive. Additionally there was no program to navigate the events in a file interactively. So the requested features were

- navigate the events in a file,
- visualize the decay,
- and access to all members of each object.

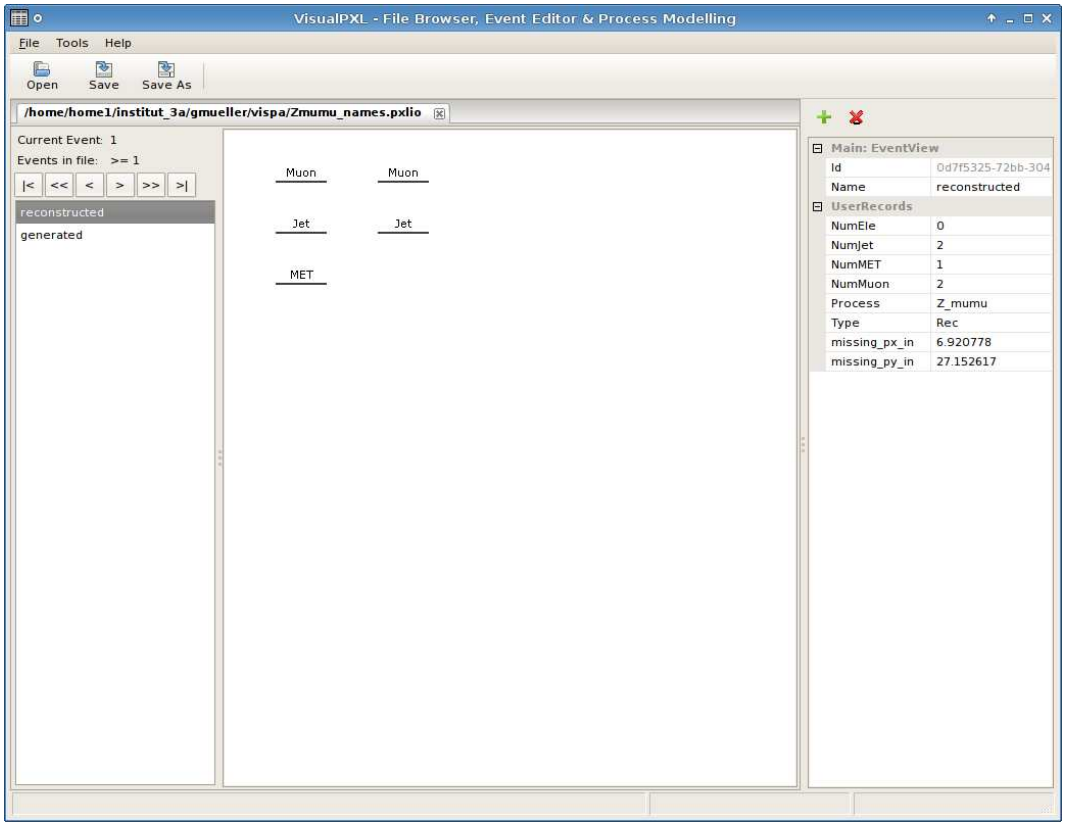


Figure 4.5: Screenshot of VisualPXL. Navigation panel on the left, EventView display in the center and the list of properties on the right.

4.2.1 Browsing

The layout of the graphical user interface reflects the fundamental requirements (Fig. 4.5). On the left side of the window the navigation panel is located. It provides information about the current event, the estimated event count in the files, buttons for the navigation (go to the first event in file, go 50 events back, go 1 event back, go to the next event, jump 50 events ahead, go to the last event in the file), and the list of the EventViews in the current event. As PXL I/O files have no index and no global header, it is not possible to know the exact number of events in a file without going through the whole file. The estimate event count thus only show the number of already visited Events.

The central panel (Fig. 4.5) visualizes the physics objects in the selected EventView. In its current implementation it supports the most commonly used objects, i.e. vertices and particles. Objects with relations are layouted and placed at the top of the panel. Objects without relations are lined up and placed below the layouted decays. The name of each object is displayed next to it. Each object can be selected by clicking on it.

The panel on the right side of the screen, called the PropertyGrid, lists all attributes of the currently selected object. If neither a particle nor a vertex is selected, the attributes of the selected EventView are displayed. Properties common to all objects are the name, the identifier and the user record. Vertices additionally have a three-vector, and particles a four-vector.

Figure 4.6: Screenshot of the PropertyGrid showing the members of a Particle.

4.2.2 Editing

Soon after the first implementation of the browsing functionality, the need to create and edit events arose as those can be used to steer analysis modules. One can either create a new event, or edit an existing one from the browser view. In editing mode the navigation panel is replaced by the editing panel. It contains a list of physics object which can be created (Vertices and Particles), buttons to create, delete and layout EventViews and the list of EventViews in the current Event. Relations can now be established via drag and drop. Vertices can be connected to the front and back of particles, whereas the front of a particle can only be connected to the back of another and vice versa. New UserRecords can be added by clicking on the plus icon right above the PropertyGrid. User records are often used to control analysis modules. That way special configuration options can be associated with each object.

VisualPXL is programmed in C++ using the wxWidgets[2] toolkit. And it incorporates the wxPropertyGrid[3] and the AntiGrain Graphics[4] libraries.

4.2.3 Automatic Layout of Decay Cascades

In order to display arbitrary decay chains a flexible layout algorithm is needed. The following requirements had to be met. First of all it should be very flexible and properly layout any

[2]http://wxwidgets.org/
[3]http://wxpropgrid.sourceforge.net/
[4]http://www.antigrain.com/

decay chain without further knowledge than the mother-daughter relations between the object. The algorithm should also be able to layout even huge decay chains like the ones generated by Pythia [15] with more than thousand particles. And last it should have no dependencies on other libraries than PXL so it can easily be integrated into existing applications.

As flexibility is the main issue, a non-deterministic approach was chosen. All vertices and joints between related particles are mapped to nodes with physical properties common to rigid bodies. Then forces and constraints are applied to these nodes and a physics simulation is run until the system reaches a stable state. Figure 4.7 outlines the algorithm. The initial layout can

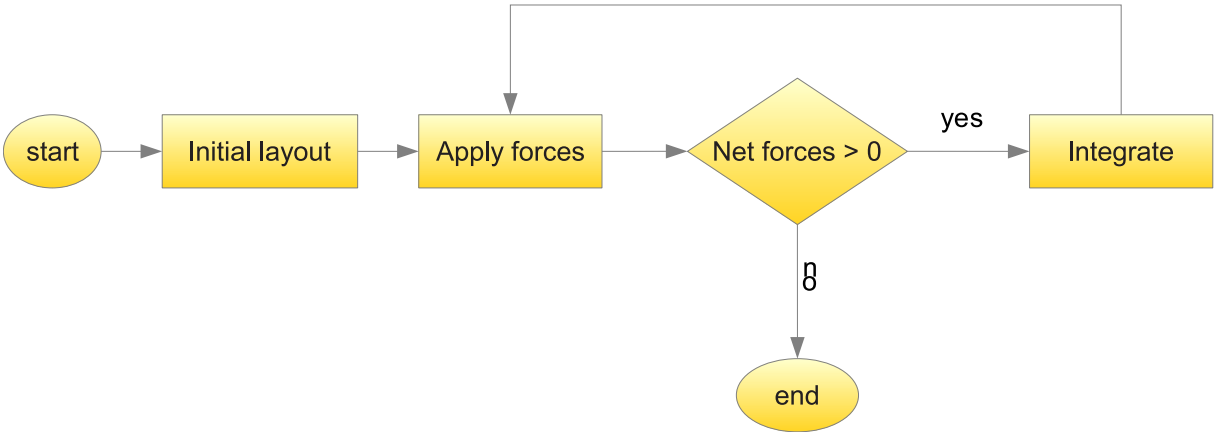


Figure 4.7: Outline of the algorithm to automatically layout decay chains.

be the existing layout information from previous calls to the algorithm or the layout the user provided when editing. If neither is present, a simple initial placing of the nodes takes place, where the child nodes are simply put right to their parents.

The first part of each iteration is to apply the following forces (Fig. 4.8):

- Constant forces to all nodes
- Repelling forces between close node
- Spring forces to all daughter nodes
- Friction forces to all moving nodes

The next step is to check if the system is in a stable state. This is the case if the magnitude of the net force for all nodes is close to zero.

If the system is not yet in a stable state the simulation integrates a fixed time step. For each node it calculates the future position based on the current velocity

$$\vec{x}_{i+1} = \vec{x}_i + \vec{v}_i \cdot \Delta t,$$

and the future velocity based on the currently acting forces

$$\vec{v}_{i+1} = \vec{v}_i + \frac{\vec{F}_{net}}{m} \cdot \Delta t.$$

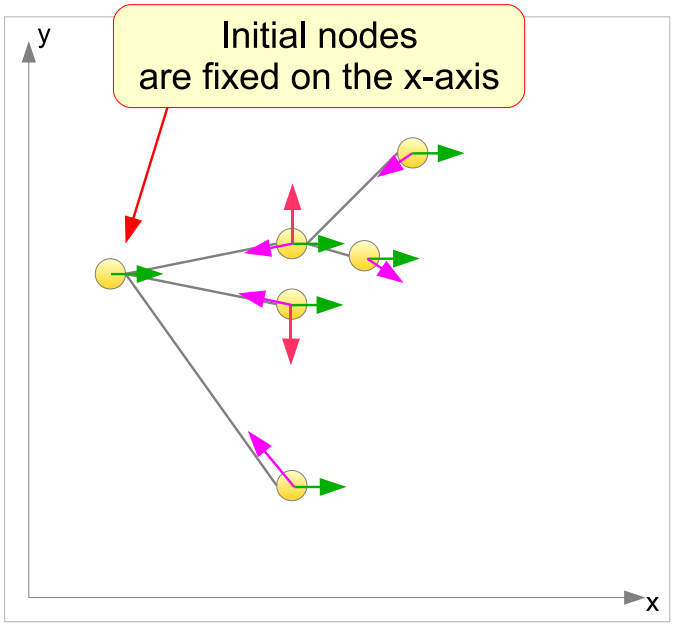


Figure 4.8: Illustration showing the different forces used in the layout algorithm

The basic algorithm so far is rather simple and easy to implement. The hard part is to find the optimal forces to apply to each node. There are a couple of things that lead to bad results.

"Explosions" can occur when distance-dependent forces are used in combination with discrete time steps as it is the case in this algorithm. To reduce this effect the magnitude of the applied forces can be limited or decrease the integrating time step.

"Wandering" of the whole decay or single nodes. This can happen when the friction forces are too low and asymmetric forces apply. This effect can, to a certain level, be reduced by increasing the friction force. Ultimately one has to reduce the asymmetric forces applied.

"Overlaps" of edges are hard to resolve once they emerge. The best approach is to use collision detection routines to avoid overlaps in the first place at all costs.

"Back pointing" - daughter nodes being left o ftheir mother nodes. This can happen when the spring forces are much stronger than the gravitational ones.

When dealing with 1000 or more nodes, performance can get an issue as well. Proximity checks for example require $N(N-1)/2$ comparisons. With 1000 nodes this are roughly half a million comparisons each time step. To overcome this huge performance hit spatial partitioning is required. In its current implementation the node hierarchy is also used for space partitioning. Each node is described by a rectangle which contains itself and all child nodes. On each proximity check the tree is traversed and only nodes where the particle to be matched lies in its rectangle is considered further.

Figure 4.9 shows a screenshot of a successfully layouted $t\bar{t}$ event. In figure 4.10 a screenshot of a layouted decay chain created by Pythia is shown. Due to the large number of particles some parts of the decay are not layouted properly.

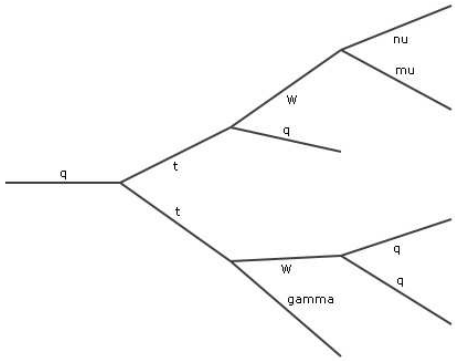


Figure 4.9: Screenshot of an automatically layouted $t\bar{t}$ event.

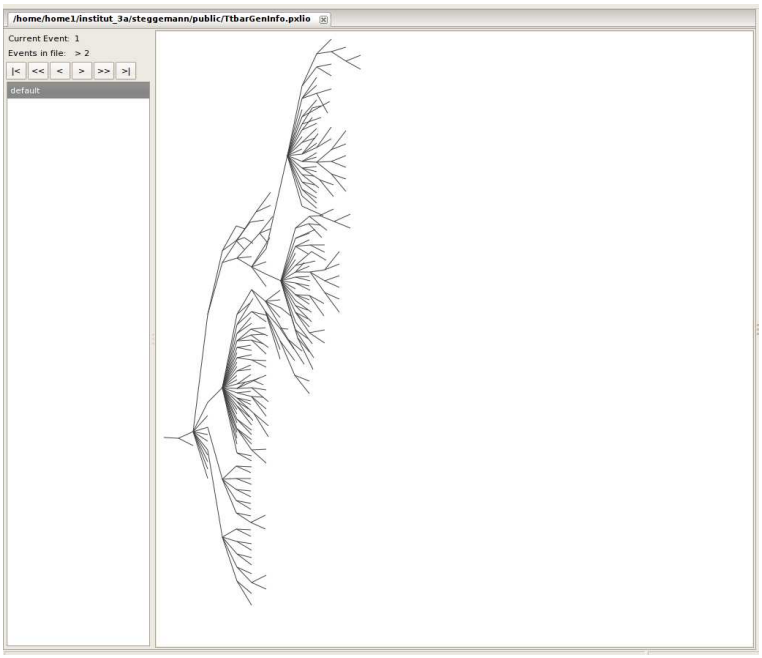


Figure 4.10: Screenshot of a Pythia event layouted in VisualPXL

5 Modular Physics Analysis Design and Graphical Steering

5.1 Modular Analysis Structure

One requirement for graphical analyses is the modularity of the analysis routines. Huge experiment specific software like CMSSW uses modules for the different analysis steps as well. Without those the complexity of such software would barely be manageable. But even in smaller settings modular programming has great benefits. It reduces code redundancy and therefore increases the reusablity of user code. This also leads to more tested code, as it is reviewed by a larger user base. But modularity itself is not yet sufficient for a graphical approach. The configuration of the modules also needs to be standardized, so they can be configured without prior knowledge.

