Thesis for the Degree
of Master

Geometrical Acceptance Analysis for RPC PAC Trigger

by
Eun-Sung Seo

Department of Physics
Graduate School
Korea University

December 2010

CERN-THESIS-2010-184
14/12/2010

朴聖根 敎授指導

碩 士 學 位 論 文

Geometrical Acceptance Analysis for RPC PAC Trigger

이 論文을 理學 碩士學位 論文으로 提出함

2010年 12月 14日

高麗大學校大學院

物理學科

徐 銀 星

徐銀星의 理學 碩士學位論文

審査를 完了함

2010年 12月 14日

委員長 박 성 근 (印)

委 員 최 우 용 (印)

委 員 이 정 일 (印)

Abstract

The CMS(Compact Muon Solenoid) is one of the four experiments that will analyze the collision results of the protons accelerated by the Large Hardron Collider(LHC) at CERN(Conseil Europen pour la Recherche Nuclaire). In case of the CMS experiment, the trigger system is divided into two stages : The Level-1 Trigger and High Level Trigger. The RPC(Resistive Plate Chamber) PAC(PAttern Comparator) Trigger system, which is a subject of this thesis, is a part of the Level-1 Muon Trigger System. Main task of the PAC Trigger is to identify muons, measures transverse momenta and select the best muon candidates for each proton bunch collision occurring every 25 ns. To calculate the value of PAC Trigger efficiency for triggerable muon, two terms of different efficiencies are needed ; acceptance efficiency and chamber efficiency. Main goal of the works described in this thesis is obtaining the acceptance efficiency of the PAC Trigger in each logical cone. Acceptance efficiency is a convolution of the chambers geometry and PAC logical segmentation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of the CMS Detector

The CMS has the form typical for the large detectors working at the particle colliders. It is a cylinder which contains sevetal layer of the subdetectors of different types surrounding the interaction point : The inner silicon tracker, the electromagnetic and hardron calorimeter (ECAL and HCAL), and the muon system in the iron yoke. One of the most important elements of the CMS detector is the superconducting solenoid, which is the source of magnetic field. The magnetic field allows to measure the momentum of the charged particles. The Lorentz force bend the trajectory of the particle, from the curvature of the track the transeverse component(perpendicular to the field lines) of momentum can be determined.

1.2 Subdetectors

Tracker

The innermost element of the CMS detector is the silicon tracking system. It composed of two part : the inner pixel detectors and the outer strip detectors. The Tracker determines the track of charged particle close to the interaction point. The tracker is composed of the thin silicon, semiconductor sensors with the readout strips or pixels. The charged particle passing through the silicon generates the electric signals, which are then amplified, readout and analyzed by dedicated electronics. The sensors are arranged in

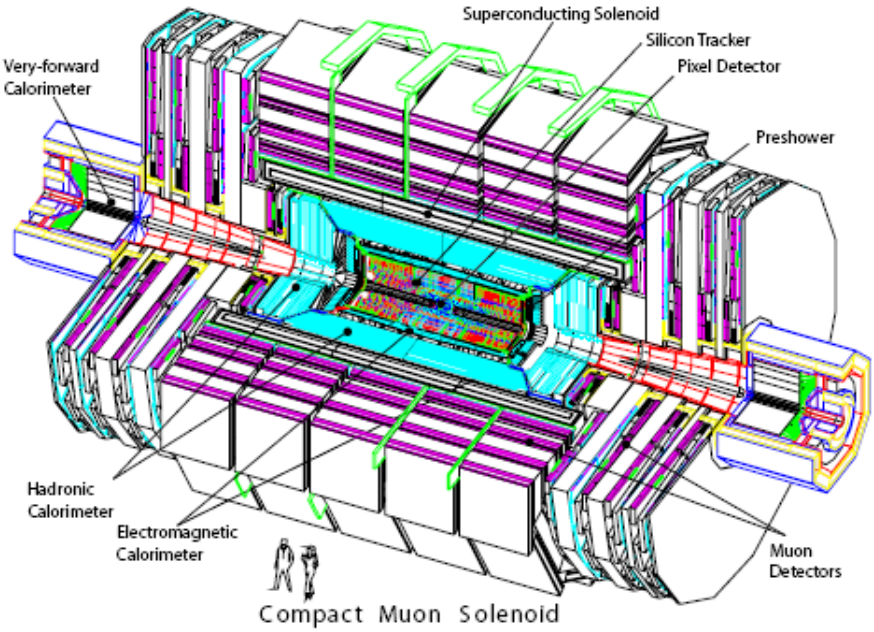

Figure 1.1: The CMS Detector

the cylindrical layers(barrel) and disc(endcaps). The inner pixel tracker consists of the three cylindrical layers in the central region closed by two discs on each side. The spatial resolution is about 10 μm for the R-ϕ measurement and about 20 μm for the Z measurement.

Electromagnetic Calorimeter

Electromagnetic Calorimeter(ECAL), subdivided into a barrel and endcap parts, measures the energy of the photons, electrons, positrons. In case of the CMS, the ECAL is based on the lead-tungstenate ($PbWO_4$) scintillating crystals. A high-energy electron or photon collides with the heavy nuclei of the lead-tungstate crystals and generates an electromagnetic shower of electrons, positrons and photons; the electrons and positrons ionise and excite the atoms of the crystals, which then emit scintillation photons (blue light). The amount of generated light is proportional to the energy that was deposited in this crystal. The light is picked up by the photodetectors attached to each crystal: silicon Avalanche PhotoDiodes (APDs) in case of the barrel and Vacuum PhotoTriodes (VPTs) in the endcaps.

APDs are similar to silicon photodiodes, with the exception of a buried

p-n junction reverse-biased at a very high electric field. Photoelectrons arriving at the junction undergo avalanche multiplication, giving the device gain. In the forward regions vacuum phototriodes (VPT) are used. Long optical fibres transport the signals digitized by voltage-sensitive analogue-to-digital converters to the counting room located adjacent to the experimental cavern. Beam tests have shown that the energy resolution of ECAL modules is excellent.

Hardron Calorimeter

The Hadron Calorimeter(HCAL) is a detector that measures the energy of strongly interacting particles (hadrons and hadronic jets). The high-energy hadron (e.g. proton, neutron, pion, kaon) interacts with the calorimeter material and initiates the cascade of secondary particles. Hadronic showers start to develop later and have larger longitudinal and lateral dimensions than electromagneticones. Therefore the hadron calorimeter (HCAL) is thicker than the ECAL, which it surrounds. The HCAL is made of alternating plates of brass and plastic scintillator read out through wavelength shifting optical fibres by photosensors in the barrel and endcap regions. The photosensors are hybrid photodiodes, which consist of a fibre-optic entrance window onto which a photocathode is deposited, followed by a gap of several millimeters over which a large applied electric field accelerates photoelectrons onto a silicon diode target. The very forward part of HCAL, the HF located at both sides of the detector, has steel absorber plates sampled by quartz fibres due to their good radiation tolerance. The readout is performed with conventional photomultipliers, since the magnetic flux density in the very forward regions is much lower than in the central part.

Muon System

In order to detect a passing muon, three different types of muon chambers form a system consisting of Drift Tube (DT) chambers in the barrel, Cathode Strip Chambers (CSC) in the endcaps and Resistive Plate chambers (RPC) glued to the DT and C CSC chambers. Four layers, also called stations, of DT/RPC and CSC/RPC chambers are embedded into the iron yoke in order to make up a redundant system that can guarantee optimal performance both in the reconstruction of tracks and in triggering.

The DT and CSC record track segments, each of which is characterized by its position and its bending angle in the magnetic field. From these seg-

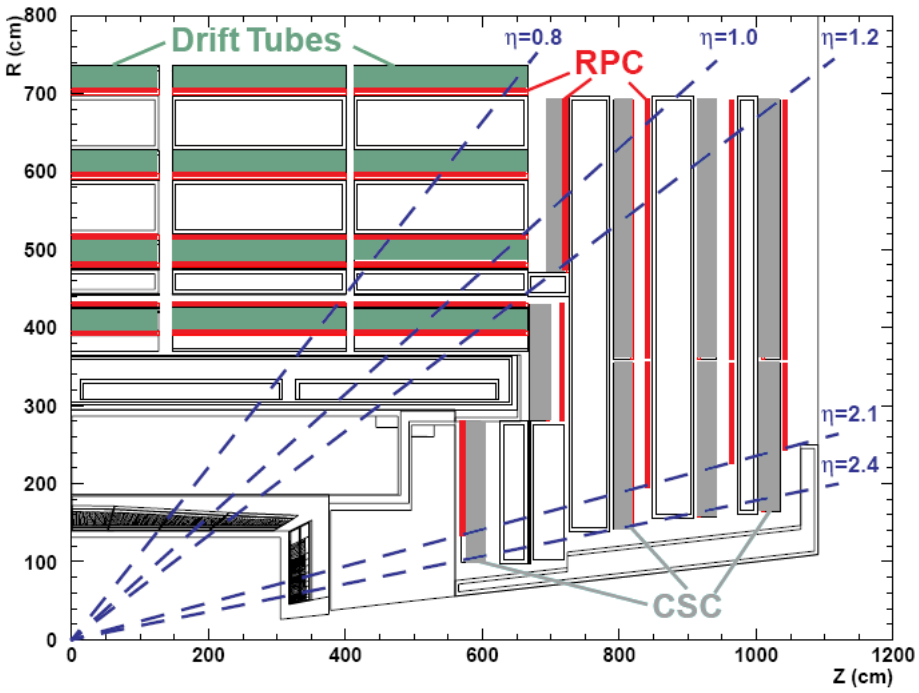

Figure 1.2: The cross-section of the CMS detector

ments the precise transverse momentum and the charge of a track can be reconstructed off-line. A coarser measurement is also available for triggering purposes. The DT chambers consist of tubes containing wires. The tubes are arranged in layers staggered by half a tube. Chambers have eight or twelve layers. The CSCs are multiwire proportional chambers, in which one cathode plane is segmented into strips and anode wires run perpendicular to them. They consist of six individual layers. The RPCs are dedicated trigger chambers, which determine a muon candidates approximate transverse momentum and the bunch crossing number from which it originated due to their excellent timing resolution of 2 to 3 ns. They are segmented in strips, which record hit patterns. The Pattern Comparator Trigger (PACT) which is dedicated trigger for RPC compares each pattern of hit strips to predefined patterns corresponding to different transverse momentum values(pT).

The principle of the operation of all three types of the detectors is similar: they consists of the boxes filled with gas, the electrodes to which the high voltage is connected produce the electric field inside the box (the configuration of the electrodes depends on the detector type). The charged particle passing through the chamber ionises the gas, the electric field multiplies the electron cascade and induces the electrons drift. The readout strips or wires

allow to determine the place, where the cascade was produced.

The DT and the CSC, working with moderate gas gains, provide precise track determination, each chamber gives the vector in space with the precision of 100 μm in position and 1 mrad in the ϕ direction for the DT and 10 mrad for the CSC. The RPC, a high gas gain device, has spatial resolution of order of a few centimetres in the ϕ direction, i.e. much worse than in case of the DT and CSC. However, the big advantage of the RPC is its excellent time resolution (of order of 2 ns), which allows for correct assignment of muon to the bunch crossing. Therefore, the RPC are very precious for the trigger system. In case of the DT and CSC the drift time is much longer than the time between two beam crossings : in some cases the bunch crossing assignment provided by them can be ambiguous, especially in conditions of high neutron background which will be present when the LHC reaches full luminosity.

1.3 Trigger and data acquisition

For the nominal LHC(Large Hardron Collider) design luminosity of 10^{34} $cm^{-2}s^{-1}$, an average of 17 events occurs at the beam crossing frequency of 25 ns. The input rate of 10^9 interactions every second should be reduced by a factor of at least 10^7 to 100 Hz, the maximum rate that can be archived by the on-line computer farm. CMS has chosen to reduce this rate in two steps. At the first level all data is stored for 3.2 μs, after which no more than 100 kHz of the stored events are forwarded to the High Level Triggers. This must be done for all channels without dead time. The Level-1 (L1) system is based on custom electronics. The High Level Trigger (HLT) system, relies upon commercial processors. The L1 system uses only coarsely segmented data from calorimeter and muon detectors, while holding all the high-resolution data in pipeline memories in the front-end electronics. The HLT is provided by a subset of the on-line processor farm which, in turn, passes a fraction of these events to the remainder of the on-line farm for more complete processing.

The physical size of the CMS detector and underground caverns imposes constraints on signal propagation that combine with electronics technology to require 3.2 μs, equivalent to 128 times of 25 ns beam crossings, for any primary decision to discard data from a particular beam crossing. During this 3.2 μs period, trigger data must be collected from the front end electronics, decisions must be developed that discard a large fraction of the data while retaining the small portion coming from interactions of interest and these decisions must be propagated to the readout electronics front end buffers.

The Trigger is the start of the physics event selection process. A decision to retain an event for further consideration has to be made every 25 ns. This decision is based on the event's suitability for inclusion in one of the various data sets to be used for analysis. The data sets to be taken are determined by CMS physics priorities as a whole. These data sets include di-lepton and multi-lepton data sets for top and higgs searches, lepton plus jet data sets for top physics, and inclusive electron data sets for calorimeter calibrations. In addition, other samples are necessary for measuring efficiencies in event selection and studying backgrounds. The trigger has to select these samples in real time along with the main data samples.

1.3.1 Level-1 Trigger

The first step of the event selection is performed by the Level-1 (L1) Trigger system. It is fully implemented in dedicated, custom electronics. Its task is to analyse each event (i.e. each bunch crossing BX) and evaluate if potentially it can contain some interesting physical process. No dead time is allowed. The assumed maximum rate of the accepted events is 100 kHz. At the first level of the selection process most of accepted events will not be interesting the expected rate of events with new physic is evaluated to be only about 10 Hz (in case of certain supersymmetry models). Therefore, the most important requirement for the Level-1 Trigger performance is to accept as many events with the interesting physic as possible (high efficiency of the trigger), keeping at the same time the rate of the selected events below the assumed level of 100 kHz (good purity of the trigger).

The time between bunch collisions (25 ns) is too short to analyse the event and work out the trigger decision. Therefore, the L1 Trigger system is based on the pipeline processing: the algorithms are divided into steps performed in 25 ns, the results of each step are passed to the next level of the algorithm. At any moment, there are many crossing being processed at the various stages of the trigger logic (the iterative algorithms are not allowed in this approach). The L1 Trigger uses selected, low granularity data from the calorimeters and the muon chambers; the tracker data are not utilised by the Trigger, as the tracker has too many channels. The complete data of each event from all subdetectors are stored in the dedicated, electronic buffers, where they are waiting for the trigger decision. In most of the cases, the readout buffers are placed in the detector cavern, near the detectors. After the positive L1 decision, the accepted events are readout by the Data Acquisition system.

The trigger decision has a form of one-bit signal (1 means accept the event,

0 - reject), and it is issued every 25 ns. The signal is called L1 Accept (L1A). It is distributed to the readout buffers by the dedicated transmission network. It was decided that the total time for working out the trigger decision and to pass it back to the readout buffers is (maximally) 3.2 μs (128 bunch crossings). This time defines also the maximum depth of the buffers, which is 128 events. As the trigger electronics is placed in the counting room, about 0.6 μs must be devoted for transmitting the data from the detector to the counting room, and then next 0.6 μs for transmitting the L1A signal back to the readout buffers on the detector.

The L1 Trigger system has a hierarchical tree-like structure. It is segmented in two main parts: the Muon Trigger and Calorimeter Trigger. The top of the L1 Trigger tree is the Global Trigger. The basic assumption is that the triggers subsystems search for the trigger candidate objects, but do not perform any threshold-based selection by themselves. The trigger decision is formed by the Global Trigger that combines the information from the Muon and Calorimeter Trigger subsystems.(But Calorimeter Trigger is outside the scope of this thesis)

Muon L1 Trigger

The First Level Muon Trigger of CMS uses all three kinds of muon detectors: Drift Tubes (DT), Cathode Strip Chambers (CSC) and Resistive Plate Chambers (RPC). The excellent spatial precision of DT and CSC ensures sharp momentum threshold. Their multilayer structure provides a possibility of effective background rejection. RPC are dedicated trigger detectors. Their superior time resolution ensures unambiguous bunch crossing identification. High granularity makes possible to work in high rate environment. Time information and both spatial coordinates of a detected particle are carried by the same signal, which eliminates ambiguities typical for wire detectors.

Complementary features of muon chambers (DT/CSC) and dedicated trigger detectors (RPC) allows us to build two trigger subsystems which deliver independent information about detected particles to the Global Muon Trigger. Advantages of having two such subsystems are numerous. The muon chambers and the dedicated trigger detectors deliver different information about particle tracks. They behave differently in difficult cases and they respond in different ways to various backgrounds. DT with long drift time (400 ns) and CSC with charge weighting are more vulnerable to muon radiation for which RPC are much less sensitive. In the DT/CSC case, a background hit or track segment can eliminate the right one and cause some inefficiency.

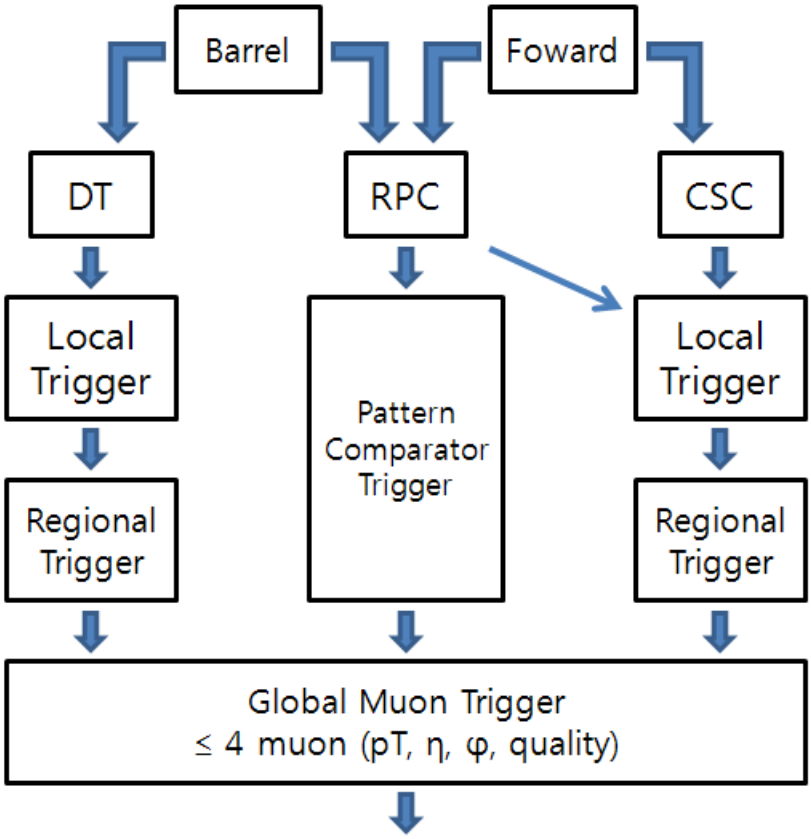

Figure 1.3: Muon Trigger Data Flow

It is just the opposite in the RPC trigger processing all hits simultaneously, which may lead to some rate increase. Accidental coincidence of three of four background hits can be recognized by the RPC trigger as a real muon. This is very unlikely for DT/CSC as they look for coincidence of several planes in each station.

The muon trigger is consist of the following items :

- Drift Tube(DT) Trigger

- Cathod Strip Chamber(CSC) Trigger

- Pattern Comparator Trigger(PACT) based on Resistive Plate Chamber(RPC)

- Global Muon Trigger

In the overap region($0.8 < |\eta| < 1.2$), the data are exhanged between DT and CSC. In this way, the barrel regional trigger covers $|\eta| < 1.0$, whereas the endcap regional trigger covers $1.0 < |\eta| < 2.4$. Optionally, RPC data can be sent to the CSC trigger in order to help solving spatial and temporal ambiguities in multimuon events.

