ON THE USEFULNESS OF THE MULTI-TRACK VALIDATOR FOR STUDYING SECONDARY VERTEXES IN THE PHASE 1 TRACKER UPGRADE OF THE SUPER LARGE HADRON COLLIDER

By

Cody Craig Johnston

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Physics

May, 2012

Nashville, Tennessee

Approved:

Dr. Paul Sheldon

Dr. Medford Webster

I dedicate this thesis to the lovely and savvy Sarah Elizabeth Avellar,
my chronologically fourth best friend Eugenio Victor Garcia,
and to all the others who helped turn my life as a Vanderbilt graduate student
from a routine experience, into an unforgettable adventure.

ACKNOWLEDGEMENTS

I first and foremost wish to thank my advisor, Dr. Paul Sheldon, for introducing me to the world of high-energy particle physics and for all his advice, guidance, and support in helping me complete this thesis.

Of nearly equal importance, I wish to thank Dr. Eric Brownson, for bringing me on board the research project that is the topic of this thesis. I am grateful for him taking the time and energy over multiple Skype chats and e-mails to guide me on my research and being patient with me while I climbed the substantial learning curve of high-energy physics research. Without Dr. Brownson's input, this thesis would most likely consist of many blank pages.

I would like to thank the Vanderbilt astronomy professors, Dr. Andreas Berlind and Dr. Kelly Holley-Bockelmann, for being excellent advisors during the summer of 2008 when I participated in the Vanderbilt physics REU program. I'd also like to thank Alyce Dobyns for being such a wonderful REU “mom” during that summer. Y'all were the ones who made me want to return to Vanderbilt for grad school. Also, I'd especially like to thank Kelly for her support and assistance in helping me get accepted into the Vanderbilt physics graduate program.

And last but not least, I'd like to thank all my friends and family for a lifetime of support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Particle physics is the study of the elementary constituents of matter and their interactions. Elementary particles are those particles which are not compositions of other particles. It has long been known that current scientific understanding of these particles is summarized in a physics model known as the Standard Model. The Standard Model is composed of three interactions; electromagnetic, weak, and strong; and twelve elementary particles called fermions. Fermions are spin half particles which satisfy Fermi-Dirac statistics, and they are divided into two sub-groups, quarks and leptons. Quarks are further divided into six types: up, down, charm, strange, top, and bottom. Leptons are also further divided into six types: electrons, electron neutrinos, muons, muon neutrinos, taus, and tau neutrinos. (1)

Despite this body of knowledge, there are still open questions in particle physics that go beyond the Standard Model. One avenue of answering these questions is the Large Hadron Collider (LHC). The LHC, constructed in Geneva, Switzerland, by the European Organization for Nuclear Research (CERN), is a proton-proton and heavy ion collider with a goal of answering a plethora of unanswered questions in high-energy particle physics, such as searching for the Higgs Boson, attempting to understand the nature of space and time at elementary levels where current theories break down, investigating the nature of dark matter, and answering questions about the existence of extra dimensions.

The LHC is opening a new energy frontier to recreate conditions present in the early Universe and understand the physics processes governing fundamental particles. However, the discovery potential of the machine is also determined by the rate at which interactions occur. Naturally after an experimental apparatus has been running for a given length of time, it comes time to upgrade the equipment. The upcoming upgrade is the Super Large Hadron Collider (SLHC). The maximum expected luminosity of the LHC is 10^{34} cm^{-2} s^{-1}, but the SLHC has set a goal to increase the luminosity by an order of magnitude. (2)

CHAPTER 2

BACKGROUND

2.1 – Introduction to CMS

The LHC has a total of six experiments studying a diverse range of topics in high-energy physics, but the only one this thesis will focus on is the Compact Muon Solenoid (CMS), shown in Figure 1. The detector consists of several sub-detectors enclosed in different layers: Silicon Pixels and Silicon Strips for tracking reconstruction, Electromagnetic and Hadronic Calorimeters for energy measurements and three different kinds of muon detectors. The tracker system, the electromagnetic calorimeter and a section of the hadronic calorimeter are enclosed in a superconducting solenoid magnet. In Figure 2, a slice of the CMS detector cross section is shown, to illustrate that different types of particles will leave different signatures as they travel through the detector. CMS is a general purpose detector, designed to study multiple aspects of proton collisions at high energies and answer those unanswered questions in particle physics. (1)

At the center of the detector is the interaction point, where the proton-proton collisions occur between two beams from the LHC. Immediately surrounding the interaction point is the tracker, which attempts to match tracks of particles from the vertex from which they originally emanated. Next is the electromagnetic calorimeter, which measures the energies of electrons and photons. Then there is the hadronic calorimeter, which measures the energies of hadrons. Next is the magnet, which determines the charge/mass ratio of particles based on the curvature produced in their track by a magnetic field. Finally, there is the muon detectors and return yoke, to identify muons and measure their momenta. (5)

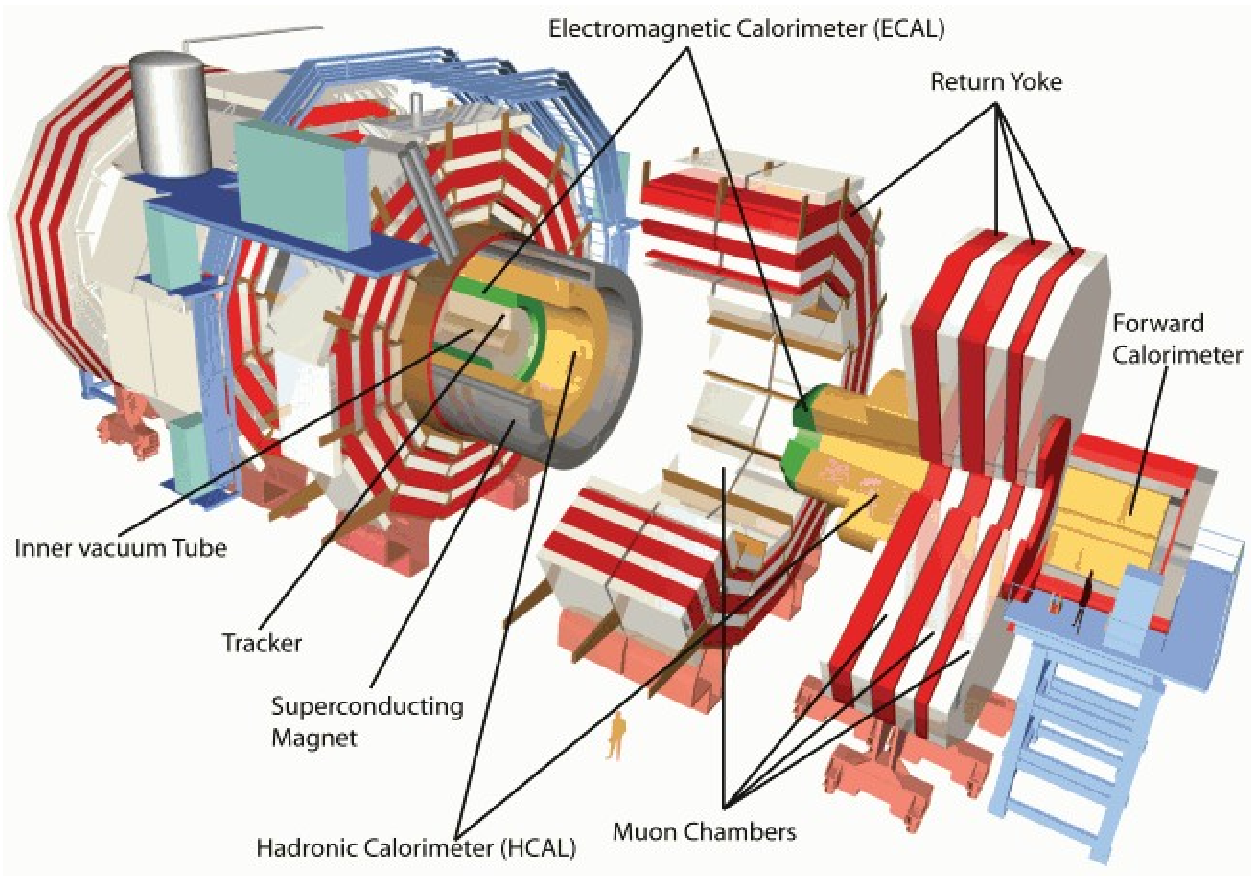
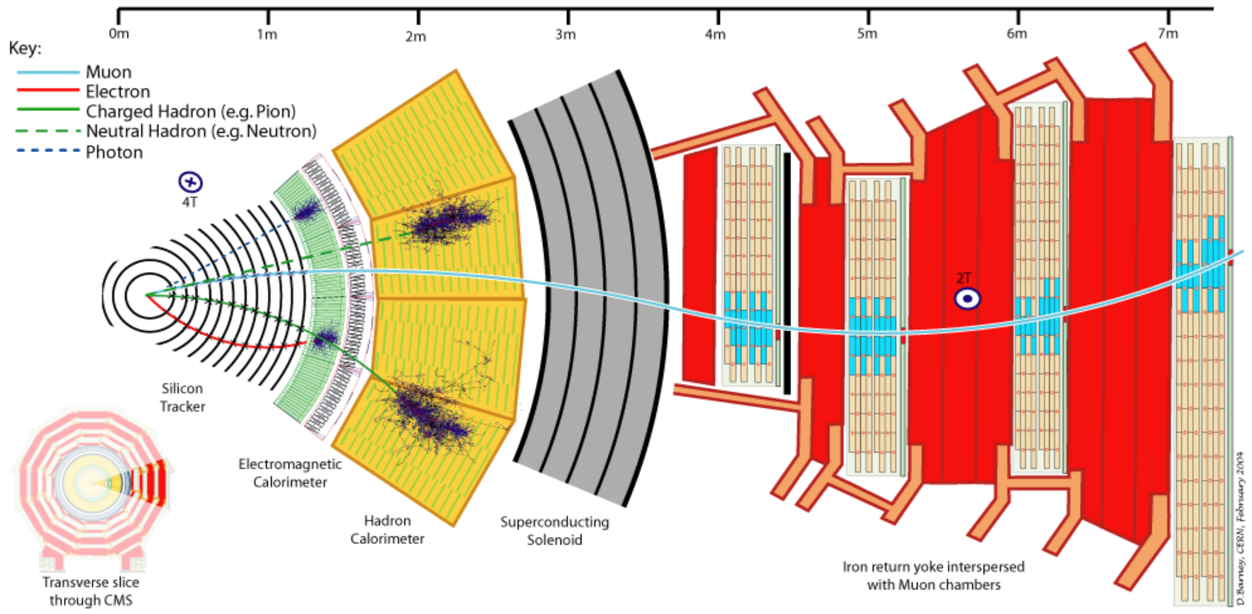