5.1.1 Common Module Interface

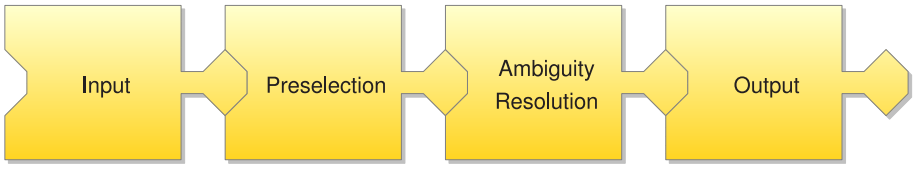


Figure 5.1: Multiple modules connected.

Each module has multiple input and output ports, called Sinks and Sources which are used to pass events from one module to another, and are identified by their names. Modules can be queried for a list of available Sinks and Sources and a list of available options. That way the list of options is contained in the module itself what reduces the required maintenance as there is no need to keep the definition and implementation synchronised. Options have a name, a type and a usage flag. The type may be a string, a vector of strings, a double, a long, or a boolean. The usage flag instructs the graphical user interface (GUI) how to present each option. Beside the default only the “filename” usage flag is currently defined. If this flag is set the GUI will show a file selection dialog when this option is edited.

The ability to have multiple Sinks and Sources enables the user to create complicated analyses with multiple paths and module configurations.

5.1.2 I/O Module

The most basic modules are the file input and file output modules. The file input module reads the next event from the file and puts it into the Source named “default”. The name of the file to be opened is set via the option named “filename”. The first event to be read can be set by the option “start”.

The output module stores the event incoming on the Sink “default” to the opened file. The name of the file to store the events to is again set via the option named “filename”.

5.1.3 Script Module

The script module allows the physicists to write analysis modules in the user friendly scripting language Python [14]. All classes and features of PXL can be used. 10 Sinks and 10 Sources are provided by the module. The scripting module calls three functions in the script if they are available.

start This function is called right after the initialisation of the module. It has one parameter: a list of parameters for this instance of the script.

evaluate This function is called for every Event that is passed to one of the 10 Sinks of the module. The first parameter is the Event and the second one is the number of the Sink the Event was passed to. The return value of the function determines what happens to the Event. When a value of 0 is returned the Event is dropped, while values of 1 to 10 will pass the Event to the corresponding Source. This allows the usage of one script module in multiple paths and therefore to split or merge a path.

finish This function has no parameters and is called just before the module is shut down.

A script simply printing the number of EventViews in each Event to the console looks like this:

```
def start (parameters)
    print 'starting ...'

def evaluate (event, sink):
    print 'Number of EventViews: ', len(event.getEventViews())
    return sink

def finish ()
    print 'finishing ...'
```

In this script the Event is passed to the Source with the same number as the Sink. That way this module can be used in 10 different paths at the same time. To merge all 10 incoming paths into one, the script should always return the same Source number.

5.2 Plugins

Modules can either be compiled statically into the analysis program or they can be loaded at runtime which is much more flexible. Multiple modules are therefore packed into so-called plugins which are basically shared libraries with a special function to query all modules in the plugin. One issue with shared libraries is the creation and deletion of objects. As the new and

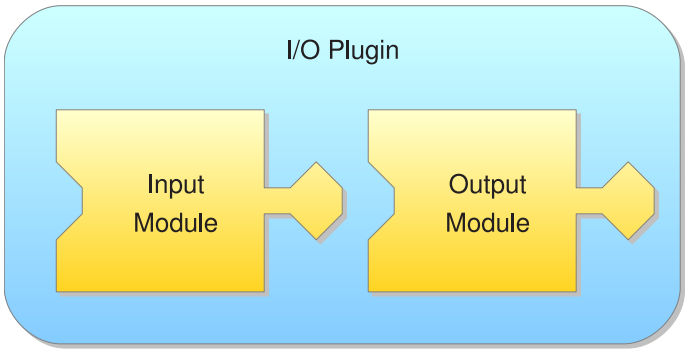


Figure 5.2: One plugin can contain multiple modules. The I/O plugin contains the input and output module.

delete operators maybe overloaded it is not safe to destroy objects in the calling code using the new and delete operators. Therefore the modules are created by a factory class and need to be deleted by their destroy function.

5.3 Visual Steering

Now that an analysis can be assembled by modules with a common interface it is also possible to design it with graphical tools. On the left side of the screen in figure 5.3 all modules found on

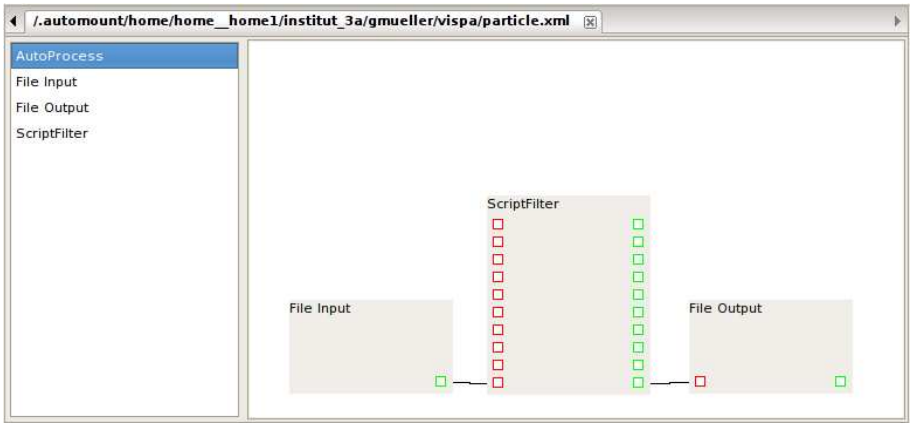


Figure 5.3: Screenshot of VisualPXL displaying a simple analysis consisting of 3 modules.

the current system are listed. The panel in the center shows the modules and their connections in the current analysis. Each module is represented by a box with its name and a list of Sinks

Figure 5.4: Screenshot of a script module in VisualPXL. At the top the name of the module is printed. On the left, in red, are the Sinks and in green and on the right are the Sources.

and Sources in it. Connections between Sinks and Sources are represented by lines going from the box of the Source to the box of the Sink. A new module can be added by double clicking on the center panel. A new instance of the selected type is created and positioned at the current position of the cursor. The module can then be moved a around by dragging it to the desired position. Connections between Sinks and Sources can be established by clicking on the Source and dragging the newly created line to the desired Sink. The Sinks are located on the bottom left and the Sources on the bottom right of each module box. The PropertyGrid on the right lists the name, the type and all options of the selected module.

5.3.1 XML Export

The designed chain of configured modules can be saved as XML file. This file has the following structure. The top level element is called “analysis”. It contains multiple “module” and “connection” elements. The “module” element has four attributes

name should describe what this module does in the analysis, e.g.“preselection”,

type specifies the type of the module, e.g. “ScriptModule”,

xPos (optional) is the position on the x axis when displayed in the GUI,

yPos (optional) is the position on the y axis when displayed in the GUI,

and it has multiple “option” child elements, containing the configuration of the module. This “option” element has the following attributes

name of the option

type of the option

and has a child text element with the value of the option.

The connection element has no attributes but two child elements: “sink” and “source”. Each of them has the attribute “name”, representing the name of the Sink or Source and the attribute “module” containing the name of the module the Sink or Source belongs to.

The XML file of the analysis displayed in figure 5.3 looks like this:

```
<analysis>

    <module name="input" type="File Input">
        <option name="filename" type="string">
            input.pxlio
        </option>
    </module>

    <module name="output" type="File Output">
        <option name="filename" type="string">
            output.pxlio
        </option>
    </module>

    <connection>
        <source name="default" module="input" />
        <sink name="default" module="output" />
    </connection>

</analysis>
```

5.4 Executing the Analysis

The analysis chain stored in the XML files can be executed by a program that reads the XML data from the file and created instances of the modules and establishes the given connections between the Sinks and Sources. This can be done from within an existing program, like CMSSW, or from a standalone tool which itself runs inside as batch job.

Figure 5.5: Workflow executing graphically designed analyses.

5.5 Python based steering

A third alternative, besides C++ and XML, is to steer the modules from Python. A Python wrapper of the plugin and module system is provided enabling the physicists which know the language to create powerful analysis chains an a very flexible way. Included is also a main loop,

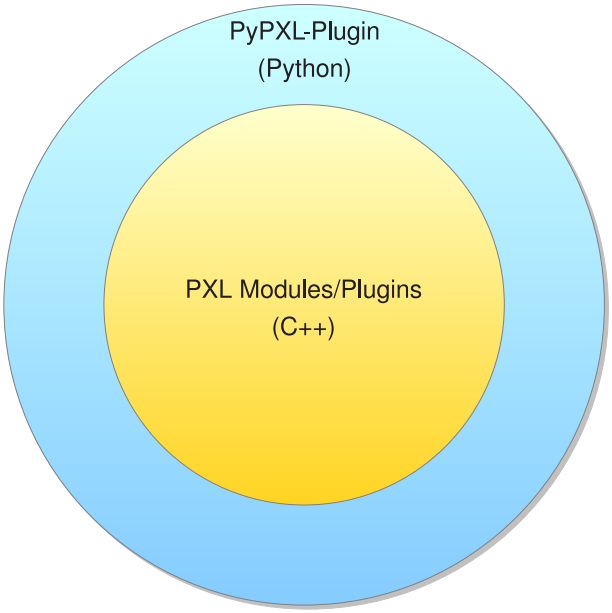


Figure 5.6: Python shell around the PXL plugin and module interfaces.

so that all the user needs to do is create and configure modules and connect them.

The analysis displayed in figure 5.3 can be programmed in Python with the following code:

```
from pxlplugin import *

manager = PluginManager()

inputmodule = manager.create('File Input')
inputmodule.setOption ('filename', 'input.pxlio')

scriptmodule = manager.create('ScriptModule')
scriptmodule.setOption ('filename', 'script.py')

outputmodule = manager.create('File Output')
outputmodue.setOption('filename', 'output.pxlio')

inputmodule.getSource('default').connect (
    scriptmodule.getSink('default'))

scriptmodule.getSource('default').connect (
    outputmodule.getSink('default'))

processor = PluginProcessor()
```

```
processor.add (inputmodule)
processor.run()
```

6 User Analysis Code and Physics Modules

The acceptance of analysis software very much depends on how easy the user can control and extend it. The module interface was designed to allow great flexibility and extensibility. In this chapter two examples are presented, one for high level analysis using Python scripting and a high performance C++ module for automated event reconstruction.

6.1 Python Modules

One of the reasons to use Python as the scripting language is that it is wide spread and therefore offers huge access to all kind of libraries and frameworks, with ROOT probably being the best known one. Figure 6.1 illustrates the dependencies of PXL, Python, ROOT, pyPXL and PyROOT.

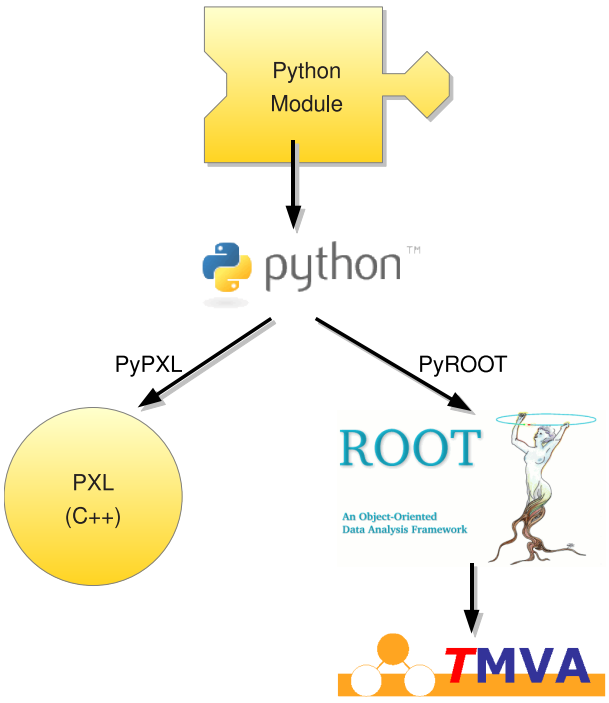


Figure 6.1: PXL, Python, PyROOT, TMVA and their dependencies [16,17]

6.1.1 $Z \rightarrow \mu\mu$

A simple $Z \rightarrow \mu\mu$ analysis is used as example for the power of the scripting module. Samples from the CMS experiment were exported to the PXL I/O format. A "File Input" module is used to read the events from disk and are then passed to a "ScriptFilter" which will reconstruct the Z mass from the two muons with the highest transverse mass. Histograms of the transverse momentum of the used muons and the mass of the reconstructed Z boson are then created.