Global Trigger

The Global Trigger receives trigger objects from the Global Muon Trigger (4 muons) and the Global Calorimeter Trigger. The objects contain information about energy or momentum, location (η, ϕ coordinates) and quality.

The GT works out the final trigger decision applying physics trigger requirements (algorithms) to those objects. Up to 128 algorithms can be programmed into the GT. The simplest algorithms are based on requirement that the pT of muons or jets is above the selected thresholds, or that the jet multiplicities exceed defined values. Additional more complex algorithms can be programmed, in which the space correlations between the trigger objects are imposed. The one-bit outputs of the algorithms are combined by a final OR function to generate the L1A signal. In addition, up to 64 so-called Technical Trigger signals (direct trigger signals from sub-detectors) can be connected to the GT, which can be included in the final OR.

Chapter 2

Level-1 RPC PAC Muon Trigger System

2.1 Tasks of RPC PAC trigger system

RPC PAC(Pattern Comparator) trigger is one of the subsystems of the CMS muon trigger. It covers both the barrel and endcap regions of the CMS detector. The PAC trigger system, based on the signals from the RPC(Resistive Plate Chambers), searches for the muons and estimates their transverse momentum. The muons recognition is based on the PAC(pattern comparator) algorithm. The system sorts the found muon candidates and sends to the GMT(Global Muon Trigger) up to four best muon candidates from the barrel region and up to four from both endcaps.

The basic physics requirements for the RPC PAC trigger can be summarized as :

- high efficiency of muons detection - accurate measurement of transverse momentum of muon - low level of false muon candidates and ghosts - no dead time - the latency from the proton-proton collision to the GMT input : no more than 96 BX.

The chambers must have high efficiency for the muon detection and good spatial resolution(small c luster size). Additionally intrinsic chamber noise must be low. The trigger algorithm should be optimized in such a way that for the actual performance of the chambers the best possible quality for the muons recognition is obtained.

2.2 RPC chambers for the CMS detector

A resistive chamber consists of two parallel plates, made out of bakelite with a bulk resistivity of 10^{10} - 10^{11} Ωcm, forming a gas gap of a few millimetres. The gap is filled with the freon-based gas mixture. The outer surfaces of the resistive material are coated with conductive graphite paint to form the high Voltage and ground electrodes. The read-out is performed by means of metal strips separated from the graphite coating by an insulating film. The charged particle ionizes the gas and initiates the electron cascade, the cascade is amplified by the applied HV. The drift of electrons towards the anode induces on the strips a charge, this charge is the output signal of the RPC.

The design of the RPC was optimized so that it can sustain the LHC experiment environment (high rate of hits), and meet the requirements of the CMS trigger system (high efficiency, low noise, good time resolution). Thus, the RPC designed for CMS consists of two gaps with common pick-up readout strips in the middle(See Figure 3.1). The gaps width is 2 mm. It was shown that the double gap RPCs are characterized by a charge spectrum and time resolution improved with respect to the single gap chambers. To reduce the intrinsic noise of the chambers, the inner surfaces of the bakelite were coated with linseed oil.

The forward RPC system consists of four RPC endcap stations designated RE1, RE2, RE3 and RE4. All the Foward RPCs have a trapezoidal shape. The size of each ϕ sector of the muon trigger is $\Delta\phi = 5/16°$. The 10 ° RPCs equipping the region of $\eta < 16$ cover 32 ϕ sectors and are segmented into three along r(RE */2 and RE */3). The 20 ° RPCs in $1.6 < \eta < 2.1$ cover 64 ϕ sectors and are segmented into four along r(RE */1).

2.3 Front-End Board

The signals from the chamber strips are transmitted to the Front-End Boards (FEB) attached to the chambers. In case of the barrel chambers the strips are connected with FEBs with the kapton foil, in case of the endcaps coaxial cables. The FEB discriminates the analogue strip signals (i.e. chooses only those signals, which charge is higher than the defined threshold) and forms them into binary pulses in the LVDS(Low-Voltage Differential Signaling) standard. The rising edge of the output pulse defines the time of the chamber hit.

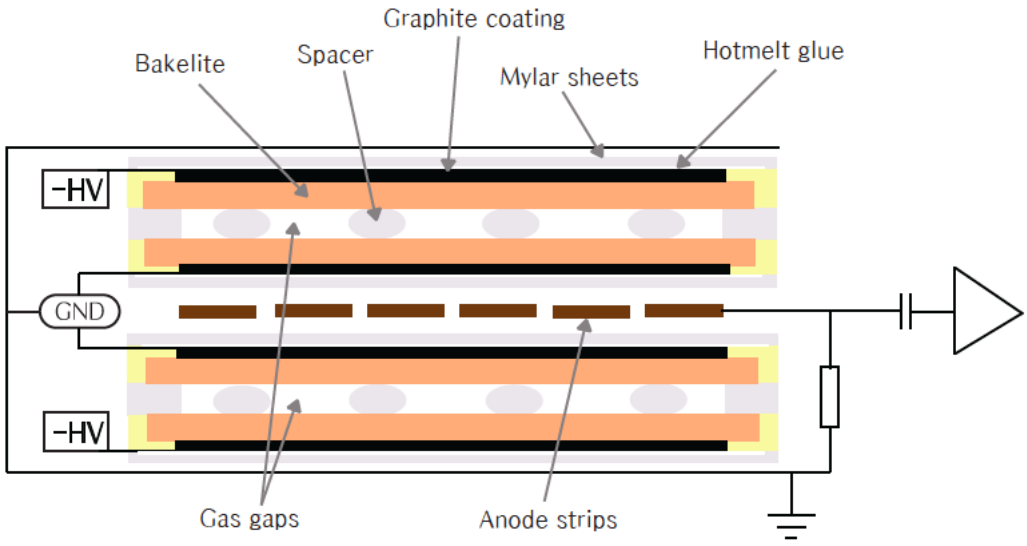

Figure 2.1: The cross-section of the RPC chamber

Above task are performed by the Front-End Chips(FEC). The FEC is a custom Application Specific Integrated Circuit(ASIC) device, it contains 8 channels, each channel corresponds to one strip. The input signals, after amplification, are processed by two discriminators working in coincidence: threshold discriminator providing selection of signals with charge above the defined level, and zero-crossing discriminator, which detects the peak of the signal. In this way, the timing of the output pulse is derived from the maximum of the strip signal, what assures, that this timing is not depending on the pulse amplitude and applied threshold.

In a RPC working in avalanche mode, an after-pulse often accompanies the particle hit signal; the delay of after-pulses is ranging from zero to some tens of ns. To block the after-pulse, a monostable circuit following the discriminators shapes the length of the output pulse to the programmed value (the range of the pulse length is 50-300 ns). The choice of the pulse length should be a compromise between the rate of the remaining after-pulses and the dead time for the true hits. A length of 100 ns, giving a dead time of 4 %, has been considered a good compromise.

2.4 Chambers segmentation, geometry and naming convention

In the CMS the muon detectors are segmented into so-called muon station (See Figure 1.2). In the barrel, the segmentation along the beam direction follows the 5 wheels of the yoke, each wheel contains four concentric layers of the muon stations (named MB1, MB2, MB3 and MB4) divided into 12 sectors in the ϕ coordinate (the logic sectors are numbered from 0 to 11). (See Figure 2.3) Each station consist of a DT chamber and attached to it RPC chambers. In case of the stations MB1 and MB2, a DT chamber is sandwiched between two RPCs, named respectively RB1in and RB1out in case of the MB1, and RB2in and RB2out in case of the MB2. In stations MB3 and MB4, each package comprises one DT chamber and one layer of RPC (it consists of one, two or four separate boxes, depending on the sector), which is placed on the inner side of the station; the RPC layers are named RB3 and RB4 respectively.

In each of the endcaps, the CSCs and RPCs are arranged in four disks perpendicular to the beam, named ME1, ME2, ME3, ME4. Each disk consists of three concentric rings of the trapezoid-shape chambers.

2.5 Algorithm of PAttern Comparator Trigger(PACT)

PACT is based on the spatial and time coincidence of hits in four RPC muon stations. PACT should recognize many spatial patterns of hits for given transverse momentum of muons. The muon identification algorithm (Pattern Comparator - PAC) that is used in the RPC PAC trigger system is based on the searching for the spatial and temporal coincidence of signals from chambers lying on the possible path of a muon coming from the interaction point. We shall call such a coincidence a track candidate. The signals from the chamber strips, which are previously digitized and time quantized (synchronized to the LHC 25 ns clock, and in this way assigned to the particular BX by the Link Boards) are compared to the predefined patterns of hits (fired strips). The PAC Patterns are obtained from Monte Carlo simulation result.

A pattern is defined in all layers laying on the path of the muon. However, in some of those layers there may be absence of hits from a given muon (due

to inefficiency of the chambers or gaps between the chambers). Therefore, to increase the efficiency of the muons detection, the coincidence of the signals from smaller number of layers is also accepted (the minimal number is 3 fired layers, for the high pT patterns in the barrel four fired layers is required). The number and layout of the fired layers fitting to a given pattern defines the quality of the track candidate found by this pattern. The quality is expressed as number of value from 0 to 7 (three bits). The assignment of a majority level (number of layers fitting to a pattern) to a given quality is a matter of the PAC algorithm optimization.

The shape of the pattern (i.e. bending of the corresponding muon track) defines the transverse momentum of the muon and its sign. Because of energy loss fluctuations and multiple scattering there are many possible hit patterns for a muon track of definitive transverse momentum emitted in a certain direction. The patterns are divided into classes with a sign and a code denoting the transverse momentum (*pTCode*, a number from 0 to 31, i.e. 5 bits, See Table 3.1) assigned to each of them.

The chamber signals produced by a given muon can fit several patterns. This is caused by two mechanisms:

- In a given chamber, a muon can fire more than one strip - For the sake of increased trigger efficiency the lower majority levels (hits in 4 or 6 out of 6 chambers) are also accepted, additional patterns with the similar shape can be activated, even though they do not fit exactly to the fired strips.

Among those active patterns, one providing best momentum estimation should be chosen. The rule adopted here is such, that the track candidate with the highest quality is selected. If there is more than one candidate with the maximum quality, the one with the highest *pTCode* is chosen.

The definition of the quality bits is important at this point. In most of the cases, at least one pattern fits to the fired strips in all fired layers.(See Table 3.1) This pattern should be chosen among all activated patterns. Studies which we have done indicate that it is enough that the quality value expresses the number of fired layers, while the layout and distribution of the fired layers is not that important.

2.6 PAC Trigger logical segmentation

The RPC chambers of the CMS detector contains over 180 000 strips. It is not possible to deliver (with frequency of 40 MHz) and process the data

$p_T Code$	$p_T [GeV/c]$	$p_T Code$	$p_T [GeV/c]$
0	No track	16	16
1	0	17	18
2	1.5	18	20
3	2	19	25
4	2.5	20	30
5	3	21	35
6	3.5	22	40
7	4	23	45
8	4.5	24	50
9	5	25	60
10	6	26	70
11	7	27	80
12	8	28	90
13	10	29	100
14	12	30	120
15	14	31	140

Figure 2.2: The definition of transverse momentum ranges for the *pTCode*. Numbers in the pT column denote the lower boundary of the range expressed in the GeV/c

Number of fired layers	Quality value
3	0
4	1
5	2
6	3

Table 2.1: Definition of the quality bits

from so many electronic channels in a single device (chip). Therefore, the Pattern Comparator algorithm must be distributed over many chips. From that it follows, that for the PAC the RPC detector has to be divided in smaller logical units.

The smallest unit of the Pattern Comparator algorithm is so-called logical cone, defined as one logical segment (ϕ segmentation) of one trigger tower (η segmentation). The logical segment is defined by 8 subsequent strips of the reference plane, thus there are 144 segments. The first strip of the logical segment number 0. In the non-reference planes, the logical segment covers up to 72 strips, the neighboring segments overlaps in the non-reference planes. The logical towers in η are defined by the length of the strips of the reference plane. The triangle defined by the interaction point and the reference strip covers usually two strips of the adjacent rolls in the non-reference planes. Therefore, the logical cone is build with the logical strips that are formed by taking logical OR of two strips of the same layer with the same ϕ (thus, the logic cones overlaps also in ϕ). The patterns are defined on the logical strips as well.

2.6.1 Strips R-ϕ segmentation

The strip angular width is assumed to be $5/16^\circ$ in the R-ϕ plane, what means that in each layer there should be 1152 strips (12 sectors $\times$ 96 strips per sector). The layout of the strips in the ideal situation should be projective (i.e. the strips in all layers should be aligned to the common radius). Only in the endcaps, where the chambers form flat discs and overlap to avoid gaps, those rules are strictly fulfilled: in each chamber in one eta-partition (row of strips) there is 32 strips of a trapezoid shape.

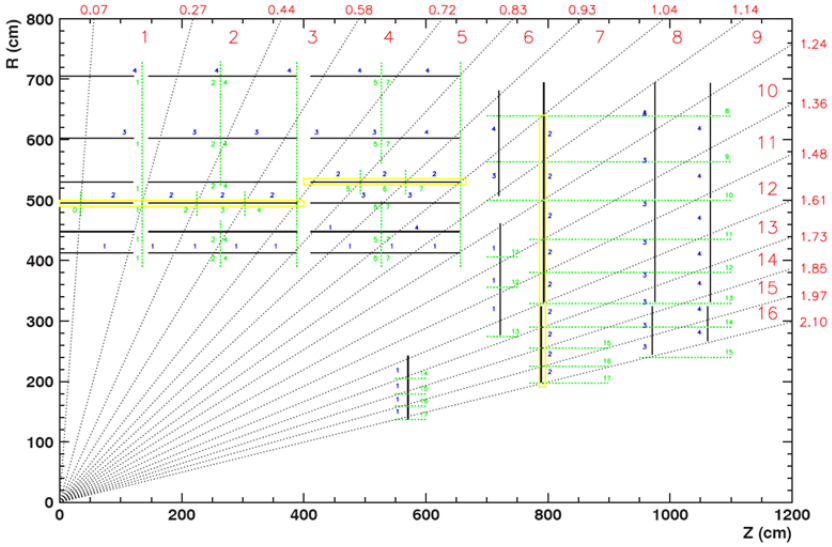

Figure 2.3: Geometry of the RPC strips and trigger towers

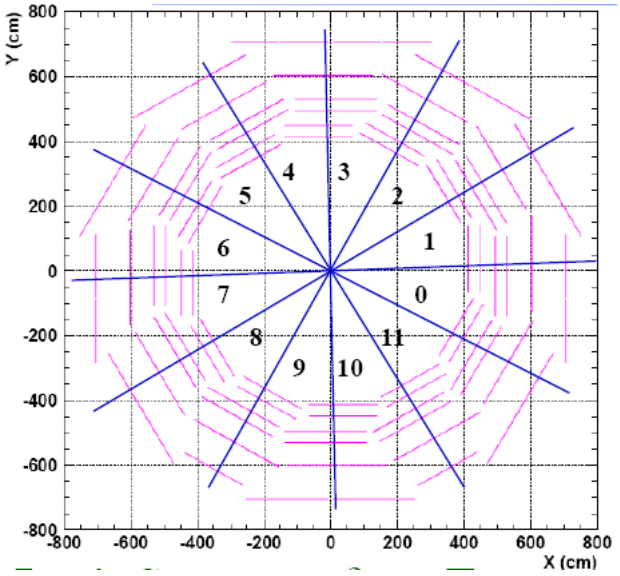

Figure 2.4: The logic sectors : phi segmentation

2.6.2 Strips η segmentation

In the barrel, due to the iron yoke construction, the chambers of a given layer cannot overlap (except in the outermost layer). Therefore, to assure the (approximately) projective geometry of the strips, there are less than 96 strips per sector in each layer; in the three innermost muon stations there are 84 or 90 strips per sector, in the muon station MB4 the number of strips in most of the sectors is 96, except in the top sector 4 where are 144 strips, and in bottom one 10 where are 120 strips.

In the barrel, the chamber length in the Z coordinate is equal to the width of the CMS wheel (260 cm). In most of the chambers, the strips are segmented in the Z coordinate into two parts 130 cm length each (so-called rolls or eta-partitions). The exception is the layer RB2in of the wheels -1, 0, +1 and the layer RB2out of the wheels -2 and +2, where the strips are segmented into three parts of length 85 cm. These chambers form so-called reference plane, their strips define trigger towers, which are units of the trigger logical segmentation in the η plane.

The endcap chambers have length of about 175 cm in the R coordinate; the strips are divided into three parts. The exception are the chambers of the innermost rings, which are divided into four parts in case of the stations RE1/1 and RE2/1, and in to two parts in case of the stations RE3/1 and RE4/1.

2.7 Ghost Busting and Sorting

As the logical cones overlaps both in the ϕ and the η, the chamber hits of a single muon may produce the track candidates in a few neighbouring logical cones. Such an additional track candidates are called ghosts; they must be eliminated. The elimination of ghost is based on the observation that the ghost track candidates have in most of the cases less layers fired, and, what follows, the lower quality than the true candidate. This effect results primary from the fact, that the strips of the reference layer belong to only one logical cone, so the reference layer is fired in only one logical cone.

