UCSD

An Easy to Use Track Comparison Tool

Ryan Kelley

Boris Mangano

Outline

- Why use this tool?
- A couple of Implementation Details
- Example: compare CTF tracks vs RS tracks

Why use this tool?

- Many different track reconstruction algorithms and configurations in CMS.
  - RS, CTF, different seeding configurations, different min p_T,etc.
- Can only compare different track reconstruction algorithms/configurations in a **statistical** manner
  - compare averages or rms.
- This tool can be used to directly compare the properties of the reconstruction on a **track by track** basis.
  - Given a TrackingParticle, was it matched to a recoTrack from configuration A?

    Matching is configurable in .cfg file

    Matching hits is default
  - Was it matched to algorithm/configuration B?
  - Well suited for use in FWLite and is quick and easy.

What's Different about this Tool?

- Redesign and extension of an earlier attempt to create a root tree based validation tool.

http://indicosearch.cern.ch/search?d2y=&c=&f=&d1y=&d1d=&p=Ryan+Kelley+650_7%3A%220%3A1l28%3A2l76%2A%22&d2m=&sc=1&d1m=&d2d=

- The two main differences:
  1. References to the original objects (reco::TrackRef(s), TrackingParticleRef, etc.) are kept.
     1. Don't have to select variables and fill a tree statically (e.g. p_T, eta, #hits, etc.)
     2. Increases the shelf-life of the tool.
     3. Doesn't significantly increase the size of original dataset.
  2. The two different reco::Tracks between the two algorithms/configurations are kept in a single branch to facilitate quick root command line comparisons.

Basic Structure of the Tool

- An Object that stores a reference to a
  - TrackingParticle
  - Reco::Track from reconstruction algorithm/configuration A (algoA)
  - Reco::Track from reconstruction algorithm/configuration B (algoB)

- An EDProducer then creates three vectors of this object and stores them as three separate branches
  1. Outer loop over TrackingParticles and inner loop determines if matched (by hits) to a reco::Track from algoA and/or algoB.
     - Direct comparison (Matched to both A and B)
     - Study where one algo/conf. finds and other misses (matched to A but NOT B, B NOT A, or neither A or B)
     - Calculate efficiencies
  2. Outer loop over algoA reco::Tracks and determine if it is matched to a TrackingParticle.
     - This can be used to determine fake rate for algoA
  3. Outer loop over algoB reco::Tracks and determine if it is matched to a TrackingParticle.
     - This can be used to determine fake rate for algoB

- twiki:
  - link

Example: CTF compared to RS

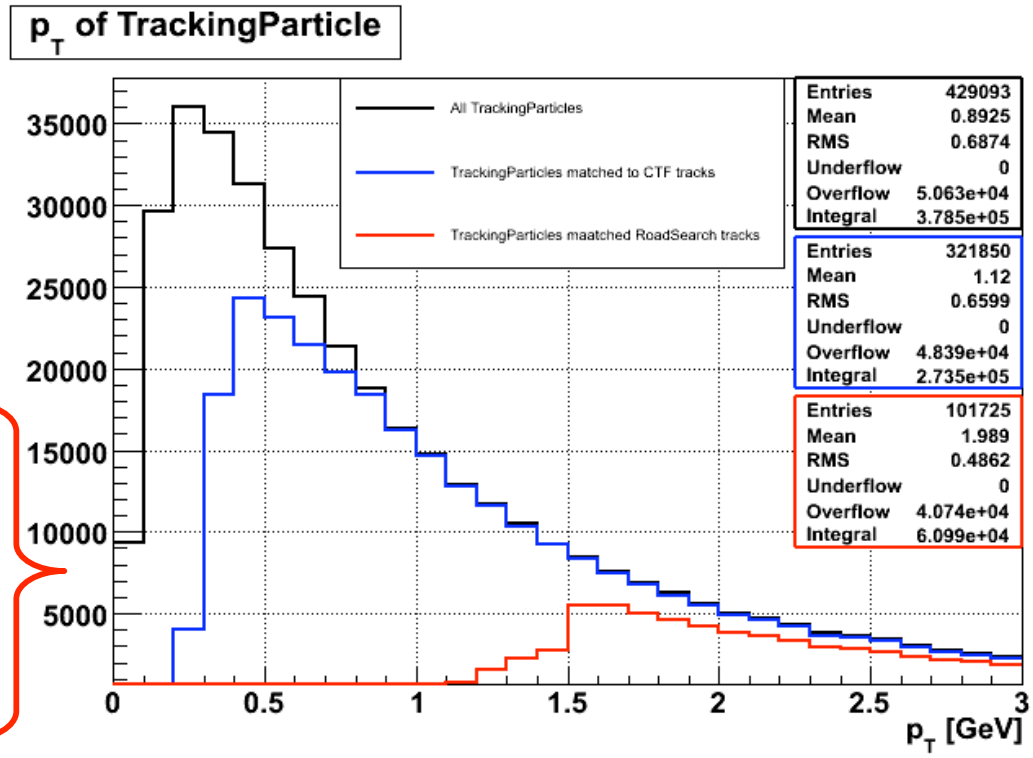
- As an example look at difference between CTF tracks and RoadSearch tracks.
- Caveat: not an in depth study
- The next few slides have some plots that illustrate some of the differences.
  - The following plots were made using
    - CMSSW_2_0_0_pre7
    - Sample: Release Validation Bjets 50-120 GeV
      - (/RelValBJets_Pt_50_120/CMSSW_2_0_0_pre7-RelVal-1206465939/RECO)
- Used “match by hits” association map
- Few steps to set up ROOT and FWLite:
  - Lauch root and load FWLite:
    - gSystem->Load(“libFWCoreFWLite);
    - AutoLibraryLoader::enable();
  - Load the file:
    - TFile* = new TFile(“filename.root”)
  - Set up an alias (to reduce typing to much):
    - Events->SetAlias(“T”,”TPtoRecoTracks_trackAlgoCompare_TP_TrackAlgoCompare.obj”)