Figure 1: Diagram of the CMS Detector


0m
1m
2m
3m
4m
5m
6m
7m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon
4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
Transverse slice through CMS
D.Barney, CERN, February 2004
Figure 2: Cross-sectional slice of the CMS detector

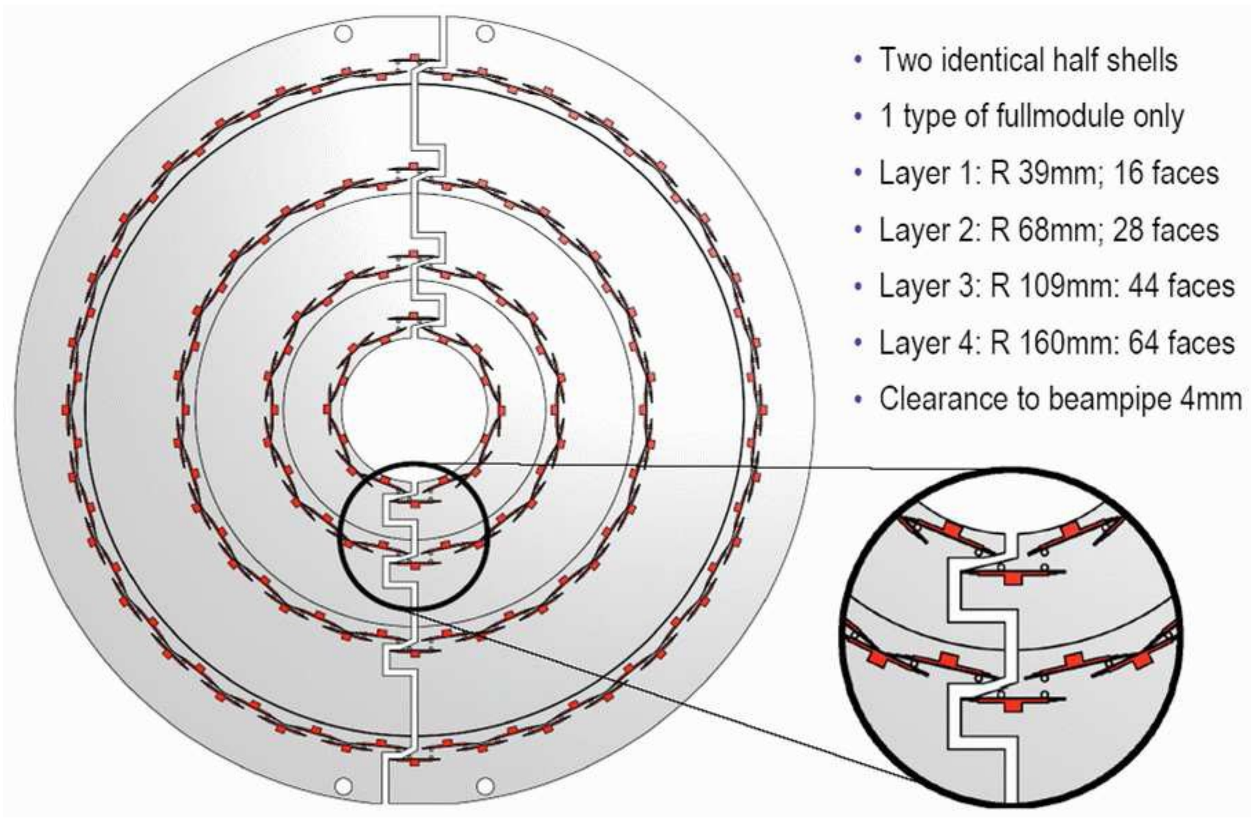
2.2 – Introduction to Phase 1

CERN is carrying out a phased approach for the SLHC upgrade. Phase 1, to be completed in 2016, would double the current maximum luminosity, and this would be pushed gradually and with minimal interruptions in operations to quadruple the current maximum luminosity. Finally, Phase 2 would eventually complete the order of magnitude increase in luminosity. The CMS detector must be upgraded in order to handle the increased luminosity. My research project focused on upgrades to the tracker, and this will be the primary focus of this thesis.

The Phase 1 upgrade will, in addition to higher luminosities, provide a smaller beam pipe at the CMS interaction point. To take advantage of this, the innermost barrel pixel layer can be moved closer to the interaction point. The addition of a fourth barrel layer and a third forward pixel disk provide an added layer of coverage throughout the central tracking region, as shown in Figure 3.

Ideally the detectors would provide at least four hits per track. That fourth hit provides a level of redundancy within the pixel detector when track seeds are constructed. The current highest purity track finding requires three hits per seed. Thus any inefficiency within the three barrel layers and/or two forward disks greatly reduces the number of tracks reconstructed at that rank. Having the possibility of four hits per track also provides an opportunity to construct track seeds requiring four hits per seed. This can greatly reduce the rate at which fake tracks are reconstructed.

The Phase 1 forward pixel disks are constructed in an inner and an outer disk. The inner radius of the forward disks is also decreased to provide better coverage. Figure 4 shows the configuration of the current forward pixel sensors, which can be compared to the Phase 1 forward pixel sensor configuration in Figure 5. (2)



• Two identical half shells
• 1 type of fullmodule only
• Layer 1: R 39mm; 16 faces
• Layer 2: R 68mm; 28 faces
• Layer 3: R 109mm: 44 faces
• Layer 4: R 160mm: 64 faces
• Clearance to beampipe 4mm

Figure 3: The four layers of the Phase 1 Barrel Pixel Detector

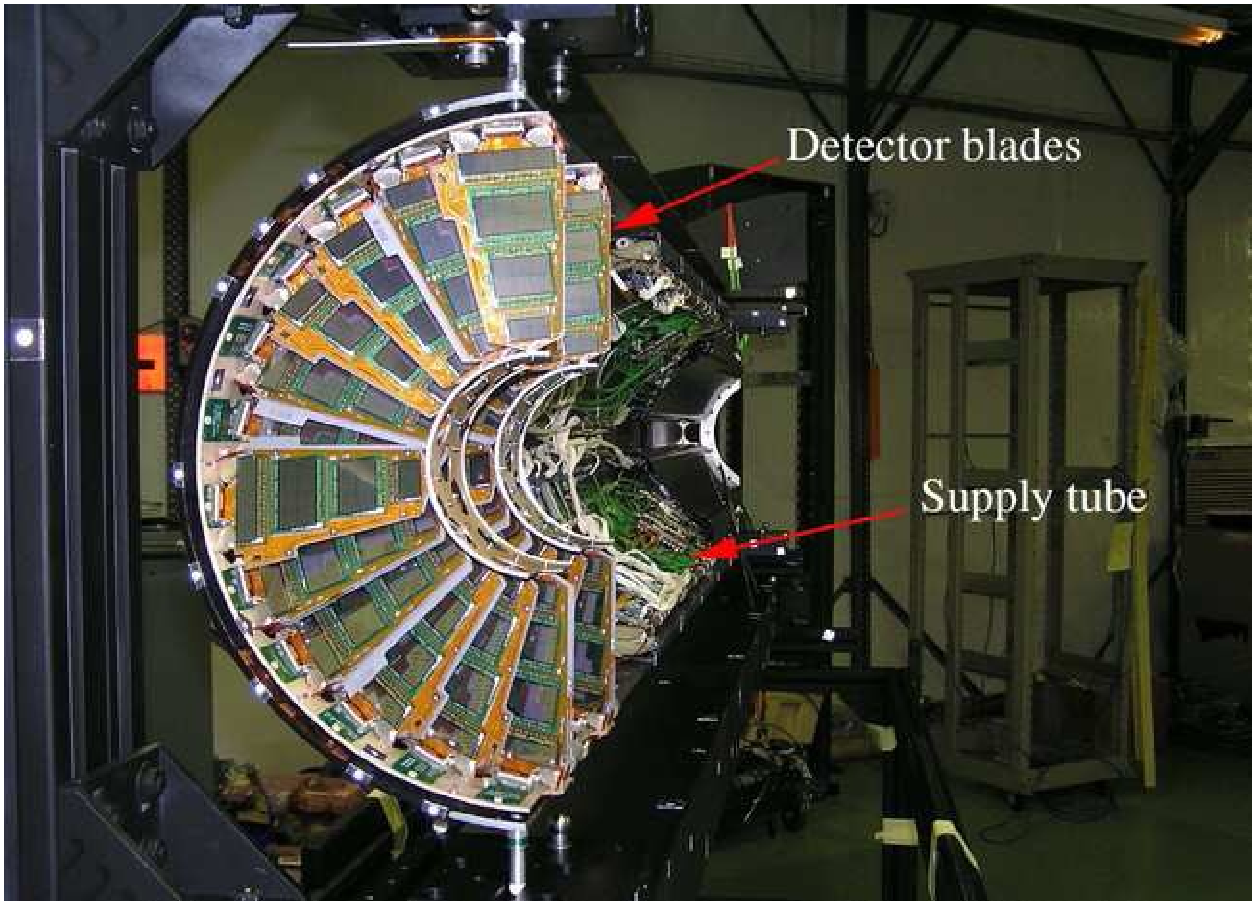


Figure 4: A “half cylinder” of the Forward Pixel detector showing two half-disks that were installed on one side of the interaction region. Panels made up of plaquettes of siliconsensors are visible