Because of lack of interconnections it is not possible to assemble the muon candidates returned by all PACs in one device which could then perform the ghost-busting, therefore the ghost-busting is performed by the tree of the devices called Ghost-Buster-Sorters (GBS). The tree has four levels, with the following types of the devices on each level: Trigger Board GBS, Trigger

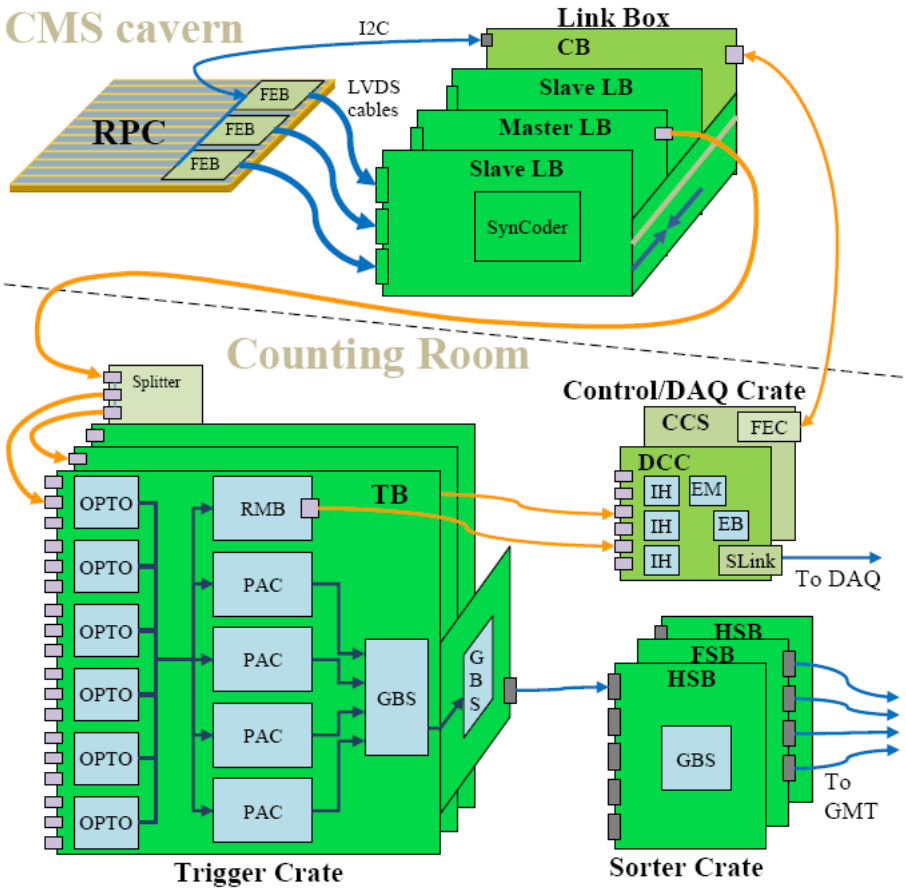

Figure 2.5: The scheme of RPC PAC Trigger system

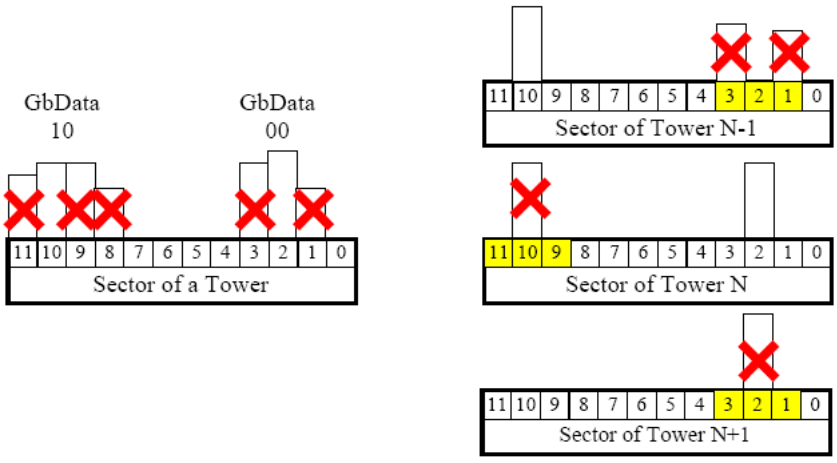

Figure 2.6: Trigger Board GBS principle : The height of the bars represents the *combined Code* of the muon candidates. Left side is ϕ ghost busting. In this process *GbData* bits are calculated which are hten used in the Half GBS. 1 means that the given candidate killed the candidate on the right or left edge of the logical sector respectively. Right side is the η ghost busting. Each candidate kills the candidates with the lower code from the other towers in the 3 neighboring segments

Crate GBS, Half GBS and Final Sorter. Both in the ghost-busting and sorting the candidates are ranked by the combined code, formed from the quality (primary criterion) and *pTCode* (secondary criterion). (See Figure 2.5, 2.6, 2.7)

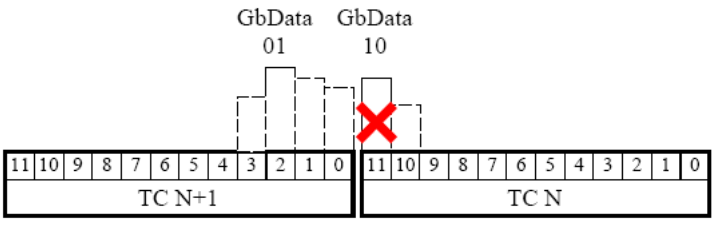

Figure 2.7: Half GBS principle : The bars with the dashed line denotes the muon killed during the TB GBS η ghost busting

Chapter 3

Geometrical Acceptance Analysis

3.1 Triggerable muon

Chamber hits are compared with predefined set of patterns obtained from the Monte Carlo simulation. The patterns correspond to muon tracks of defined transeverse momentum. And patterns are defined inside Logic Cones that geometrical units of the PAC segmentation. Muon candidates are generated in logical cones.

Chamber hits should fit to at least one pattern and must be in same bunch crossing(25 ns). Triggerable muon can be sorted by regions of chamber hits as listed in the following sections.

3.1.1 Barrel region

For low pT muon, 3 out of 4 inner chamber layers must be fired then those muons can be defined the triggerable muon. For high pT muon, 4 out of 6 chamber layers are to be fired to become triggerable muons.

3.1.2 Endcap region

Just only for high pT muon, 3 out of 3 chamber layers are to be fired.

3.2 Efficiency calculation of RPC PAC trigger

RPC PAC trigger efficiency can be obtained by multiplying terms of specific efficiencies.

$\varepsilon = \varepsilon_{acceptance} \otimes \varepsilon_{chambers} \otimes \varepsilon_{PAC} \otimes \varepsilon_{synchronization} \otimes \varepsilon_{datataking}$

$\varepsilon_{acceptance}$: Probability that muon crosses chambers required by the PAC patterns, at least 3 (4) layers belonging to a logical cone.

$\varepsilon_{chamber}$: Probability that a muon crossing the chamber leaves hits. It includes gas chamber performance, front-end board performance, and strip masking.

ε_{PAC} : patterns efficiency. Probability that the chamber hits of a triggerable muon fits to any pattern.

$\varepsilon_{synchronization}$: Probability that a hit is assigned to the correct bunch crossing. Soon it will be 99.xxx %.

$\varepsilon_{datataking}$: Probability that a hit is correctly processed from the Link Boards to the PAC logic. It includes temporarily disabled hardware (Link Boards, Trigger Boards or Trigger Crates, optical link) and Transmission errors.

$\varepsilon_{acceptance}$ is a convolution of the chamber geometry and PAC logical segmentation(Logic Cone shape). It depends on gaps of wheels(η), gap of sectors(ϕ), transeverse momentum and charge. $\varepsilon_{acceptance}$ can be obtained from the Monte Carlo simulation dataset that contains the simulated muon hits with 100 % chamber efficiency. The especially calculated value that $\varepsilon_{acceptance} \otimes \varepsilon_{chambers}$ is the efficiency of triggerrable muon. It is important to check the performance of PAC trigger.

$\varepsilon_{acceptance} \otimes \varepsilon_{chambers} = \varepsilon_{triggerable}$

3.3 Obtaining information from dataset

The used MC dataset is 40 million muons generated by Particle Generator(Path of dataset is Tier-2, Warsaw Group of Poland/RPCPatGen0911/fruboes-RPCPatGen0911-d5f3140110dd9197bf2e8d38ef5dbefb/USER). There are several containers

of muon. SimTrack and TBMuons are used for this analysis. From the SimTrack, the information about eta-phi address and transverse momentum(pT) are obtained. From the TBMuons, Firedplanes information are collected. Because the resolution of simTrack about pT, eta, phi information was better than those of TBMuons, the SimTrack container is chosen in this analysis. In this analysis, the number of muons in an event is kept to be 1.

3.3.1 Binning for the histograms

Binning of eta region

An natural binning is by trigger towers. There are 33 towers from -16 to +16. Barrel region is $-7 \leq \eta \leq +7$ and endcap region is $\eta \geq +8, \eta \leq -8$. Transition region(or overap region) is defined towers = $\pm$ 7, 8, 9. (See Figure 2.3)

Binning of phi region

Logic sectors consist of $2\pi/12$ segmentation. Logic Segments are separated by $2\pi/144$ segmentation. Actually these are shifted by 5 degree, the first strip of the logical segment number 0 is placed on the $\pi = +5°$. In this analysis, a bin for ϕ region means one logical segment. (See Figure 2.4)

Binning of pT

Basically, transeverse momentum of muon is defined by the $pTCode$ in PAC Trigger. From 0 to 31, 32 number of $pTCode$ bins are used.(See Figure 2.2)

3.4 Acceptance Efficiency about 40M dataset

As defined at the previous section, $\varepsilon_{acceptance}$ is the probability that muon crossing a Logic Cone of chamber layers. For the normalization, the 2 dimensional histograms are drawn for eta-pT and eta-phi region. And the Muons are sorted by layer of RPC using the firedplanes information that obtained by PAC Trigger. Drawn histograms(Figure 3.7, 3.9 and 3.8, 3.11) six of them(Figure 3.9, 3.11) are sorted by layers and one(Figure 3.7, 3.8)

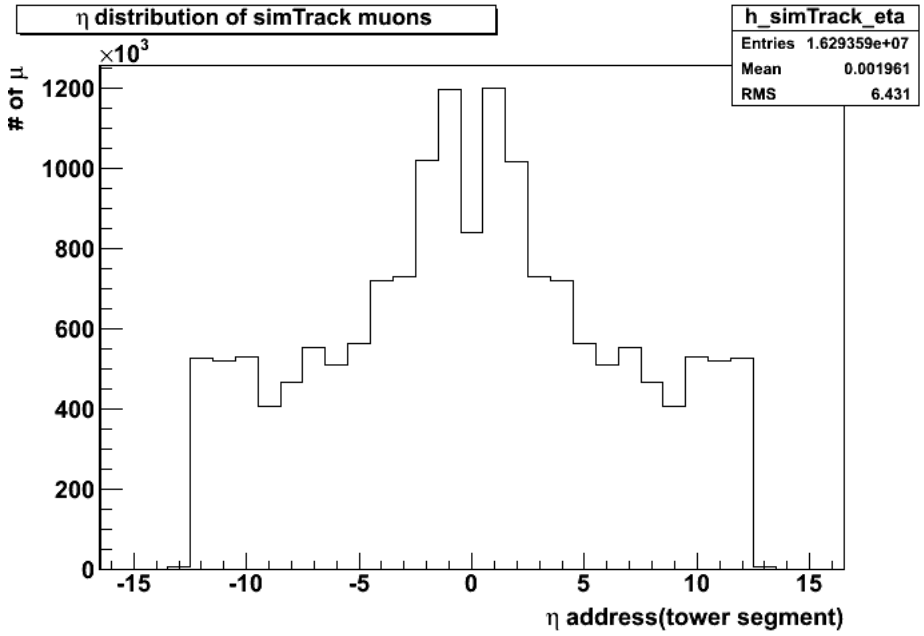

Figure 3.1: Eta distribution of the muons in SimTrack

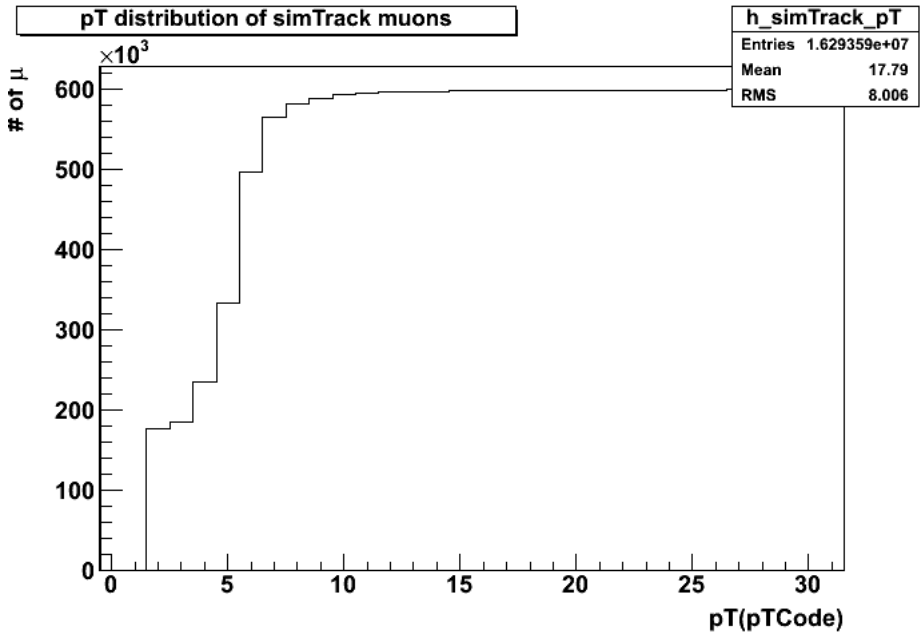

Figure 3.2: pT distribution of the muons in SimTrack

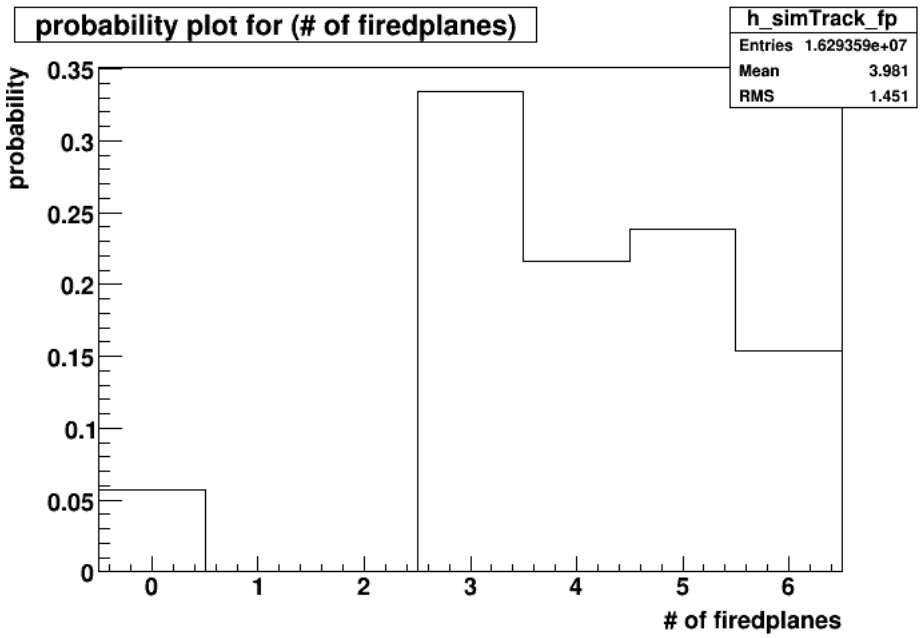

Figure 3.3: Probability plot for the number of firedplane information about simTrakc muon

is not sorted. Each bin of histrograms in eta-phi region means the unit of segmentation in the **Logic Cone**.

After the collecting muons, by using the function of ROOT : **Divide()** the histogram can be normailized easily. The sorted histograms by layers(Figure 3.9, 3.11) are the subset of total distribution of muons(Figure 3.7, 3.8). And to obtain the value of geometrical acceptance efficiency, the sorted histograms are divided by the total distribution which is the histogram of not sorted one.

The reason of drawing the probability plots in eta-pT region is to select the pT cut for the probability plot in eta-phi. (See Figure 3.11) As pT increasing, probability on trigger tower is stabled. Selected pT cut is $pTCode \geq 25$. Then crossing probabilities in each Logic Cone are obtained. Along the x and y axis, the inefficiency lines appeared, and it occurs from the gaps of RPC chamber. (See Figure 1.2)

3.4.1 Firedplane information : Bitset Operator

To get information of firedplane, a function *getFiredPlanes()* was called. This function brings a number that contains the firedplane information of

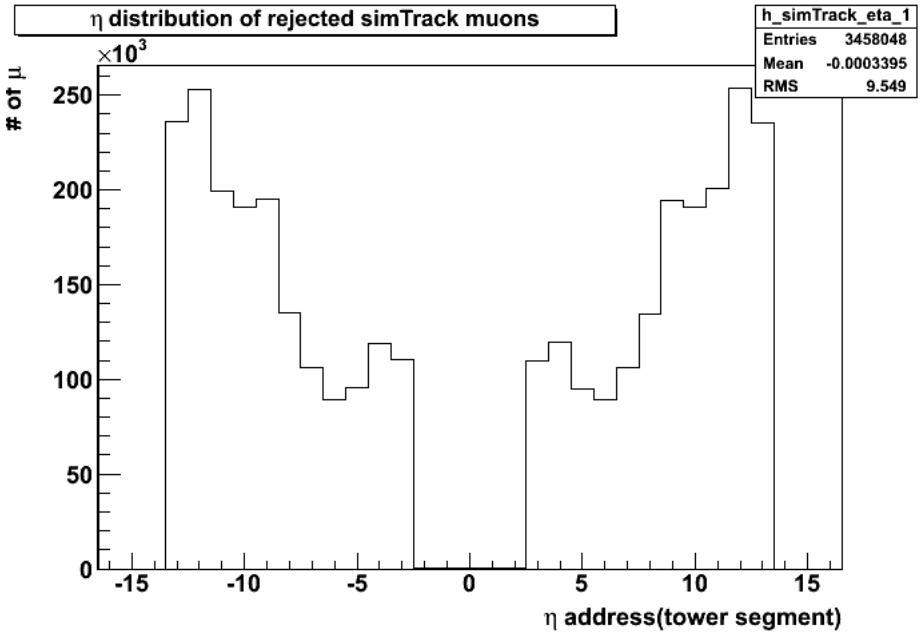

Figure 3.4: Eta distribution of the rejected muons

the muon. Range of the number is from 0 to 63, and it can be converted to **binary representation**. Bitset Operator(or Bitwise Operator) is basic of the C++ language. C++ provides operators to work with the individual bits in ints. The *bitset* is very similar to bool type of *vector* : it contatins a collection of bits and provides constant-time access to each bit. There are two main differences between *bitset* and *vector*. First, the size of a *bitset* cannot be changed. Second, *bitset* is not a sequence ; in fact it's not an STL container at all.

Using *bitset* operator, obtained numbers(0 to 63) can be converted to a bitset form. It contains 6 bits of digit and one bit correspond to a RPC layer. 1 means fired, 0 means not fired and most rightside bit is firing information of innermost layer of RPC. Figure 3.3 shows the probabilities about the number of firedplanes information of the muon. The number of fired planes = 1 or 2 muons have zero value of probability because PAC trigger passes only the muon at least 3 planes fired. 1 or 2 fired muons are moved into the bin of zero fired muon.

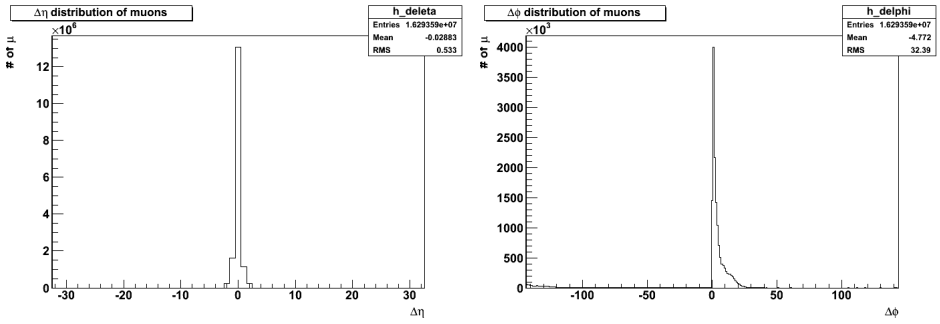

Figure 3.5: $\Delta\eta$ and $\Delta\phi$ distribution

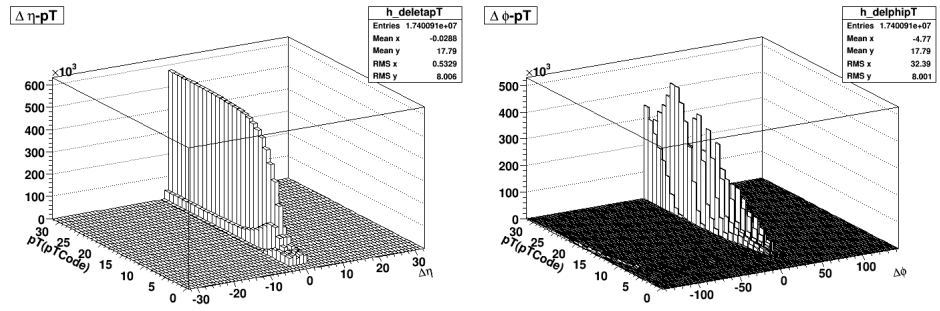

Figure 3.6: $\Delta\eta - pT$ and $\Delta\phi - pT$ distribution

3.4.2 Muon selection and constraints

TBMuons contains the muons that are the output of the Trigger Board. It consist of four number of muon candadate. TBMuon is a type of the *vector*. **TBMuons[0]** means the muon candidate from barrel, **TBMuons[1]** is endcap muon candidates. These candidates are aligned by the *Code*(obtained from *quality* + *pTCode*), most leftside one has best quality and nonzero *pTCode*. Then compare the leftside muon of TBMuons[0] and TBMuons[1] candidate, select the muon have higher value of the *Code*. In Analysis, only this selected muons(called selMu) are used. Meaningful muons are only triggerable muons that at least 3 layers fired.

