Tools for MC truth matching

Dom Barker

University of Sheffield

December 18, 2018

The University Of Sheffield.

Introduction

- Both SBND and ICARUS use RecoUtils, written by Dom Brailsford. The code is a bunch of c++ higher level functions in the RecoUtils namespace used to link events back to Monte Carlo Information.
- The functions are algorithms only. You don't need to store the output of the functions in the art::event to make use of them.
- The functions usually return upper level truth information.
- A quick example:

  `int TrueParticleIDFromTotalRecoHits(const std::vector<art::Ptr<recob::Hit> > hits).`

  This returns the Geant4 ID which contributes the most to the vector of hits. (More detail to follow)

Introduction

- In the past couple of months we have been adding to the list of functions and added a further set of functions called ShowerUtils. We would like to move the code into larreco, so there is one source code for both detectors.
- Also we see the functions as a useful way to ensure consistency between developers and prevent duplication.
- We also view it as a valuable piece of source code for beginners to get acquainted this side of LArSoft.

RecoUtils

```
namespace RecoUtils{
  int TrueParticleID(const art::Ptr<recob::Hit>& hit); //Returns the geant4 ID which contributes t\
he most to a single reco hit.  The matching method looks for true particle which deposits the most\
 true energy in the reco hit
  int TrueParticleIDFromTotalTrueEnergy(const std::vector<art::Ptr<recob::Hit> >& hits); //Returns\
 the geant4 ID which contributes the most to the vector of hits.  The matching method looks for wh\
ich true particle deposits the most true energy in the reco hits
  int TrueParticleIDFromTotalRecoCharge(const std::vector<art::Ptr<recob::Hit> >& hits);  //Return\
s the geant4 ID which contributes the most to the vector of hits.  The matching method looks for w\
hich true particle contributes the most reconstructed charge to the hit selection (the reco charge\
 of each hit is correlated with each maximally contributing true particle and summed)
  int TrueParticleIDFromTotalRecoHits(const std::vector<art::Ptr<recob::Hit> >& hits);  //Returns \
the geant4 ID which contributes the most to the vector of hits.  The matching method looks for whi\
ch true particle maximally contributes to the most reco hits
  bool IsInsideTPC(TVector3 position, double distance_buffer); //Checks if a position is within an\
y of the TPCs in the geometry (user can define some distance buffer from the TPC walls)

  int NumberofHitsFromTrack(int TrackID, const std::vector<art::Ptr<recob::Hit> >& hits); //Return\
s the number of hits in the vector that are associated to the MC track.

  std::map<geo::PlaneID,int> NumberofPlaneHitsFromTrack(int TrackID, const std::vector<art::Ptr<re\
cob::Hit> >& hits); //Returns the number of hits in the vector that are ssociated to the MC trakc \
split into planes.

  std::map<int,std::map<geo::PlaneID,int> > NumberofPlaneHitsPerTrack(const std::vector<art::Ptr<r\
ecob::Hit> >& hits); //Returns a map of all the number of hits and the respetive track id they are\
 associated to.
```

RecoUtils

```
int RecoUtils::TrueParticleIDFromTotalRecoHits(const std::vector<art::Ptr<recob::Hit> >& hits) {
  // Make a map of the tracks which are associated with this object and the number of hits they ar\
e the primary contributor to
  std::map<int,int> trackMap;
  for (std::vector<art::Ptr<recob::Hit> >::const_iterator hitIt = hits.begin(); hitIt != hits.end(\
); ++hitIt) {
    art::Ptr<recob::Hit> hit = *hitIt;
    int trackID = TrueParticleID(hit);
    trackMap[trackID]++;
  }

  // Pick the track which is the primary contributor to the most hits as the 'true track'
  int objectTrack = -99999;
  int highestCount = -1;
  for (std::map<int,int>::iterator trackIt = trackMap.begin(); trackIt != trackMap.end(); ++trackI\
t) {
    if (trackIt->second > highestCount) {
      highestCount = trackIt->second;
      objectTrack  = trackIt->first;
    }
  }
  return objectTrack;
}
```

RecoUtils

```
#include "RecoUtils.h"

int RecoUtils::TrueParticleID(const art::Ptr<recob::Hit>& hit) {
  double particleEnergy = 0;
  int likelyTrackID = 0;
  art::ServiceHandle<cheat::BackTrackerService> bt_serv;
  std::vector<sim::TrackIDE> trackIDs = bt_serv->HitToTrackIDEs(hit);
  for (unsigned int idIt = 0; idIt < trackIDs.size(); ++idIt) {
    if (trackIDs.at(idIt).energy > particleEnergy) {
      particleEnergy = trackIDs.at(idIt).energy;
      likelyTrackID = trackIDs.at(idIt).trackID;
    }
  }
  return likelyTrackID;
}
```

ShowerUtils

```
namespace ShowerUtils{

  std::pair<int,double> TrueParticleIDFromTrueChain(std::map<int,std::vector<int> >& ShowersMother\
s,const std::vector<art::Ptr<recob::Hit> >& hits, int planeid);
  std::map<geo::PlaneID,int> NumberofWiresHitByShower(std::vector<int> &TrackIDs, const std::vecto\
r<art::Ptr<recob::Hit> >& hits);

}
```

ShowerUtils

```
std::map<geo::PlaneID,int> ShowerUtils::NumberofWiresHitByShower(std::vector<int> &TrackIDs, const std::vector<art::Ptr<recob::Hit> >& hits){

  art::ServiceHandle<cheat::BackTrackerService> bt_serv;

  std::vector<geo::WireID> WiresUsed;
  std::map<geo::PlaneID,int> HitWirePlaneMap;

  for (std::vector<art::Ptr<recob::Hit> >::const_iterator hitIt = hits.begin(); hitIt != hits.end(); ++hitIt) {

    art::Ptr<recob::Hit> hit = *hitIt;

    //Find the wire and  plane id
    geo::WireID wireid = hit->WireID();
    geo::PlaneID  PlaneID = wireid.planeID();

    //Check to see if the wire is already been continued.
    if(std::find(WiresUsed.begin(),WiresUsed.end(),wireid) != WiresUsed.end()){continue;}

    std::vector<sim::TrackIDE> trackIDs = bt_serv->HitToTrackIDEs(hit);
    for (unsigned int idIt = 0; idIt < trackIDs.size(); ++idIt) {
      if(std::find(TrackIDs.begin(), TrackIDs.end(),TMath::Abs(trackIDs.at(idIt).trackID)) != TrackIDs.end()){
        WiresUsed.push_back(wireid);
        ++HitWirePlaneMap[PlaneID];
        break;
      }
    }
  }
  return HitWirePlaneMap;
}
```

Conclusions

- We wish to port RecoUtils and ShowerUtils into a subdirectory called MCRecoUtils in larreco/RecoAlg.
- This will centralise the code for SBND and ICARUS.
- It is detector agnostic and can be used by any liquid argon experiment that want to.
- We think its a good idea as it also helps to prevent duplication and maintain consistency between higher level matching between reconstruction and truth.
- It is also easy to add further functions that people might find belong with this code.
- Are there any questions/feedback?