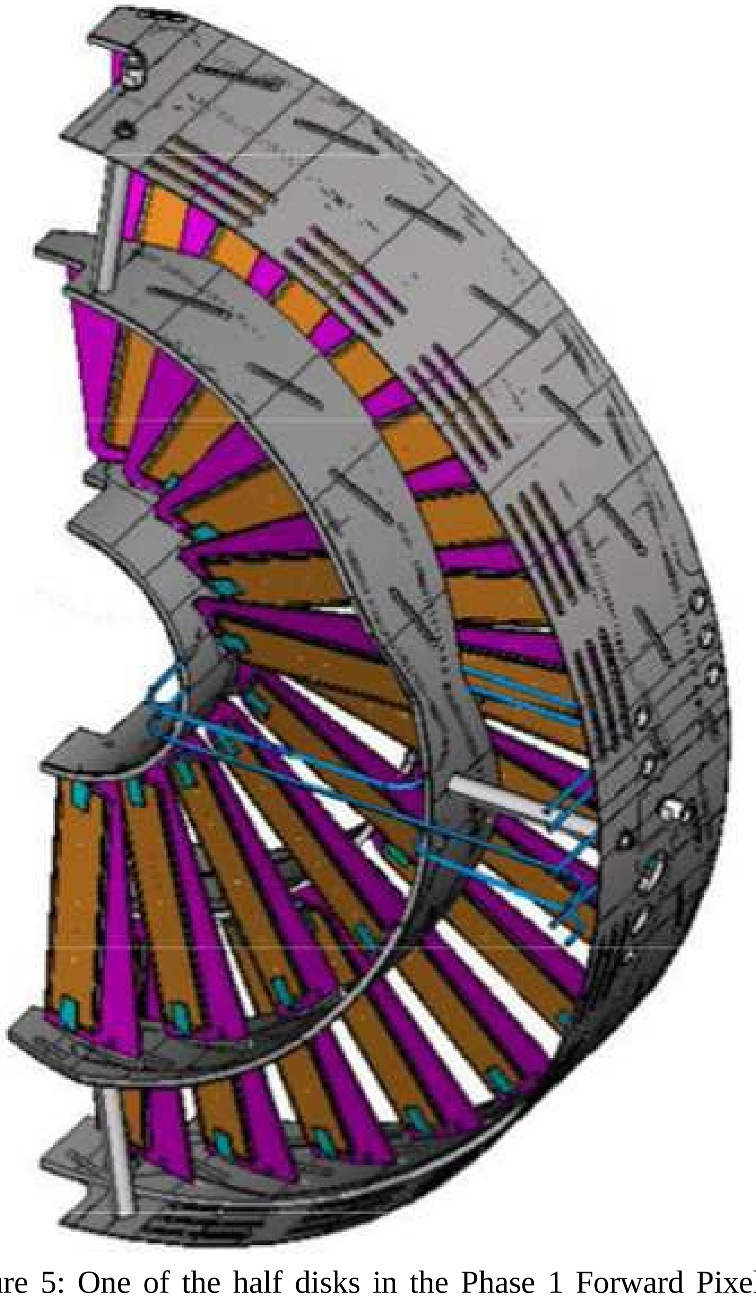


Figure 5: One of the half disks in the Phase 1 Forward Pixel Detector

2.3 – Design of a Tracker-Based Trigger

One of the challenges that CMS must meet when running at the SLHC is to upgrade the L1 trigger to handle the increased luminosity. The trigger is the set of algorithms designed to filter events in the detector and select for only “interesting” events. Since proton-proton collisions result in numerous types of events, some of which have been completely and thoroughly studied and others which are too complex to understand, there is no need and not enough available computational resources to store every single piece of experimental data from the detector. Thus, we have need for a trigger to remove these “boring” results.

There are currently three levels of triggers, with the first level L1 being “quick” and the third level L3 spending much more time sorting through the data. When the luminosity is upgraded, as Figure 6 shows and I will explain below, it will “break” the current trigger. We would like to get the sophistication of L3 information at L1. So we are motivated to change the geometry, such as to Phase 1. Changing the geometry requires studies to be done to make sure everything works correctly.

Figure 6 illustrates the single muon rates for various L1 and L3 selections as shown for a luminosity of 10^{34} $cm^{-2}s^{-1}$. It can be seen that a pT threshold of about 20 GeV/c is required to keep the rate below 10 KHz, which is 10% of the maximum L1 rate. If one assumes that the trigger rate for a given threshold increases by a factor of 10 if one increases the luminosity by a factor of 10, it appears from Figure 6 that the L1 trigger would likely no longer work. Without additional considerations, simply increasing thresholds to account for a 10-fold increase in luminosity would inhibit physics studies. And as the figure shows, the rejection curve actually flattens out, so that it appears it might not even be possible to obtain the required rejection at the SLHC.

This motivates the use of tracking information in the L1 trigger at the SLHC. The momentum of this track, as determined by the pixel tracker, is also used to sharpen the lepton pT threshold; the pixel tracker has ten times better resolution than the muon system. Looking at the L3 rejection curve in Figure 6, there appears to be hope that requiring tracking information will reduce the trigger rates to acceptable levels. The L3 trigger adds tracking information from the pixel tracker, and requires the presence of a track that is spatially matched to the high-pT lepton triggered on. However, implementing even this minimal requirement in the L1 trigger introduces significant technological challenges. The L3 trigger data shown in Figure 6 uses the offline tracking algorithm to sharpen the muon pT measurement. Detailed studies are needed using a realistic L1 tracking algorithm to check the performance at L1. (2)

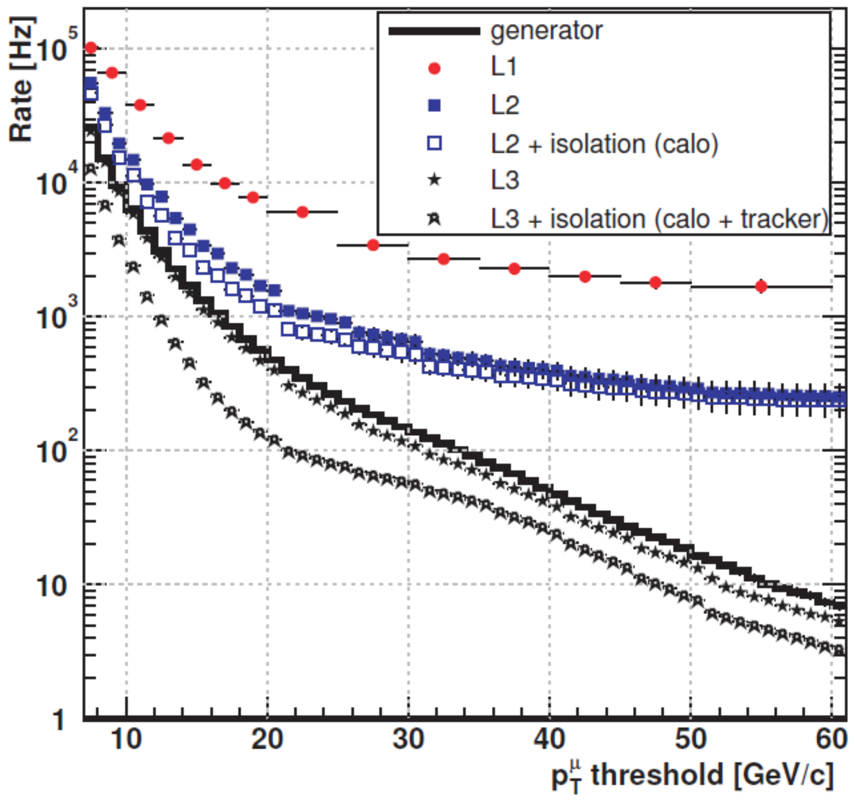


Figure 6: Single muon rates for various L1 and L3 selections

2.4 – Studies of Trigger Stack Layers

Providing trigger algorithms fast enough for use in the CMS L1 trigger system will require a geometry change in at least part of the pixel tracking system. Finding track stubs from only high pT tracks requires some association of data between different layers. This in turn requires a readout of a fraction of the data off-detector, thereby introducing complications of cabling, power, and additional material in the tracking detector. The data rate for the barrel pixel system has been estimated to be about 10 Gb/cm^2/s. Some reduction in the amount of data to be readout would probably be required either on-chip or on-detector. One suggestion of how this could be achieved is via closely stacked sensors to create hit pairs before the off-detector readout. Another possibility is to implement a specialized trigger readout for groups of pixels. An additional and significant challenge is to keep the amount of material to a minimum; the introduction of any additional detector layers or other material would be a serious concern.

A promising L1 tracking trigger strategy has been proposed for SLHC called "Stacked Triggers", illustrated in Figure 7. The main idea is to use stacks of closely spaced sensors to quickly find mini-vectors and reduce, by a factor of 10-100, the data that needs to be readout off-detector. This reduces power and cabling. Furthermore, with an appropriate choice of the separation between the stacked layers and pixel sizes, reconstructed mini-vectors would meet suitable minimum pT requirements. This would be done using either one stacked barrel layer, or several sets of stacked layers. Two sets of stacked layers would be needed to infer the track pT .

Although the proposed Stacked Trigger is a promising idea, there is a lot of work needed to show that the idea can work in a real detector under real conditions. In addition closely stacked layers are not ideal for offline track reconstruction. While the pixels of the trigger stacks provide excellent position resolution it is not the most efficient use of the space available. Studies are needed to show that a buildable pixel doublet can be used to produce a workable L1 trigger and to ensure that the tracking performance of the complete tracking system is not compromised by the addition of one (or more) trigger pixel doublets. (2)