Used MC dataset was generated by the condition ; one muon in an event. Because SimTrack and TBMuon contatiners are not matched but there is just one muon in the same event, muons are selected by constraints : $\Delta\eta \leq +2$ and $\Delta\eta \geq -2$, $\Delta\eta$ = SelMu η - SimTrack η.

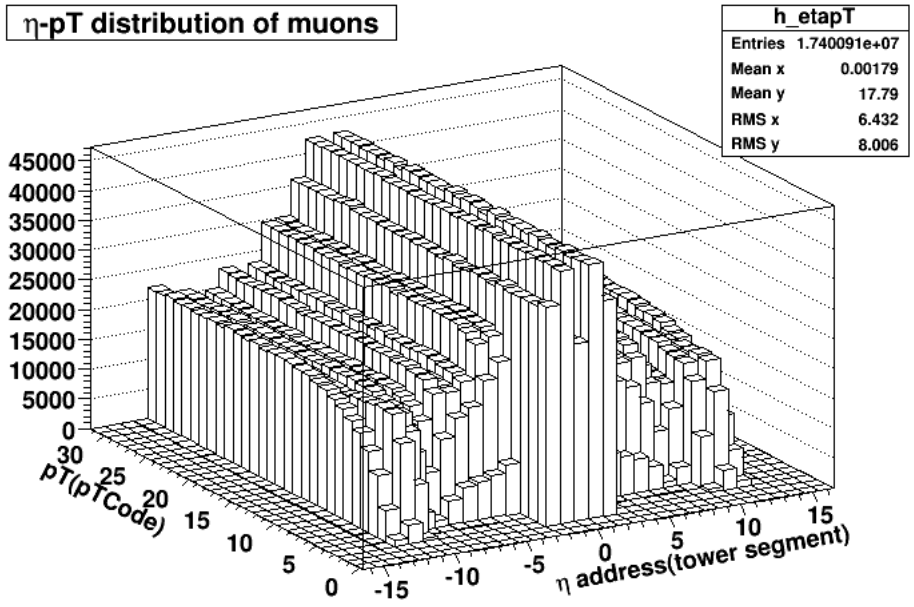

Figure 3.7: eta-pT distribution of SimTrack muon

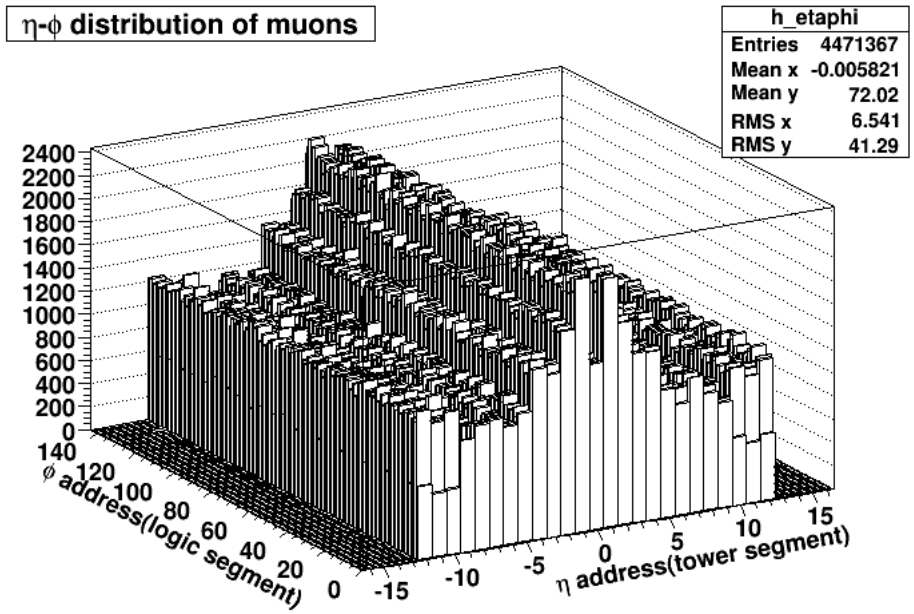

Figure 3.8: eta-phi distribution of SimTrack muon

3.4.3 Probability plots : eta-pT and eta-phi

Using the ROOT macro(entire code is attached on chapter 5), the 2 dimensional probability plots in eta-pT and eta-phi region for each RPC layer are obtained(See Figure 3.10 and 3.12). The values are described by color, each value means the probability of a bin. The probabilities are stabled as the pT increasing and the inefficiency lines are appeared(Because of the chamber gaps). The histrograms in eta-phi region collected only $pTCode \geq 25$ muons(See Figure 3.11).

Figure 3.12 shows the values of geometrical acceptance efficiency for each logic cone. By comparing with the definition of trigger tower(See Figure 2.3), it can be checked what is the reason of the inefficiencies of acceptance. Also the inefficiencies in the phi region, it can be verified from the gaps of the chambers(12 gaps, See Figure 2.4).

3.5 Verification of the Geometry and PAC Trigger Algorithm

3.5.1 Probabilities about the number of firedplanes in each tower

Using the information of eta address and firedplanes, the muons can collected and distributed in eta-fp region. By normalizing it for each logic cone, the probability about the number of firedplanes in each cone can be estimated. Figure 3.13 is the probability plot as discribed and Figure 3.14 is obtained in order to check the probabilities in condition $pTCode \geq 25$.

3.5.2 PAC Trigger Algorithm verification

RPC PAC Trigger uses the 4 out of 6 algorithm that related with definition of triggerable muon. By collecting the muons firing at least 4 RPC layers and normailizing collected plot, the probabilities for each tower can be obtained(See Figure 3.15). Similar analysis can be performed about 3 out of 3 algorithm in forward region(See Figure 3.16). The meaning of probabilities is the geometrical acceptance efficiency only for each tower. And the result of 4/6 algorithm in barrel can be added by the result of the innermost 3/4 algorithm to obtain the total geometrical efficiency of the trigger.

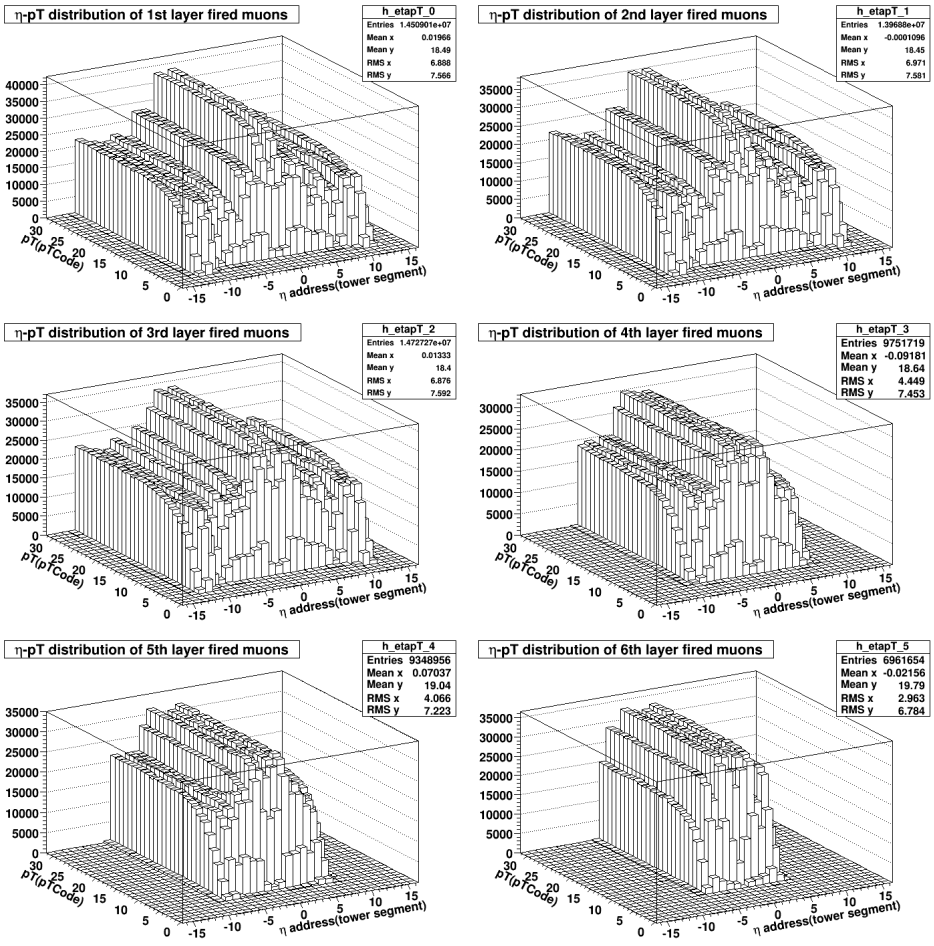

Figure 3.9: eta-pT distribution of SimTrack muon for each RPC layer

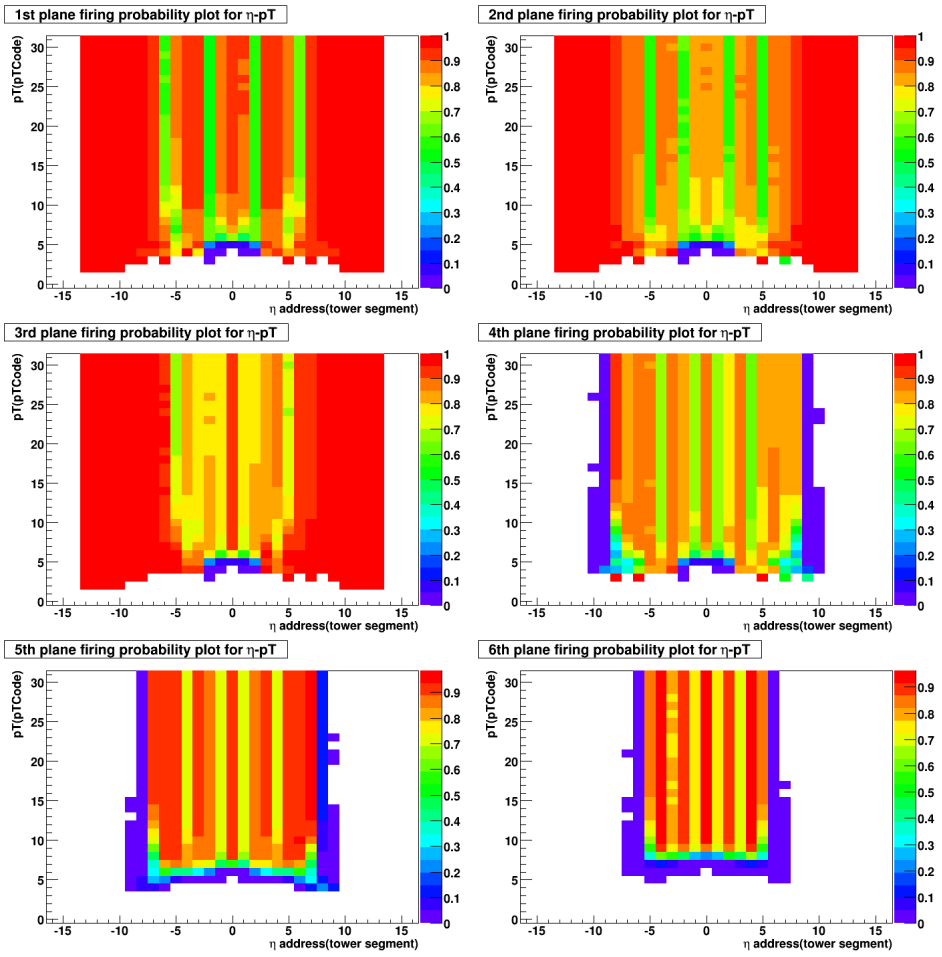

Figure 3.10: Probability plots in eta-pT region for each layer

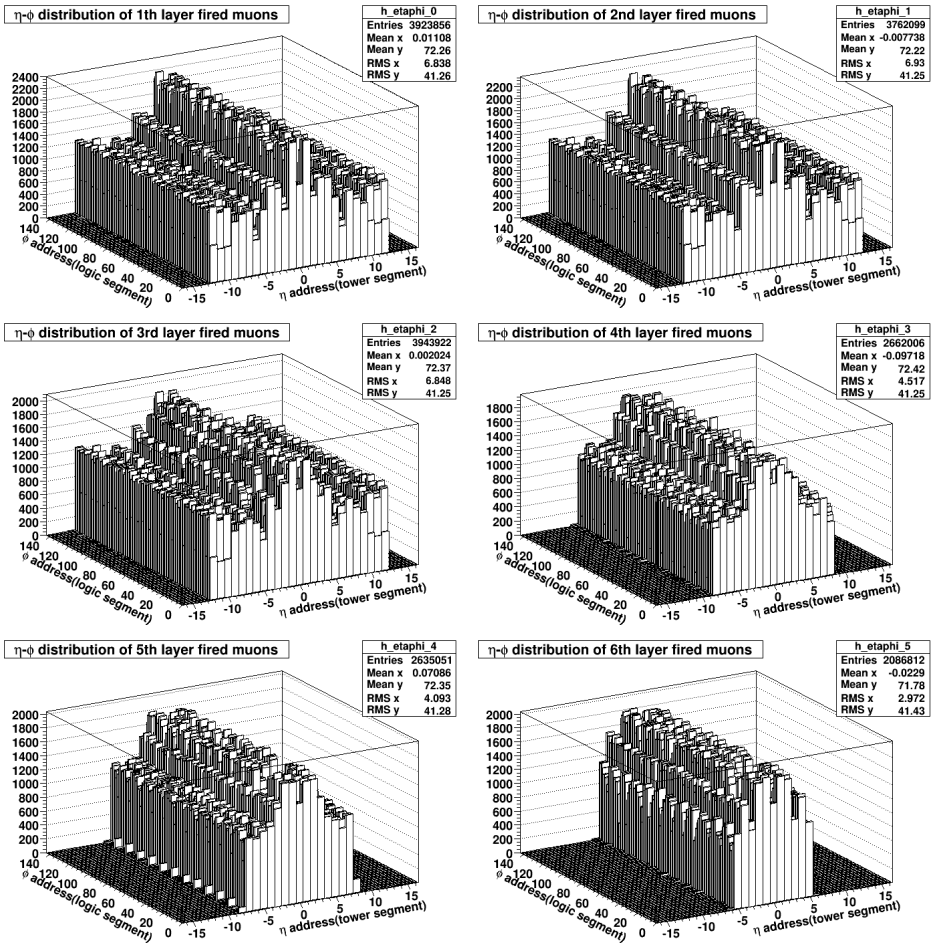

Figure 3.11: eta-phi distribution of SimTrack muon for each RPC layer

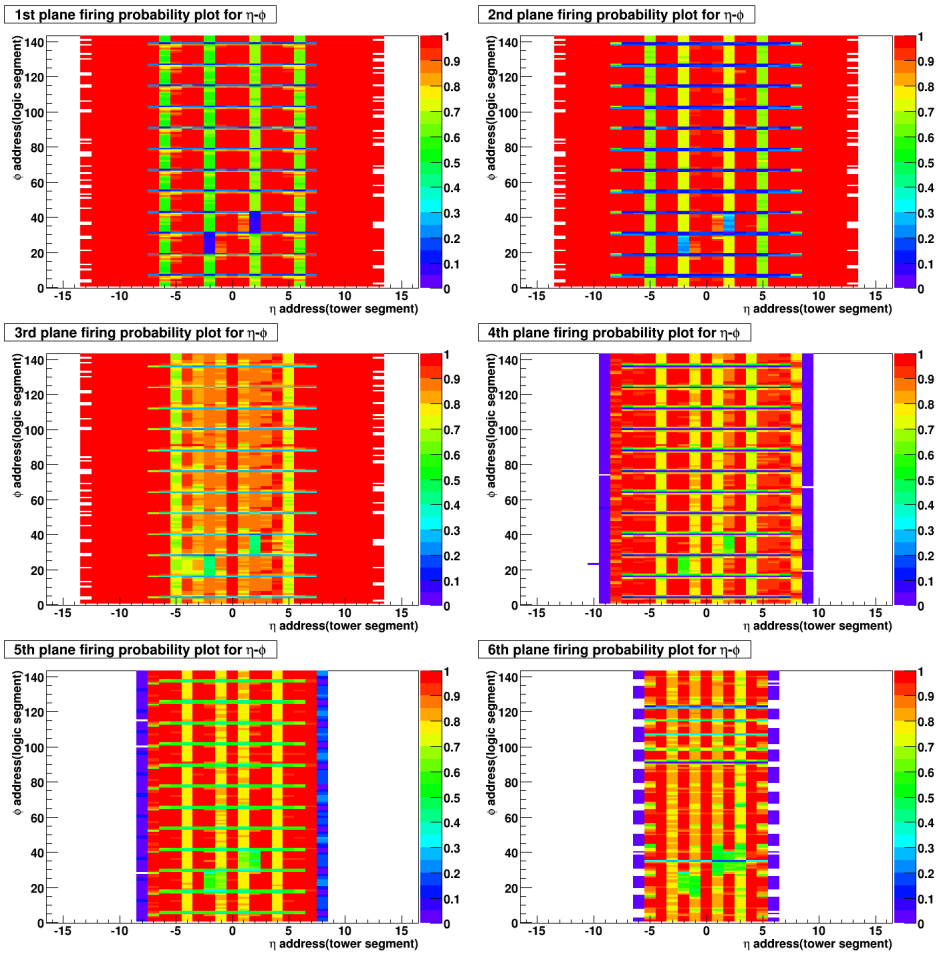

Figure 3.12: Probability plots in eta-phi region for each layer

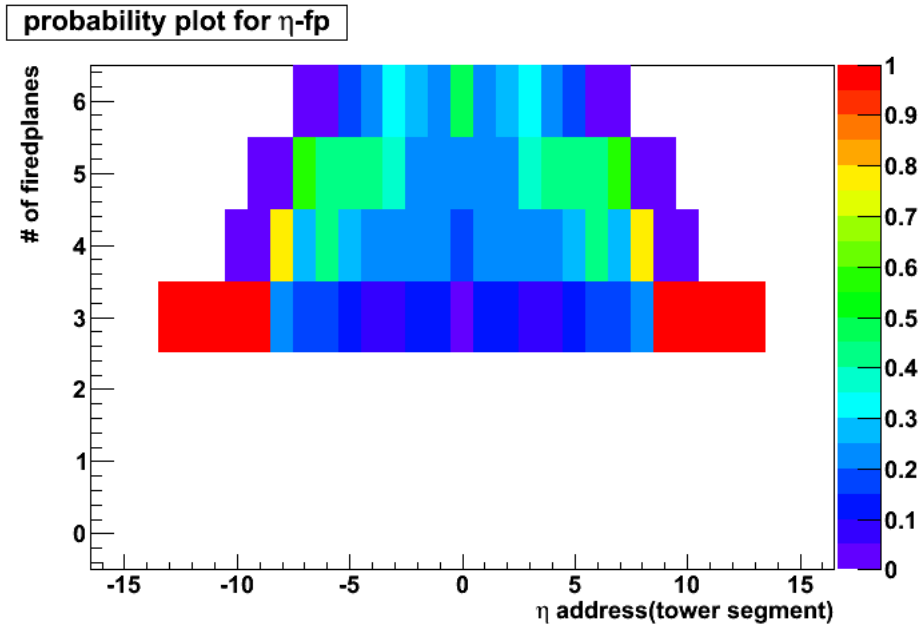

Figure 3.13: Probability plot about the number of firedplanes for each Logic Cone

