A New Track Interface

The old Track

The current Track interface suffers from the following problems

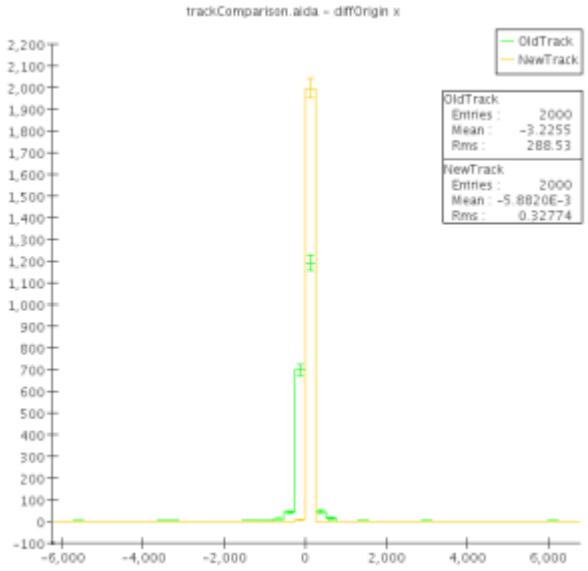
- Inconsistent accessors: A complete representation is given by the reference point, the POCA on the track to the reference point and the momentum at that point. However, org.lcsim.event.Track only provides reference point and momentum.
- The Track parameters are only accessible by index

What is more important is that the only implementation of Track that exists in the main trunk, ReconTrack, adds the following problems

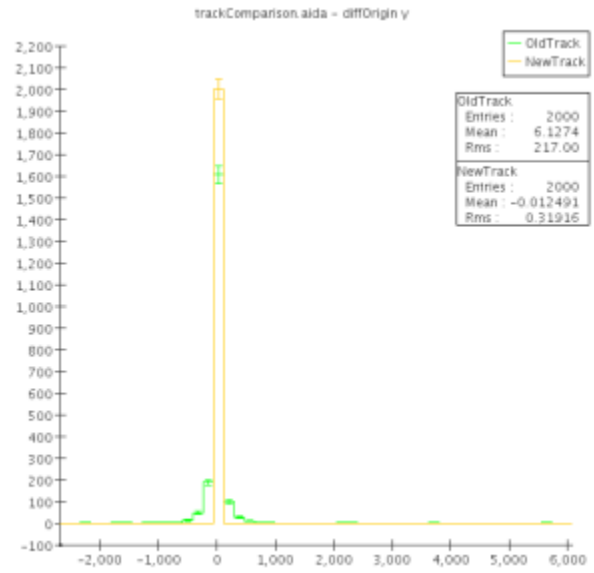
- The Track parameters are always evaluated at the origin
- the reference point of the Track is therefore (0, 0, 0), but getReferencePoint() returns the same point as getOrigin of the particle that the track was instantiated with.
- The Parameters of the Track are never swum to the reference point and therefore the track parameters are therefore wrong by design for all particles that don't come from the origin

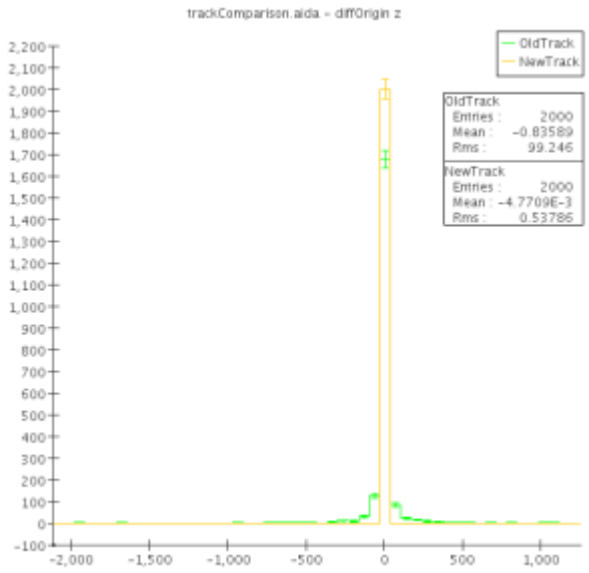
Issues

This has the result that you cannot reliably swim the Track to a given point. Because the reference point for the track is 0, the Swimmer claims that the tracklength to the origin is 0. However, since the Track has been instantiated from a particle that did not originate at the IP, the track parameters do not actually correspond to the origin.
In other words: The POCA on the track to the IP results in the wrong point.