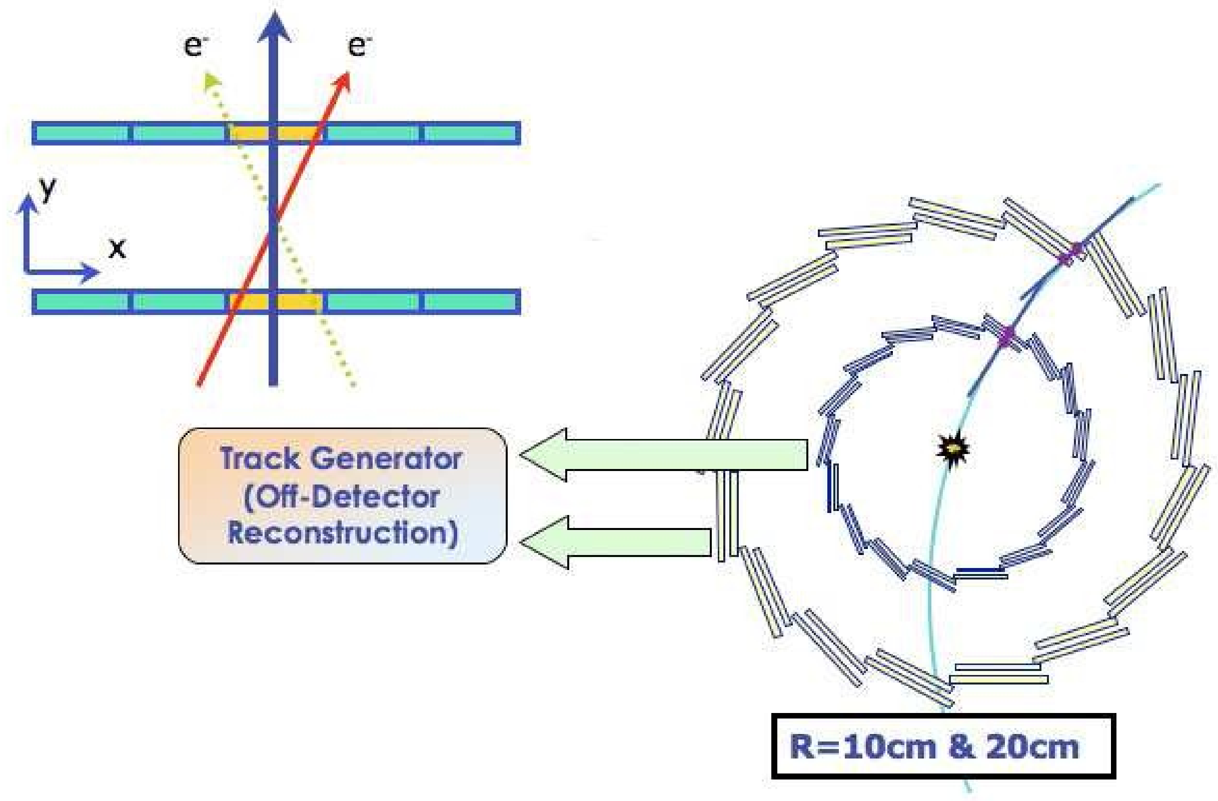


Figure 7: Illustration of “Stacked Triggers”

CHAPTER 3

MY WORK ON THE PHASE 1 TRACKER UPGRADE

3.1 – Phase 1 Upgrade

My area of research during the course of 2011, and the topic of this thesis, was looking into one area of the Phase 1 upgrade for the SLHC, under the guidance and supervision of Dr. Paul Sheldon (Vanderbilt), Dr. Eric Brownson (Fermilab), and the rest of the high-energy physics group at Vanderbilt. Then-postdoc Dr. Brownson was leading our efforts on the CMS detector upgrades required by the SLHC. He was stationed at the LHC Physics Center (LPC) at Fermilab where he could interact with other physicists working on these upgrades, in particular the Tracker Upgrade Simulations Working Group co-led by Harry Cheung of Fermilab.

Upgrading to the SLHC first requires going from the current Standard Geometry to the Phase 1 upgrades. The more hostile and challenging environment of the SLHC will require new designs for the CMS tracking system. Two distinct phases of upgrade are proposed. The initial Phase 1 upgrade would be a replacement of the Pixel Tracking System. This upgrade is driven by both the higher multiplicities and occupancies in the tracker and by increased radiation exposure of pixel detector. The extreme intensities produced by the later phases of LHC accelerator upgrades have motivated CMS to explore the possibility of including the tracker in the first level trigger. Dr. Brownson was performing studies to help evaluate the possible gains from both phases of tracker upgrades. He also became the release manager for the SLHC upgrade simulations. This work provided an excellent introduction to CMS analysis methods and techniques, and this is one reason why I was brought on-board the project.

Detailed studies are being carried out on the Phase 1 upgrade tracking system. The Phase 1 geometry contains a fourth barrel layer and a third forward disk in the pixel detector. A significant effort has been made to model the anticipated radiation damage in the tracker. In addition to radiation damage the high multiplicities of hits will be difficult to read out within the time between collisions. At an average pile up of 50, the current pixel detector is expected to have an inefficiency from 3% to 16%. The details behind pile up aren't particularly necessary, as one may just think of it as a measurement of the “junk” left over in the detector from previous or current collisions. Track finding is complex and a drop in pixel efficiency can be tempered by other mitigating factors within the tracker. For this reason, detailed analyses of tracking efficiencies and fake rates for different sources of tracks at high pile up must be made. The track seeding, resolutions, b-tagging, electron and photon performance for luminosities up to 10^{35} cm^{-2} s^{-1} must also be studied. The performance statistics for the Phase 1 & Phase 2 strawmen geometries will be compared to those for the standard CMS geometry.

The current CMS tracker geometry has a battery of packages that can be used to study its effectiveness. The one I utilized for the topic of this thesis is the MultiTrack Validator. The MultiTrack Validator is a validation utility and tool used to generate histograms and performance plots (e.g. efficiency, fake rates, resolution) by the Tracking Physics Object Group (Tracking POG), which are used to test, validate, and debug the track reconstruction chains. It analyzes the tracking performance by comparing every reconstructed Track with the corresponding TrackingParticle. It takes as input one or more ROOT files containing previously produced tracks, and gives as output a ROOT file containing a multitude of plots based on the reconstruction of the tracks. (6)

For the strawmen geometries, standardized packages must be modified or developed to verify tracking and triggering performance. As they are developed, studies are carried out to further refine improved strawmen geometries. The number and location in radius of the trigger stacks, the stack separation and the pixel size will all be studied to see how well trigger stack layers can work. Once completed, the packages will be used as part of an automated package for producing validation like plots. The immediate goals of the task are to provide input on the Phase 1 Pixel Upgrade performance & whether buildable trigger stacks can give sufficient information to create a working L1 trigger when running at a luminosity of 10^{35} cm^{-2} s^{-1}. If so, we can use the parameter space of acceptable trigger stack geometries to inform our longer term task of reaching a baseline design for an upgrade tracking system for CMS for the SLHC Phase 2 upgrade. (2)

3.2 – Upgrade Tools

Beginning work on the Phase 1 upgrades is by no means an easy task, as there is a vast infrastructure in place that one must first learn to navigate, and I needed to become intimately familiar with the tools, methods, and equipment involved in doing research with CMS. First, I had to successfully clear a few significant international bureaucratic hurdles, such as obtaining a grid certificate with the proper virtual organization credentials. Next, I needed to acquaint myself with several of the software tools in use by high-energy physicists from around the globe. These primarily include CMSSW, CRAB, and ROOT, which I will now describe.

I began my research with CMSSW, the CMS software package. My task involved helping the group upgrade their analysis software from an aging release of CMSSW (3.6) to a proposed future release (4.2) by developing a standard set of tracking efficiency plots used to compare the simulation of the proposed upgrades to the CMS detector between the two releases, and I was also tasked with varying simulation and reconstruction parameters to study how they change the tracking performance of various proposed upgraded tracking detectors.

However, the power of CMSSW cannot be truly exploited without utilizing the power of CRAB. CRAB is the utility that submits and runs CMSSW jobs across a grid of distributed computational resources located at various institutions, such as CERN, Fermilab, and Vanderbilt. By using CRAB, one is able to access CMS data distributed to CMS aligned centers worldwide and exploit their CPU and storage resources. (3)

Results obtained from CRAB and CMSSW cannot be fully appreciated without the use of ROOT. ROOT is an analysis package containing built-in functions and user-compiled code to produce graphs, histograms, and trees with data objects. ROOT has the benefit of being able to generate graphs of CMSSW job results from either the command line, a user-created script, or by easily navigating its GUI Object Browser. (4)

3.3 – Studying Secondary Vertexes

The primary goal of my project, as the title of my thesis would suggest, was to use the MultiTrack Validator to study secondary vertexes in the Phase 1 Upgrade. I wanted to make comparisons of the parameters between tracks originating from primary vertexes with those originating from secondary vertexes. The primary vertex is the one produced by the initial collision of two protons from the LHC beam. Particles from the primary vertex can later decay into other particles, and the point at which these tracks split are called secondary vertexes.

Having a good method for differentiating between primary and secondary vertexes is quite important, because in addition to improving the vertex finding for primary vertexes, the Phase 1 Tracker Upgrade is expected to contribute to the locating and evaluation of secondary vertexes. With increasingly complex physics topologies being searched for, secondary vertexing will play an increasing role in searches for new physics. We are very interested in knowing, how much will our secondary vertex finding efficiency and resolution increase with the Phase 1 Tracker Upgrade? Thus, we must have a full understanding of secondary vertexes from our simulated and reconstructed tracks before beginning to run actual experiments.

There are two different, equally valid methods available for tackling the problem of searching for new physics: the MultiTrack Validator, as I previously discussed, and B-tagging.

B-tagging is a set of algorithms used to tag b-jets for the purposes of physics analysis. Here, b stands for the bottom quark in a jet produced from a proton-proton collision. B-jets are important because they are an important signature for new physics, such as the Higgs Boson. B-tagging works by associating a single, real number called a discriminator with each jet. B-jets will always tend to show higher values of the discriminator, but the details depend on the specific algorithm. All algorithms produce a discriminator for each jet to distinguish b-jets from non b-jets. CMSSW has already implemented several b-tag algorithms, with some exploiting the long B hadron lifetime, some exploiting its semi-leptonic decay mode, and some using kinematic variables. All of the algorithms require two inputs: the primary vertex (in a sorted collection, the first element is used as the signal vertex) and the jets to be tagged and their associated charged tracks. (7)(8)

CHAPTER 4

THE “TIP_INNER” SAGA

Over the course of the spring and summer of 2011, as I've already stated, I utilized the MultiTrack Validator to study a potential upgrade to the SLHC, going from the Standard Geometry to Phase 1. At the time of my project, I was using the most up-to-date version of the CMSSW software package, CMSSW_4_2_3_SLHC2. The primary focus of my particular project was to study how well simulated tracks were reconstructed and investigate if there were any statistical differences between tracks originating from primary vertexes, occurring near the center, and tracks originating from secondary vertexes, occurring at the fringes.

4.1 – Choosing a test sample and setting parameters

Before beginning work, it was necessary for me to choose a test sample to work with. A single, consistent test sample was necessary as I needed a standard in order to make proper comparisons. I needed a sample with a consistently high efficiency when tracks were reconstructed with the original CMSSW software, so that when making my own modifications to the code, a significant drop in efficiency would be a warning sign that the code had a bug. To that end, I decided to use the 4-muon sample.

I also needed to choose a standard plot for making my comparisons, so that I would always make sure I was comparing the same two variables before diverging out into comparing other variables. To that end, I selected the “efficiency vs η” set of graphs as my standard, primarily because it was a simple graph to select. Here, efficiency is how well tracks are reconstructed compared to the Monte Carlo simulated tracks. η is the pseduorapidity, a spatial coordinate used in particle physics to describe the angle of a particle relative to the beam axis. The full details behind η beyond it being a spatial coordinate are not important for this discussion. What is of primary importance is the efficiency, because an increase in efficiency is directly tied to the Phase 1 Upgrade.