Example: CTF compared to RS (p_T)

- Let's look at some plots to make sure that this tool is working properly

- No need for an analyzer (all the info is stored in one branch)
- Root CINT command are below

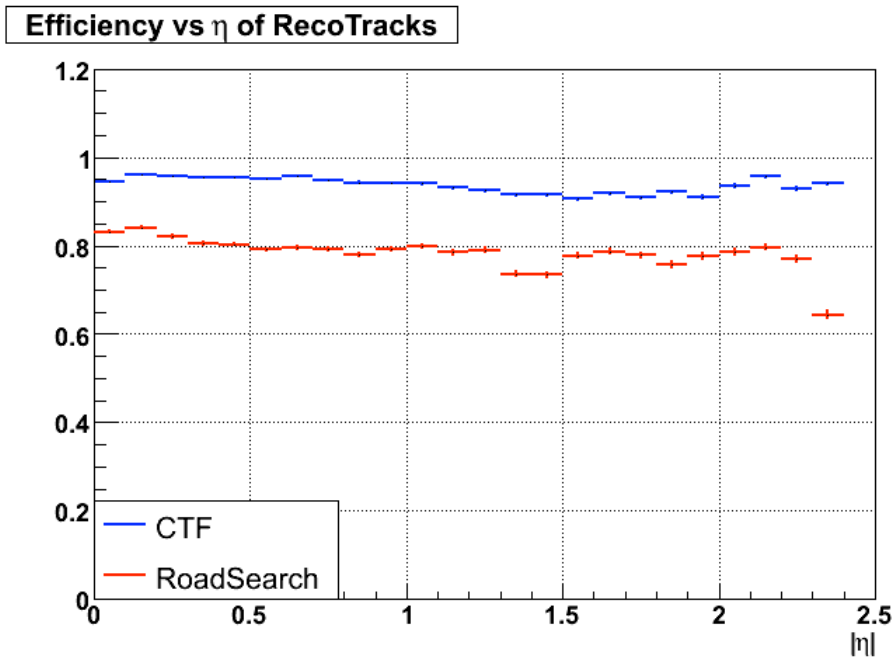
- Difference in min p_T between due to the difference in the seed requirements (CTF: 0.3 GeV vs RS: 1.5 GeV)

