A JAS Package for LCD Muon Reconstruction

Richard Markeloff
Northern Illinois Center for Accelerator and Detector Development
Northern Illinois University
DeKalb, Illinois
July 31, 2002

Abstract

A new package has been developed to provide classes for identifying and analyzing muons in simulated LCD events using Java Analysis Studio. The classes in the package are described. Sample code for performing a JAS LCD muon analysis is presented.

1. Introduction

The Java Analysis Studio (JAS) is a software tool, implemented in Java, for graphically analyzing high energy physics data [1]. It can run as either a standalone application or as client for analyzing data stored on a remote server. JAS is currently being used to analyze simulated data for the proposed Linear Collider Detector (LCD) [2].

The JAS packages available for LCD studies include classes to accomplish tasks such as track reconstruction and jet finding, but do not contain classes specifically designed for muon identification. This note describes a new package, intended for inclusion in the hep.lcd package hierarchy [3] as hep.lcd.recon.muon, which provides classes for reconstructing muons in simulated LCD events. The goal of this work is to provide the LCD community of JAS users with a standard set of robust, easy-to-use tools for muon analyses.

2. LCD Muon Identification

Various techniques for LCD muon detection are under consideration, ranging from instrumenting the solenoid return yoke to create a conventional muon system to doing away with a dedicated muon system entirely and relying on a "digital" hadronic calorimeter for identifying muons. In the latter case, the calorimeter would have fine enough spatial resolution to see the MIP trace from a single muon and would have to be thick enough to reduce hadronic punch-through background to an acceptable level at the outer layers. The muon reconstruction classes have been designed to accommodate variations in detector design with minimal coding effort.

An LCD muon candidate object contains detector data consistent with what would be produced by a muon originating from the event vertex. These data are grouped into three segments: a reconstructed track in the central tracker, a set of hits in the hadron calorimeter associated with the track, and a set of hits in the muon system that are also associated with the track. The number of hits in the muon system, and perhaps also the number of hits in the calorimeter, are required to exceed some minimum value in order for the muon candidate to be considered valid. The decomposition of a muon candidate
into three segments is not dependent on specific details of the detector design. For example, if a digital calorimeter were used for muon identification, then the "muon system" would be the outermost layers of the calorimeter.

To identify muons, it is sufficient to require that a track match with hits in the muon system. However, requiring a muon candidate to also pass a cut on the number of hits in the hadron calorimeter reduces the background from junk tracks that happen to point to muon system hits. Even if no cut is made on the calorimeter segment, the data might be useful for studying calorimeter MIP response.

3. Muon Reconstruction Classes

The hep.lcd.recon.muon package contains the classes MuonReco, SegmentFinder, BarrelCalSegmentFinder, MuonCalSegmentFinder, MuonCandidate, and MuonList. The class diagram is shown in Figure 1. Descriptions of the classes are provided in the following subsections.

3.1 MuonReco

This class extends the AbstractProcessor class, and muon finding is done in the process() method.

The first actions taken within the process() method are to instantiate a MuonList object to hold the muon candidates found in the event, and to instantiate segment finder objects for the hadron calorimeter and muon system. These are instances of classes derived from the abstract class SegmentFinder, and their purpose is to do the actual work of matching detector hits with a given track.

Next, a loop is done over the tracks in the event. For each track, hadron calorimeter and muon system SegmentFinder findSegment() methods are invoked. Then getNHits() methods, from the CalorimeterHits interface, are used to obtain the number of hits in the calorimeter and muon system segments. If the number of hits passes the cuts, then a MuonCandidate object is created and added to the MuonList object. Finally, when the loop over tracks is completed, the MuonList close() method is invoked. This sorts the list of muon candidates by $p_T$. The MuonList object is then added to the LCDEvent object via its put() method.

The constructor for the MuonReco class takes four parameters: the cut on the number of calorimeter hits, the cut on the number of muon hits, the name of the hadron calorimeter detector component, and the name of the muon system detector component. There is another version of the constructor that accepts the parameters representing the cuts but assumes the detector components are HAD_Barrel and MU_Barrel. There is also a default constructor that not only assumes that HAD_Barrel and MU_Barrel are the detector components, it also sets the cuts to the arbitrary values 0 and 12 respectively.

3.2 SegmentFinder
The abstract class SegmentFinder provides a means for the MuonReco class to communicate with segment finder objects. The SegmentFinder class has two public methods: findSegment() and reset(). The latter method is invoked after findSegment() has processed a track, and its purpose is to reinitialize internal data to ready the SegmentFinder object for the next track.