The first few lines of the script import functions from ROOT and book the the histograms for the transverse momentum of the muons and the mass of the Z boson.

```
from ROOT import gROOT, TH1F, TCanvas
from math import sqrt

gROOT.Reset()

h_pt    = TH1F ("h_pt", "muon transverse momentum", 40, 0, 100)
h_Zmass = TH1F ("h_Zmass", "reconstructed Z mass", 50, 0, 150)
```

The start function just prints a message to the console. That way one can be sure it is correctly initialized when executing the analysis.

```
def start (parameters):
    print "*** starting analysis "
```

The evaluate function is the most important part of the script. Here the EventView with the name 'reconstructed' is looked up and a list of all Particles with the name 'Muon' is extracted from it. The event is dropped if it contains less then 2 muons. If 2 or more were found they are sorted by their transverse momentum. The p_T values of the the highest muons are filled into the corresponding histogram. And finally the mass of the Z boson is calculated from those 2 muons and filled into the second histogram.

```
def evaluate (event, sink):
    for eventview in filter(lambda e: e.getName()
    == 'reconstructed',  event.getEventViews()):
        muons = filter(lambda p: p.getName()
        == 'Muon',  eventview.getParticles())

    if muons == None or len(muons) < 2:
        return 0

    muons.sort (lambda a, b: -cmp(a.getPt(), b.getPt()))

    h_pt.Fill(muons[0].getPt())
    h_pt.Fill(muons[1].getPt())
```

```
    Zmass2 = muons[0].getE() * muons[1].getE()
           - muons[0].getPx() * muons[1].getPx()
           - muons[0].getPy() * muons[1].getPy()
           - muons[0].getPz() * muons[1].getPz()
    h_Zmass.Fill(sqrt(Zmass2))

    return sink
```

The finish function is called when all events are read from the input file. The histograms are now printed and stored on disk.

```
def finish():
    print "*** finishing analysis"

    c1 = TCanvas('c1')
    c1.Divide(2,1)
    c1.cd(1)
    h_pt.Draw()
    c1.cd(2)
    h_Zmass.Draw()
    c1.Print("Zmumu.eps")
```

In figure 6.2 the resulting distributions are shown.

6.2 C++ Modules

As using PXL through the Python interface is a lot slower than using it directly from C++ it is not suited for CPU intensive tasks. But as it is not trivial to write efficient C++ code there is a need for high performance modules written in C++ which are generic and easily configurable. If these modules implement the common module interface they are also usable from within the graphical analysis tool. On the next pages such a module is presented.

6.2.1 AutoProcess - Automated Hypothesis Generation

In high energy physics experiments the whole decay cascade of heavy particles often needs to be reconstructed from the final state particles measured in the detector [18]. But the reconstructed particles can, in many cases, not be assigned to the corresponding parton without ambiguities. In figure 6.3(a) a screenshot of a so-called parton picture template modelling the decay cascade of a t-channel single top event is shown. The list of reconstructed particles from such an event is shown in picture 6.3(b). The “Muon” and “MET” particles can be assigned to the corresponding partons (“lepton” and “neutrino”) in the template without ambiguities. But as the light quark as well as the b quark are detected in the form of jets, it is not obvious which jet corresponds

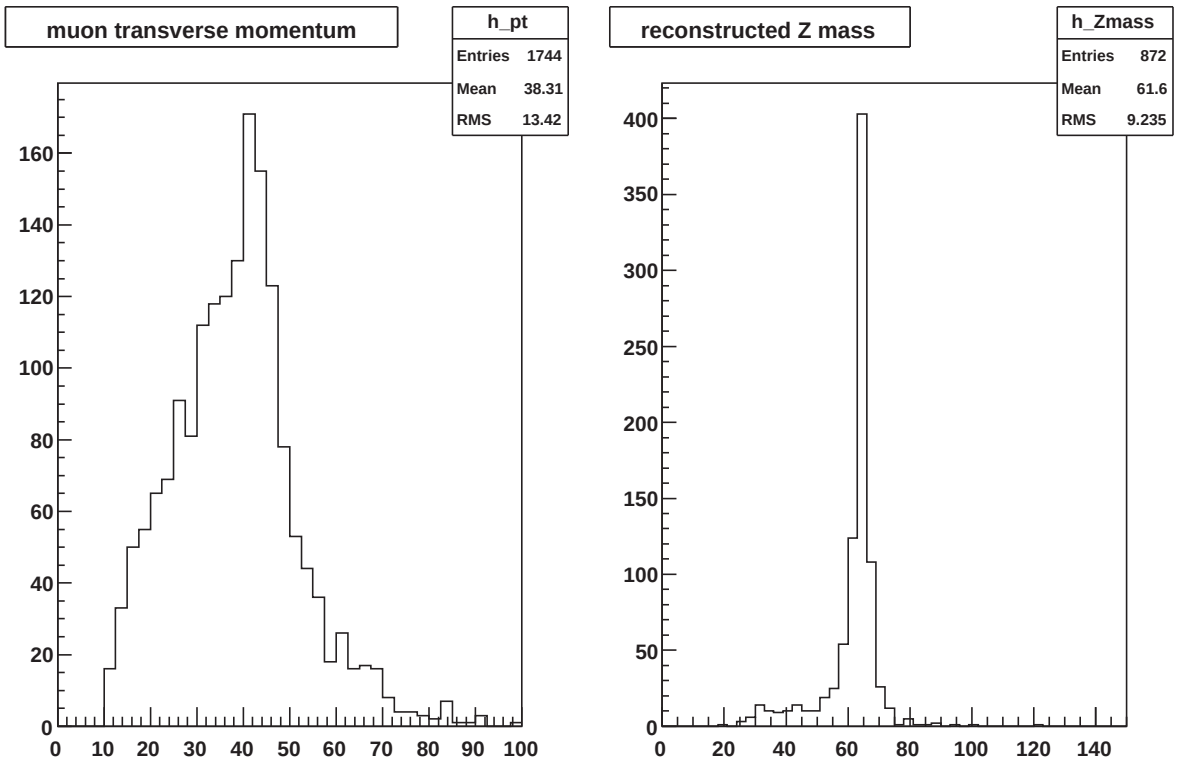
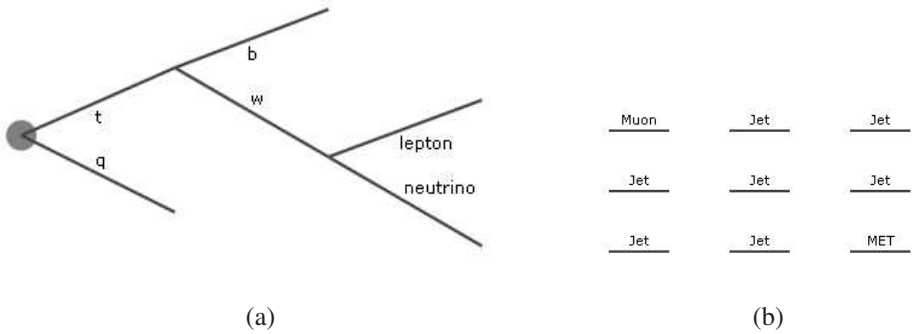


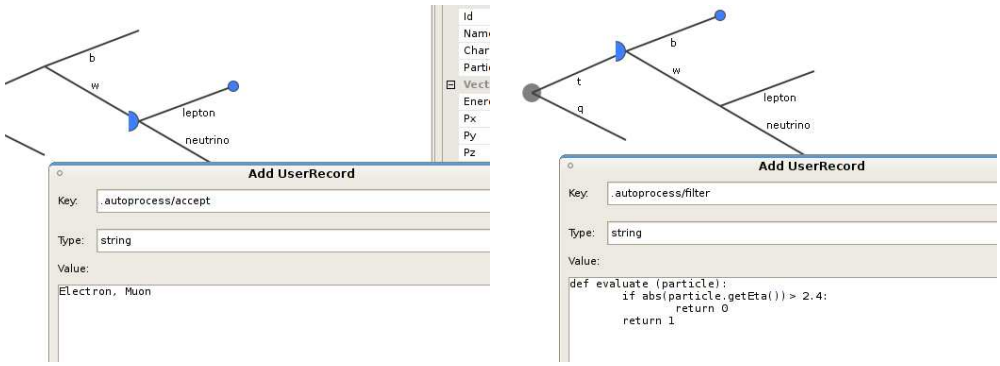
Figure 6.2: Transverse momentum of the two muons used to reconstruct the Z boson. And the mass of the reconstructed Z boson.



(a) (b)

Figure 6.3: Parton picture template (a) and a list of reconstructed particles (b) of t-channel single top decay.

to which parton. Preselecting cuts and b-tagging can reduce the number of ambiguities but in many cases they can not completely be removed. Which reconstructed particle should be assign to which parton in the template is defined in the UserRecord ".autoprocess/access" of the parton, see figure 6.4(a). Additionally parton based cuts maybe applied. In figure 6.4(b) a script limiting the jets treated as the one coming from the b quark is shown. Here the pseudorapidity is required to be less than 2.4. These cuts are defined in the form of small Python scripts which are executed for each reconstructed particle with the required name for that parton. The script has the same structure as the ones used in the script modules. The only difference are the parameters for the evaluate function. Here a reference to the particle is passed. The return value of the evaluate function decides if the reconstructed particle is considered as candidate for the parton containing the script in its UserRecord. For all possible permutations a copy of the



(a) (b)

Figure 6.4: Screenshot of VisualPXL showing how to edit the UserRecord controlling the assignment of particles (a) and editing a parton based cut ($|\eta(\mathrm{bquark})| < 2.4$) in the form of a Python script (b).

template is made and the final state partons are replaced by the reconstructed particles according to their names and cuts as described before. In addition this module is capable of reconstructing neutrinos from missing transverse energy information. The neutrino reconstruction is enabled by the UserRecord ".autoprocess/mass_constraint" which tells the algorithm what mass should be used for the W boson as well. A new hypothesis is added for each p_z solution of the neutrino. When all partons are replaced the four-vectors of the parent particles are recalculated based on the kinematics of their children.

Another feature of this module is the ability to compare the created hypotheses with generator decays if available. The name of the parton in the generator decay the reconstructed particle should be compared with is stored in the according template partons UserRecord with the name ".autoprocess/compare". If, e.g, the reconstructed b jet particle should be compared with the parton named "bquark" the UserRecord would contain "bquark". The distance in the (η, ϕ) space is calculated and stored in the UserRecord of the reconstructed particle. The sum of all compared particles is stored for each hypothesis in the UserRecord of the EventView.

7 A Demonstration Analysis of Electroweak Top Quark Production

The purpose of this analysis is a proof of concept: complex analyses can be done using the presented graphical tools and they provide correct results. The analysis consists of 3 consecutive parts due to the various steps needed in multivariate analysis training. Each step is designed using VisualPXL and a screenshot of each diagram is shown before each step is described on the following pages.

The following software packages were used

- CMSSW 1_6_12
- PXL and Accessories rev. 262
- Python 2.5.2
- ROOT 5.20.0
- TMVA 3.9.5

7.1 Event Samples

All samples have been filled from PAT Layer 1 objects to PXL objects. Additional information, like isolation parameters, is stored in the UserRecord.

7.1.1 Signal events of t-channel single top production

For the single top quark signal the MC@NLO event generator was used [19] CMS detector simulation and reconstruction were applied. The sample is stored in the CMS DBS system under the name

`/mcatnlo_tbq_mu/CMSSW_1_6_7-CSA07-1203846994/RECO`

It was produced with a single top cross section in the t-channel with the W-decay to μ, ν of 27 pb and 44503 events have been filled to PXL I/O files from PAT Layer1 objects, corresponding to roughly 1700/pb integrated luminosity.

This sample contains several errors and one of the consequences is that there is no b-tagging available. Also the random number generator was incorrectly treated. This event sample was the only sample available at this time with the current CMS software version, and it has been verified as far as possible for the use in this demonstration analysis.

7.1.2 Background

As background a so-called muon soup sample is used which contains $t\bar{t}$, Z + Jets and W + Jets events with the counts and weights listed in table 7.1. The whole sample contains about 1 million events of which one quarter were filled to PXL I/O files resulting in an integrated luminosity of 250/pb. The corresponding data files are listed as

```
/CSA07Muon/CMSSW_1_6_7-CSA07-Chowder-A3-PDMuon
-ReReco-100pb-Skims4-topSemiLepMuon/USER
```

in the CMS DBS system.

7.2 Preparation of the Samples

The first step of the analysis is the preparation the signal and background samples, see figure 7.1 for a screenshot of the diagram in VisualPXL. Both signal and background samples are read from PXL I/O files and passed event by event through the same preselection. Then they are passed to automatic event reconstruction modules. The only difference between the background and signal event reconstruction modules is that the one for the background does not do a matching with the generator samples. The signal events are additionally passed to the TMVA training and a plotting module.

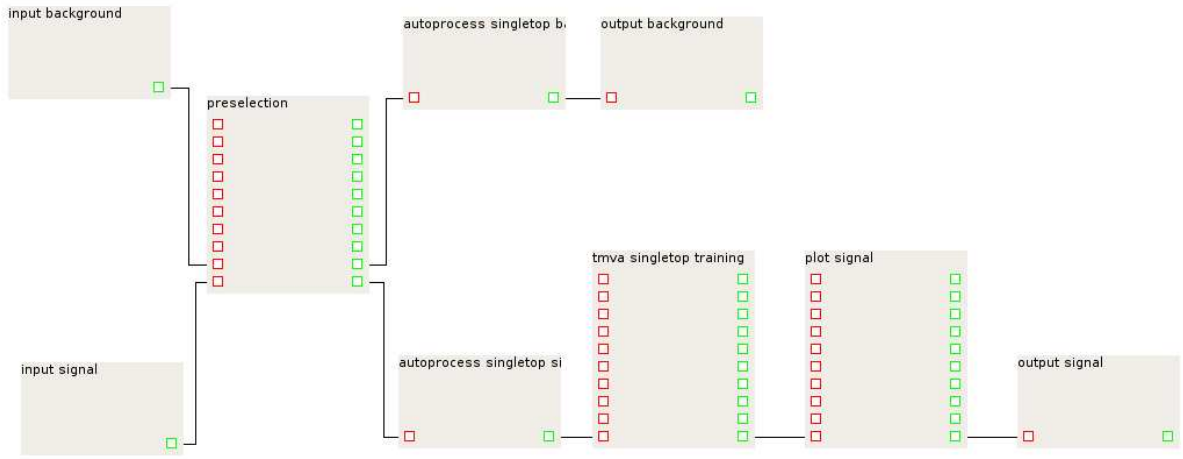


Figure 7.1: Screenshot of the diagram of the first part of the analysis.