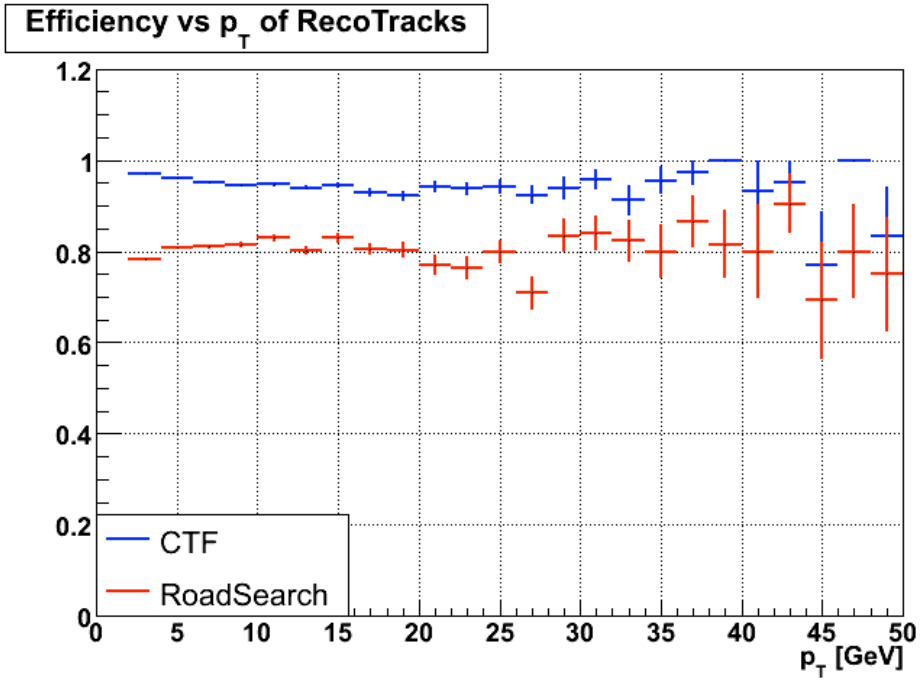


- Note: The difference in the cuts for the TP p_T plot is due to vertex cuts.
- **TrackingParticle** p_T:
  Draw("T.TP().pt()", "T.TP().charge()!=0 && T.fabs(T.TP().eta())<2.4 && fabs(T.T().vertex().rho() <3.5 && fabs(T.TP().vertex().z())<20")
- **CTF Track** p_T:
  Draw("T.TP().pt()", "T.TP().charge()!=0 && T.MatchedA()","sames")
- **RoadSearch** Track p_T:
  Draw("T.TP().pt()", "T.TP().charge()!=0 && T.MatchedB()","sames")