Ideally, the MuonReco class should be coupled to segment finding objects only through the methods of the SegmentFinder class and other abstract interfaces such as CalorimeterHits. This frees the MuonReco class from the need to have any knowledge of the internal implementation of the segment finders. It also facilitates updating MuonReco as detector designs evolve.

### 3.3 BarrelCalSegmentFinder

This class extends SegmentFinder and implements the CalorimeterHits interface. It can find muon segments in any calorimeter within the solenoid that is represented by a class derived from BarrelCalorimeter. The name of the specific detector component to find segments in is passed to the constructor as a parameter.

Within the findSegment() method, the $\phi$ and $\theta$ indices of the cell that the track passes through are determined for each layer of the detector. BarrelCalSegmentFinder uses the services of the HelicalSwimmer class to propagate a track through the calorimeter. Calorimeter hits are added to the segment if the absolute value of the difference between their cell indices and the cell indices of the track are within certain ranges. The default values of the $\phi$ and $\theta$ nearest neighbor cuts are 3 and 1 respectively, and users can change them via the public methods setPhiNNCut() and setThetaNNCut().

BarrelCalSegmentFinder inherits the methods getHits() and getNHits() from the CalorimeterHits interface. MuonReco uses the getNHits() method to check to see if the muon candidate passes the cut on the number of calorimeter hits.

### 3.4 MuonCalSegmentFinder

Reference [2] describes a detector design that incorporates RPCs with strip electrode readout for muon detection, and classes such as MUStripSystem exist in the hep.lcd package hierarchy. However, the current version of the LCD simulation has for the muon system another barrel calorimeter outside the solenoid. MuonCalSegmentFinder extends BarrelCalSegmentFinder, and finds muon segments in this muon calorimeter. The principal difference between this class and its immediate ancestor is how tracks are extrapolated. MuonCalSegmentFinder uses the methods of a nested class TrackExtrapolator to extend tracks through the muon calorimeter. TrackExtrapolator uses HelicalSwimmer to find the point at which a track intersects the inner surface of the solenoid, and then extends it as a straight line. This assumes that the magnetic field is zero beyond the inner radius of the solenoid, which is probably not true for the simulation and certainly would not be true for a real detector. But it serves as a first approximation.

Perhaps because of the uncertainty regarding the value of the field within the return yoke used in the simulation, and because tracks at the muon calorimeter have a longer lever
arm than tracks at the hadron calorimeter, the allowed range for associating nearest
neighbor cells with the track is greater for MuonCalSegmentFinder than
BarrelCalSegmentFinder. The $\phi$ and $\theta$ nearest neighbor cut default values are set to 4 and
2 respectively.

3.5 MuonCandidate

This class holds all the relevant information about a muon candidate. Its instance fields
include:
- The muon candidate's Track object
- The number of calorimeter hits
- The number of layers in the calorimeter that the track passes through
- An Enumeration containing the individual calorimeter hits
- The number of muon system hits
- The number of layers in the muon system that the track passes through
- An Enumeration containing the individual muon system hits
- The track $p_T$

Public methods are provided to access these data.

3.6 MuonList

This class holds the set of valid muon candidates found in an event. Internally, the
MuonCandidate objects are stored in a Vector. The public method close() trims the
Vector to size and sorts the objects in order of descending $p_T$. Other public methods add a
MuonCandidate to the list, return the number of MuonCandidates in the list, return the
MuonCandidate stored at a particular index in the list, and return an Enumeration of all
MuonCandidate objects in the list.

4. Using the MuonReco Class

Sample code for using the MuonReco class is presented in Appendix A. Note that
TrackReco must be added to the Driver before MuonReco to ensure that track
reconstruction is done. The sample code shows how to get the MuonList from the
LCDEvent object, and loop over muon candidates to perform whatever muon analysis is
desired.

5. Suggestions for Further Development

Although the muon reconstruction package described in this note is useful in its current
form, there is always room for improvement and the need for further studies. In this case,
these are:
- The optimal cuts on the number of hits, and the optimal nearest neighbor cuts
  for the SegmentFinders, needs to ascertained for both the large and small LCD
designs.
- The efficiency for finding muons, as a function of $\theta$ and muon $p_T$, needs to be
determined.
• The amount of fake muon background found in various data sets needs to be measured.
• A better way to extrapolate tracks that more closely corresponds to the model of the magnetic field used in the simulation should be developed, and inserted into the extrapolate() method of the TrackExtrapolator class in MuonCalSegmentFinder.java.