7.3 Preselection

A common set of preselection criteria was used in this analysis [20]. First the event is required to have at least 40 GeV of missing transverse energy. The next requirements are two jets with

(a) W+Jets

ProcessID	Weight	# Events
1000	5.140	523
1001	1.017	72301
1001	1.006	11
1001	1.038	7999
1002	1.013	9
1002	0.783	11614
1002	1.066	40725
1003	1.666	8407
1003	0.913	4927
1003	0.987	8
1004	0.982	3839
1004	0.977	4
1004	0.952	1799
1005	1.010	4
1005	1.354	2268
1005	0.903	1171

(b) Z + Jets

ProcessID	Weight	# Events
2000	1.384	2426
2001	0.984	12644
2001	0.830	1426
2002	0.982	1
2002	0.799	1299
2002	0.933	7588
2003	0.943	2720
2003	0.531	1340
2003	0.975	1
2004	0.635	8
2004	0.417	633
2005	0.855	603
2005	0.717	11

(c) $t\bar{t}$

ProcessID	Weight	# Events
3000	0.425	45734
3001	0.488	9349
3002	0.415	3386
3003	0.422	457
3004	0.277	334

Table 7.1: Composition of the background soup.

more than 35 GeV p_T and a pseudorapidity of less than 5. And last, a lepton with $p_T > 20$ GeV and $|\eta| < 2.4$ is required.

These cuts reduce the signal sample to 1834 events, according to a signal efficiency of $\approx 4.1\%$. The background sample is reduced to 6797 events, so the background rejection for these cuts is about 97.3 %.

To identify the jet coming from the b quark b-tagging is normally used. But as this information is missing in this sample another condition introduced. Figure 7.2 shows the pseudorapidity distribution of the b quark as a function of the light quark pseudorapidity. One can see that in most events the pseudorapidity of the light quark is above 2 while at the same the the pseudorapidity of the b quark is less than 2. This suggest a cut for the light quark of $|\eta| > 2$ and $|\eta| < 2$ for the b quark. These exclusive cuts also remove ambiguities but reduce the signal efficiency to about 4.0% or 1770 events.

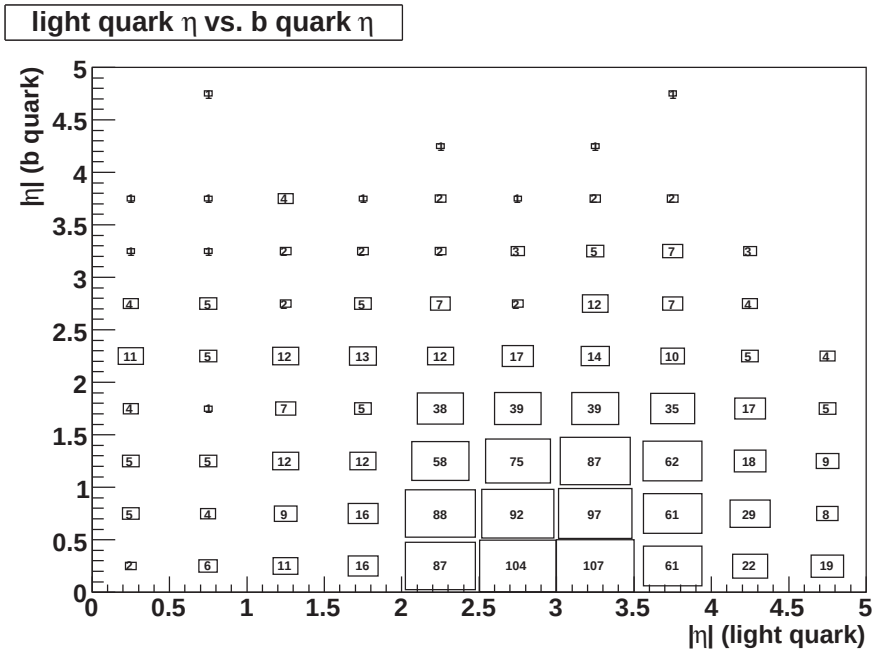


Figure 7.2: Pseudorapidity distribution of the b quark vs light quark on generator level.

In some events additional jets were reconstructed that have exactly the same direction as the reconstructed leptons but have a very low mass. These jets are also removed by a preselection script under the following conditions. The cone size between the jet and the lepton needs to be less than 0.5 and the mass of the jet less than 8 GeV.

7.4 Generator

To understand and verify the signal sample the following plots were produced after the preselection. In figure 7.3 the 3 kinematic variables p_T, η and ϕ of the light quark are plotted. In figures 7.4, 7.5 and 7.6, the same kinematic variables are shown for b quark, the lepton and the neutrino.

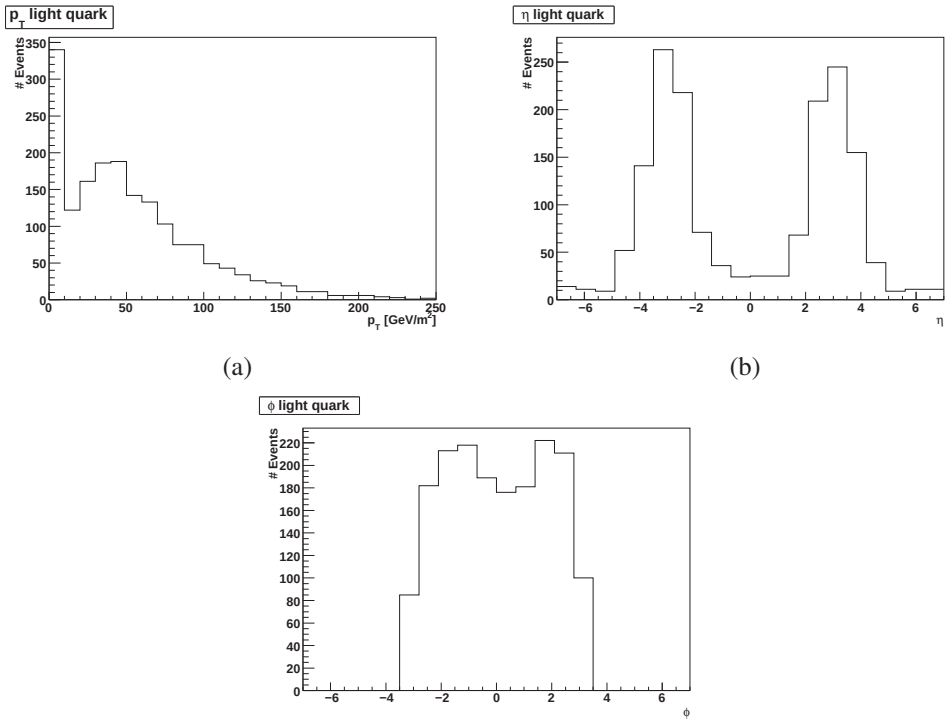


Figure 7.3: Kinematic variables of the light quark on generator level: a) p_T, b) η and c) ϕ

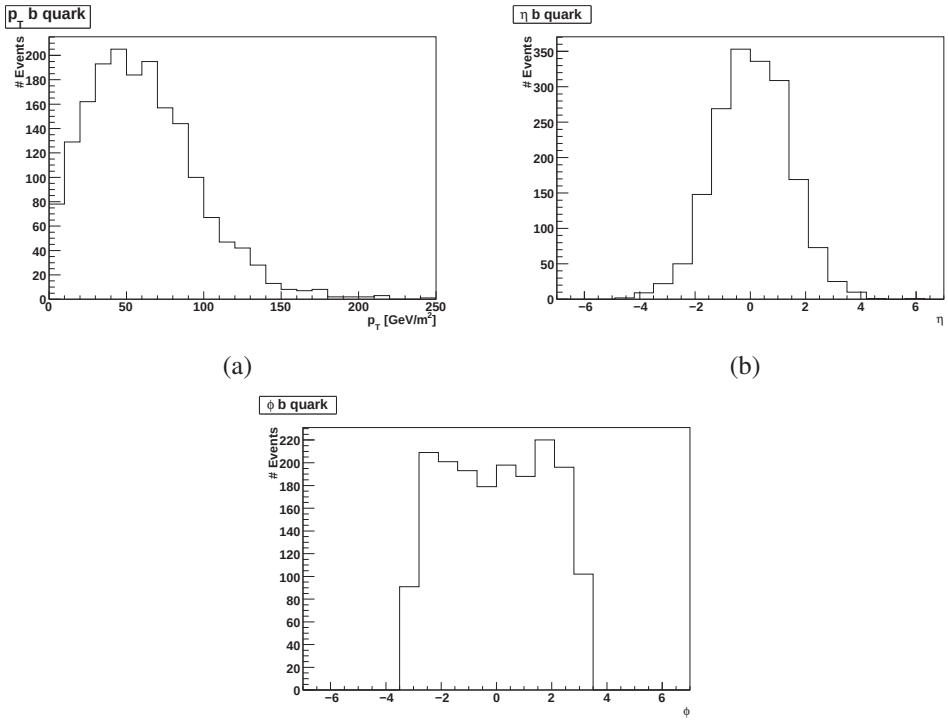
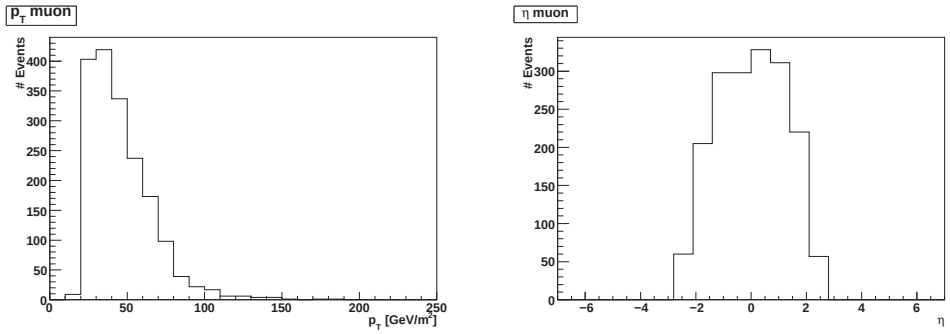
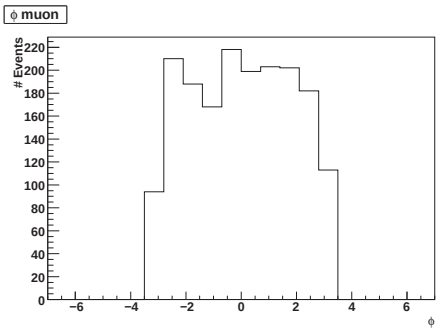


Figure 7.4: Kinematic variables of the b quark on generator level: a) p_T, b) η and c) ϕ



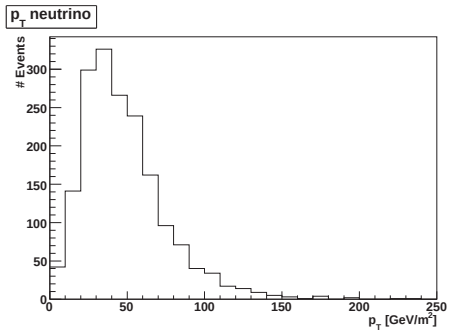
(a)

(b)

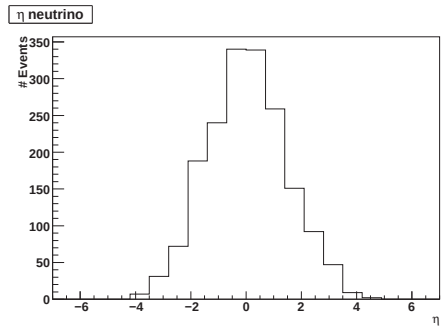


(c)

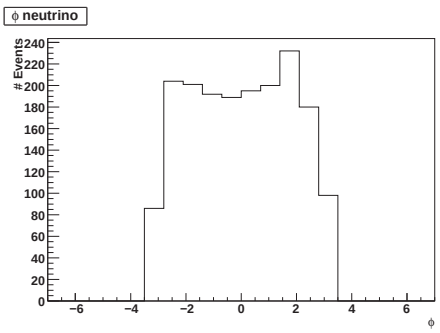
Figure 7.5: Kinematic variables of the muon on generator level: a) p_T, b) η and c) ϕ



(a)



(b)

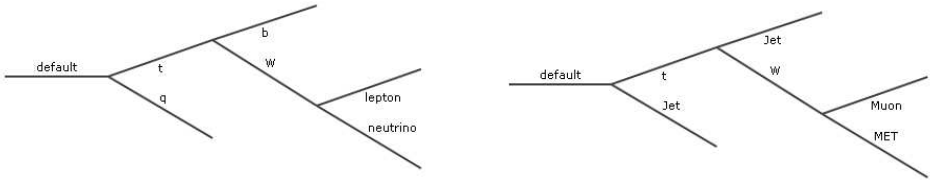


(c)

Figure 7.6: Kinematic variables of the neutrino on generator level: a) p_T, b) η and c) ϕ

7.5 Event Reconstruction

For the neutrino p_z solutions and easy access to event shape variables the AutoProcess module is used. The parton picture template in figure 7.7(a) is used as steering card and incorporates the two η cuts for the jets in form of Python scripts. In figure 7.7(b) the matched reconstructed particles are attached to the decay cascade. Technically they are flagged by the UserRecord with the key ".autoprocess/accept" in the template. The two hypotheses generated by the AutoProcess are stored as EventViews in the same Event. Their names are "AutoProcess" and the one with the smallest ΔR value is called "AutoProcess BestMatch". Here ΔR denotes the sum of the distances in (η, ϕ) space between the reconstructed objects and the final state quarks and leptons respectively.



(a) (b)

Figure 7.7: a) Parton picture template used for the AutoProcess. b) A copy of the template where the reconstructed particles are placed at the corresponding positions.

In figures 7.8, 7.9, 7.10 and 7.11 the kinematic variables p_T, η and ϕ and the difference to their corresponding partons are shown for the light jet, b jet, lepton and neutrino. The distributions of the muon and the b jet are as expected but the p_T distribution of the light jet and the reconstructed neutrino are obviously shifted to higher values. This may be caused by missing corrections in the reconstruction software.

The neutrino p_z solutions are extracted from the quadric equation

$$M_W^2 = 2\left(E_\mu\sqrt{p_{z,\nu}^2 + \not{E}_T^2} - \vec{P}_{T,\mu}\cdot\vec{\not{E}}_T - P_{z,\mu}P_{z,\nu}\right) \tag{7.1}$$

which has the two solutions:

$$p_{z,\nu}^{(1,2)} = \frac{Ap_{z,\mu} \pm \sqrt{\Delta}}{p_{T,\mu}^2}. \tag{7.2}$$

With $A = \frac{m_W^2}{2} + \vec{p}_{T,\mu}\cdot\vec{\not{E}}_T$ and $\Delta = E_\mu^2(A^2 - \not{E}_T^2 p_{T,\mu}^2)$.