Efficiency

- As another sanity check, I've included the efficiency plots

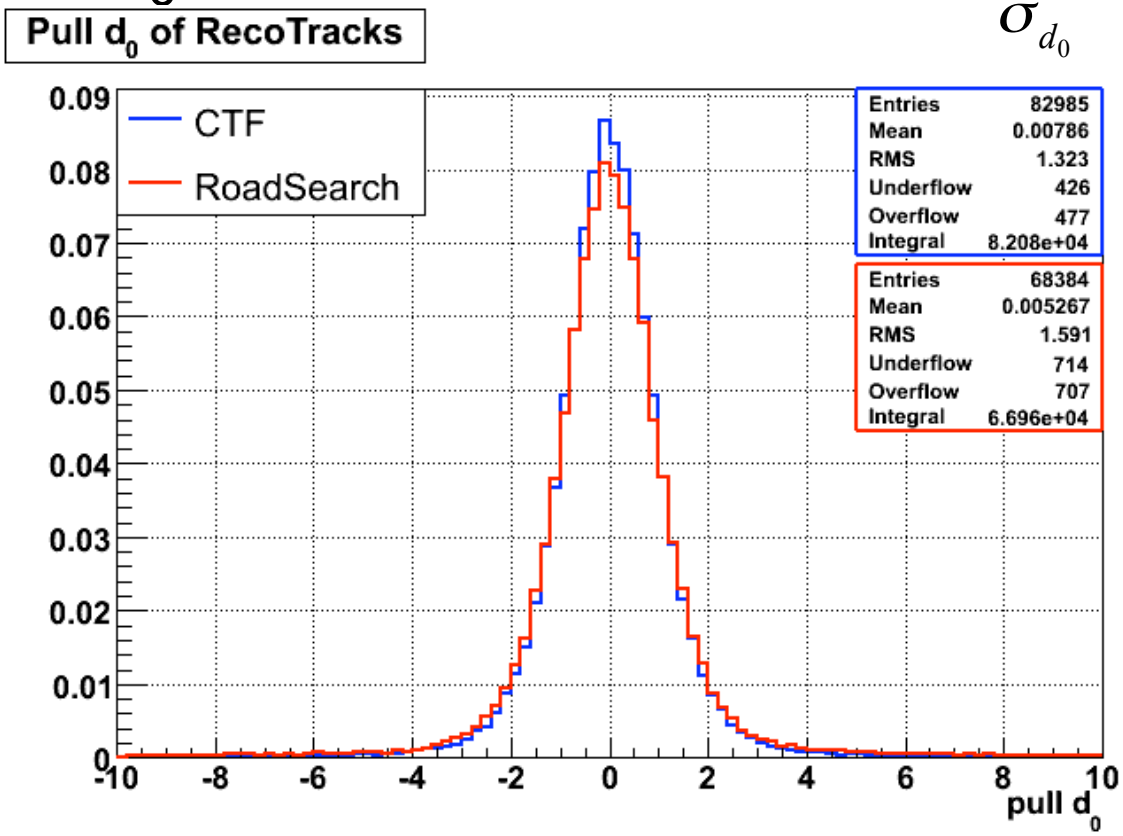




- **Efficiency vs p_T (CTF)**:
  Draw("T.TP().pt()>>hDen", "T.TP().charge()!=0 && T.TP().pt() && T.fabs(T.TP().eta())<2.4 && T.vertex().rho()<3.5 && fabs(T.vertex().z())<20")
  Draw("T.TP().pt()>>hNum", "T.TP().charge()!=0 && T.TP().pt() > 2 && T.MatchedA()")
  hEffic->Divide(hNum, hDen,1,1,"B")

Pull d_0

- D_0 Pull Distributions $\frac{d_0^{Reco} - d_0^{Sim}}{\sigma_{d_0}}$

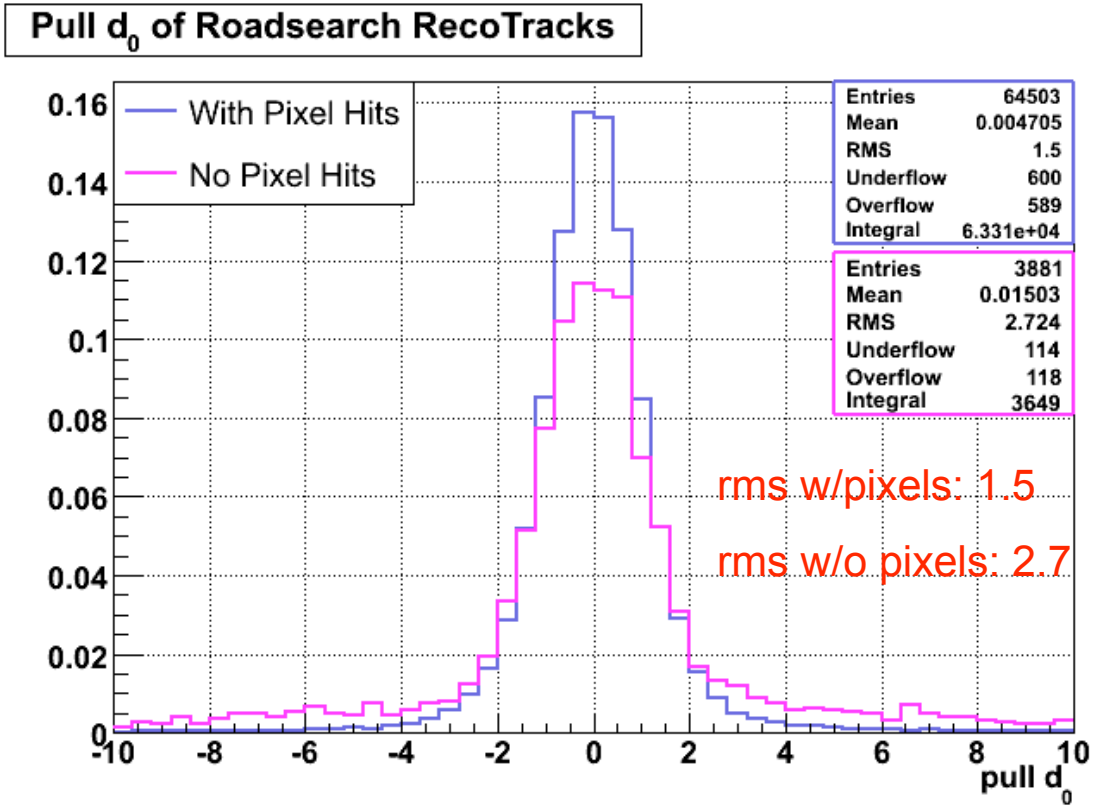


- 20% difference in rms for RS w.r.t CTF.
  - RS doesn't require pixels in its seeding.
  - The errors may not be properly accounted for the material between the interaction point and the 1st layer in the TEC.