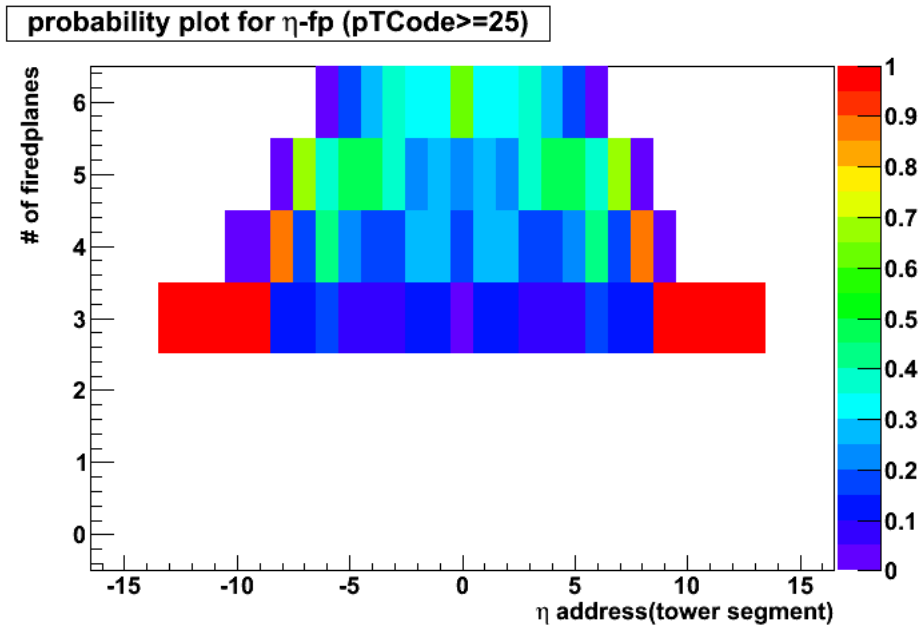

Figure 3.14: Probability plot about the number of firedplanes for each Logic Cone($pTCode \geq 25$)

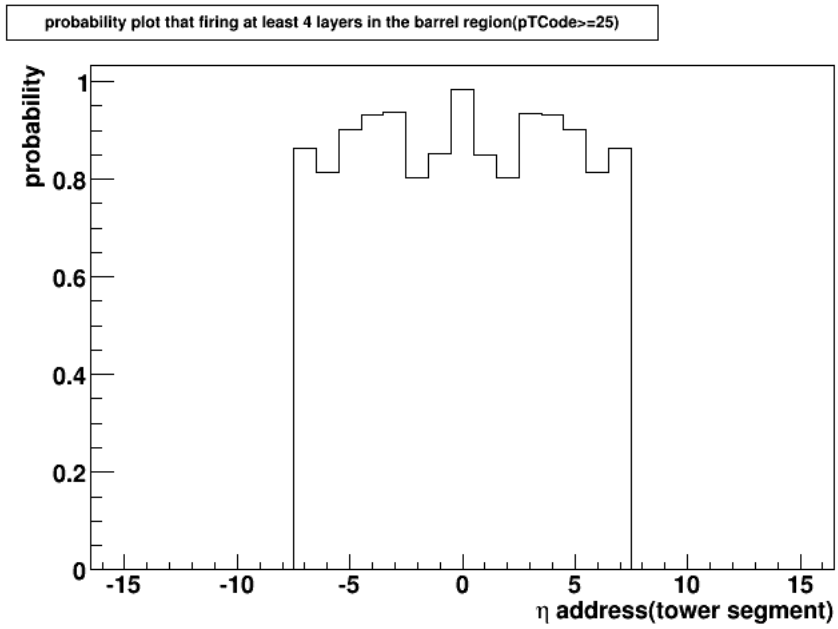

Figure 3.15: Probability plot about the muon firing at least 4 layers in barrel region($pTCode \geq 25$)

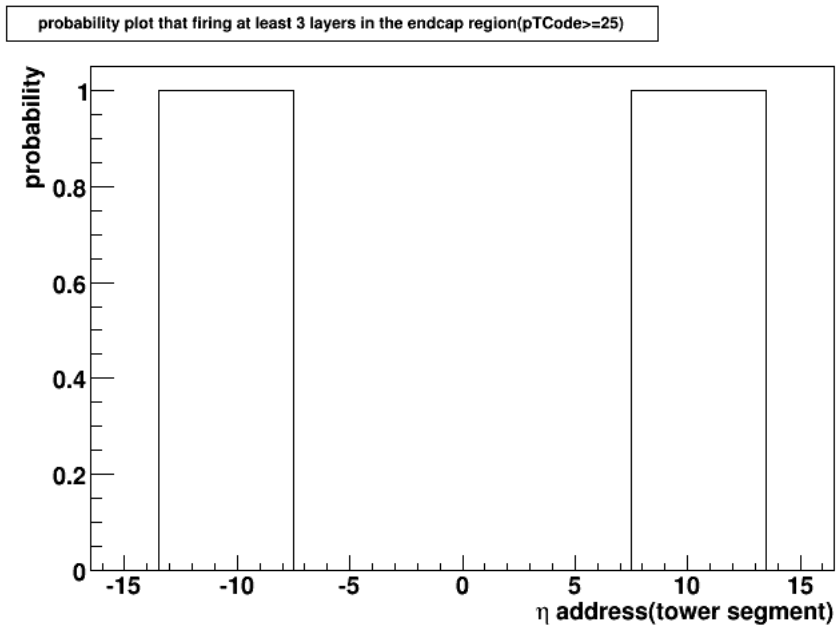

Figure 3.16: Probability plot about the muon firing at least 3 layers in forwad region($pTCode \geq 25$)

Chapter 4

Conclusion and Discussion

Analysis to obtaing the geometrical acceptance efficiency is performed well. The results of analysis are so reasonable and calculated values of the acceptance efficiency for each logical cone can be used to obtaining total PAC trigger efficiency value.

The probability plot for 3rd RPC layer in eta-phi region need to be concerned. The inefficiencies are distributed more wider than expected and it still cannot be explained. It was issued in CMS Trigger Group, need to have discussion and additional analysis. And if it needs, additional verification of the gerometry also can be archived.

Because this analysis used Monte Carlo Simulation dataset, efficiency of the RPC chamber is assumed all 100 %. Real inputs of PAC Trigger are of course real efficiency of the chambers included data. So the most realistic analysis can be performed by having this analysis with real proton-proton collision dataset. But it will not exactly same.(i.e. real dataset not includes simTrack container. Instead of simTrack, the container of global muon can be used.)

Chapter 5

Analysis Code

This chapter includes the codes to perform the analysis in this thesis.

RPCTrigger.cc is main code for analysis. On the web site for reference, there is equal name of cc file. RPCTrigger.cc in this thesis is based on that reference code, but it was added and corrected many of parts.

RPCTrigger.h is the header file for RPCTrigger.cc. Also it based on the header file in reference.

probability.c is the ROOT macro that in order to obtain the probability plots from the result(root file).

5.1 RPCTrigger.cc

```
#include "L1Trigger/RPCTrigger/interface/RPCTrigger.h"

// Configuration via eventsetup:
#include "CondFormats/DataRecord/interface/L1RPCConfigRcd.h"
#include "CondFormats/L1TObjects/interface/L1RPCConfig.h"

#include "CondFormats/DataRecord/interface/L1RPCConeBuilderRcd.h"
#include "CondFormats/RPCObjects/interface/L1RPCConeBuilder.h"

#include "CondFormats/DataRecord/interface/L1RPCHwConfigRcd.h"
#include "CondFormats/RPCObjects/interface/L1RPCHwConfig.h"
```

```
#include "CondFormats/DataRecord/interface/L1RPCBxOrConfigRcd.h"
#include "CondFormats/L1TObjects/interface/L1RPCBxOrConfig.h"
#include "SimDataFormats/RPCDigiSimLink/interface/RPCDigiSimLink.h"

#include "SimDataFormats/Track/interface/SimTrackContainer.h"
#include "SimDataFormats/Track/interface/SimTrack.h"
#include "SimDataFormats/Track/interface/CoreSimTrack.h"
#include "L1Trigger/RPCTrigger/interface/RPCConst.h"
//#define ML_DEBUG



#include "L1Trigger/RPCTrigger/interface/MuonsGrabber.h"

RPCTrigger::RPCTrigger(const edm::ParameterSet& iConfig):
   outputFile_(iConfig.getUntrackedParameter<std::string>("hOutputFile")),
   m_trigConfig(0),m_pacTrigger(0)
{

  produces<std::vector<L1MuRegionalCand> >("RPCb");
  produces<std::vector<L1MuRegionalCand> >("RPCf");

  produces<std::vector<RPCDigiL1Link> >("RPCb");
  produces<std::vector<RPCDigiL1Link> >("RPCf");

  m_firstRun = true;
  m_cacheID = 0;

  m_triggerDebug = iConfig.getUntrackedParameter<int>("RPCTriggerDeb
ug",0);
  if ( m_triggerDebug != 1){
     m_triggerDebug = 0;
  }
  m_label = iConfig.getParameter<std::string>("label");
}

RPCTrigger::~RPCTrigger(){
   if (m_pacTrigger != 0) delete m_pacTrigger;
   if (m_trigConfig != 0) delete m_trigConfig;
}
```

```
void RPCTrigger::beginJob() {

  hOutputFile = new TFile( outputFile_.c_str(), "RECREATE");
  TTree *T = new TTree("T", "ses tree");

  T -> Branch("eta", &eta, "eta/F");
  T -> Branch("phi", &phi, "phi/F");
  T -> Branch("pT", &pT, "pT/D");
  T -> Branch("fp", &fp, "fp/F");
  T -> Branch("qual", &eta, "qual/F");
  T -> Branch("num_fp", &num_fp, "num_fp/F");

//define the histograms

  h_testTri_pT = new TH1F( "h_selMu_pT", "pT distribution of selected
TBmuons", 32, -0.5, 31.5);
  h_testTri_fp = new TH1F( "h_selMu_firedPlanes", "(# of fp) distribution
 of selected TBmuons", 7, -0.5, 6.5);

  h_selMu_firedPlanes_pb = new TH1F("h_selMu_firedPlanes_pb", "prob
ability of firing # of planes", 7, -0.5, 6.5);

  h_deleta = new TH1F( "h_deleta", "#Delta#eta distribution of muons", 6
5, -32.5, 32.5);
  h_delphi = new TH1F( "h_delphi", "#Delta#phi distribution of muons", 2
87, -143.5, 143.5);

  h_simTrack_fp = new TH1F( "h_simTrack_fp", "(# of fp) distribution of
simTrack muons", 7, -0.5, 6.5);
  h_simTrack_fp_1 = new TH1F( "h_simTrack_fp_1", "(# of fp) distributio
n of rejected simTrack muons", 7, -0.5, 6.5);

  h_simTrack_pT_1 = new TH1F( "h_simTrack_pT_1", "pT distribution of
 rejected simTrack muons", 32, -0.5, 31.5);
  h_simTrack_pT = new TH1F( "h_simTrack_pT", "pT distribution of sim
Track muons", 32, -0.5, 31.5);
  h_simTrack_eta_1 = new TH1F( "h_simTrack_eta_1", "#eta distribution
 of rejected simTrack muons", 33, -16.5, 16.5);
  h_simTrack_eta = new TH1F( "h_simTrack_eta", "#eta distribution of s
imTrack muons", 33, -16.5, 16.5);
  h_simTrack_etapT_1 = new TH2F( "h_simTrack_etapT_1", "#eta-pT di
```

```
stribution of rejected simTrack muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_simTrack_etapT = new TH2F( "h_simTrack_etapT", "#eta-pT distrib
ution of simTrack muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_simTrack_etaphi_1 = new TH2F( "h_simTrack_etaphi_1", "#eta-#phi
 distribution of rejected simTrack muons", 33, -16.5, 16.5, 144, -0.5, 143.5);
  h_simTrack_etaphi = new TH2F( "h_simTrack_etaphi", "#eta-#phi dist
ribution of muons", 33, -16.5, 16.5, 144, -0.5, 143.5);

  h_deletapT = new TH2F( "h_deletapT", "#Delta #eta-pT", 65, -32.5, 32.
5, 32, -0.5, 31.5);
  h_delphipT = new TH2F( "h_delphipT", "#Delta #phi-pT", 287, -143.5,
143.5, 32, -0.5, 31.5);
  h_etafp = new TH2F("h_etafp", "#eta-(# of fp) distribution of muons", 3
3, -16.5, 16.5, 7, -0.5, 6.5);
  h_etafp_25pT = new TH2F("h_etafp_25pT", "#eta-#fp distribution of pT
>= 25 muons", 33, -16.5, 16.5, 7, -0.5, 6.5);
  h_pTfp_br = new TH2F("h_pTfp_br", "pT-(# of fp) distribution of muon
s in barrel region", 32, -0.5, 31.5, 7, -0.5, 6.5);
  h_pTfp_ec = new TH2F("h_pTfp_ec", "pT-(# of fp) distribution of muo
ns in endcap region", 32, -0.5, 31.5, 7, -0.5, 6.5);
  h_etapTfp3_1or2_3or4 = new TH2F( "h_etapTfp3_1or2_3or4", "the muon
s that (# of fp) = 3 and (1 or 2) & (3 or 4) layer is fired", 33, -16.5, 16.5,
32, -0.5, 31.5);
  h_etapTfp3_5or6 = new TH2F( "h_etapTfp3_5or6", "the muons that (# o
f fp) = 3 and (5 or 6) layer is fired", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_br_innermost3of4_eta = new TH1F( "h_br_innermost3of4_eta", "the
muons that firing at least 3 of 4 innermost layers", 33, -16.5, 16.5);

  h_etapT_3_01 = new TH2F( "h_etapT_3_01", "#eta-pT distribution of (
# of fp) >= 3 muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapT_3_23 = new TH2F( "h_etapT_3_23", "distribution for #eta-pT,
fp >= 3", 33, -16.5, 16.5, 32, -0.5, 31.5);

  h_etapTfp_0 = new TH2F( "h_etapTfp_0", "#eta-pT distribution of (# of
 fp) = 0 muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapTfp_1 = new TH2F( "h_etapTfp_1", "#eta-pT distribution of (# of
 fp) = 1 muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapTfp_2 = new TH2F( "h_etapTfp_2", "#eta-pT distribution of (# of
 fp) = 2 muons", 33, -16.5, 16.5, 32, -0.5, 31.5)
  h_etapTfp_3 = new TH2F( "h_etapTfp_3", "#eta-pT distribution of (# of
 fp) = 3 muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
```

```
  h_etapTfp_4 = new TH2F( "h_etapTfp_4", "#eta-pT distribution of (# of
 fp) = 4 muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapTfp_5 = new TH2F( "h_etapTfp_5", "#eta-pT distribution of (# of
 fp) = 5 muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapTfp_6 = new TH2F( "h_etapTfp_6", "#eta-pT distribution of (# of
 fp) = 6 muons", 33, -16.5, 16.5, 32, -0.5, 31.5);

  h_etapT = new TH2F( "h_etapT", "#eta-pT distribution of muons", 33,
-16.5, 16.5, 32, -0.5, 31.5);
  h_etapT_0 = new TH2F( "h_etapT_0", "#eta-pT distribution of 1st layer
 fired muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapT_1 = new TH2F( "h_etapT_1", "#eta-pT distribution of 2nd layer
 fired muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapT_2 = new TH2F( "h_etapT_2", "#eta-pT distribution of 3rd layer
 fired muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapT_3 = new TH2F( "h_etapT_3", "#eta-pT distribution of 4th layer
fired muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapT_4 = new TH2F( "h_etapT_4", "#eta-pT distribution of 5th layer
fired muons", 33, -16.5, 16.5, 32, -0.5, 31.5);
  h_etapT_5 = new TH2F( "h_etapT_5", "#eta-pT distribution of 6th layer
 fired muons", 33, -16.5, 16.5, 32, -0.5, 31.5);

  h_etaphi = new TH2F( "h_etaphi", "#eta-#phi distribution of muons", 33
, -16.5, 16.5, 144, -0.5, 143.5);
  h_etaphi_0 = new TH2F( "h_etaphi_0", "#eta-#phi distribution of 1th lay
er fired muons", 33, -16.5, 16.5, 144, -0.5, 143.5);
  h_etaphi_1 = new TH2F( "h_etaphi_1", "#eta-#phi distribution of 2nd la
yer fired muons", 33, -16.5, 16.5, 144, -0.5, 143.5);
  h_etaphi_2 = new TH2F( "h_etaphi_2", "#eta-#phi distribution of 3rd la
yer fired muons", 33, -16.5, 16.5, 144, -0.5, 143.5);
  h_etaphi_3 = new TH2F( "h_etaphi_3", "#eta-#phi distribution of 4th lay
er fired muons", 33, -16.5, 16.5, 144, -0.5, 143.5);
  h_etaphi_4 = new TH2F( "h_etaphi_4", "#eta-#phi distribution of 5th lay
er fired muons", 33, -16.5, 16.5, 144, -0.5, 143.5);
  h_etaphi_5 = new TH2F( "h_etaphi_5", "#eta-#phi distribution of 6th lay
er fired muons", 33, -16.5, 16.5, 144, -0.5, 143.5);

}

void
RPCTrigger::produce(edm::Event& iEvent, const edm::EventSetup& iS
```

```
etup)
{

    edm::Handle<edm::SimTrackContainer> simTracks;
    iEvent.getByLabel("g4SimHits",simTracks);

  if ( m_triggerDebug == 1)  MuonsGrabber::Instance().startNewEve
nt(iEvent.id().event(), iEvent.bunchCrossing());

  if (m_firstRun){

     m_cacheID = iSetup.get<L1RPCConfigRcd>().cacheIdentifier();
     m_firstRun = false;
     edm::ESHandle<L1RPCConfig> conf;
     iSetup.get<L1RPCConfigRcd>().get(conf);
     const L1RPCConfig *rpcconf = conf.product();


     m_pacManager.init(rpcconf);
     m_trigConfig = new RPCBasicTrigConfig(&m_pacManager);
     m_trigConfig->setDebugLevel(m_triggerDebug);
     m_pacTrigger = new RPCPacTrigger(m_trigConfig);
     if ( m_triggerDebug == 1)  MuonsGrabber::Instance().setRPCBa
sicTrigConfig(m_trigConfig);

std::cout<<"middle of produce"<<std::endl;
  }

  if (m_cacheID != iSetup.get<L1RPCConfigRcd>().cacheIdentifier()) {

      std::cout << " New pats: " << iSetup.get<L1RPCConfigRcd>()
.cacheIdentifier() << std::endl ;
      m_cacheID = iSetup.get<L1RPCConfigRcd>().cacheIdentifier();

     edm::ESHandle<L1RPCConfig> conf;
     iSetup.get<L1RPCConfigRcd>().get(conf);
     const L1RPCConfig *rpcconf = conf.product();

     m_pacManager.init(rpcconf);
     delete m_trigConfig;
     m_trigConfig = new RPCBasicTrigConfig(&m_pacManager);
```

```
     m_trigConfig->setDebugLevel(m_triggerDebug);

     delete m_pacTrigger;
     m_pacTrigger = new RPCPacTrigger(m_trigConfig);

     if ( m_triggerDebug == 1)  MuonsGrabber::Instance().setRPCBa
sicTrigConfig(m_trigConfig);

std::cout << "produce end" << std::endl;
  }

  edm::Handle<RPCDigiCollection> rpcDigis;
  iEvent.getByLabel(m_label, rpcDigis);

  std::auto_ptr<std::vector<L1MuRegionalCand> > candBarell(new
 std::vector<L1MuRegionalCand>);
  std::auto_ptr<std::vector<L1MuRegionalCand> > candForward(n
ew std::vector<L1MuRegionalCand>);

  if (!rpcDigis.isValid())
  {
     LogDebug("RPCTrigger")
          << "\nWarning: RPCDigiCollection with input tag " << m_l
abel
          << "\nrequested in configuration, but not found in the eve
nt. Emulator will produce empty collection \n ";

          iEvent.put(candBarell, "RPCb");
          iEvent.put(candForward, "RPCf");

          return;
  }

  if (rpcDigis->begin() == rpcDigis->end() )
  {
     LogDebug("RPCTrigger")
          << "\nWarning: RPCDigiCollection with input tag " << m_la
bel
          << "\n seems to be empty for this event. Emulator will run
 on empty collection ";
```

```
  }

  std::auto_ptr<std::vector<RPCDigiL1Link> >  brlLinks(new std::v
ector<RPCDigiL1Link>);
  std::auto_ptr<std::vector<RPCDigiL1Link> >  fwdLinks(new std:
:vector<RPCDigiL1Link>);

  for (int iBx = 0; iBx < 1; ++ iBx) {

    L1RpcLogConesVec ActiveCones;

    edm::ESHandle<L1RPCConeBuilder> coneBuilder;
    iSetup.get<L1RPCConeBuilderRcd>().get(coneBuilder);

    edm::ESHandle<L1RPCConeDefinition> l1RPCConeDefinition;
    iSetup.get<L1RPCConeDefinitionRcd>().get(l1RPCConeDefinition);

    edm::ESHandle<L1RPCHwConfig> hwConfig;
    iSetup.get<L1RPCHwConfigRcd>().get(hwConfig);

    edm::ESHandle<L1RPCBxOrConfig> bxOrConfig;
    iSetup.get<L1RPCBxOrConfigRcd>().get(bxOrConfig);

    ActiveCones = m_theLinksystemFromES.getConesFromES(rpcDi
gis, coneBuilder, l1RPCConeDefinition, bxOrConfig, hwConfig, iBx);

    edm::ESHandle<L1RPCHsbConfig> hsbConfig;
    iSetup.get<L1RPCHsbConfigRcd>().get(hsbConfig);

    L1RpcTBMuonsVec2 finalMuons = m_pacTrigger->runEvent(Activ
eCones, hsbConfig);
    std::cout<<"finalMuons : "<<finalMuons.size()<<std::endl;

    std::vector<RPCDigiL1Link> dlBrl;
    std::vector<RPCDigiL1Link> dlFwd;

    std::vector<L1MuRegionalCand> RPCb = giveFinallCandindates
(finalMuons[0],1, iBx, rpcDigis, dlBrl, simTracks);
    std::vector<L1MuRegionalCand> RPCf = giveFinallCandindates
(finalMuons[1],3, iBx, rpcDigis, dlFwd, simTracks);
```

```
    std::cout<<"size of 0 :"<<finalMuons[0].size()<<std::endl;
    std::cout<<"size of 1 :"<<finalMuons[1].size()<<std::endl;

//TBmuon selection by comparing Code of finalMuons

    RPCTBMuon selMu;
    if (finalMuons[0][0].getCode() > 0 && finalMuons[1][0].getCode(
) > 0) {
        if (finalMuons[0][0].getCode() <= finalMuons[1][0].getCode()
) {
            selMu = finalMuons[1][0];
        }
        else {
            selMu = finalMuons[0][0];
        }
    }
    else if (finalMuons[0][0].getCode() > 0) {
            selMu = finalMuons[0][0];
    }
    else if (finalMuons[1][0].getCode() > 0) {
            selMu = finalMuons[1][0];
    }

    std::bitset<6> fp_bitset(selMu.getFiredPlanes());
    if (selMu.getCode() == 0) {
        fp_bitset.reset();
    }

    num_fp = fp_bitset.count();
    fp = selMu.getFiredPlanes();

    LogDebug("RPCTrigger") << "pt of Muons "
        << selMu.getPtCode()
        << " fp " << selMu.getFiredPlanes()
        << " binary_fp " << std::bitset<6>(selMu.getFiredPlanes())
        << " number of firedplanes : " << num_fp
        << " phi_selMu " << selMu.getPhiAddr()
        << " tower " << selMu.getTower();

    std::cout << "pt of Muons "
        << selMu.getPtCode()
```

```
        << " fp " << selMu.getFiredPlanes()
        << " binary of fp : " << std::bitset<6>(selMu.getFiredPlanes())
        << " 3rd fp : " << std::bitset<6>(selMu.getFiredPlanes())[2]
        << " number of firedplanes : " << num_fp
        << " phi_selMu " << selMu.getPhiAddr()
        << " tower " << selMu.getTower()<<std::endl;

        if (selMu.getCode() > 0) {
            h_testTri_pT->Fill(selMu.getPtCode());
            h_testTri_fp->Fill(num_fp);
        }

        double simpT = 0;
        double simeta = 0;
        double simphi = 0;

        int num_sim = 0;

        edm::SimTrackContainer::const_iterator simTrack;

        for (simTrack = simTracks->begin(); simTrack != simTrack
s->end(); ++simTrack) {
            if (abs((*simTrack).type()==13)) {
                ++num_sim;
                simeta = (*simTrack).momentum().eta();
                simphi = (*simTrack).momentum().phi();
                simpT = sqrt((*simTrack).momentum().perp2());
                pT = RPCConst::iptFromPt(simpT);
                eta = RPCConst::towerNumFromEta(simeta);
                phi = RPCConst::segmentNumFromPhi(simphi);

                if (phi > 144) {
                    phi = phi - 144;
                }
                else {
                    phi = phi;
                }

                double deleta = selMu.getTower() - eta;
                double delphi = selMu.getPhiAddr() - phi;
```

```
//declare the constraints and fill up

                if ((num_sim == 1) && (pT != 1) && (deleta <= +2 && de
leta >= -2)) {

                h_deleta->Fill(deleta);
                h_delphi->Fill(delphi);

                h_etafp->Fill(eta,num_fp);

                if ((eta >= -7) && (eta <= +7)) {
                    h_pTfp_br->Fill(pT,num_fp);
                }
                else {
                    h_pTfp_ec->Fill(pT,num_fp);
                }

                if ((num_fp == 3) && (fp_bitset[0] == 1 || fp_bitset[1] ==
 1) && (fp_bitset[2] == 1 || fp_bitset[3] == 1)) {
                    h_etapT_3_01->Fill(eta,pT);
                }
                if ((num_fp == 3) && (fp_bitset[0] == 1 || fp_bitset[1] ==
 1) && (fp_bitset[2] == 1 || fp_bitset[3] == 1)) {
                    h_etapTfp3_1or2_3or4->Fill(eta,pT);
                }
                if ((num_fp == 3) && (fp_bitset[4] == 1 || fp_bitset[5] ==
 1)) {
                    h_etapTfp3_5or6->Fill(eta,pT);
                }
                if (fp == 7 || fp == 11 || fp == 13 || fp == 14) {
                h_br_innermost3of4_eta -> Fill(eta);
                }

                h_deletapT->Fill(deleta,pT);
                h_delphipT->Fill(delphi,pT);
                h_simTrack_pT->Fill(pT);
                h_simTrack_fp->Fill(num_fp);
                h_simTrack_eta->Fill(eta);
                h_simTrack_etapT->Fill(eta, pT);
                h_simTrack_etaphi->Fill(eta, phi);
```

```
if (num_fp == 0){
    h_etapTfp_0->Fill(eta, pT);
}
if (num_fp == 1){
    h_etapTfp_1->Fill(eta, pT);
}
if (num_fp == 2){
    h_etapTfp_2->Fill(eta, pT);
}
if (num_fp == 3){
    h_etapTfp_3->Fill(eta, pT);
}
if (num_fp == 4){
    h_etapTfp_4->Fill(eta, pT);
}
if (num_fp == 5){
    h_etapTfp_5->Fill(eta, pT);
}
if (num_fp == 6){
    h_etapTfp_6->Fill(eta, pT);
}

h_etapT->Fill(eta, pT);
if (fp_bitset[0] == 1){
    h_etapT_0->Fill(eta, pT);
}
if (fp_bitset[1] == 1){
    h_etapT_1->Fill(eta, pT);
}
if (fp_bitset[2] == 1){
    h_etapT_2->Fill(eta, pT);
}
if (fp_bitset[3] == 1){
    h_etapT_3->Fill(eta, pT);
}
if (fp_bitset[4] == 1){
    h_etapT_4->Fill(eta, pT);
}
if (fp_bitset[5] == 1){
    h_etapT_5->Fill(eta, pT);
}
```

```
                if (pT >= 25) {

                    h_etafp_25pT->Fill(eta, num_fp);

                    h_etaphi->Fill(eta, phi);
                    if (fp_bitset[0] == 1){
                        h_etaphi_0->Fill(eta, phi);
                    }
                    if (fp_bitset[1] == 1){
                        h_etaphi_1->Fill(eta, phi);
                    }
                    if (fp_bitset[2] == 1){
                        h_etaphi_2->Fill(eta, phi);
                    }
                    if (fp_bitset[3] == 1){
                        h_etaphi_3->Fill(eta, phi);
                    }
                    if (fp_bitset[4] == 1){
                        h_etaphi_4->Fill(eta, phi);
                    }
                    if (fp_bitset[5] == 1){
                        h_etaphi_5->Fill(eta, phi);
                    }
                }
            }
            else {
                h_simTrack_fp_1->Fill(num_fp);
                h_simTrack_pT_1->Fill(pT);
                h_simTrack_eta_1->Fill(eta);
                h_simTrack_etapT_1->Fill(eta, pT);
                h_simTrack_etaphi_1->Fill(eta, phi);
                    }
            }
        std::cout<<"Number of simtrack muons : "<<num_sim<<st
d::endl;
        }

    brlLinks->insert(brlLinks->end(), dlBrl.begin(), dlBrl.end() );
    fwdLinks->insert(fwdLinks->end(), dlFwd.begin(), dlFwd.end(
) );
```

```
    candBarell->insert(candBarell->end(), RPCb.begin(), RPCb.en
d());
    candForward->insert(candForward->end(), RPCf.begin(), RPC
f.end());

    if ( m_triggerDebug == 1)  MuonsGrabber::Instance().writeDataF
orRelativeBX(iBx);
  }

  iEvent.put(fwdLinks, "RPCf");
  iEvent.put(brlLinks, "RPCb");
  iEvent.put(candBarell, "RPCb");
  iEvent.put(candForward, "RPCf");

}

std::vector<L1MuRegionalCand> RPCTrigger::giveFinallCandinda
tes(L1RpcTBMuonsVec finalMuons, int type, int bx, edm::Handle
<RPCDigiCollection> rpcDigis, std::vector<RPCDigiL1Link> & retR
PCDigiLink, edm::Handle<edm::SimTrackContainer> simTracks)
{

  std::vector<L1MuRegionalCand> RPCCand;

  return RPCCand;
}

void RPCTrigger::endJob() {

//write the histograms

        hOutputFile->SetCompressionLevel(2);
        hOutputFile->cd();

        h_testTri_pT -> Write();
        h_testTri_fp -> Write();
        h_selMu_firedPlanes_pb -> Write();

        h_etafp -> Write();
        h_etafp_25pT -> Write();
```

```
        h_deleta -> Write();
        h_delphi -> Write();
        h_deletapT -> Write();
        h_delphipT -> Write();
        h_pTfp_br -> Write();
        h_pTfp_ec -> Write();
        h_etapT_3_01 -> Write();
        h_etapT_3_23 -> Write();

        h_etapTfp_0 -> Write();
        h_etapTfp_1 -> Write();
        h_etapTfp_2 -> Write();
        h_etapTfp_3 -> Write();
        h_etapTfp_4 -> Write();
        h_etapTfp_5 -> Write();
        h_etapTfp_6 -> Write();

        h_etapT -> Write();
        h_etapT_0 -> Write();
        h_etapT_1 -> Write();
        h_etapT_2 -> Write();
        h_etapT_3 -> Write();
        h_etapT_4 -> Write();
        h_etapT_5 -> Write();
h_etaphi -> Write();
        h_etaphi_0 -> Write();
        h_etaphi_1 -> Write();
        h_etaphi_2 -> Write();
        h_etaphi_3 -> Write();
        h_etaphi_4 -> Write();
        h_etaphi_5 -> Write();
        h_br_innermost3of4_eta -> Write();
        h_etapTfp3_1or2_3or4 -> Write();
        h_etapTfp3_5or6 -> Write();

        h_simTrack_fp_1->Write();
        h_simTrack_pT_1->Write();
        h_simTrack_eta_1->Write();
        h_simTrack_etapT_1->Write();
        h_simTrack_etaphi_1->Write();
        h_simTrack_fp->Write();
```

```
        h_simTrack_pT->Write();
        h_simTrack_eta->Write();
        h_simTrack_etapT->Write();
        h_simTrack_etaphi->Write();

hOutputFile->Close();

}
```