To avoid negative values for Δ, which would cause imaginary values for p_z, the W mass is increased until a valid solution is found. The W mass distribution is plotted in figure 7.12(a), the top quark mass in figure 7.12(b). The m_{top} plot contains a second distribution on which a ΔR cut was performed to see how well the reconstruction worked. In about 50% of the events ΔR has negative values leading to an increased m_W. This effect is caused by the finite resolution of the detector and additions to the missing energy like neutrinos from meson decays and particles outside acceptance [20].

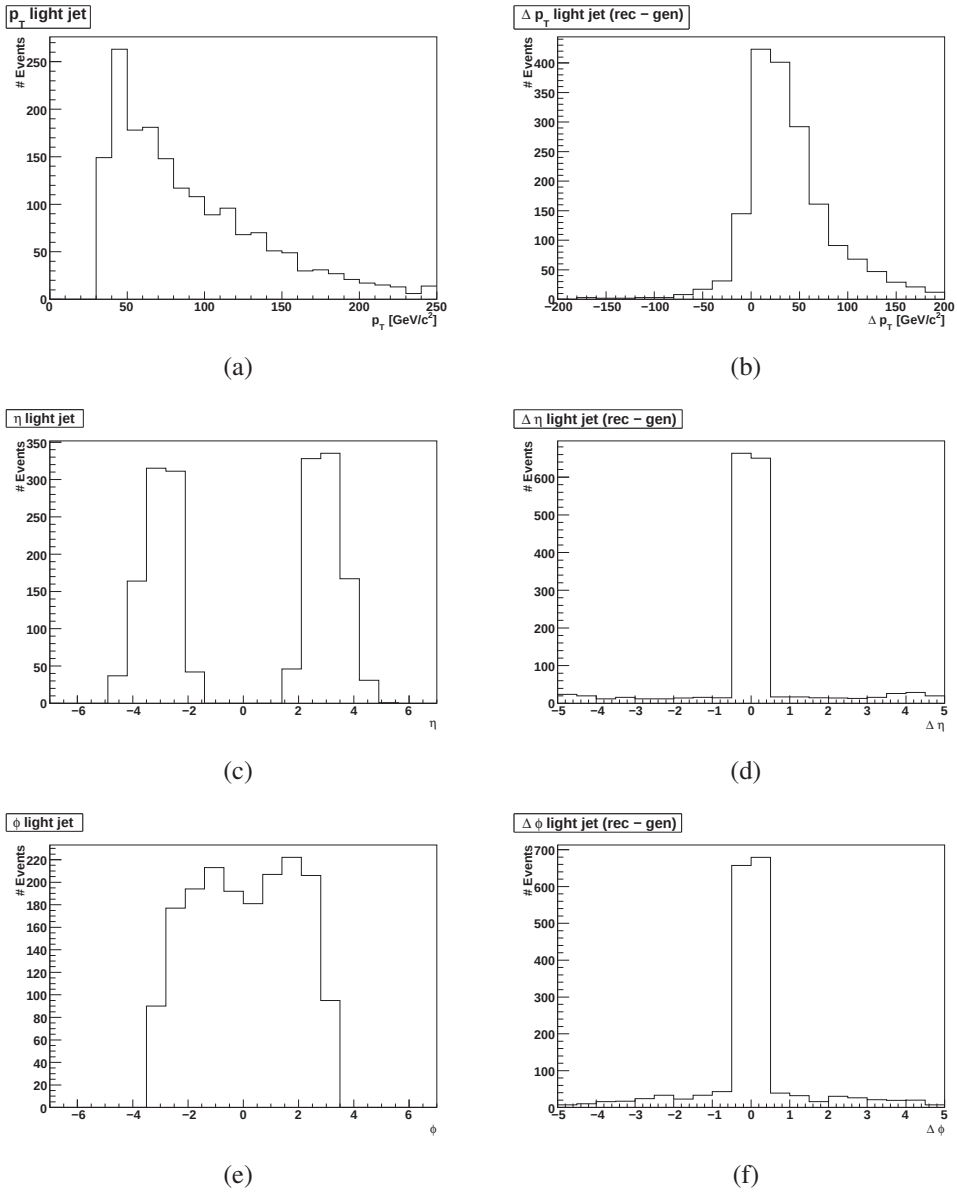


Figure 7.8: Kinematic variables of the reconstructed light jet. a) p_T, b) p_T difference between rec. particle and gen. parton, c) η, d) η difference between rec. particle and gen. parton, e) ϕ and f) ϕ difference between rec. particle and gen. parton

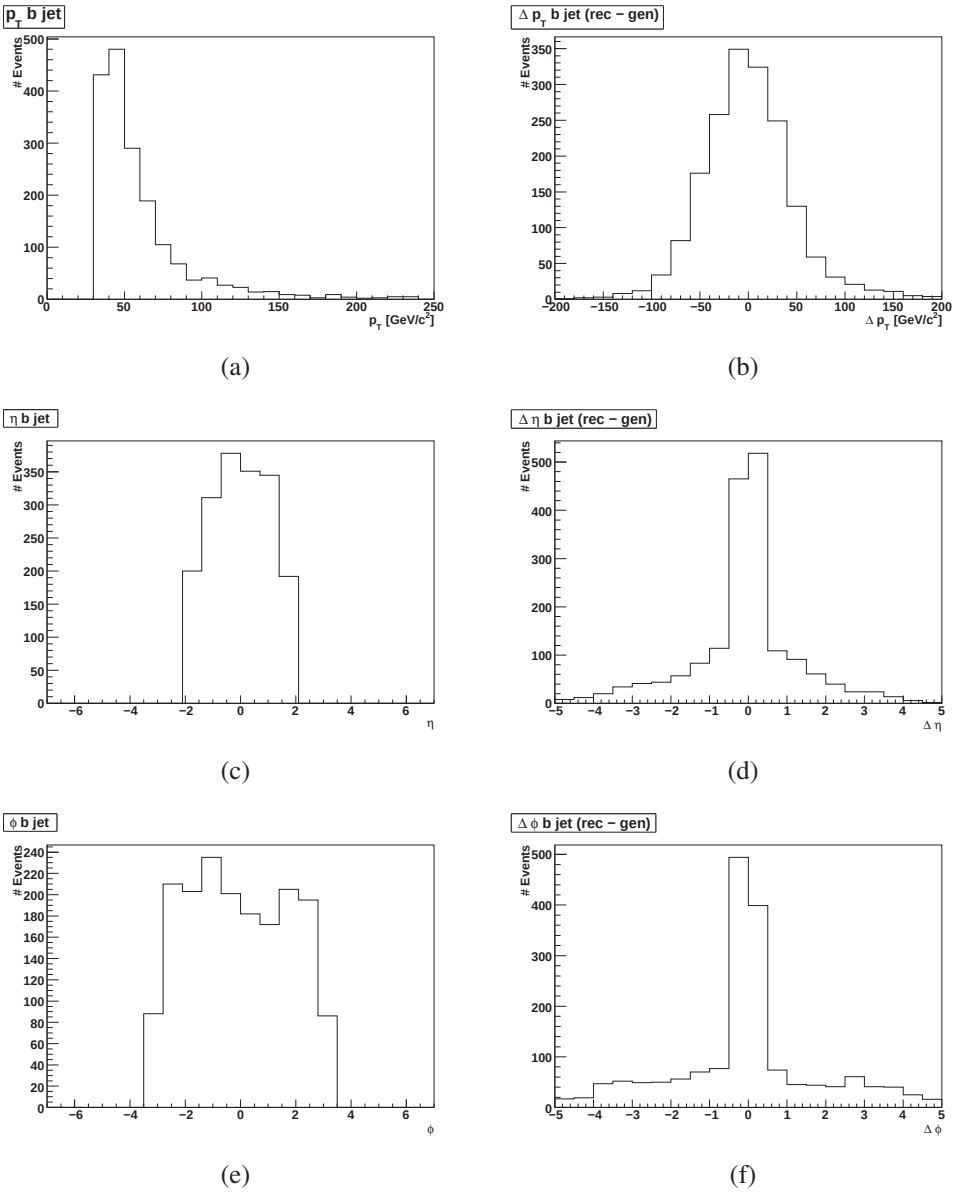


Figure 7.9: Kinematic variables of the reconstructed b jet. a) p_T, b) p_T difference between rec. particle and gen. parton, c) η, d) η difference between rec. particle and gen. parton, e) ϕ and f) ϕ difference between rec. particle and gen. parton

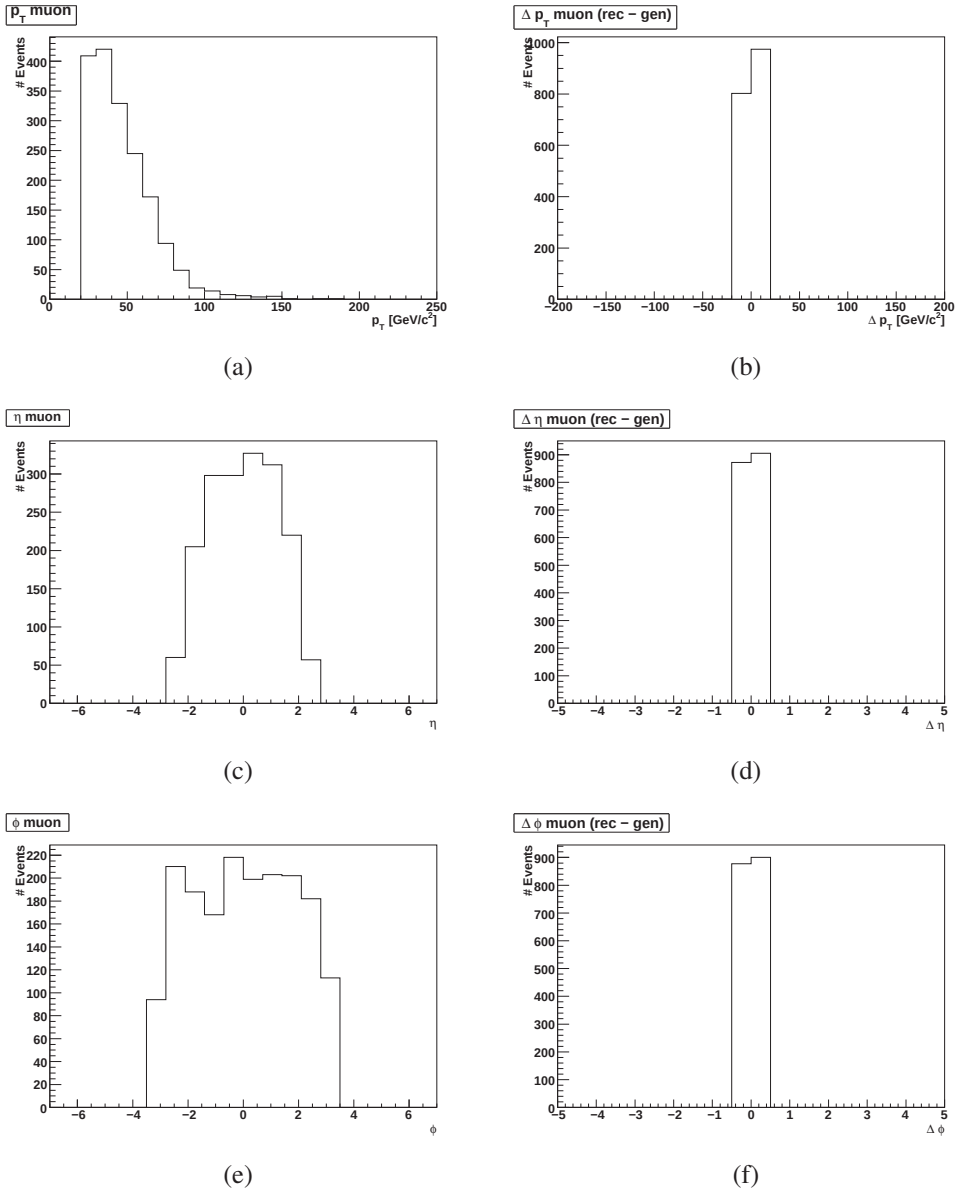


Figure 7.10: Kinematic variables of the reconstructed muon. a) p_T, b) p_T difference between rec. particle and gen. parton, c) η, d) η difference between rec. particle and gen. parton, e) ϕ and f) ϕ difference between rec. particle and gen. parton