When working on this sample, it was up to me to choose certain parameters for running jobs with CRAB, such as the number of events per job and the number to use for pile-up. At the time I began my work, there was a problem with the configuration file, causing up to 50% of jobs to fail due to a vertex error. Given this, I needed to keep the numbers low, so I initially started using a pile-up of 25, bringing it down from its default value of 50, and kept the pile-up at 25 for most future jobs.

4.2 – Checking the Beam-Spot

Since the purpose of the project was to see if there was a statistical difference in track reconstruction near the center of the beam-pipe vs. much further away, the first thing I needed to check was the position of the beam-spot. If the beam was not directly centered along the beam pipe, collisions would not happen at the origin, and if they were off-center, this would obviously affect the results we wanted. I modified the code to output the position of the beam-spot over 1000 events of the 4-muon sample. Each event resulted in showing all tracks originated at (0,0,0), so that meant we could safely say the beam-spot was perfectly centered.

4.3 – Introducing “tip_inner”

When I came onboard the project, the software was written to represent the detector surrounding the beam pipe as a cylinder. Tracks would originate from the center of the beam pipe and travel through the layers of the detector. The CMSSW software would reconstruct these tracks and provide a wide range of statistical information about them as output from jobs ran over the test samples. The software was compiled by different people on top of pre-existing software just to do the types of analyses they wanted to do. So my job was to look at this code and figure out what I needed to modify and how I needed to modify it to study the new parameters I was interested in and have the intertwined programs continue to work with each other. My parameter of interest was the transverse impact parameter; the default name of this parameter in the software packages was “tip”. This is the distance of closest approach of a track to the center of the beam pipe, and it is assumed to be where the tracks originate in the beampipe.

My approach to studying the reconstructed tracks originating from near the center or out on the fringes was to divide tracks with a “tip” value into two separate zones, meaning I needed to divide the original “tip” parameter into an outer transverse impact parameter, “tip_outer”, and an inner transverse impact parameter, “tip_inner.” Both are simply two different values for the transverse impact parameter emerging from the center of the beam pipe, with “tip_inner” < “tip_outer”. The “tip_outer” parameter was simply the former “tip” parameter, but renamed. Within “tip_outer” was a second, inner transverse impact parameter, named “tip_inner”. Separating “tip” into two separate zones like this enables several ways of studying the results of simulations.

First, I could set “tip_inner” = 0 and keep “tip_outer” set at the default value for “tip”, which hypothetically would provide the same results as the unmodified code, as I'm considering the detector and beam pipe as one entire entity without making separate zones. This was a great way to check my modifications for bugs. For studying results close to the center of the beam pipe, I could set “tip_inner” to some small value much less than “tip_outer”, with the final value being set by experimentally toying with various values and looking at the results, and then focus on only the portions of reconstructed tracks falling within “tip_inner”. For studying results at the fringes far from the center of the beam pipe, I could keep “tip_inner” set at the value previously determined, and only focus on the portions of reconstructed tracks in the areas greater than “tip_inner” and less than “tip_outer”. I started envisioning this setup as being similar to a hollowed-out cylinder, where I could remove reconstructed tracks near the center of the beam pipe and only examine those at the fringes.

The first place for me to add the new “tip_inner” parameter was the file *CommonTools / RecoAlgos / interface / RecoTrackSelector.h* in the CMSSW software package. This configuration file was already in place to declare and process the multitude of parameters being used in current research on reconstructed tracks, such as the transverse impact paramter from the beampipe, “tip”, the longitudinal impact parameter along the beampipe, “lip”, min and max rapidity, ptMin, etc. And given this as the parameter file, it was naturally convenient to add my “tip_inner” parameter in a similar fashion as the rest.

However, *RecoTrackSelector.h* is only the file that sets up the parameters; numerical values are assigned in a python script, *PhysicsTools / RecoAlgos / python / recoTrackSelector_cfi.py* in the CMSSW software package. This use of a Python script to assign values was convenient, since every time I modified *RecoTrackSelector.h*, I needed to recompile the software package before running a job, but not so for modifying the Python script. This made it convenient for me to experiment with different values for “tip_inner” and observe the results.

It was at this stage that I began to realize adding “tip_inner” to the configuration file was going to be a challenge, because the CMSSW software package had been written to depend on a multitude of interconnected programs and scripts that were dependent on each other. This meant that the configuration file and Python script I just modified would have cascading effects in the long term, so I couldn't necessarily trust any immediate results. And since adding a parameter was challenging enough, it also meant that I couldn't necessarily change the name of a parameter in one spot without completely breaking the software. At this point, for reasons I will discuss later, I decided it would be in my best interest, as well as the best interest of the entire CMS team, to not change the default “tip” parameter to “tip_outer”, as I had originally planned. And since “tip” and “tip_outer” were the same thing (I just wanted to change the name for purposes of clarity), I made the decision to leave that parameter name unchanged, even though I personally still thought of it as “tip_outer” to keep it clear for my own work.

4.4 – A Statistical Problem

Now that I had added “tip_inner” to the configuration files, the next step was to compare the results of setting tip_inner=0 with the results of the code without any modifications. My purpose in doing so was to check that my modifications and additions to the code had not had unintended consequences. It was a reasonable hypothesis to assume that setting tip_inner to be 0 should be the same as the default code where it was non-existent. Thus, I assumed these two cases should give exactly the same results. In Figure 8, I show efficiency of reconstructing simulated tracks vs η, from running the code over the 4-muon sample for two cases: the unmodified default code (black) and my modified code with tip_inner=0 (red). What one should immediately notice is that both are slightly different, when my hypothesis was that the black and red lines should completely overlap, as they're hypothetically the same thing.

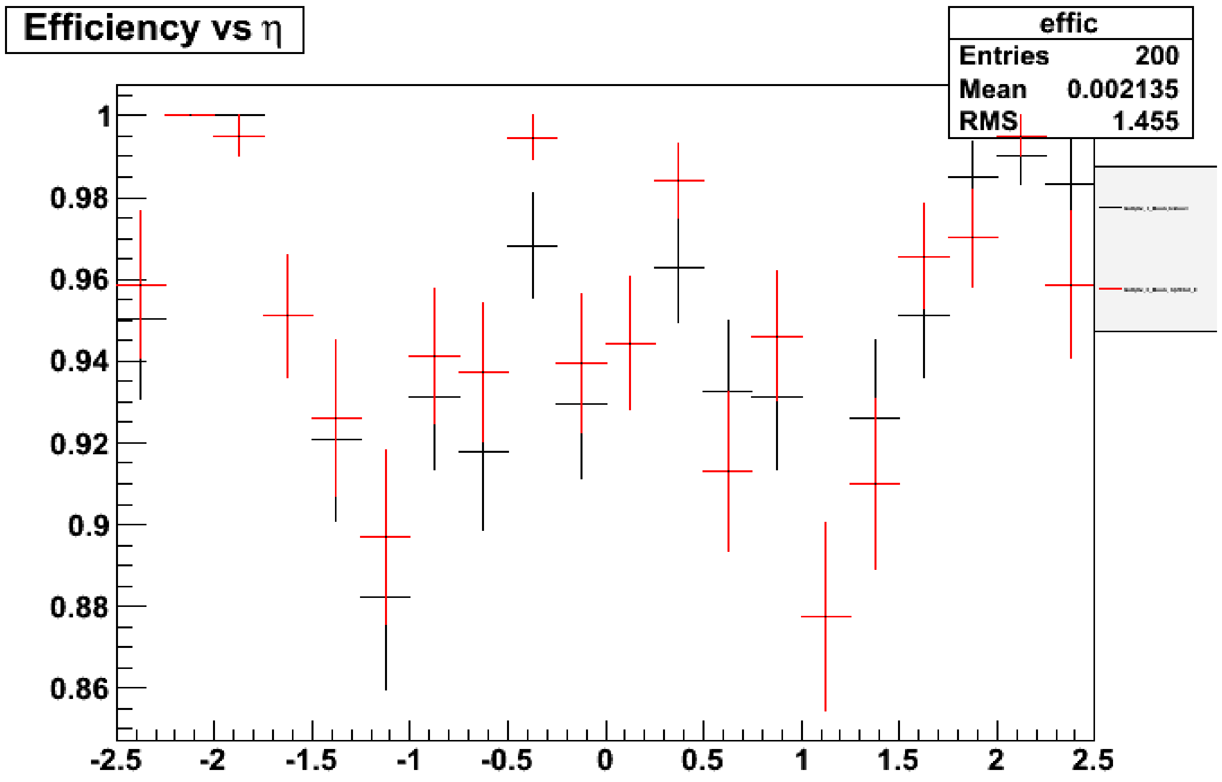


Figure 8: Efficiency of reconstructed tracks in default code vs. tip_inner=0

So this was a problem. Did I inadvertently change something, causing the code to run differently with my modifications? Or was there something else going on of which I was previously unaware? If there was an inadvertent change, it seemed very suspicious to me that the results would be so close to overlapping.

The next step in my investigation was to see what happened when I ran the default code over the same sample two different times. I was currently comparing my modifications to the code with the default code, but I realized I had no idea what results looked like from comparing results of two different jobs done in the same way and over the same sample. I have shown a plot of those results in Figure 9. As one should notice, once again the two results are slightly different in that they are close to overlapping, but they don't when I assumed they should. But this one had me stumped, because I literally did nothing different between the two jobs, so shouldn't doing the same thing always give the same result?