- **CTF** Track d_0:
  Draw("(T.rA_d0()-T.s_d0())/T.RTA().d0Error()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedA()")
- **RoadSearch** Track d_0:
  Draw("(T.rB_d0()-T.s_d0())/T.RTB().d0Error()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedB()","sames")

Pull d_0 with/without pixels

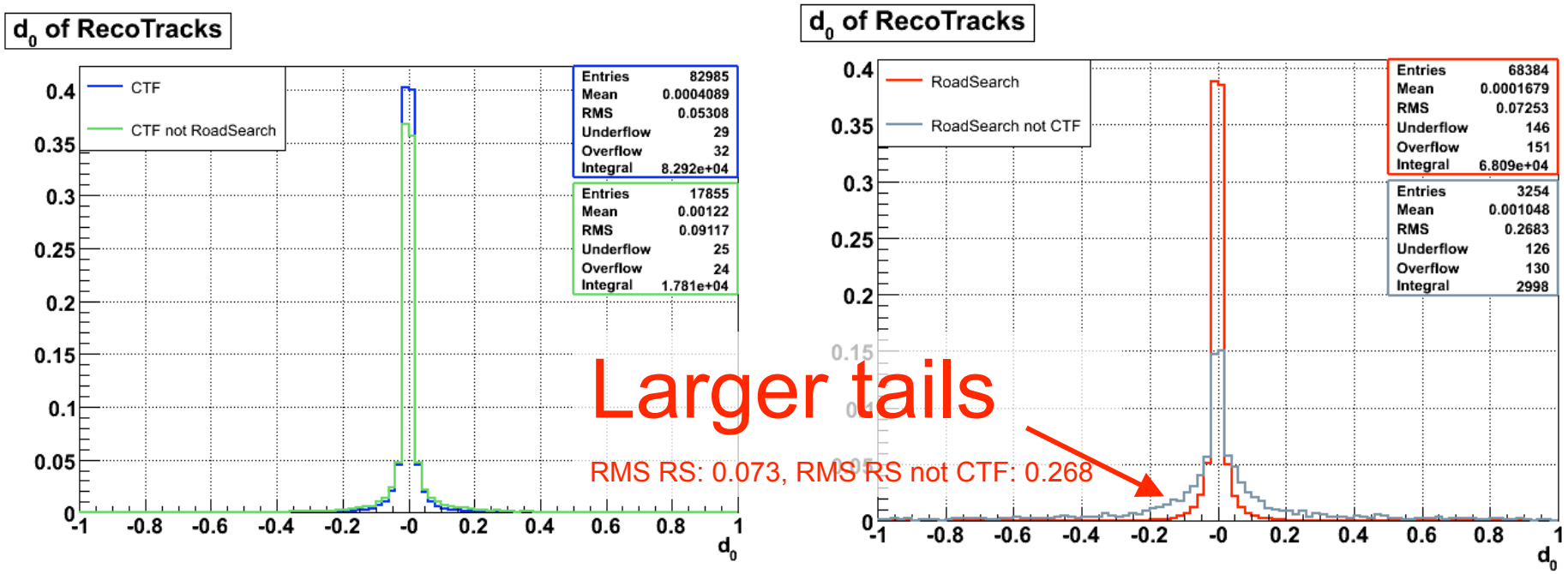
- The large tails on the pull d0
  - Miscalculation of material amount when pixel layers are missed



- **RoadSearch** Track d_0 (no Pixels):
  - Draw("(T.rB_d0()-T.s_d0())/T.RTB().d0Error()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedB() && T.hitPattern().numberOfValidPixelHits()==0")
- **RoadSearch** Track d_0 (with Pixels):
  - Draw("(T.rB_d0()-T.s_d0())/T.RTB().d0Error()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedB() && T.hitPattern().numberOfValidPixelHits()>0)