5.2 RPCTrigger.h

```
#ifndef L1Trigger_RPCTrigger_h
#define L1Trigger_RPCTrigger_h

#include "FWCore/Framework/interface/Frameworkfwd.h"
#include "FWCore/Framework/interface/EDProducer.h"

#include "FWCore/Framework/interface/Event.h"
#include "FWCore/Framework/interface/MakerMacros.h"

#include "FWCore/ParameterSet/interface/ParameterSet.h"

#include "DataFormats/MuonDetId/interface/RPCDetId.h"
#include "DataFormats/RPCDigi/interface/RPCDigiCollection.h"


#include <FWCore/Framework/interface/ESHandle.h>
#include "FWCore/MessageLogger/interface/MessageLogger.h"

#include "DataFormats/L1GlobalMuonTrigger/interface/L1MuRegi
onalCand.h"


// L1RpcTrigger specific includes
#include "L1Trigger/RPCTrigger/interface/RPCConeBuilderFromE
S.h"

#include "L1Trigger/RPCTrigger/interface/RPCPacManager.h"
```

```
#include "L1Trigger/RPCTrigger/interface/RPCPacTrigger.h"
#include "L1Trigger/RPCTrigger/interface/RPCBasicTrigConfig.h"
#include "L1Trigger/RPCTrigger/interface/RPCPacData.h"
#include "L1Trigger/RPCTrigger/interface/RPCConst.h"
#include "L1Trigger/RPCTrigger/interface/RPCPacManager.h"
#include "CondFormats/DataRecord/interface/L1RPCHsbConfigRc
d.h"
#include "CondFormats/L1TObjects/interface/L1RPCHsbConfig.h"
#include "DataFormats/RPCDigi/interface/RPCDigiL1Link.h"
#include "SimDataFormats/RPCDigiSimLink/interface/RPCDigiSim
Link.h"

#include "SimDataFormats/Track/interface/SimTrackContainer.h"
#include "SimDataFormats/Track/interface/SimTrack.h"
#include "SimDataFormats/Track/interface/CoreSimTrack.h"
#include "L1Trigger/RPCTrigger/interface/RPCConst.h"

#include <memory>
#include <vector>

#include <TH1.h>
#include <TFile.h>
#include <TString.h>
#include <stdio.h>
#include <TH2F.h>
#include <TNtuple.h>
#include <TROOT.h>
#include <TTree.h>

//class RPCTriggerGeo;

class RPCTrigger : public edm::EDProducer {
  public:
    explicit RPCTrigger(const edm::ParameterSet&);
    ~RPCTrigger();

    virtual void produce(edm::Event&, const edm::EventSetup&);
  private:
      virtual void beginJob();
      virtual void endJob();
```

```
      // ----------member data ---------------------------
    std::string theDataType;

    std::string outputFile_;
    TFile *hOutputFile;

    float eta, phi, fp, qual, num_fp;
    double pT;

    TH1F *h_testTri_pT, *h_testTri_fp, *h_selMu_firedPlanes_pb, *h
_br_innermost3of4_eta;
    TH1F *h_simTrack_pT, *h_simTrack_pT_1, *h_simTrack_eta, *h
_simTrack_eta_1, *h_simTrack_fp, *h_simTrack_fp_1;
    TH1F *h_deleta, *h_delphi;

    TH2F *h_etapTfp_0, *h_etapTfp_1, *h_etapTfp_2, *h_etapTfp_3,
*h_etapTfp_4, *h_etapTfp_5, *h_etapTfp_6;
    TH2F *h_simTrack_etapT, *h_simTrack_etaphi, *h_simTrack_et
apT_1, *h_simTrack_etaphi_1;
    TH2F *h_etapT, *h_etapT_0, *h_etapT_1, *h_etapT_2, *h_etapT_
3, *h_etapT_4, *h_etapT_5, *h_etaphi, *h_etaphi_0, *h_etaphi_1, *h_
etaphi_2, *h_etaphi_3, *h_etaphi_4, *h_etaphi_5;

    TH2F *h_etafp, *h_etafp_25pT, *h_pTfp_br, *h_pTfp_ec, *h_etap
T_3_01, *h_etapT_3_23, *h_deletapT, *h_delphipT;
    TH2F *h_etapTfp3_1or2_3or4, *h_etapTfp3_5or6;

    static int iptFromPt(const double pt);

    RPCConeBuilderFromES m_theLinksystemFromES;

    RPCPacManager<RPCPacData> m_pacManager;

    RPCBasicTrigConfig* m_trigConfig;

    RPCPacTrigger* m_pacTrigger;

    bool m_firstRun;
    int m_triggerDebug;
    unsigned long long m_cacheID;
```

```
    std::vector<L1MuRegionalCand> giveFinallCandindates(L1RpcT
BMuonsVec finalMuons, int type, int bx, edm::Handle<RPCDigiCol
lection> rpcDigis, std::vector<RPCDigiL1Link> & retRPCDigiLink, e
dm::Handle<edm::SimTrackContainer> simTracks);

    std::string m_label;
};


#endif
```