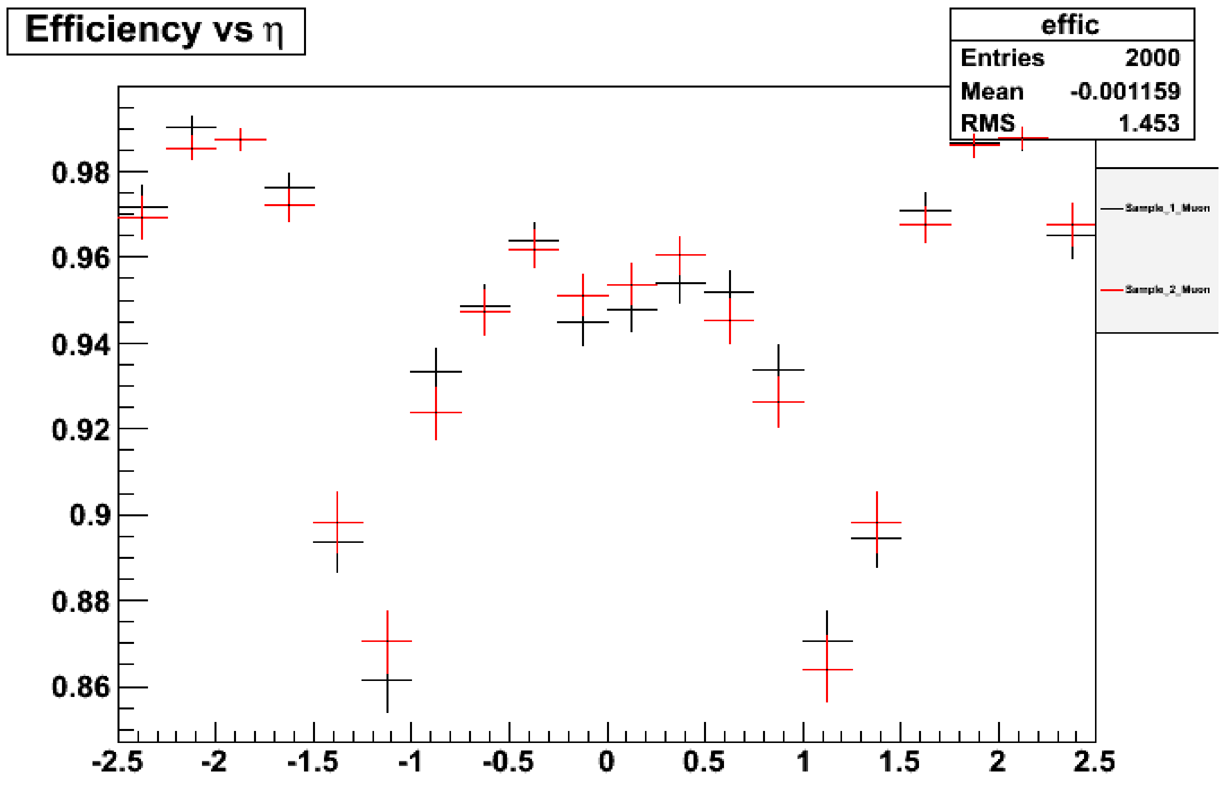


Figure 9: Two identical runs over the 4-muon sample

Upon further investigation, I eventually discovered that each time a series of jobs is run, CRAB chooses a different seed from a Poisson distribution depending on the numerical value for pileup, which introduces a small degree of randomness into the output. Thus, it would be impossible for me to make direct comparisons over different runs. At most, the closest together a result from two different sets of jobs could ever be is shown in Figure 9. I attempted to completely remove any notions of pile-up in a feeble attempt to still be able to make direct comparisons, but these efforts proved futile, as the jobs would not run without some value set for pile-up. Thus, I would have to force myself to be comfortable with not being able to make direct comparisions.

This means my method of comparison must be based on the statistics of the results, not the results themselves. Thus, I can only make judgements based on whether results are statistically the same or different. As one may also notice in Figures 8 and 9, the error bars overlap for most of the points, which implies they are both statistically the same. Therefore, we must accept that we can't make direct comparisons like I had preferred. For the purposes of this research project, statistical comparisons would have to suffice.

Fortunately, this also has the implication that my introduction of “tip_inner” into the code did not disrupt the overall cohesion of the CMSSW software like I feared upon seeing my first result in Figure 8. Based on the results in Figure 9, I judged the comparisons between the default code and my modifications to be statistically the same. Given that, I could now proceed further with my project.

4.5 – A Purity Problem

When getting back reconstructed track collections, samples could be divided into different categories, like low-purity and high-purity, depending upon extra requirements placed upon them. Up until this point, my research had focused on the general tracks (i.e. everything), as any time I ran a non-zero tip_inner value on high-purity tracks, the output that came back was blank. I was only able to obtain results with a nonzero value for “tip_inner” from keeping the general tracks. So my next step was to investigate this.

My first method to solve this problem was to modify the file, *CommonTools / RecoAlgos / interface / TrackingParticleSelector.h*, found in the CMSSW software package, along with its associated configuration file, *PhysicsTools / RecoAlgos / python / trackingParticleSelector_cfi.py*. These files were needed for initially setting up the true tracks, as opposed to the reconstructed tracks. I had not added the “tip_inner” parameter to these files, which meant the true tracks were not properly being set up for an inner and outer transverse impact parameter, so clearly there would be problems in getting back an inner and outer transverse impact parameter when the tracks were reconstructed. So the next step was to add the “tip_inner” parameter to these files, just like I had to the previous files.

Unfortunately, this idea turned into a dead end. All jobs I ran with the modifications to *TrackingParticleSelector.h* still resulted in the high-purity tracks coming back blank, with the general tracks still giving similar results as before. I used the “python -i” command to double-check that the scripts were recognizing the new “tip_inner” paramter, and they were. The next step was setting signalonlytp to false, but this was also a dead end.

Clearly, all of this indicated to me that there was a major problem in adding the new “tip_inner” parameter to the established code that went beyond just a few minor bugs. In order to solve this, I would need to figure out exactly how the original “tip” parameter had been coded so that I could replicate its results.

4.6 – Tip Tracking

As I previously mentioned, I ran into a few problems when attempting to rename the original “tip” parameter to “tip_outer”. This was most evident when attempting to extract information after modifying *TrackingParticleSelector.h* by running “cmsRun Harvesting_cfg.py”. Everything worked as expected when I kept the parameter as “tip”, but it all fell apart when I attempted to change it to “tip_outer”.

After a brief investigation, I discovered that when running the Python script “Harvesting_cfg.py”, it called another connected script that required the variable name to remain “tip” as was the default. And this script called another script, which called another script, etc., all of which required the default name. I eventually backtracked everything to a file called *Validation / RecoEgamma / python / tkConvValidator_cfi.py* in the CMSSW software package. I continued on and tracked “tip” to *Validation / RecoEgamma / python / tpSelection_cfi.py*, which seemed to enable “cmsRun Harvesting_cfg.py” to now work when I modified the name of “tip there.” It seems that “tip” is given a value of 3.5 in the default cfg file, but that gets modified to 120 to match the Reco file in the two Python files noted here.

Next, Figure 10 shows a piece of code that I found in *Validation / RecoTrack / plugins / MultiTrackValidator.cc* and *Validation / RecoMuon / plugins / MuonTrackValidator.h*. This call to “tpSelector = TrackingParticleSelector(...)” is set up for all the original parameters, but as one can see, this clearly breaks because it does not contain a reference to “tip_inner” after I modified TrackingParticleSelector.h since this call now no longer pointed to the correct parameters. Thus, I added a new line “pset.getParameter<double>("tip_innerTP"),” to the above code in the correct spot to match my modifications to *TrackingParticleSelector.h*. I attempted to run CRAB jobs with the edited Validation files, but I get back the error “Error occurred while creating for module of type 'MultiTrackValidator' with label 'trackValidator'” and “MissingParameter: Parameter 'tip_inner' not found” in the output files. There was no indication what file is trying to be called at this instance, so this left another mystery to be solved.

```
tpSelector = TrackingParticleSelector(pset.getParameter<double>("ptMinTP"),

                                  pset.getParameter<double>("minRapidityTP"),

                                  pset.getParameter<double>("maxRapidityTP"),

                                  pset.getParameter<double>("tipTP"),

                                  pset.getParameter<double>("lipTP"),

                                  pset.getParameter<int>("minHitTP"),

                                  pset.getParameter<bool>("signalOnlyTP"),

                                  pset.getParameter<bool>("chargedOnlyTP"),

                                  pset.getParameter<bool>("stableOnlyTP"),

                                  pset.getParameter<std::vector<int>
>("pdgIdTP"));
```

Figure 10: Example of code where I needed to add the new “tip_inner” parameter

My next approach was to hard-code “tip_inner” into *RecoIter_Fullsim_Phase1_cfg.py*, as I discovered certain configuration files weren't recognizing the Python scripts where I originally added “tip_inner” since they had their own set of the same original parameters. My first result with “tip_inner” set to 0 worked with my modified *MultiTrackValidator.cc*, which indicated I was finally starting to get somewhere. I next compared tip_inner=0 with tip_inner=2 for new modifications to *MultiTrackValidator.cc*. I was testing to see if this gets nonzero tip_inner values recognized for anything but the general track. However, CRAB jobs were still flatlining for the nonzero “tip_inner”, and now even the general tracks were flatlining for nonzero “tip_inner” values. This at least indicated some progress as I was getting consistent results for all tracks.

But this started making me wonder, were my jobs really failing for all non-zero “tip_inner” values, or just the values I had selected? By default, the original software had, in different places, programmed “tip” to values of 3.5 and 120, so I was selecting my “tip_inner” parameters to be less than that. Thus I decided to make a comparison, which I show in Figure 11, between results with the 4-muon sample on the boundary between “tip_inner” being greater than 0 (black and red) and greater than or equal to 0 (green) for the general tracks. As one can see, nearly all the error bars overlap for both cases, so I can conclude they are statistically the same. This implies that there are indeed some cases where non-zero tip_inner values are allowed.