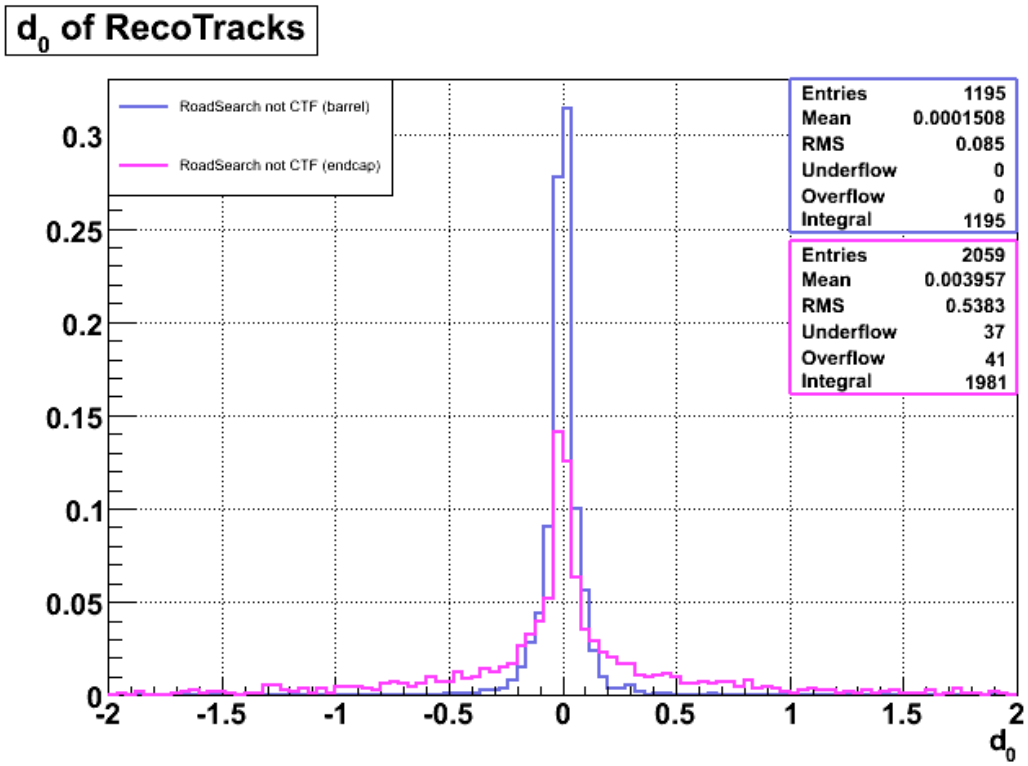
One Algo but NOT the Other

- d_0 for CTF reconstructs a charged particle that RS does not? (left)
- Conversely, RS reconstruct a charged particle that CTF does not? (right)
- Right plot was most interesting pull distribution for the 5 track parameters (q/p, λ, φ, d_0, d_z)



- **RoadSearch** Track d_0 (all):
  - **Draw("T.rB_d0()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedB()")**
- **RoadSearch** Track d_0 (not matched to a CTF track):
  - **Draw("T.rB_d0()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedBnotA()")**

Barrel versus Endcap



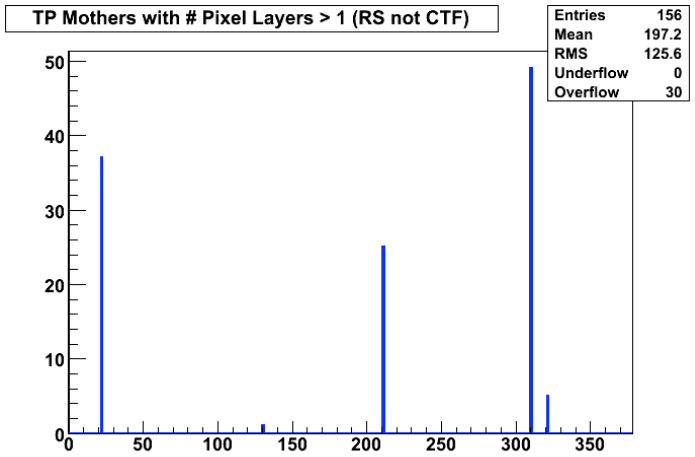
- Look at barrel and endcap separately.
- RS has a looser endcap seeding requirement (2mm barrel vs 12mm endcap)
  - Lower occupancy of hits in the barrel?
- CTF is symmetric between the barrel and endcap.

