Photon-finding

Photon-finding in a PFA

Introduction

One of the most basic and conceptually simple parts of a PFA is finding electromagnetic showers from photons and electrons. Electromagnetic showers are better understood, better contained, and behave more consistently than hadronic showers. So often we aim to identify and isolate them early in a PFA.

Photon showers have a distinctive cigar-like shape, with a needle-like core and a less dense halo of energy. They also occur early in the EM calorimeter (typically starting in the first few layers). However, the same is more or less true of MIPs – in particular, soft photons can look very MIP-like. So PFAs will either need to distinguish charged and neutral clusters (e.g. by track association), use other cluster information to positively identify photons (e.g. H-matrix longitudinal shower ID), or both.

We do not discuss pi0 reconstruction here.

Worked examples

Example without strong photon ID

Here is a code snippet showing a photon-finder in action:

```
// Some of the import statements:
import org.lcsim.recon.cluster.mipfinder.TrackClusterDriver;
import org.lcsim.recon.cluster.util.ClusterFirstLayerDecision;
import org.lcsim.recon.pfa.identifier.MIPChargedParticleMaker;
import org.lcsim.recon.pfa.identifier.SimpleTrackMIPClusterMatcher;
import org.lcsim.recon.cluster.mst.MSTPhotonFinderDriver;
import org.lcsim.recon.pfa.identifier.SimpleNeutralParticleMaker;

// Find MIP candidates in ECAL, starting near the front:
TrackClusterDriver ecalFrontSideMIPs = new TrackClusterDriver("input hit map ecal", "front-side mips ecal",
"hit map ecal without front-side mips");
ecalFrontSideMIPs.filterOutputClusters(new ClusterFirstLayerDecision(4));
add(ecalFrontSideMIPs);

// Check if they match a track (ugly... don't need to make the particles!)
MIPChargedParticleMaker mipHadID = new MIPChargedParticleMaker();
SimpleTrackMIPClusterMatcher mipMatch = new SimpleTrackMIPClusterMatcher();
add(mipMatch);
mipHadID.setTrackMatcher(mipMatch);
mipHadID.setInputTrackList(trackList);
mipHadID.setOutputTrackList("tracks left over from front-side mips");
mipHadID.setInputMIPList("front-side mips ecal");
mipHadID.setOutputMIPList("mips minus front-side mips ecal");
mipHadID.setOutputParticleList("front-side mip particles");
add(mipHadID);

// Isolate the genuine (charged) MIPs from the rest
ClusterListFilterDriver filterRemoveChargedMIPs = new ClusterListFilterDriver();
VetoClustersFromParticles vetoChargedMIPs = new VetoClustersFromParticles("front-side mip particles"); // veto
MIPs
filterRemoveChargedMIPs.setInputDecision(vetoChargedMIPs);
filterRemoveChargedMIPs.setInputList("front-side mips ecal");
filterRemoveChargedMIPs.setOutputList("neutral front-side mips ecal");
add(vetoChargedMIPs);
add(filterRemoveChargedMIPs);

// Merge the non-charged "MIP" hits back in
ClusterListToHitMapDriver convertFakeMIPsToHitMap = new ClusterListToHitMapDriver("neutral front-side mips
ecal", "hit map ecal of neutral front-side mips");
add(convertFakeMIPsToHitMap);
HitMapAddDriver addFakeMIPHitsBack = new HitMapAddDriver();
addFakeMIPHitsBack.addInputHitMap("hit map ecal of neutral front-side mips");
addFakeMIPHitsBack.addInputHitMap("hit map ecal without front-side mips");
addFakeMIPHitsBack.setOutputHitMap("hit map ecal without charged front-side mips");
add(addFakeMIPHitsBack);

// Find photons
MSTPhotonFinderDriver photonFinder = new MSTPhotonFinderDriver();
photonFinder.setCoreThreshold(7.5);
photonFinder.setFragmentThreshold(7.5);
photonFinder.setLayerProximityThreshold(2);
photonFinder.setCoreSizeMinimum(10);
photonFinder.setFragmentSizeMaximum(6);
photonFinder.setInputHitMap("hit map ecal without charged front-side mips");
photonFinder.setOutputHitMap("hit map ecal without photons or charged front-side mips");
photonFinder.setOutputClusterList("photon clusters");
add(photonFinder);

// Make into particles
SimpleNeutralParticleMaker myPhotonIdentifier = new SimpleNeutralParticleMaker(22); // 22 = photon
myPhotonIdentifier.setInputClusterList("photon clusters");
myPhotonIdentifier.setOutputParticleList("photon particles");
add(myPhotonIdentifier);
```

Related Drivers

(add some)