6. Conclusions

A JAS package for LCD muon reconstruction has been developed. The classes in this package make it possible for LCD JAS users to perform muon analyses quickly and easily. This package should be made publicly available at the earliest possible date. Effort should be devoted to enhancing and extending the classes in this package, beginning with the suggested list of work items in the previous section.

7. References

Figure 1: Class diagram for the proposed hep.lcd.recon.muon package
Appendix A. Sample Code for Muon Analyses

```java
import hep.lcd.recon.muon.*;
import hep.lcd.recon.tracking.*;
import hep.lcd.util.driver.*;
import hep.lcd.event.*;
import hep.analysis.*;
import java.util.*;

final public class LCDMuonDriver extends Driver
{
    public LCDMuonDriver()
    {
        add(new TrackReco());
        // First parameter for MuonReco constructor is the minimum
        // number of hadron calorimeter hits required for a muon
        // candidate, the second parameter is the minimum number of
        // muon system hits
        add(new MuonReco(0,12));
        add(new MuonAnalyzer());
    }
}

class MuonAnalyzer extends AbstractProcessor
{
    public void process(LCDEvent event)
    {
        // Get a list of the muon candidates found in this event
        MuonList muonList = (MuonList)event.get("MuonList");
        // Loop over the muon candidates
        for (Enumeration e = muonList.getMuonCandidates();
             e.hasMoreElements();)
        {
            MuonCandidate mu = (MuonCandidate)e.nextElement();
            // Get the transverse momentum of this muon candidate
            double pT = mu.getPT();
            // Get the number of hits found in the muon system
            int nMuHits = mu.getNMuonSystemHits();
            // Get the number of layers the track passed through
            int nLayersHit = mu.getNMuonSystemLayersHit();
            // Get the number of hits found in the hadron calorimeter
            int nCalHits = mu.getNCalorimeterHits();
            // Fill some histograms
            pT_hist.fill(pT);
            nMuHits_hist.fill(nMuHits);
            HitsVsLayers_hist.fill(nMuHits, nLayersHit);
            MuHitsVsCalHits_hist.fill(nMuHits, nCalHits);
            // Get the track associated with this muon candidate
            Track t = mu.getTrack();
            // Separate real muons from background and fill histograms
            // for each
        }
    }
}
```
if (t.fitSuccess() &&
    Math.abs(t.getMCParticle().getType().getPDGID()) == 13)
{
    // This is a muon
    pTreal_hist.fill(pT);
    nMuHitsReal_hist.fill(nMuHits);
    HitsVsLayersReal_hist.fill(nMuHits, nLayersHit);
    MuHitsVsCalHitsReal_hist.fill(nMuHits, nCalHits);
}
else
{
    // This is background
    pTfake_hist.fill(pT);
    nMuHitsFake_hist.fill(nMuHits);
    HitsVsLayersFake_hist.fill(nMuHits, nLayersHit);
    MuHitsVsCalHitsFake_hist.fill(nMuHits, nCalHits);
}

// Book some histograms
private Histogram1D pT_hist = new Histogram1D("PT (All Muon Candidates)" );
private Histogram1D pTreal_hist = new Histogram1D("PT (Real Muons)" );
private Histogram1D pTfake_hist = new Histogram1D("PT (Background)" );
private Histogram1D nMuHits_hist = new Histogram1D("Number of muon hits (All Muon Candidates)" );
private Histogram1D nMuHitsReal_hist = new Histogram1D("Number of muon hits (Real Muons)" );
private Histogram1D nMuHitsFake_hist = new Histogram1D("Number of muon hits (Background)" );
private Histogram2D HitsVsLayers_hist = new Histogram2D("Muon hits found vs. muon layers hit (All Muon Candidates)" );
private Histogram2D HitsVsLayersReal_hist = new Histogram2D("Muon hits found vs. muon layers hit (Real Muons)" );
private Histogram2D HitsVsLayersFake_hist = new Histogram2D("Muon hits found vs. muon layers hit (Background)" );
private Histogram2D MuHitsVsCalHits_hist = new Histogram2D("Muon hits vs. had cal hits (All Muon Candidates)" );
private Histogram2D MuHitsVsCalHitsReal_hist = new Histogram2D("Muon hits vs. had cal hits (Real Muons)" );
private Histogram2D MuHitsVsCalHitsFake_hist = new Histogram2D("Muon hits vs. had cal hits (Background)" );