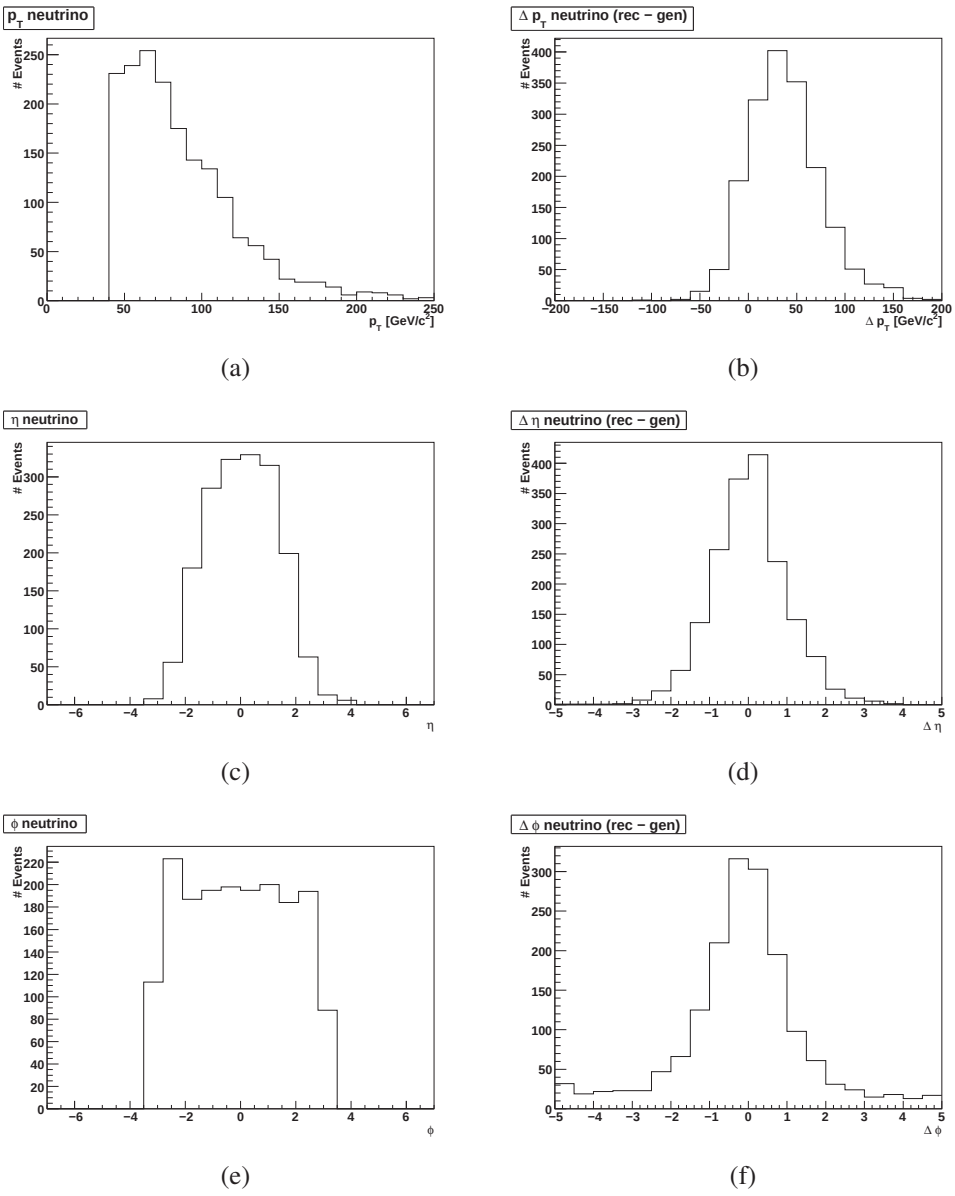


Figure 7.11: Kinematic variables of the reconstructed neutrino. a) p_T, b) p_T difference between rec. particle and gen. parton, c) η, d) η difference between rec. particle and gen. parton, e) ϕ and f) ϕ difference between rec. particle and gen. parton

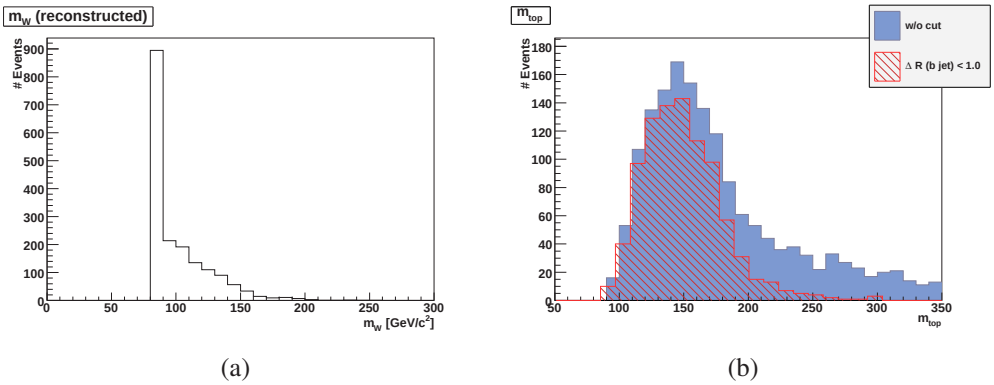
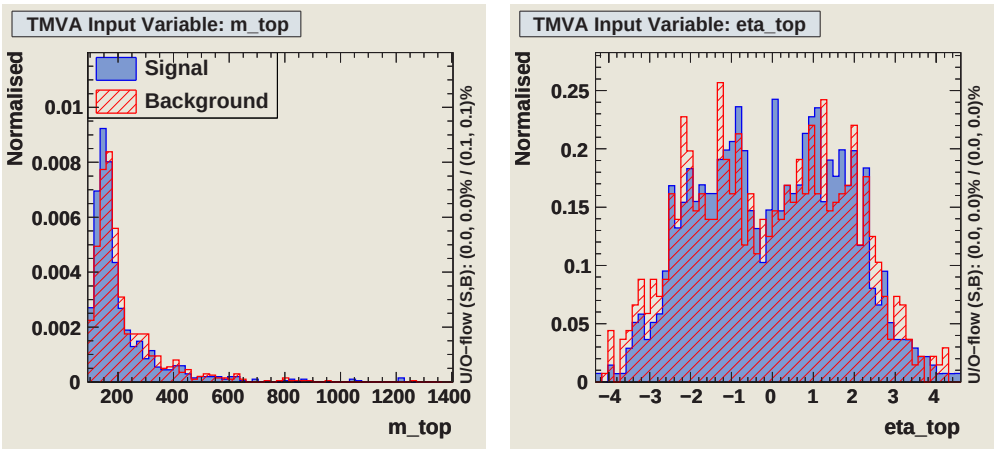


Figure 7.12: Masses of the W boson and the top quark after reconstruction.

7.6 Longitudinal Momentum of the Neutrino

The Boosted Decision Tree (BDT) method of the TMVA package is used for the determination of the correct neutrino solution from equation 7.2. The p_z solution resulting in the overall smallest ΔR calculated by the AutoProcess module is considered as signal while all other hypotheses are considered background. Half of the events are used for training and the other half for testing. In figures 7.13(a) and 7.13(b) the event shape variables m_{top} and η_{top} are plotted.



(a) (b)

Figure 7.13: Event shape variables used for the determination of the correct neutrino p_z solution: the mass (a) and the pseudorapidity (b) of the reconstructed top quark.

To check if the TMVA delivers reasonable results the BDT response is compared with the ΔR values. Figure 7.14 confirms that the TMVA rates the same hypothesis highest that also has the highest ΔR value in almost 72%.

7.7 Separation of Signal and Background

The second part of the analysis, fig. 7.15, focuses on the signal and background separation using the BDT of the TMVA package a second time. First the signal and background samples which passed the preselection and went through automatic reconstruction are read event by event from disk. Next the TMVA trained in the first part of the analysis evaluates each reconstructed hypothesis looking for the one which is most likely a single top event. After the evaluation both signal and background samples are split up into 3 equal parts. The first part serves as training sample, the second one as testing sample of the signal/background separating TMVA, while the third part is stored on disk and will serve as "data" sample in the last part of the analysis.

Figures 7.16 to 7.18 show the histograms of the following input variables for the TMVA.

- p_T, η, ϕ of the top quark
- p_T and η of the b jet and the light jet

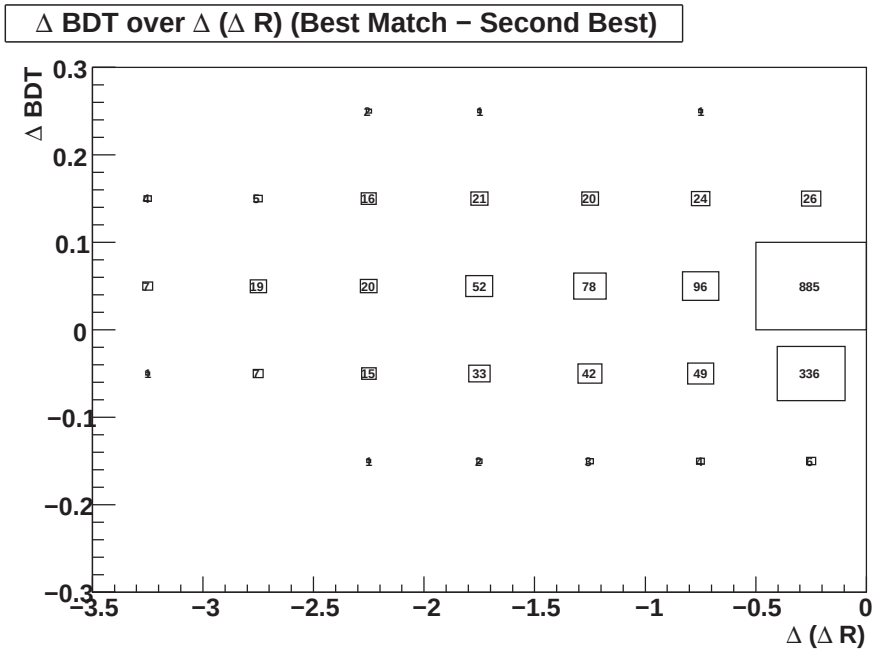


Figure 7.14: Comparison plot of the BDT response with the ΔR matching.

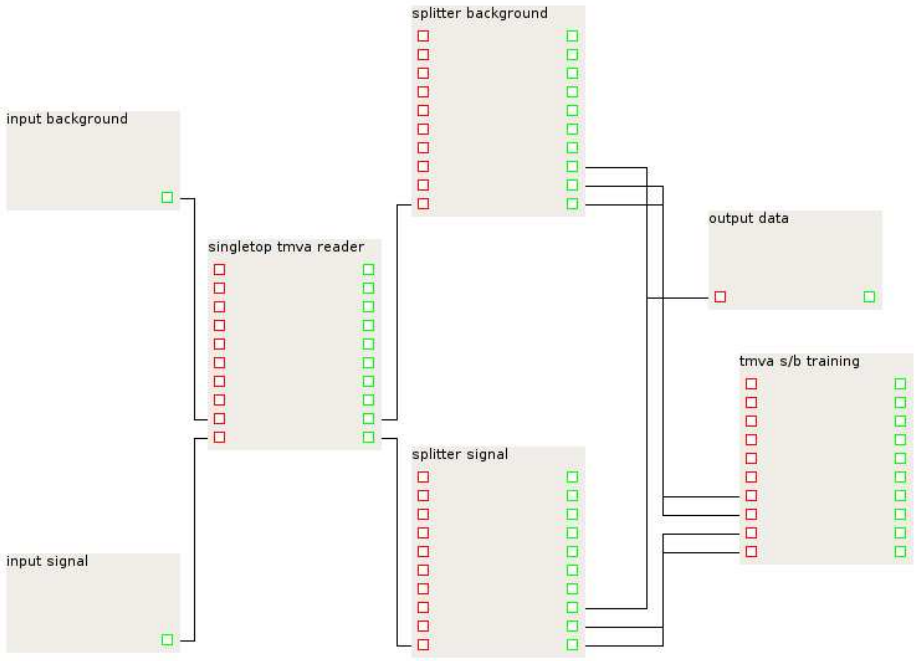
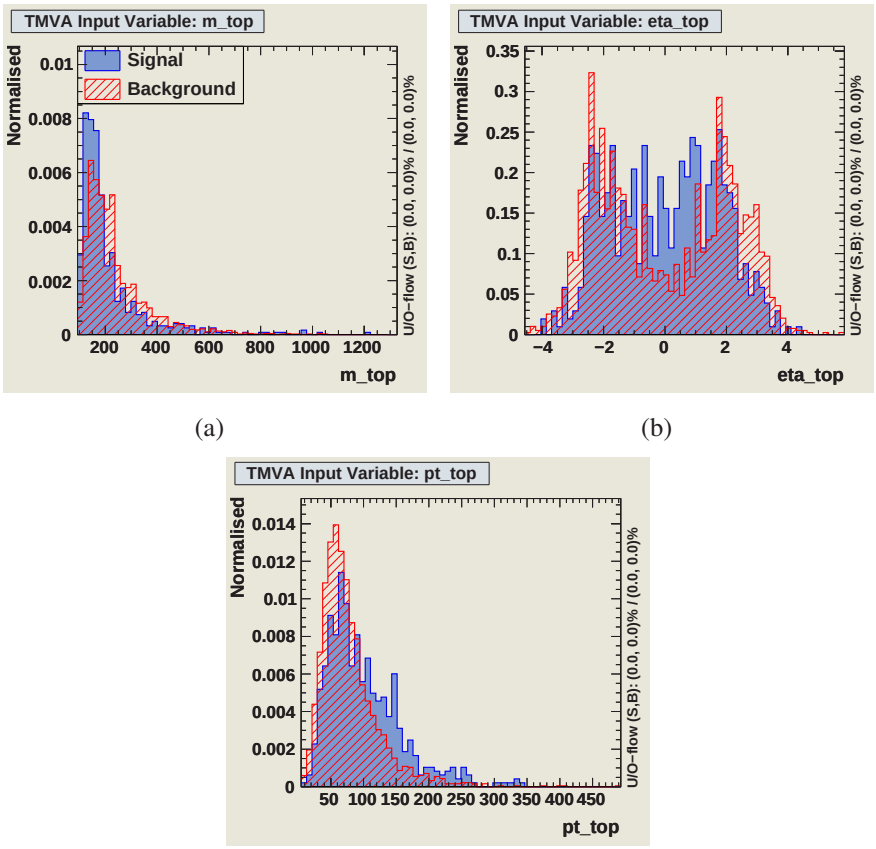


Figure 7.15: Screenshot of the analysis flow diagram, part 2.

- $\Delta\phi$ and $\Delta\eta$ between the two jets
- $H_T = \sum p_{T,i}$ of the 4 final state particles

Figure 7.19 shows the correlations between them.



(a) (b)

(c)

Figure 7.16: Event shape variables used for the separation of signal and background: the mass (a), the pseudorapidity (b) and the azimuthal angle (c) of the reconstructed top quark.

Using the test samples the TMVA predicts the signal efficiency and background rejection shown in figure 7.20. The corresponding BDT response is plotted in figure 7.21.

7.8 Enhanced Single Top Quark Contribution in a Simulated CMS Measurement

In the last part of the analysis the trained Boosted Decision Tree is applied to the simulated "data" sample. The screenshot of the diagram from VisualPXL (figure 7.22) is rather short now. The BDT response is stored for the best hypothesis of each event in the UserRecord and read by the last module in the analysis, the data plotting module.