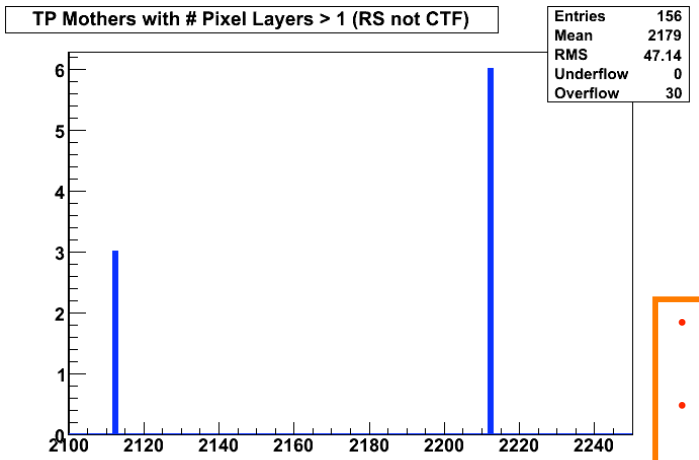
- **RoadSearch** Track d_0 (Barrel):
  - Draw("T.rB_d0()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedBnotA() && fabs(T.RTB().eta())<0.9")
- **RoadSearch** Track d_0 (Encap):
  - Draw("T.rB_d0()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedBnotA() && fabs(T.RTB().eta())>=0.9 ")

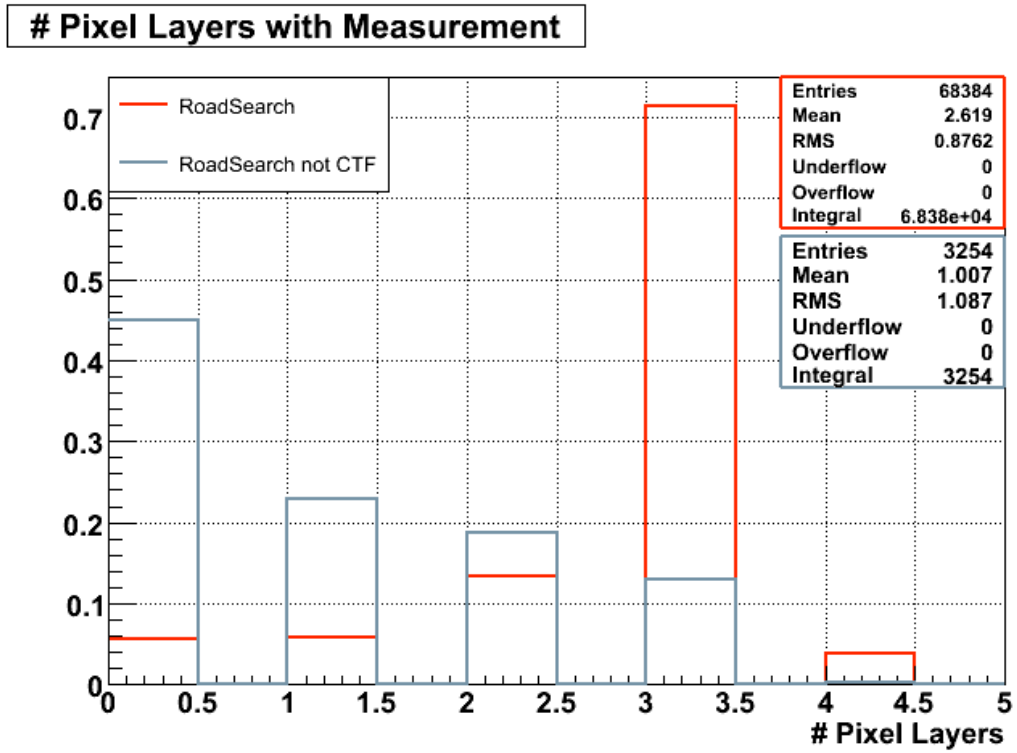
What kind of Tracks will have this larger d_0?

- The are tracks from decay products from long lived particles (e.g. K_S) or converted γ's.
- They miss the pixel layers but hit the first few layers of strip detectors.