5.3 probability.c

```
#include <Riostream.h>
#include <TSystem.h>
#include <TProfile.h>
#include <TBrowser.h>
#include <TROOT.h>
#include <TGraph.h>
#include <TString.h>
#include <TH2.h>
#include <TFile.h>
#include <TMath.h>
#include <TCanvas.h>
#include <TLegend.h>
#include <TStyle.h>
#include <TInterpreter.h>
#include <TGraphAsymmErrors.h>

void macro_probability(){

    gROOT->Reset();
    gROOT->Clear();
    gROOT->SetStyle("Plain");

    TFile *f1 = new TFile("testl1_histo.root");
    TFile f("h_probability.root", "new");

    TH1F *h_selMu_pT = (TH1F*)f1->Get("h_selMu_pT");
```

```
    TH1F *h_selMu_firedPlanes = (TH1F*)f1->Get("h_selMu_firedPla
nes");
    TH1F *h_selMu_firedPlanes_pb = (TH1F*)f1->Get("h_selMu_fired
Planes_pb");
    TH1F *h_deleta = (TH1F*)f1->Get("h_deleta");
    TH1F *h_delphi = (TH1F*)f1->Get("h_delphi");
    TH1F *h_simTrack_pT = (TH1F*)f1->Get("h_simTrack_pT");
    TH1F *h_simTrack_fp = (TH1F*)f1->Get("h_simTrack_fp");
    TH1F *h_simTrack_fp_1 = (TH1F*)f1->Get("h_simTrack_fp_1");
    TH1F *h_simTrack_eta_1 = (TH1F*)f1->Get("h_simTrack_eta_1");
    TH2F *h_simTrack_etapT_1 = (TH2F*)f1->Get("h_simTrack_etap
T_1");
    TH2F *h_simTrack_etaphi_1 = (TH2F*)f1->Get("h_simTrack_etap
hi_1");
    TH1F *h_simTrack_pT_1 = (TH1F*)f1->Get("h_simTrack_pT_1");
    TH1F *h_simTrack_eta = (TH1F*)f1->Get("h_simTrack_eta");
    TH1F *h_br_innermost3of4_eta = (TH1F*)f1->Get("h_br_innermost
3of4_eta");

    TH2F *h_etapTfp3_1or2_3or4 = (TH2F*)f1->Get("h_etapTfp3_1or2_
3or4");
    TH2F *h_etapTfp3_5or6 = (TH2F*)f1->Get("h_etapTfp3_5or6");
    TH2F *h_deletapT = (TH2F*)f1->Get("h_deletapT");
    TH2F *h_delphipT = (TH2F*)f1->Get("h_delphipT");
    TH2F *h_etafp = (TH2F*)f1->Get("h_etafp");
    TH2F *h_etafp_25pT = (TH2F*)f1->Get("h_etafp_25pT");
    TH2F *h_pTfp_br = (TH2F*)f1->Get("h_pTfp_br");
    TH2F *h_pTfp_ec = (TH2F*)f1->Get("h_pTfp_ec");

    TH2F *h_etapT_3_01 = (TH2F*)f1->Get("h_etapT_3_01");

    TH2F *h_etapTfp_0 = (TH2F*)f1->Get("h_etapTfp_0");
    TH2F *h_etapTfp_1 = (TH2F*)f1->Get("h_etapTfp_1");
    TH2F *h_etapTfp_2 = (TH2F*)f1->Get("h_etapTfp_2");
    TH2F *h_etapTfp_3 = (TH2F*)f1->Get("h_etapTfp_3");
    TH2F *h_etapTfp_4 = (TH2F*)f1->Get("h_etapTfp_4");
    TH2F *h_etapTfp_5 = (TH2F*)f1->Get("h_etapTfp_5");
    TH2F *h_etapTfp_6 = (TH2F*)f1->Get("h_etapTfp_6");

    TH2F *h_etapT = (TH2F*)f1->Get("h_etapT");
    TH2F *h_etapT_0 = (TH2F*)f1->Get("h_etapT_0");
```

```
    TH2F *h_etapT_1 = (TH2F*)f1->Get("h_etapT_1");
    TH2F *h_etapT_2 = (TH2F*)f1->Get("h_etapT_2");
    TH2F *h_etapT_3 = (TH2F*)f1->Get("h_etapT_3");
    TH2F *h_etapT_4 = (TH2F*)f1->Get("h_etapT_4");
    TH2F *h_etapT_5 = (TH2F*)f1->Get("h_etapT_5");

    TH2F *h_etaphi = (TH2F*)f1->Get("h_etaphi");
    TH2F *h_etaphi_0 = (TH2F*)f1->Get("h_etaphi_0");
    TH2F *h_etaphi_1 = (TH2F*)f1->Get("h_etaphi_1");
    TH2F *h_etaphi_2 = (TH2F*)f1->Get("h_etaphi_2");
    TH2F *h_etaphi_3 = (TH2F*)f1->Get("h_etaphi_3");
    TH2F *h_etaphi_4 = (TH2F*)f1->Get("h_etaphi_4");
    TH2F *h_etaphi_5 = (TH2F*)f1->Get("h_etaphi_5");

    TH1F *h_simTrack_fp_pb = new TH1F("h_simTrack_fp_pb", "", 7, -
0.5, 6.5);
    TH1F *h_eta_4br_pb = new TH1F("h_eta_4br_pb", "", 33, -16.5, 16.5
);
    TH1F *h_eta_3ec_pb = new TH1F("h_eta_3ec_pb", "", 33, -16.5, 16.
5);
    TH1F *h_br_innermost3of4_eta_pb = new TH1F("h_br_innermost3of
4_eta_pb", "", 33, -16.5, 16.5);

    TH2D *h_etafp3_1or2_3or4_pb = new TH2D("h_etafp3_1or2_3or4_pb
", "", 33, -16.5, 16.5, 32, -0.5, 31.5);
    TH2D *h_etafp3_5or6_pb = new TH2D("h_etafp3_5or6_pb", "", 33, -1
6.5, 16.5, 32, -0.5, 31.5);

    TH2F *h_pTfp_br_pb = new TH2F("h_pTfp_br_pb", "", 32, -0.5, 31.5,
 7, -0.5, 6.5);
    TH2F *h_pTfp_ec_pb = new TH2F("h_pTfp_ec_pb", "", 32, -0.5, 31
.5, 7, -0.5, 6.5);
    TH2D *h_etafp_pb = new TH2D("h_etafp_pb", "", 33, -16.5, 16.5, 7,
-0.5, 6.5);
    TH2D *h_etafp_25pT_pb = new TH2D("h_etafp_25pT_pb", "", 33, -1
6.5, 16.5, 7, -0.5, 6.5);
    TH2F *etapT_pb_0 = new TH2F("etapT_pb_0", "", 33, -16.5, 16.5, 32,
-0.5, 31.5);
    TH2F *etapT_pb_1 = new TH2F("etapT_pb_1", "", 33, -16.5, 16.5, 32,
-0.5, 31.5);
    TH2F *etapT_pb_2 = new TH2F("etapT_pb_2", "", 33, -16.5, 16.5, 32,
```

```
-0.5, 31.5);
    TH2F *etapT_pb_3 = new TH2F("etapT_pb_3", "", 33, -16.5, 16.5, 32,
-0.5, 31.5);
    TH2F *etapT_pb_4 = new TH2F("etapT_pb_4", "", 33, -16.5, 16.5, 32,
-0.5, 31.5);
    TH2F *etapT_pb_5 = new TH2F("etapT_pb_5", "", 33, -16.5, 16.5, 32,
-0.5, 31.5);

    TH2F *etaphi_pb_0 = new TH2F("etaphi_pb_0", "", 33, -16.5, 16.5, 144,
-0.5, 143.5);
    TH2F *etaphi_pb_1 = new TH2F("etaphi_pb_1", "", 33, -16.5, 16.5, 144,
-0.5, 143.5);
    TH2F *etaphi_pb_2 = new TH2F("etaphi_pb_2", "", 33, -16.5, 16.5, 144,
-0.5, 143.5);
    TH2F *etaphi_pb_3 = new TH2F("etaphi_pb_3", "", 33, -16.5, 16.5, 144,
-0.5, 143.5);
    TH2F *etaphi_pb_4 = new TH2F("etaphi_pb_4", "", 33, -16.5, 16.5, 144,
-0.5, 143.5);
    TH2F *etaphi_pb_5 = new TH2F("etaphi_pb_5", "", 33, -16.5, 16.5, 144,
-0.5, 143.5);

    TCanvas *c1 = new TCanvas("c1");

    etapT_pb_0 -> Divide(h_etapT_0,h_etapT,1,1);
    etapT_pb_1 -> Divide(h_etapT_1,h_etapT,1,1);
    etapT_pb_2 -> Divide(h_etapT_2,h_etapT,1,1);
    etapT_pb_3 -> Divide(h_etapT_3,h_etapT,1,1);
    etapT_pb_4 -> Divide(h_etapT_4,h_etapT,1,1);
    etapT_pb_5 -> Divide(h_etapT_5,h_etapT,1,1);

    etaphi_pb_0 -> Divide(h_etaphi_0,h_etaphi,1,1);
    etaphi_pb_1 -> Divide(h_etaphi_1,h_etaphi,1,1);
    etaphi_pb_2 -> Divide(h_etaphi_2,h_etaphi,1,1);
    etaphi_pb_3 -> Divide(h_etaphi_3,h_etaphi,1,1);
    etaphi_pb_4 -> Divide(h_etaphi_4,h_etaphi,1,1);
    etaphi_pb_5 -> Divide(h_etaphi_5,h_etaphi,1,1);

    gStyle->SetTitleXOffset(1.05);
    gStyle->SetTitleYOffset(1.05);
    gStyle->SetPadTopMargin(0.15);
    gStyle->SetPadBottomMargin(0.15);
```

```
gStyle->SetPadLeftMargin(0.15);
gStyle->SetPalette(1);

h_selMu_pT->GetYaxis()->SetTitle("# of #mu");
h_selMu_pT -> Draw();
h_selMu_pT->GetXaxis()->SetTitleOffset(1.2);
h_selMu_pT->GetYaxis()->SetTitleOffset(1.4);
c1 -> SaveAs("h_selMu_pT.png");

h_selMu_firedPlanes->GetXaxis()->SetTitle("# of firedplanes");
h_selMu_firedPlanes->GetYaxis()->SetTitle("# of #mu");
h_selMu_firedPlanes -> Draw();
h_selMu_firedPlanes->GetXaxis()->SetTitleOffset(1.2);
h_selMu_firedPlanes->GetYaxis()->SetTitleOffset(1.4);
c1 -> SaveAs("h_selMu_firedPlanes.png");

h_deleta->GetXaxis()->SetTitle("#Delta#eta");
h_deleta->GetYaxis()->SetTitle("# of #mu");
h_deleta -> Draw();
h_deleta->GetXaxis()->SetTitleOffset(1.2);
h_deleta->GetYaxis()->SetTitleOffset(1.4);
c1 -> SaveAs("h_deleta.png");

h_delphi->GetXaxis()->SetTitle("#Delta#phi");
h_delphi->GetYaxis()->SetTitle("# of #mu");
h_delphi -> Draw();
h_delphi->GetXaxis()->SetTitleOffset(1.2);
h_delphi->GetYaxis()->SetTitleOffset(1.4);
c1 -> SaveAs("h_delphi.png");

h_deletapT->GetXaxis()->SetTitle("#Delta#eta");
h_deletapT->GetYaxis()->SetTitle("pT(pTCode)");
h_deletapT -> Draw("LEGO");
h_deletapT->GetXaxis()->SetTitleOffset(1.4);
h_deletapT->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_deletapT.png");

h_delphipT->GetXaxis()->SetTitle("#Delta#phi");
h_delphipT->GetYaxis()->SetTitle("pT(pTCode)");
h_delphipT -> Draw("LEGO");
h_delphipT->GetXaxis()->SetTitleOffset(1.4);
```

```
    h_delphipT->GetYaxis()->SetTitleOffset(1.5);
    c1 -> SaveAs("h_delphipT.png");

    h_simTrack_fp->GetXaxis()->SetTitle("# of firedplanes");
    h_simTrack_fp->GetYaxis()->SetTitle("# of #mu");
    h_simTrack_fp -> Draw();
    h_simTrack_fp->GetXaxis()->SetTitleOffset(1.2);
    h_simTrack_fp->GetYaxis()->SetTitleOffset(1.4);
    c1 -> SaveAs("h_simTrack_fp.png");

    h_simTrack_pT->GetXaxis()->SetTitle("pT(pTCode)");
    h_simTrack_pT->GetYaxis()->SetTitle("# of #mu");
    h_simTrack_pT -> Draw();
    h_simTrack_pT->GetXaxis()->SetTitleOffset(1.2);
    h_simTrack_pT->GetYaxis()->SetTitleOffset(1.4);
    c1 -> SaveAs("h_simTrack_pT.png");

    h_simTrack_eta->GetXaxis()->SetTitle("#eta address(tower segm
ent)");
    h_simTrack_eta->GetYaxis()->SetTitle("# of #mu");
    h_simTrack_eta -> Draw();
    h_simTrack_eta->GetXaxis()->SetTitleOffset(1.2);
    h_simTrack_eta->GetYaxis()->SetTitleOffset(1.4);
    c1 -> SaveAs("h_simTrack_eta.png");

    h_simTrack_fp_1->GetXaxis()->SetTitle("# of firedplanes");
    h_simTrack_fp_1->GetYaxis()->SetTitle("# of #mu");
    h_simTrack_fp_1 -> Draw();
    h_simTrack_fp_1->GetXaxis()->SetTitleOffset(1.2);
    h_simTrack_fp_1->GetYaxis()->SetTitleOffset(1.4);
    c1 -> SaveAs("h_simTrack_fp_1.png");

    h_simTrack_pT_1->GetXaxis()->SetTitle("pT(pTCode)");
    h_simTrack_pT_1->GetYaxis()->SetTitle("# of #mu");
    h_simTrack_pT_1 -> Draw();
    h_simTrack_pT_1->GetXaxis()->SetTitleOffset(1.2);
    h_simTrack_pT_1->GetYaxis()->SetTitleOffset(1.4);
    c1 -> SaveAs("h_simTrack_pT_1.png");

    h_simTrack_eta_1->GetXaxis()->SetTitle("#eta address(tower seg
ment)");
```

```
    h_simTrack_eta_1->GetYaxis()->SetTitle("# of #mu");
    h_simTrack_eta_1 -> Draw();
    h_simTrack_eta_1->GetXaxis()->SetTitleOffset(1.2);
    h_simTrack_eta_1->GetYaxis()->SetTitleOffset(1.4);
    c1 -> SaveAs("h_simTrack_eta_1.png");

    h_simTrack_etapT_1->GetXaxis()->SetTitle("#eta address(tower s
egment)");
    h_simTrack_etapT_1->GetYaxis()->SetTitle("pT(pTCode)");
    h_simTrack_etapT_1 -> Draw("LEGO");
    h_simTrack_etapT_1->GetXaxis()->SetTitleOffset(1.4);
    h_simTrack_etapT_1->GetYaxis()->SetTitleOffset(1.5);
    c1 -> SaveAs("h_simTrack_etapT_1.png");

    h_simTrack_etaphi_1->GetXaxis()->SetTitle("#eta address(tower s
egment)");
    h_simTrack_etaphi_1->GetYaxis()->SetTitle("#phi address(logic s
egment)");
    h_simTrack_etaphi_1 -> Draw("LEGO");
    h_simTrack_etaphi_1->GetXaxis()->SetTitleOffset(1.4);
    h_simTrack_etaphi_1->GetYaxis()->SetTitleOffset(1.5);
    c1 -> SaveAs("h_simTrack_etaphi_1.png");

    h_etafp->GetXaxis()->SetTitle("#eta address(tower segment)");
    h_etafp->GetYaxis()->SetTitle("# of firedplanes");
    h_etafp -> Draw("LEGO");
    h_etafp->GetXaxis()->SetTitleOffset(1.4);
    h_etafp->GetYaxis()->SetTitleOffset(1.5);
    c1 -> SaveAs("h_etafp.png");

    h_etafp_25pT->GetXaxis()->SetTitle("#eta address(tower segment)
");
    h_etafp_25pT->GetYaxis()->SetTitle("# of firedplanes");
    h_etafp_25pT -> Draw("LEGO");
    h_etafp_25pT->GetXaxis()->SetTitleOffset(1.4);
    h_etafp_25pT->GetYaxis()->SetTitleOffset(1.5);
    c1 -> SaveAs("h_etafp_25pT.png");

    h_pTfp_br->GetXaxis()->SetTitle("pT(pTCode)");
    h_pTfp_br->GetYaxis()->SetTitle("# of firedplanes");
    h_pTfp_br -> Draw("LEGO");
```

```
h_pTfp_br->GetXaxis()->SetTitleOffset(1.4);
h_pTfp_br->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_pTfp_br.png");

h_pTfp_ec->GetXaxis()->SetTitle("pT(pTCode)");
h_pTfp_ec->GetYaxis()->SetTitle("# of firedplanes");
h_pTfp_ec -> Draw("LEGO");
h_pTfp_ec->GetXaxis()->SetTitleOffset(1.4);
h_pTfp_ec->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_pTfp_ec.png");

h_etapTfp_0->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapTfp_0->GetYaxis()->SetTitle("pT(pTCode)");
h_etapTfp_0 -> Draw("LEGO");
h_etapTfp_0->GetXaxis()->SetTitleOffset(1.4);
h_etapTfp_0->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapTfp_0.png");

h_etapTfp_1->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapTfp_1->GetYaxis()->SetTitle("pT(pTCode)");
h_etapTfp_1 -> Draw("LEGO");
h_etapTfp_1->GetXaxis()->SetTitleOffset(1.4);
h_etapTfp_1->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapTfp_1.png");

h_etapTfp_2->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapTfp_2->GetYaxis()->SetTitle("pT(pTCode)");
h_etapTfp_2 -> Draw("LEGO");
h_etapTfp_2->GetXaxis()->SetTitleOffset(1.4);
h_etapTfp_2->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapTfp_2.png");

h_etapTfp_3->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapTfp_3->GetYaxis()->SetTitle("pT(pTCode)");
h_etapTfp_3 -> Draw("LEGO");
h_etapTfp_3->GetXaxis()->SetTitleOffset(1.4);
h_etapTfp_3->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapTfp_3.png");

h_etapTfp_4->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapTfp_4->GetYaxis()->SetTitle("pT(pTCode)");
```

```
h_etapTfp_4 -> Draw("LEGO");
h_etapTfp_4->GetXaxis()->SetTitleOffset(1.4);
h_etapTfp_4->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapTfp_4.png");

h_etapTfp_5->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapTfp_5->GetYaxis()->SetTitle("pT(pTCode)");
h_etapTfp_5 -> Draw("LEGO");
h_etapTfp_5->GetXaxis()->SetTitleOffset(1.4);
h_etapTfp_5->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapTfp_5.png");

h_etapTfp_6->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapTfp_6->GetYaxis()->SetTitle("pT(pTCode)");
h_etapTfp_6 -> Draw("LEGO");
h_etapTfp_6->GetXaxis()->SetTitleOffset(1.4);
h_etapTfp_6->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapTfp_6.png");

h_etapT_3_01->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapT_3_01->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT_3_01 -> Draw("LEGO");
h_etapT_3_01->GetXaxis()->SetTitleOffset(1.4);
h_etapT_3_01->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT_3_01.png");

h_etapT->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapT->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT -> Draw("LEGO");
h_etapT->GetXaxis()->SetTitleOffset(1.4);
h_etapT->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT.png");

h_etapT_0->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapT_0->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT_0 -> Draw("LEGO");
h_etapT_0->GetXaxis()->SetTitleOffset(1.4);
h_etapT_0->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT_0.png");

h_etapT_1->GetXaxis()->SetTitle("#eta address(tower segment)");
```

```
h_etapT_1->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT_1 -> Draw("LEGO");
h_etapT_1->GetXaxis()->SetTitleOffset(1.4);
h_etapT_1->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT_1.png");

h_etapT_2->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapT_2->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT_2 -> Draw("LEGO");
h_etapT_2->GetXaxis()->SetTitleOffset(1.4);
h_etapT_2->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT_2.png");

h_etapT_3->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapT_3->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT_3 -> Draw("LEGO");
h_etapT_3->GetXaxis()->SetTitleOffset(1.4);
h_etapT_3->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT_3.png");

h_etapT_4->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapT_4->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT_4 -> Draw("LEGO");
h_etapT_4->GetXaxis()->SetTitleOffset(1.4);
h_etapT_4->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT_4.png");

h_etapT_5->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etapT_5->GetYaxis()->SetTitle("pT(pTCode)");
h_etapT_5 -> Draw("LEGO");
h_etapT_5->GetXaxis()->SetTitleOffset(1.4);
h_etapT_5->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etapT_5.png");

h_etaphi->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etaphi->GetYaxis()->SetTitle("#phi address(logic segment)");
h_etaphi -> Draw("LEGO");
h_etaphi->GetXaxis()->SetTitleOffset(1.4);
h_etaphi->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etaphi.png");
```

```
h_etaphi_0->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etaphi_0->GetYaxis()->SetTitle("#phi address(logic segment)");
h_etaphi_0 -> Draw("LEGO");
h_etaphi_0->GetXaxis()->SetTitleOffset(1.4);