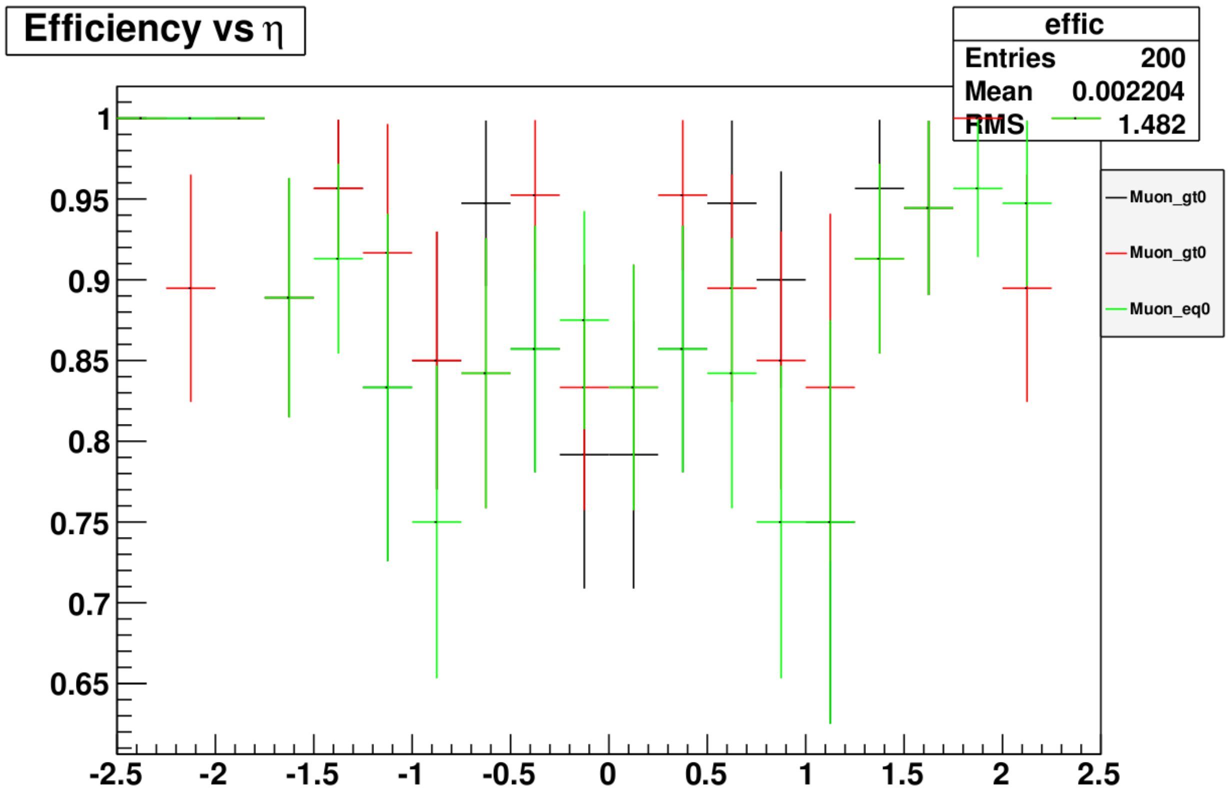


Figure 11: Comparison of tip_inner greater than 0 with greater than or equal to 0

With this new discovery in mind, I wanted to see if I could place a limit on just how much greater than 0 that "tip_inner" could be before all the results flatlined at 0. In Figure 12, I compare 7 samples of 100 events for various small values of tip_inner, changing order of magnitude from 0.0 to 0.1. One should first immediately notice that for all results that did not flatline, they are all statistically similar. Only the two samples for 0.1 and 0.01 flatlined. This implied to me that a value of 0.01 was an upper bound on the maximum allowed value of tip_inner, and perhaps this meant I could only pick up on tracks that were reconstructed very, very close to the center. And perhaps the default values of 3.5 and 120 set for "tip" were absurdly large upper bounds put in place by the original programmers, which would also explain why the values vary by approximately 2 orders of magnitude.

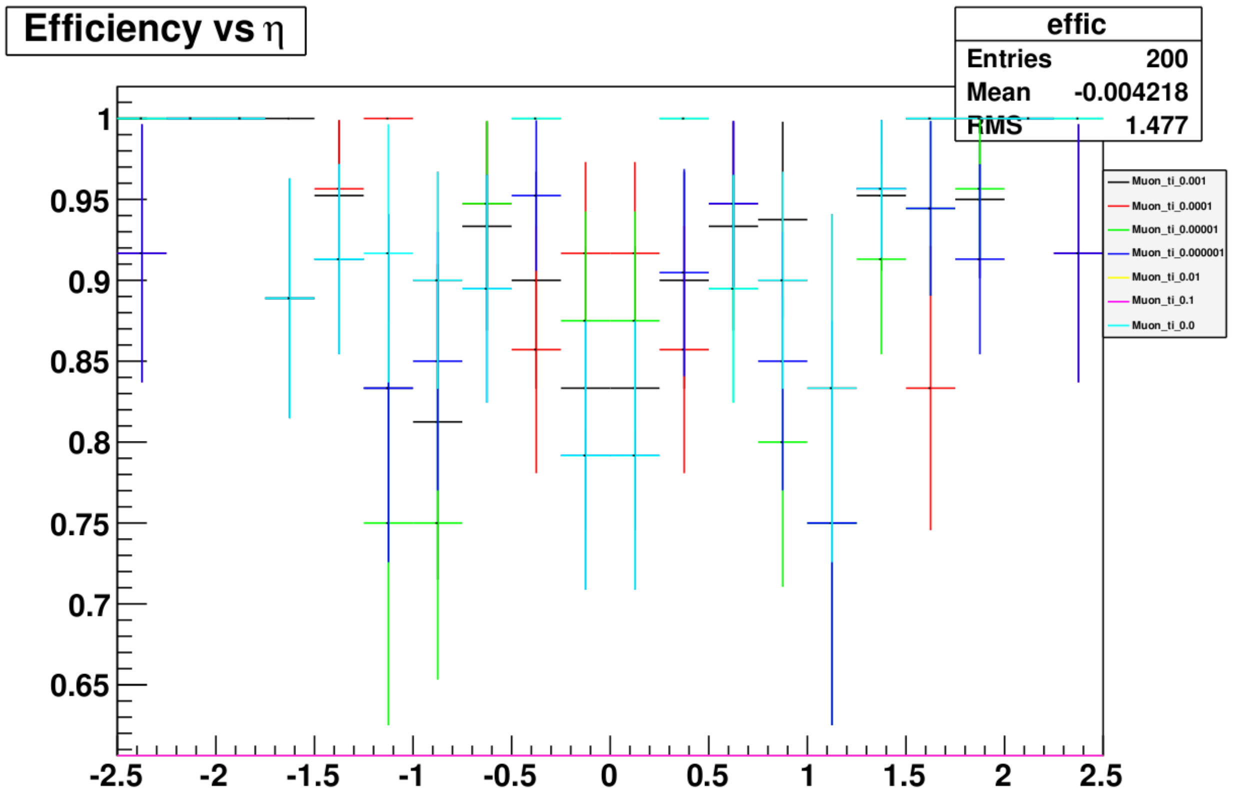


Figure 12: Looking at tip_inner for small values

I decided to test my suspicions on how the value for "tip" was initially set by altering my scripts to set the tip_inner selection to be <= instead of >= and then look for differences in the output. I compared 4 samples altering the value of tip from 0.1, 3.5, 60, & 120, and my results are shown in Figure 13. As one will notice, again there were no statistical differences, which provided further evidence that I was correct in my suspicions. It was important to constrain a "real" upper bound on "tip" so that I could be more accurate in setting "tip_inner" and also better understand what the results from adding "tip_inner" would actually mean.

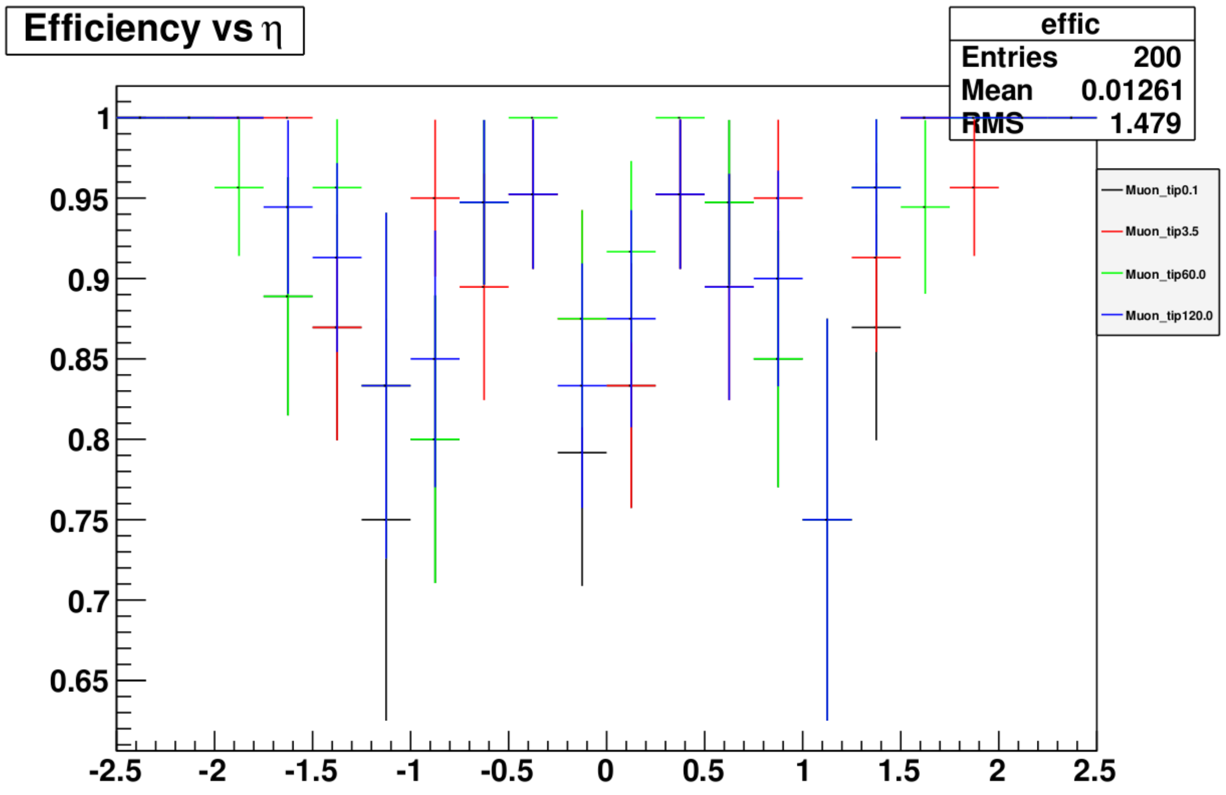


Figure 13: Comparing various values of “tip” to look for any statistical differences

After discussing my findings with Dr. Brownson, my suspicions about “tip” having an absurdly large upper bound seemed to be justified. However, he also pointed out to me another possibility. The beam has some smearing involved, which seemed to be around the same value I found where events were occurring. This makes it harder to separate the primary vertexes from the secondary vertexes, which was the primary motivation behind this entire project. Thus, things were starting to look grim for this particular direction of research. However, I was still holding out hope as I attempted to make a few last, desperate moves...

4.7 – I Want My MTV

It was starting to look more and more likely that the MultiTrack Validator was not going to be useful for finding secondary vertexes in the Phase 1 Upgrade. But I wasn't ready to completely give up just yet.