22 = γ, 130 = K_L, 211 = $\pi^{\pm}$, 310 = K_S, 321 = $K^{\pm}$, 2112 = n, 2212 = p



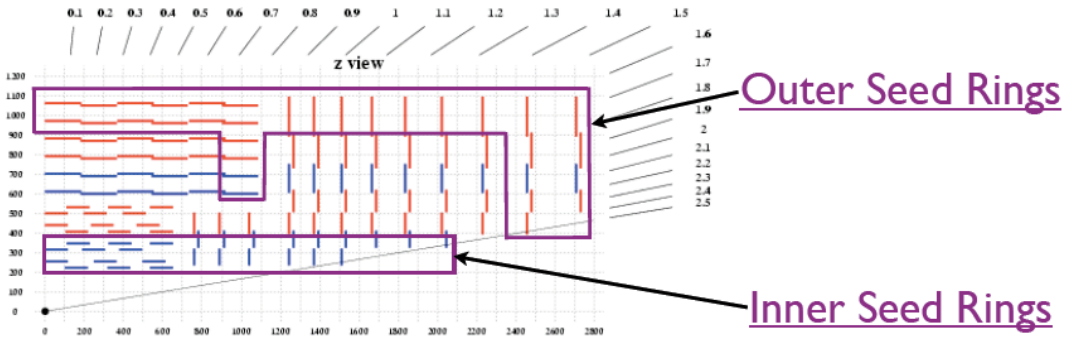


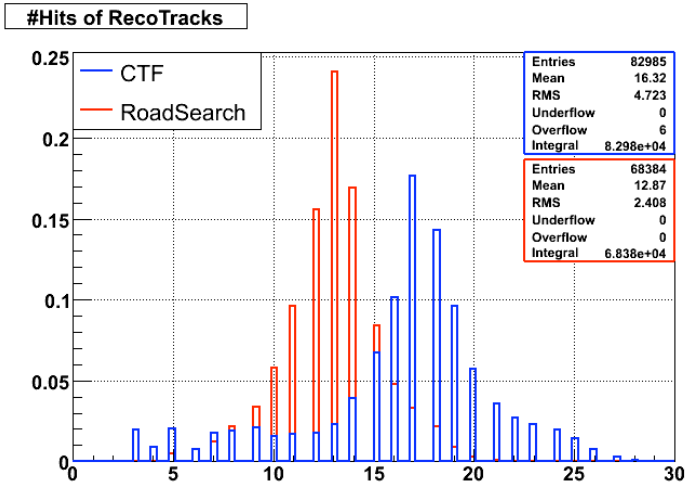


- **Pixel Layers with measuremnt**
  - Draw("T. RTB().hitpattern().pixelLayersWithMeasurment()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedBnotA()")
- **TP mothers**
  - Draw("T.TPmother()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedBnotA()")

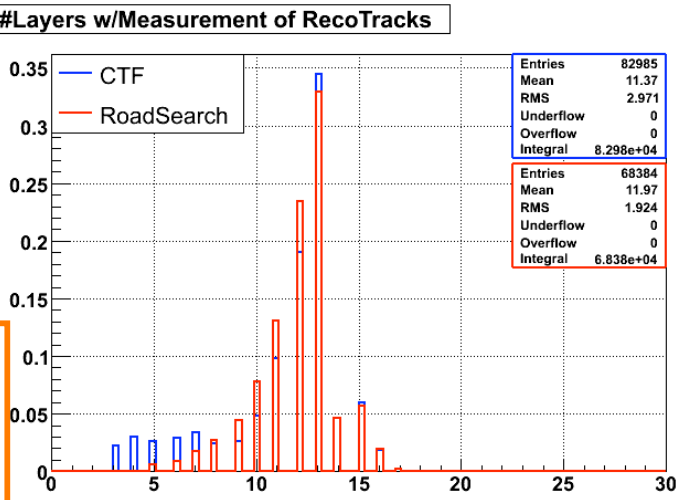
Where could RS be losing tracks that are picked up by CTF?

- RS requires a inner and outer seed ring.





- Conversion in the middle of the tracker
  no seed is created for the RoadSearch



- **#hits**
  - **Draw("T.RTA().found()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedA()")**
  - **Draw("T.RTB().found()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedB()")**
- **#layers**
  - **Draw("T.RTA().hittPattern().trackerLayersWithMeasurement()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedA()")**
  - **Draw("T.RTB(). hittPattern().trackerLayersWithMeasurement()", "T.TP().charge()!=0 && T,PT().pt()>2 && T.MatchedB()")**

Conclusion

- Previous tools only allowed for a statistical comparison between recoTracks produced from different algorithms/configurations.
- This tool give you a root tree so that plots can be made on the fly during a FWLite ROOT session.
- Also, allows one to do tracking studies and direct comparisons on a track by track basis.