h_etaphi_0->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etaphi_0.png");

h_etaphi_1->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etaphi_1->GetYaxis()->SetTitle("#phi address(logic segment)");
h_etaphi_1 -> Draw("LEGO");
h_etaphi_1->GetXaxis()->SetTitleOffset(1.4);
h_etaphi_1->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etaphi_1.png");

h_etaphi_2->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etaphi_2->GetYaxis()->SetTitle("#phi address(logic segment)");
h_etaphi_2 -> Draw("LEGO");
h_etaphi_2->GetXaxis()->SetTitleOffset(1.4);
h_etaphi_2->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etaphi_2.png");

h_etaphi_3->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etaphi_3->GetYaxis()->SetTitle("#phi address(logic segment)");
h_etaphi_3 -> Draw("LEGO");
h_etaphi_3->GetXaxis()->SetTitleOffset(1.4);
h_etaphi_3->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etaphi_3.png");

h_etaphi_4->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etaphi_4->GetYaxis()->SetTitle("#phi address(logic segment)");
h_etaphi_4 -> Draw("LEGO");
h_etaphi_4->GetXaxis()->SetTitleOffset(1.4);
h_etaphi_4->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etaphi_4.png");

h_etaphi_5->GetXaxis()->SetTitle("#eta address(tower segment)");
h_etaphi_5->GetYaxis()->SetTitle("#phi address(logic segment)");
h_etaphi_5 -> Draw("LEGO");
h_etaphi_5->GetXaxis()->SetTitleOffset(1.4);
h_etaphi_5->GetYaxis()->SetTitleOffset(1.5);
c1 -> SaveAs("h_etaphi_5.png");
```

```
    gStyle->SetOptStat(0);

    double total_fp;
    total_fp = h_simTrack_fp->Integral();

    h_simTrack_fp_pb = (TH1F*)h_simTrack_fp->DrawCopy();
    h_simTrack_fp_pb -> Scale(1/total_fp);

    h_simTrack_fp_pb->SetTitle("probability plot for (# of firedplanes)");
    h_simTrack_fp_pb->GetXaxis()->SetTitle("# of firedplanes");
    h_simTrack_fp_pb->GetYaxis()->SetTitle("probability");
    h_simTrack_fp_pb -> Draw();
    h_simTrack_fp_pb->GetXaxis()->SetTitleOffset(1.2);
    h_simTrack_fp_pb->GetYaxis()->SetTitleOffset(1.4);
    c1 -> SaveAs("h_simTrack_fp_pb.png");

    double im34_a;
    double im34_b;
    double im34_c;

    for(int i=10; i<25; i++) {
        im34_a = h_br_innermost3of4_eta->GetBinContent(i);
        im34_b = h_simTrack_eta->GetBinContent(i);
        if (im34_b == 0) {
            im34_c = 0;
        }
        else {
        im34_c = im34_a/im34_b;
        }
        h_br_innermost3of4_eta_pb->SetBinContent(i,im34_c);
    }

    h_br_innermost3of4_eta_pb->SetTitle("probability plot for the muons
that 3 out of 4 innermost layers fired in barrel region");
    h_br_innermost3of4_eta_pb->GetXaxis()->SetTitle("#eta address
(tower segment)");
    h_br_innermost3of4_eta_pb->GetYaxis()->SetTitle("probability");
    h_br_innermost3of4_eta_pb -> Draw();
    h_br_innermost3of4_eta_pb->GetXaxis()->SetTitleOffset(1.2);
    h_br_innermost3of4_eta_pb->GetYaxis()->SetTitleOffset(1.4);
```

```
    c1 -> SaveAs("h_br_innermost3of4_eta_pb.png");

    double fp312_a;
    double fp312_b;
    double fp312_c;

    for(int i=10; i<25; i++) {
        for(int k=1;k<33;k++){
            fp312_a = h_etapTfp_3->GetBinContent(i,k);
            std::cout<<"fp312_a : "<<fp312_a<<std::endl;
            fp312_b = h_etapTfp3_1or2_3or4->GetBinContent(i,k);
            std::cout<<"fp312_b : "<<fp312_b<<std::endl;
            if(fp312_a == 0) {
                fp312_c = 0;
            }
            else {
            fp312_c = fp312_b/fp312_a;
            std::cout<<"fp312_c : "<<fp312_c<<std::endl;
            }
            h_etafp3_1or2_3or4_pb->SetBinContent(i,k,fp312_c);
        }
    }

    h_etafp3_1or2_3or4_pb->SetTitle("probability plot for the muons that
(1st or 2nd) && (3rd or 4th) layer fired in barrel region ; (# of fp) = 3");
    h_etafp3_1or2_3or4_pb->GetXaxis()->SetTitle("#eta address(tower
segment)");
    h_etafp3_1or2_3or4_pb->GetYaxis()->SetTitle("pTCode");
    h_etafp3_1or2_3or4_pb -> Draw("colz");
    c1 -> SaveAs("h_etafp3_1or2_3or4_pb.png");

    double fp356_a;
    double fp356_b;
    double fp356_c;

    for(int i=10; i<25; i++) {
        for(int k=1;k<33;k++){
            fp356_a = h_etapTfp_3->GetBinContent(i,k);
            fp356_b = h_etapTfp3_5or6->GetBinContent(i,k);
            if(fp356_a == 0) {
                fp356_c = 0;
```

```
            }
            else {
                fp356_c = fp356_b/fp356_a;
            }
            h_etafp3_5or6_pb->SetBinContent(i,k,fp356_c);
        }
    }

    h_etafp3_5or6_pb->SetTitle("probability plot for the muons that 5th or
 6th layer fired in barrel region ; (# of fp) = 3");
    h_etafp3_5or6_pb->GetXaxis()->SetTitle("#eta address(tower segment)");
    h_etafp3_5or6_pb->GetYaxis()->SetTitle("pTCode");
    h_etafp3_5or6_pb -> Draw("colz");
    c1 -> SaveAs("h_etafp3_5or6_pb.png");

    double eta_4br_a;
    double eta_4br_b;
    double eta_4br_c;
    double eta_4br_tot1 = 0.0;
    double eta_4br_tot2 = 0.0;

    for(int i=10; i<25; i++) {
        eta_4br_tot1 = 0;
        eta_4br_tot2 = 0;
        for(int k=1; k<8;k++){
        eta_4br_a = h_etafp_25pT->GetBinContent(i,k);
        eta_4br_tot1 = eta_4br_tot1 + eta_4br_a;
        }
        for(int j=5;j<8;j++) {
            eta_4br_b = h_etafp_25pT->GetBinContent(i,j);
            eta_4br_tot2 = eta_4br_tot2 + eta_4br_b;
        }
        if(eta_4br_tot1 == 0) {
            eta_4br_c = 0;
        }
        else {
            eta_4br_c = eta_4br_tot2/eta_4br_tot1;
        }
        h_eta_4br_pb->SetBinContent(i,eta_4br_c);
    }
```

```
    h_eta_4br_pb->SetTitle("probability plot that firing at least 4 layers
in the barrel region(pTCode>=25)");
    h_eta_4br_pb->GetXaxis()->SetTitle("#eta address(tower segment)");
    h_eta_4br_pb->GetYaxis()->SetTitle("probability");
    h_eta_4br_pb -> Draw();
    c1 -> SaveAs("h_eta_4br_pb.png");

    double eta_3ec_a;
    double eta_3ec_b;
    double eta_3ec_c;
    double eta_3ec_tot1 = 0.0;
    double eta_3ec_tot2 = 0.0;

    for(int i=1; i<10; i++) {
        eta_3ec_tot1 = 0;
        eta_3ec_tot2 = 0;
        for(int k=1; k<8;k++){
        eta_3ec_a = h_etafp_25pT->GetBinContent(i,k);
        eta_3ec_tot1 = eta_3ec_tot1 + eta_3ec_a;
        }
        for(int j=4;j<8;j++) {
            eta_3ec_b = h_etafp_25pT->GetBinContent(i,j);
            eta_3ec_tot2 = eta_3ec_tot2 + eta_3ec_b;
        }
        if(eta_3ec_tot1 == 0) {
            eta_3ec_c = 0;
        }
        else {
            eta_3ec_c = eta_3ec_tot2/eta_3ec_tot1;
        }
        h_eta_3ec_pb->SetBinContent(i,eta_3ec_c);
    }

    double eta_3ec_a;
    double eta_3ec_b;
    double eta_3ec_c;
    double eta_3ec_tot1 = 0.0;
    double eta_3ec_tot2 = 0.0;

    for(int i=25; i<34; i++) {
        eta_3ec_tot1 = 0;
```

```
        eta_3ec_tot2 = 0;
        for(int k=1; k<8;k++){
        eta_3ec_a = h_etafp_25pT->GetBinContent(i,k);
        eta_3ec_tot1 = eta_3ec_tot1 + eta_3ec_a;
        }
        for(int j=4;j<8;j++) {
            eta_3ec_b = h_etafp_25pT->GetBinContent(i,j);
            eta_3ec_tot2 = eta_3ec_tot2 + eta_3ec_b;
        }
        if(eta_3ec_tot1 == 0) {
            eta_3ec_c = 0;
        }
        else {
            eta_3ec_c = eta_3ec_tot2/eta_3ec_tot1;
        }
        h_eta_3ec_pb->SetBinContent(i,eta_3ec_c);
    }

    h_eta_3ec_pb->SetTitle("probability plot that firing at least 3 layers
in the endcap region(pTCode>=25)");
    h_eta_3ec_pb->GetXaxis()->SetTitle("#eta address(tower segment)");
    h_eta_3ec_pb->GetYaxis()->SetTitle("probability");
    h_eta_3ec_pb -> Draw();
    c1 -> SaveAs("h_eta_3ec_pb.png");

    double etafp_a;
    double etafp_b;
    double etafp_c;
    double etafp_tot = 0.0;

    for(int i=1; i<34; i++) {
        etafp_tot = 0;
        for(int k=1; k<8;k++){
        etafp_a = h_etafp->GetBinContent(i,k);
        etafp_tot = etafp_tot + etafp_a;
        }
        for(int j=4; j<8; j++) {
            etafp_b = h_etafp->GetBinContent(i,j);
            if(etafp_tot == 0) {
                etafp_c = 0;
            }
```

```
            else {
                etafp_c = etafp_b/etafp_tot;
            }
            h_etafp_pb->SetBinContent(i,j,etafp_c);
        }
    }

    h_etafp_pb->SetTitle("probability plot for #eta-fp");
    h_etafp_pb->GetXaxis()->SetTitle("#eta address(tower segment)");
    h_etafp_pb->GetYaxis()->SetTitle("# of firedplanes");
    h_etafp_pb -> Draw("colz");
    c1 -> SaveAs("h_etafp_pb.png");

    double etafp_25pT_a;
    double etafp_25pT_b;
    double etafp_25pT_c;
    double etafp_25pT_tot = 0.0;

    for(int i=1; i<34; i++) {
        etafp_25pT_tot = 0;
        for(int k=1; k<8;k++){
        etafp_25pT_a = h_etafp_25pT->GetBinContent(i,k);
        etafp_25pT_tot = etafp_25pT_tot + etafp_25pT_a;
        }
        for(int j=4; j<8; j++) {
            etafp_25pT_b = h_etafp_25pT->GetBinContent(i,j);
            if(etafp_25pT_tot == 0) {
                etafp_25pT_c = 0;
            }
            else {
                etafp_25pT_c = etafp_25pT_b/etafp_25pT_tot;
            }
            h_etafp_25pT_pb->SetBinContent(i,j,etafp_25pT_c);
        }
    }

    h_etafp_25pT_pb->SetTitle("probability plot for #eta-fp (pTCode>=25)");
    h_etafp_25pT_pb->GetXaxis()->SetTitle("#eta address(tower segment)");
    h_etafp_25pT_pb->GetYaxis()->SetTitle("# of firedplanes");
    h_etafp_25pT_pb -> Draw("colz");
    c1 -> SaveAs("h_etafp_25pT_pb.png");
```

```
double pTfp_br_a;
double pTfp_br_b;
double pTfp_br_c;
double pTfp_br_tot = 0.0;

for(int i=1; i<33; i++) {
    pTfp_br_tot = 0;
    for(int k=1; k<8;k++){
    pTfp_br_a = h_pTfp_br->GetBinContent(i,k);
    pTfp_br_tot = pTfp_br_tot + pTfp_br_a;
    }
    for(int j=4; j<8; j++) {
        pTfp_br_b = h_pTfp_br->GetBinContent(i,j);
        if(pTfp_br_tot == 0) {
            pTfp_br_c = 0;
        }
        else {
            pTfp_br_c = pTfp_br_b/pTfp_br_tot;
        }
        h_pTfp_br_pb->SetBinContent(i,j,pTfp_br_c);
    }
}

h_pTfp_br_pb->SetTitle("probability plot for pT-fp in barrel");
h_pTfp_br_pb->GetXaxis()->SetTitle("pT(pTCode)");
h_pTfp_br_pb->GetYaxis()->SetTitle("# of firedplanes");
h_pTfp_br_pb -> Draw("colz");
c1 -> SaveAs("h_pTfp_br_pb.png");

double pTfp_ec_a;
double pTfp_ec_b;
double pTfp_ec_c;
double pTfp_ec_tot = 0.0;

for(int i=1; i<33; i++) {
    pTfp_ec_tot = 0;
    for(int k=1; k<8;k++){
    pTfp_ec_a = h_pTfp_ec->GetBinContent(i,k);
    pTfp_ec_tot = pTfp_ec_tot + pTfp_ec_a;
    }
```

```
        for(int j=4; j<8; j++) {
            pTfp_ec_b = h_pTfp_ec->GetBinContent(i,j);
            if(pTfp_ec_tot == 0) {
                pTfp_ec_c = 0;
            }
            else {
                pTfp_ec_c = pTfp_ec_b/pTfp_ec_tot;
            }
            h_pTfp_ec_pb->SetBinContent(i,j,pTfp_ec_c);
        }
    }

    h_pTfp_ec_pb->SetTitle("probability plot for pTfp in endcap");
    h_pTfp_ec_pb->GetXaxis()->SetTitle("pT(pTCode)");
    h_pTfp_ec_pb->GetYaxis()->SetTitle("# of firedplanes");
    h_pTfp_ec_pb -> Draw("colz");
    c1 -> SaveAs("h_pTfp_ec_pb.png");

    etapT_pb_0->SetTitle("1st plane firing probability plot for #eta-pT");
    etapT_pb_0->GetXaxis()->SetTitle("#eta address(tower segment)");
    etapT_pb_0->GetYaxis()->SetTitle("pT(pTCode)");
    etapT_pb_0 -> Draw("colz");
    c1 -> SaveAs("etapT_pb_0.png");

    etapT_pb_1->SetTitle("2nd plane firing probability plot for #eta-pT");
    etapT_pb_1->GetXaxis()->SetTitle("#eta address(tower segment)");
    etapT_pb_1->GetYaxis()->SetTitle("pT(pTCode)");
    etapT_pb_1 -> Draw("colz");
    c1 -> SaveAs("etapT_pb_1.png");

    etapT_pb_2->SetTitle("3rd plane firing probability plot for #eta-pT");
    etapT_pb_2->GetXaxis()->SetTitle("#eta address(tower segment)");
    etapT_pb_2->GetYaxis()->SetTitle("pT(pTCode)");
    etapT_pb_2 -> Draw("colz");
    c1 -> SaveAs("etapT_pb_2.png");

    etapT_pb_3->SetTitle("4th plane firing probability plot for #eta-pT");
    etapT_pb_3->GetXaxis()->SetTitle("#eta address(tower segment)");
    etapT_pb_3->GetYaxis()->SetTitle("pT(pTCode)");
    etapT_pb_3 -> Draw("colz");
    c1 -> SaveAs("etapT_pb_3.png");
```

```
etapT_pb_4->SetTitle("5th plane firing probability plot for #eta-pT");
etapT_pb_4->GetXaxis()->SetTitle("#eta address(tower segment)");
etapT_pb_4->GetYaxis()->SetTitle("pT(pTCode)");
etapT_pb_4 -> Draw("colz");
c1 -> SaveAs("etapT_pb_4.png");

etapT_pb_5->SetTitle("6th plane firing probability plot for #eta-pT");
etapT_pb_5->GetXaxis()->SetTitle("#eta address(tower segment)");
etapT_pb_5->GetYaxis()->SetTitle("pT(pTCode)");
etapT_pb_5 -> Draw("colz");
c1 -> SaveAs("etapT_pb_5.png");

etaphi_pb_0->SetTitle("1st plane firing probability plot for #eta-#phi");
etaphi_pb_0->GetXaxis()->SetTitle("#eta address(tower segment)");
etaphi_pb_0->GetYaxis()->SetTitle("#phi address(logic segment)");
etaphi_pb_0 -> Draw("colz");
c1 -> SaveAs("etaphi_pb_0.png");

etaphi_pb_1->SetTitle("2nd plane firing probability plot for #eta-#phi");
etaphi_pb_1->GetXaxis()->SetTitle("#eta address(tower segment)");
etaphi_pb_1->GetYaxis()->SetTitle("#phi address(logic segment)");
etaphi_pb_1 -> Draw("colz");
c1 -> SaveAs("etaphi_pb_1.png");

etaphi_pb_2->SetTitle("3rd plane firing probability plot for #eta-#phi");
etaphi_pb_2->GetXaxis()->SetTitle("#eta address(tower segment)");
etaphi_pb_2->GetYaxis()->SetTitle("#phi address(logic segment)");
etaphi_pb_2 -> Draw("colz");
c1 -> SaveAs("etaphi_pb_2.png");

etaphi_pb_3->SetTitle("4th plane firing probability plot for #eta-#phi");
etaphi_pb_3->GetXaxis()->SetTitle("#eta address(tower segment)");
etaphi_pb_3->GetYaxis()->SetTitle("#phi address(logic segment)");
etaphi_pb_3 -> Draw("colz");
c1 -> SaveAs("etaphi_pb_3.png");

etaphi_pb_4->SetTitle("5th plane firing probability plot for #eta-#phi");
etaphi_pb_4->GetXaxis()->SetTitle("#eta address(tower segment)");
etaphi_pb_4->GetYaxis()->SetTitle("#phi address(logic segment)");
etaphi_pb_4 -> Draw("colz");
```

```
    c1 -> SaveAs("etaphi_pb_4.png");

    etaphi_pb_5->SetTitle("6th plane firing probability plot for #eta-#phi");
    etaphi_pb_5->GetXaxis()->SetTitle("#eta address(tower segment)");
    etaphi_pb_5->GetYaxis()->SetTitle("#phi address(logic segment)");
    etaphi_pb_5 -> Draw("colz");
    c1 -> SaveAs("etaphi_pb_5.png");

    etapT_pb_0 -> Write("etapT_pb_0");
    etapT_pb_1 -> Write("etapT_pb_1");
    etapT_pb_2 -> Write("etapT_pb_2");
    etapT_pb_3 -> Write("etapT_pb_3");
    etapT_pb_4 -> Write("etapT_pb_4");
    etapT_pb_5 -> Write("etapT_pb_5");

    etaphi_pb_0 -> Write("etaphi_pb_0");
    etaphi_pb_1 -> Write("etaphi_pb_1");
    etaphi_pb_2 -> Write("etaphi_pb_2");
    etaphi_pb_3 -> Write("etaphi_pb_3");
    etaphi_pb_4 -> Write("etaphi_pb_4");
    etaphi_pb_5 -> Write("etaphi_pb_5");
}
```

Bibliography

[1] CMS Collaboration, http://cmsinfo.cern.ch

[2] The Muon Project, Technical Design Report, CERN/LHCC 97-32 (1997)

http://cmsdoc.cern.ch/cms/TDR/MUON/muon.html

[3] The TriDAS Project, The Level-1 Trigger Technical Design Report, CERN/LHCC 2000-38 (2000)

http://cmsdoc.cern.ch/cms/TDR/TRIGGER-public/CMSTrigTDR.pdf

[4] Karol Bumkowski, University of Warsaw, Thesis for the degree of Doctor (2009)

[5] S. Park, Nucl. Instr. and Meth. A 443 (2000)

[6] S. Park, Nucl. Instr. and Meth. A 469 (2001)

[7] S. Park, Nucl. Instr. and Meth. A 550 (2005)

[8] Claudia-Elisabeth Wulz, Nikolsdorfergassse 18, A-1050 Vienna, Austria (2007)

[9] the bitset operator

http://www.sgi.com/tech/stl/bitset.html