My next method involved attempting to try to hardcode “tip_inner” values directly into *.h files in order to bypass needing configuration files. But I kept getting errors when trying to compile. When hardcoded inside the main function, the errors imply I couldn't do that. When hardcoded outside the main function, errors keep referring to multiple definitions of the tip_inner parameter in the file, even though there shouldn't be any other definitions. I even changed the variable name to test this and got back the same error. I had no idea what this was about, though I suspected this was because header files can't have variables declared in them. Regardless, I decided to just live with needing the configuration files, but I needed to find a way to make them “cleaner”.

I reconfigured my setup for “tip_inner” so that the value for it was solely contained in a single parameter file, *PhysicsTools / RecoAlgos / python / tip_cfi.py*. I also added “tip_outer” to this file, so that I could set a single value for “tip” for everything. This method seemed better than hardcoding, since I wouldn't have to recompile everything every time I wanted to run the script. I now only had one place to change “tip_inner” and “tip” for each run. I set “tip” to be equal to “tip_outer” in all of the other parameter files I could find that set a value for “tip”. Once the new parameter was in place, I initiated another set of CRAB jobs to test out the new parameter file. However, once again, this failed to produce any meaningful results.

I discussed these latest findings with Dr. Brownson, and I learned that apparently the 4-muon sample I had been using this entire time might not have been the best choice for a test sample. Apparently the 4-muon sample doesn't produce many secondary vertexes, which is one possible reason for why I was not seeing tracks outside a small “tip_inner”. Instead, I should be using the TTBar sample. However, I tried that for 100 events, and found for “tip_inner” >= 0.1 and tip_outer <=3.5, I could only reconstruct events for the general tracks, not the high-purity tracks.

If you haven't been paying attention, that is the same problem I had been facing this entire time with the 4-muon test sample. This lead me to conclude there was an inherent problem in configuring the software to introduce “tip_inner”, rather than a problem with any specific test sample. I had exhausted nearly every idea and option at my disposal to create the new “tip_inner” parameter, but all my efforts were proving to be futile. I was finally forced to admit the inevitable...

4.8 – The Kobayashi Maru

I was in a no-win scenario. This particular project was a dead end in the wrong direction. The MultiTrack Validator simply cannot be used effectively to distinguish primary and secondary vertexes in the Phase 1 Upgrade. The software just is not set up properly to be configured for it.

I strongly suspect the primary problem is inherent in how the code is set up. In my opinion, it seems the software was configured for a specific, pre-determined set of parameters, and the entire CMSSW software package was constructed around that. It is not designed to have new parameters, like “tip_inner” inserted into it. It essentially feels like the software scripts were written by various scientists for the specific jobs they wanted to do, so there is unfortunately no single unifying script. Everything has become a hodge-podge of scripts that were “good enough” at the time they were originally created and needed. And this idea is consistent with the notion that high-energy particle physics research is a fast-paced field, where the software needs to primarily be “usable” instead of “pretty”.

The main problem, and what I suspect is the problem that made this project into a dead end, is that all of the scripts in the CMSSW software package are too heavily interconnected and dependent on each other. Remember Figure 10 that showed a sample of code from *Validation / RecoTrack / plugins / MultiTrackValidator.cc*? Other scripts in the software package, like *Common Tools / RecoAlgos / interface / TrackingParticleSelector.h*, rely on that set of code for taking in the necessary parameters in a specified order. It also seems to rely on the code remaining unchanged from how it was initially written. So when I modify one script, e.g. *MultiTrackValidator.cc*, to add “tip_inner”, the software doesn't work properly because the other script, e.g. *TrackingParticleSelector.h,* breaks because it is unable to interpret the input. Or vice versa. When I modify both scripts to add “tip_inner”, the software still doesn't work properly, presumably because another unidentified third script somewhere else was relying on those two first scripts to remain static.

With such a high level of dependency on each script to be written in a certain way and in a specific order, it becomes a nearly impossible task to track down each and every script that needs to be modified to make the software work properly if one wants to introduce a new parameter. For example, I found one place where I needed to insert “tip_inner”, but doing so caused an error in compiling the code, but if I fixed it so it could compile, the script wouldn't run properly. I eventually found a work-around, as there was no way to tell which of the hundreds of scripts in the CMSSW software was responsible for this “damned if you do, damned if you don't” scenario. But even with the work-around, the jobs still would not run correctly.

Even when I could get the software to compile successfully and tell me that my new “tip_inner” parameter is being recognized, there's still no guarantee that it's *actually* being recognized. It could just mean that one set of scripts is recognizing it, but another set of scripts that are connected further along the chain won't recognize “tip_inner” once jobs are actually running because my initial methods to check could only probe a maximum number of degrees of separation. I make the analogy that attempting to track everything down is like playing the classic arcade game of Whac-A-Mole. I could fix errors caused by one script not being modified, only to have that fix reveal another error in a different script that couldn't be previously revealed when the first script wasn't working correctly.

And in the end, I'm forced to admit, this just is not an efficient or effective way of proceeding on this project. But, perhaps there is another way...

CHAPTER 5

HOW I LEARNED TO STOP WORRYING AND LOVE B-TAGGING

Having concluded that the MultiTrack Validator was the wrong approach to the project, Dr. Brownson and I concluded that a better approach would be to switch to B-tagging. However, by the time the project had advanced to this point, it was now the end of the summer of 2011. This meant that Dr. Brownson's post-doc term with the Vanderbilt HEP group had ended, which unfortunately forced an end to this particular project. However, I was able to begin looking into a few rudimentary ideas for making the switch to use the new coding routines of B-tagging, which I will now describe.

The first hurdle I had to overcome was learning how much different B-tagging was from the MultiTrack Validator. I initially thought both could be treated as validation packages, as the initial script just required me to run *recbtag_validation_cfg.py* instead of *Harvesting_cfg.py*. However, I quickly learned there were major differences between B-tagging and the MTV, such as completely different results for ROOT files.

Dr. Brownson and I outlined a plan of attack for using B-tagging. First, I should begin by primarily looking at two variables in the outputted ROOT files, flightdistance2dsigall and sigb and figure out what they actually physically represent. Looking into a path named IPTag_GLOBAL, we suspected flightdistance2D may be filled with resolution of the secondary vertex as one of the parameters. Though we had we no concrete evidence of that, just a hunch.

However, I never got a chance to really look into this. I started attempting to run jobs, but kept getting back the error message:

```
cms::Exception going through module CastorDigiProducer/simCastorDigis run: 1
lumi: 666685 event: 1901
If you wish to continue processing events after a ProductNotFound exception,
add "SkipEvent = cms.untracked.vstring('ProductNotFound')" to the "options"
PSet in the configuration.
```

My task after that was to track down what this error meant and where I needed to add the suggested line. I followed the lead to *SimCalorimetry / CastorSim / data / CastorDigiReco.py*, but it did not work. After conferring with Dr. Brownson, I discovered I needed to add the line “pdigi.remove(simCastorDigis)” to the end of *SLHCUpgradeSimulations / Geometry / python / Digi_stdgeom_cff.py* and the lines “process.DigiToRaw.remove(process.castorRawData)”, “process.DigiToRaw.remove(process.siPixelRawData)”, and “process.RawToDigi.remove(process.siPixelDigis)” to *SLHCUpgradeSimulations / Geometry / test / RecoFull_Fullsim_stdgeom_cfg.py*. I still don't fully understand why the error told me to add a line that did no good, but I wasn't questioning it since all my CRAB jobs ran successfully after following the advice of Dr. Brownson.

Of course, “ran successfully” and “worked successfully” are not necessarily the same thing, as I learned when actually trying to retrieve output from certain CRAB jobs and the output kept coming back as corrupt even though the jobs ran successfully and did not crash. What I ended up discovering was that generating the results of these CRAB jobs produced excessively large output files that were bumping up against my disk quota. For example, a 2000 event sample ended up being 21 GB in size. Obviously, this isn't an efficient or effective way when the research would require me to study multiple files similar to this. It is clear that much additional work will be required before a complete understanding of this phenomenon occurs.

But that is the point where this research project stops. Literally.

CHAPTER 6

FOR FURTHER STUDY

As it so often goes with scientific research, wrong turns do happen. I still learned a lot from my time spent on this research project about high-energy particle physics, the inner workings of Fermilab, CMS, SLHC upgrades, the various software packages used in such research, etc. And in that regard, I fully believe this project was a successful and worthwhile endeavor. But it is hoped that this study will stimulate further investigations in this field.

In conclusion, my findings show that the MultiTrack Validator is NOT useful for studying secondary vertexes in the Phase 1 upgrade. I presume it would be possible to somehow add "tip_inner" as I originally intended—after all, some collective of scientists originally wrote the CMSSW software package and made all the default parameters work cohesively—but future physicists should be warned that modifying that software for a new parameter is no easy task. If you still wish to do it, the main advice I can offer is that you should be prepared to essentially rewrite the entire CMSSW software from the ground up.

It is my sincerest wish that this thesis serve as a warning to all who read it that B-tagging is ~~the best~~ a more suitable way to approach this goal. If someone wishes to continue this project from where I left off, I strongly recommend proceeding in the direction of B-tagging, as I outlined in Chapter 5. I wasn't able to make it work, but I suspect that's only because I spent a mere few weeks looking into B-tagging. I simply was not able to devote the necessary amount of time to it as I was to the MultiTrack Validator. The next problem that needs to be overcome is figuring out why I was getting such absurdly large output files and/or find a work-around for it. Of course, I'm sure there will still be more unforeseen problems ahead pursuing this path, but I won't be going along for the ride.

But to any high-energy particle physicists pursuing this goal in the future: good luck and godspeed.

REFERENCES

(1) *Search For High Mass Resonances Decaying to Tau Pairs at the CMS Experiment*, Carlos Andres Florez Bustos, Ph.D. Thesis, Vanderbilt University, 2011.

(2) *Annual Report Number 2 for NSF PHY-0855651*, Vanderbilt High-Energy Physics Group, Internal Publication, 2011.

(3) https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideCrab. Retrieved 26 March 2012.

(4) https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookBasicROOT. Retrieved 26 March 2012.

(5) http://en.wikipedia.org/wiki/Compact_Muon_Solenoid. Retrieved 21 March 2012.

(6) https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideMultiTrackValidator. Retrieved 27 March 2012.

(7) https://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideBTagging. Retrieved 27 March 2012.

(8) http://www.quantumdiaries.org/2011/05/12/to-b-or-not-to-bbar-b-jet-identification