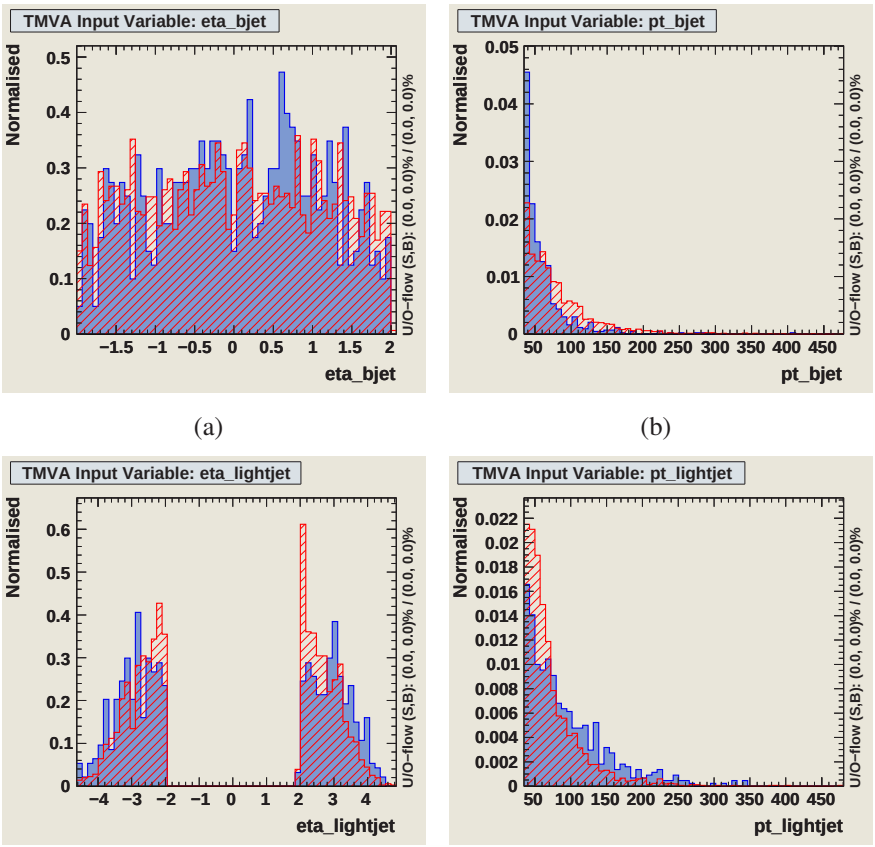
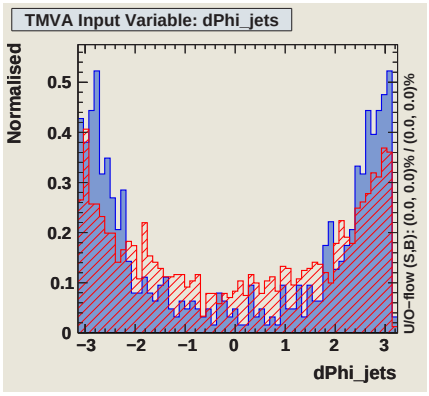
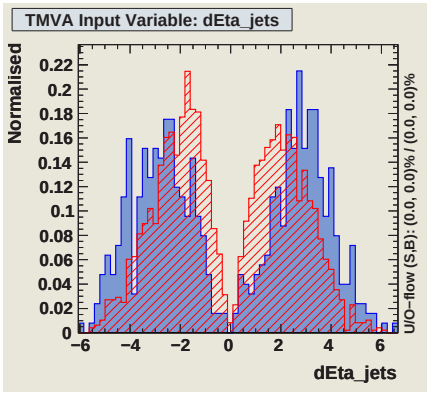


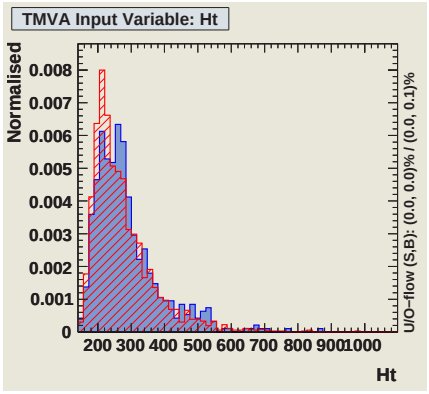
Figure 7.17: Event shape variables used for the separation of signal and background: the pseudorapidity (a, c) and the azimuthal angle (b, d) of the reconstructed b and light jets.



(a)

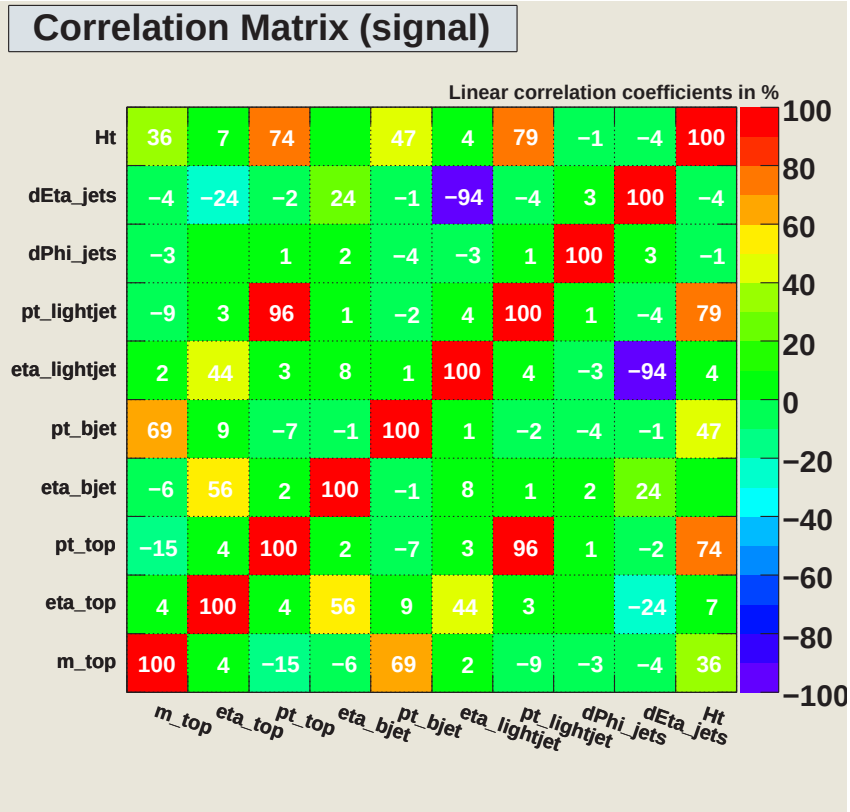


(b)

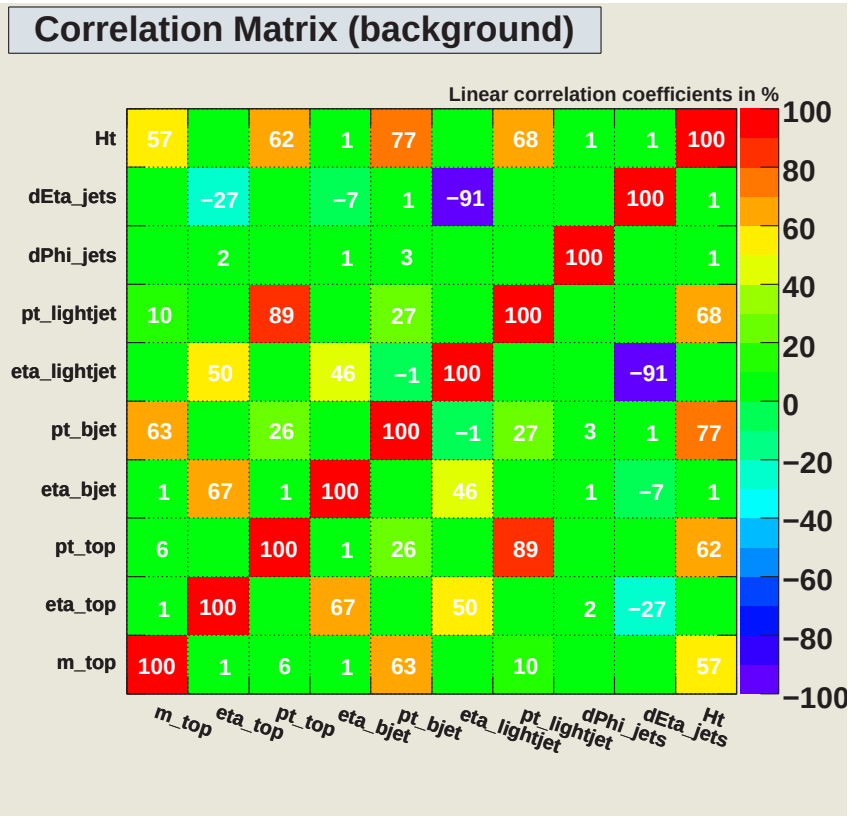


(c)

Figure 7.18: Event shape variables used for the separation of signal and background: the difference between the azimuthal angle (a) and the difference between the pseudorapidity (b) of the b and light jets, and the sum of the transverse momentum of the reconstructed particles (c).



(a) Signal



(b) Background

Figure 7.19: Correlation between the event shape variables of the signal events (a) and the background events (b).

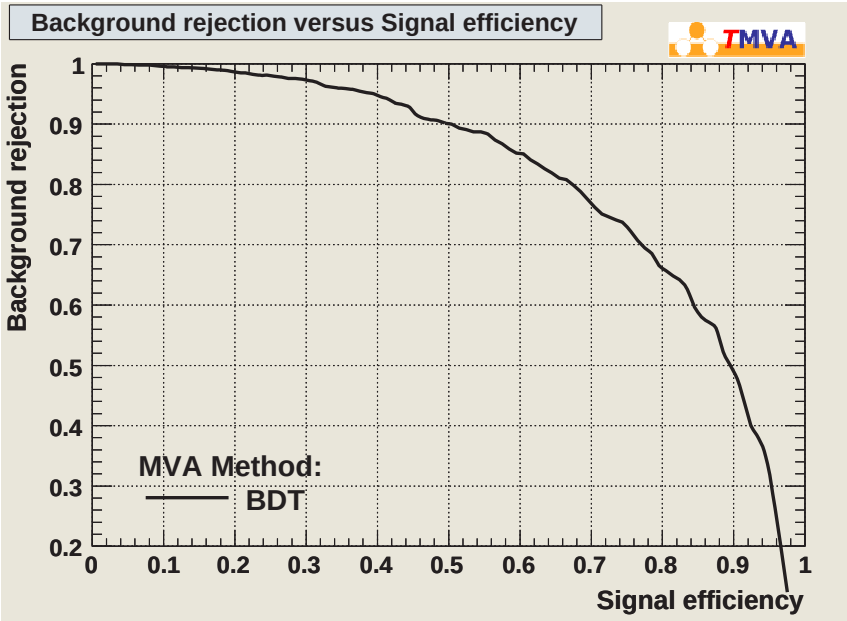


Figure 7.20: Background rejection vs Signal efficiency.

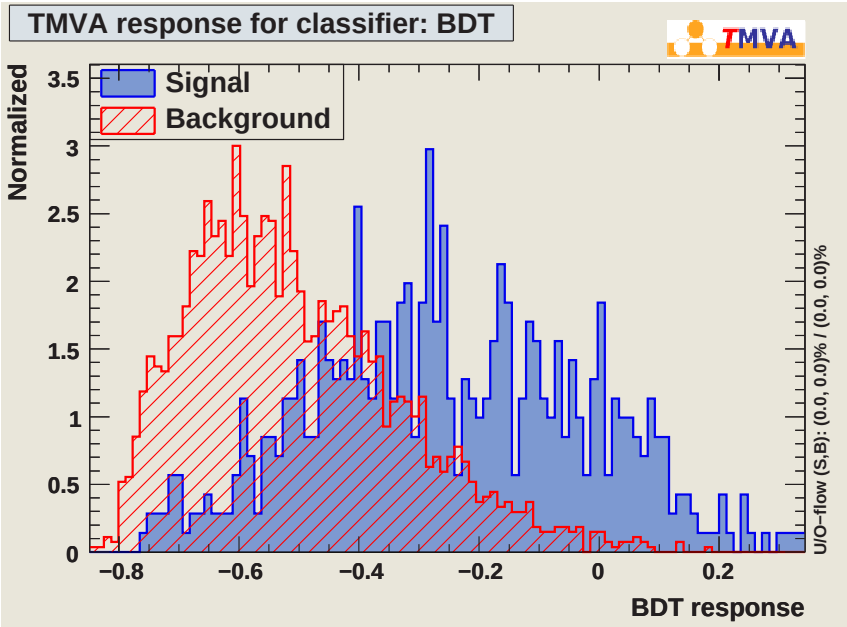


Figure 7.21: BDT response.

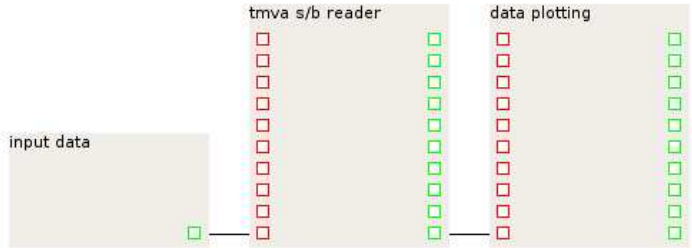


Figure 7.22: Screenshot of the analysis flow diagram, part 3.

Figure 7.23 shows the top mass distribution for the whole simulated data sample. The contribution of the signal is barely visible. In figure 7.24 the BDT response distribution is plotted. A cut on the BDT value can be used to increase the signal/background ratio. This cut on the BDT response of -0.3 is performed in plot 7.25. This removes many background events and makes the signal just visible even without b-tagging which would reduce the number of background events even further.

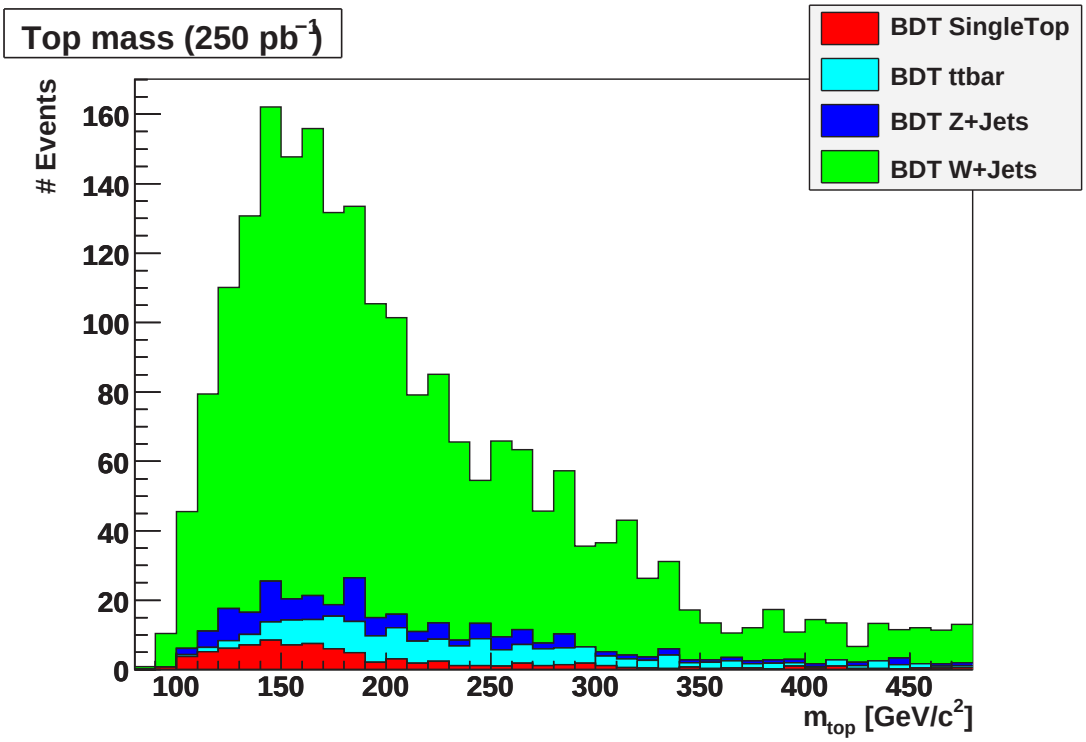


Figure 7.23: Top mass distribution.

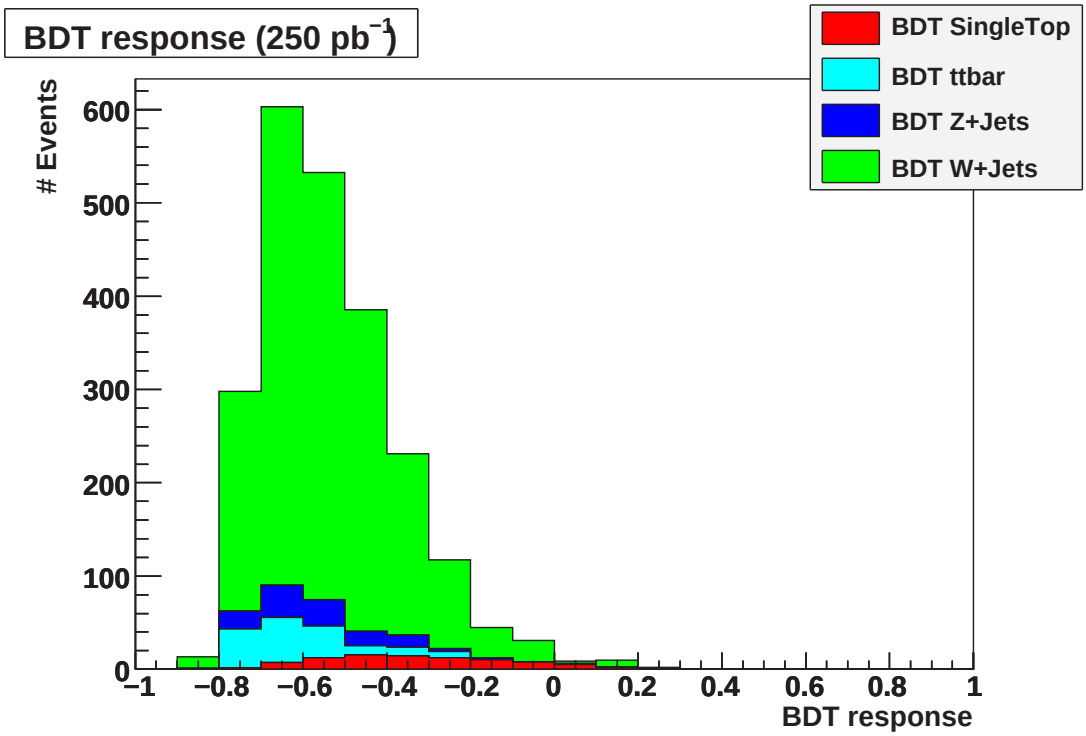


Figure 7.24: BDT response for the simulated "data" sample. This stacked histogram show the contribution of the signal and the different background types.

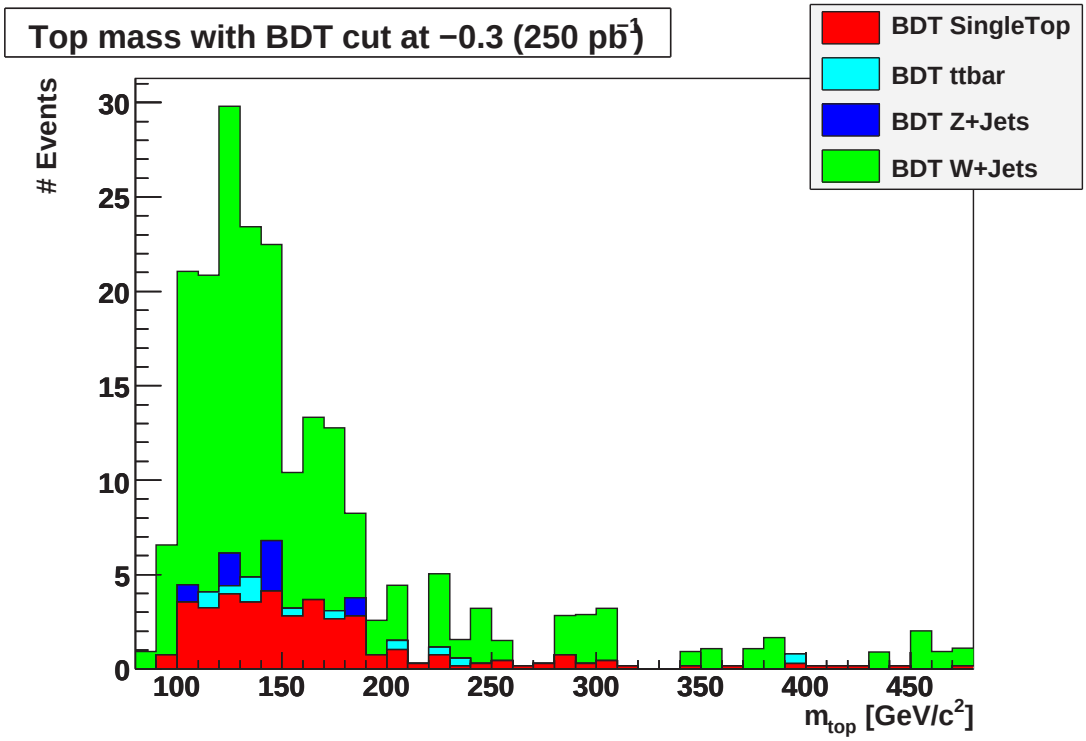


Figure 7.25: Top mass distribution after BDT cut.

8 Summary

The scope of this thesis work was to develop a novel environment for prototyping and performing physics data analyses in a combination of graphical and textural methods. Such a combination has been successfully developed, e.g., for driving hardware configurations using the LabVIEW program [21], but nothing comparable was developed for physics analysis so far. To enable this physics analysis environment, several major components are required some of which were developed within this work.

As the basis for the physics analysis, the four-vector analysis package PXL is used. The main characteristics are a general event container enabling physicists to store all information required in a physics data analysis including multiple reconstruction versions of a single event, and management of relations, such as mother-daughter relations. While PXL is based on the C++ language, an interface to the dynamically typed language Python was established which enables physicists to use the powerful concepts of PXL with a far easier to learn user command syntax compared to the C++ language.

In order to visualize the contents of a particle collision event with all its particles, vertices, and possibly decay cascades, an event browser was developed to enable both a global view of the event, and detailed views on each of the contained objects. To visualize decay cascades, a dedicated layout algorithm was developed which enables arbitrary cascades to be drawn.

Within the same program, a window enabling users to edit and draw a complete event by mouse clicks, e.g., drawing a Feynman type diagram with the corresponding particles, and their decays was created.

To support physicists in a modular programming of their analysis, a C++ based module steering based on a common module interface was integrated into VisualPXL. This windows allows the analysis to be designed in an graphical manner. Modules can be added, connected and configured using VisualPXL. Several default modules are provided such as an input file module, and an output file module, a module which allows to write user code for the analysis using Python.

Furthermore, a module for automated reconstruction of decay cascades was developed and integrated into the graphical module system. This module allows to explore different physics interpretations of the same event on short time scales. The steering card of the module consists of the template of a particle process including decay trees which can be graphically developed using the above mentioned editor. User analysis cuts, re-calibration commands, etc. can be incorporated into this steering card through the Python language, and give maximum flexibility to the individual analysis.

A simple ($Z \rightarrow \mu\mu$) and a complex (single top) demonstration physics analysis have been performed to proof the readiness of the developed tools and concepts for high energy physics analyses.

List of Figures

List of Tables

Bibliography

[1] CMS Collaboration, CMS Physics Technical Design Report Volume I: Detector Performance and Software, *CERN/LHCC*, 001, 2006.

[2] A. Quadt, Top quark physics at hadron colliders, *Eur. Phys. J. C*, 48:835–1000, 2006.

[3] CERN PhotoLab, The CERN accelerator complex, `http://cdsweb.cern.ch/record/979035`.

[4] CMS Outreach, `http://cmsinfo.cern.ch/outreach/CMSdocuments/DetectorDrawings/DetectorD%rawings.html`.

[5] CMS Collaboration, CMS MUON Technical Design Report, *CERN/LHCC*, 32, 1997.

[6] Danek Kotlinski, `http://people.web.psi.ch/danek/CMS/Figures/`.

[7] Duccio Abbaneo, `http://abbaneo.web.cern.ch/abbaneo/cms/layout/whole.html`.

[8] W. Wagner, Top quark physics in hadron collisions, *Rep. Prog. Phys.*, 68:2409–2494, 2005.

[9] The Tevatron Electroweak Working Group for the CDF and DØ Collaborations, A Combination of CDF and DØ Results on the Mass of the Top Quark, 2007, hep-ex/0703034.

[10] W.-M. Yao et al., The Review of Particle Physics, *J. Phys. G*, 33:1, (2006) and 2007 partial update for the 2008 edition.

[11] Werner Bernreuther, Top quark physics at the LHC, *J. Phys.*, G35:083001, 2008, 0805.1333.

[12] Network Working Group, A Universally Unique IDentifier (UUID) URN Namespace, `http://tools.ietf.org/html/rfc4122`.

[13] Makoto Matsumoto and Takuji Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[14] G. van Rossum et al., Python Language Website, `http://www.python.org`.

[15] T. Sjöstrand et al., High-energy-physics event generation with PYTHIA 6.1, *Comput. Phys. Commun.*, 135:238, 2001, hep-ph/0010017.

[16] R. Brun and F. Rademakers, ROOT - An Object Oriented Data Analysis Framework, *Nucl. Inst. & Meth. in Phys. Res. A*, 398:81–86, 1996, `http://root.cern.ch`.

[17] A. Hocker, P. Speckmayer, J. Stelzer, F. Tegenfeldt, H. Voss, K. Voss, A. Christov, S. Henrot-Versille, M. Jachowski, A. Krasznahorkay Jr, Y. Mahalalel, R. Ospanov, X. Prudent, M. Wolter, and A. Zemla, Tmva - toolkit for multivariate data analysis, 2007.

[18] O. Actis, M. Erdmann, A. Henrichs, A. Hinzmann, M. Kirsch, G. Muller, and J. Steggemann, An algorithm for automated reconstruction of particle cascades in high energy physics experiments, 2008.

[19] Stefano Frixione, Eric Laenen, Patrick Motylinski, and Bryan R. Webber, Single-top production in MC@NLO, *JHEP*, 03:092, 2006, hep-ph/0512250.

[20] V. Abramov et al., Selection of single top events with the CMS detector at LHC, *CMS Note*, 064, 2006.

[21] National Instruments Corporation., LabVIEW Development Systems, `http://www.ni.com/labview/`.

Danksagung (Acknowledgements)

Zunächst möchte ich mich ganz herzlich bei Prof. Dr. Martin Erdmann für die ausgezeichnete Betreuung dieser Diplomarbeit bedanken. Die wöchentlichen Besprechungen der Arbeitsgruppe unter seiner Leitung sowie die individuelle Betreuung haben das Gelingen dieser Arbeit erst ermöglicht.

Bei Prof. Dr. Thomas Hebbeker möchte ich mich für die Übernahme der Zweitkorrektur bedanken.

Außerdem danke ich Oxana Actis, Anna Henrichs, Matthias Kirsch, Andreas Hinzmann, Tatsiana Klimkovich und Jan Steggemann für die Hilfe, die Diskussionen und die konstruktive Kritik in und außerhalb der Besprechungen.

Bei Jan Steggemann und Andreas Hinzmann möchte ich mich zusätzlich für das Korrekturlesen bedanken.

Meinen Eltern danke ich für die Unterstützung während des gesamten Studiums.

Für die ruhigen Nächte und die schöne Zeit danke ich meiner Tochter Tamara. Und bei meiner Frau Katharina Tondera möchte ich mich für die unendliche Geduld und Unterstützung in den vergangenen Monaten bedanken.

Erklärung

Hiermit versichere ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt zu haben.

Aachen, den 31. August 2008