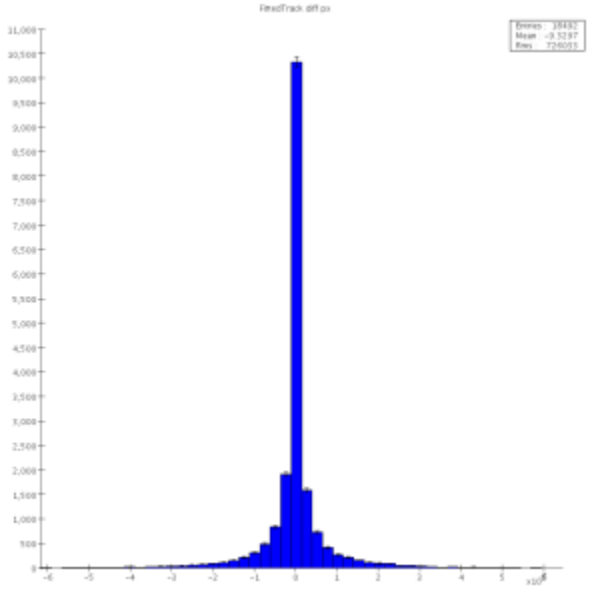


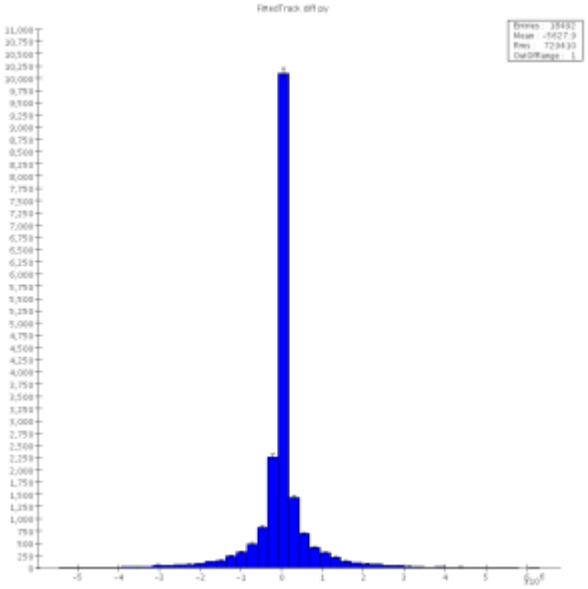
The result of this can be seen in ,

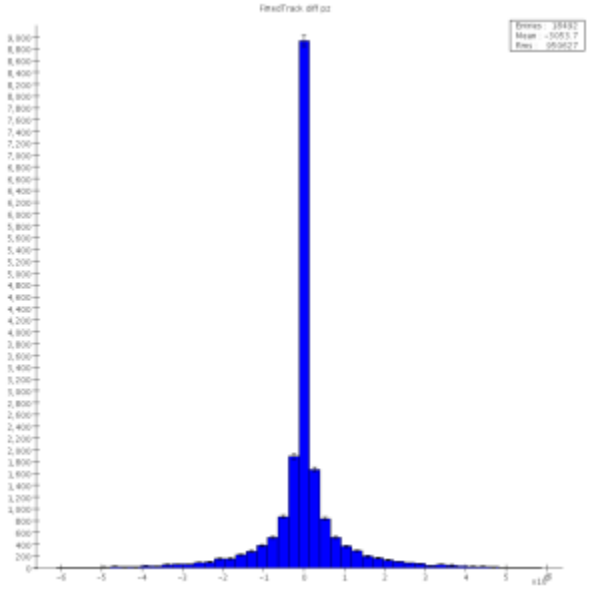




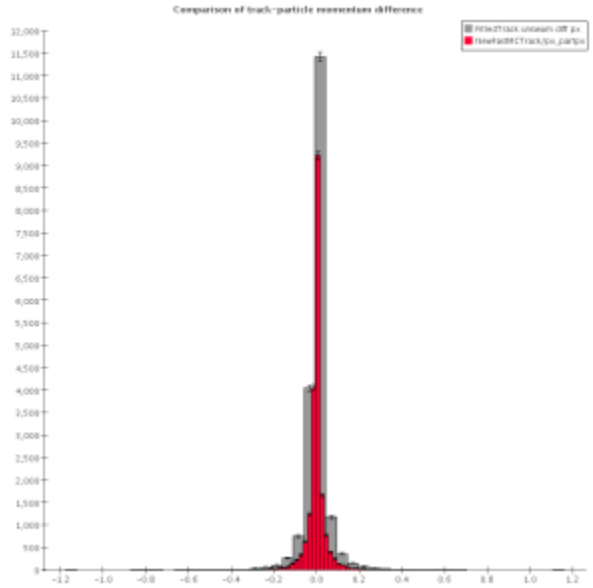
What is more: The following is an example of how to instantiate Tracks from Particles.
The approach is simple: Get a Particle, make a Track from it and compare Track momentum and particle momentum. This is kind of the whole idea of a track.
Naive approach 1: Swim the Track to the Origin of the Particle and compare momenta.

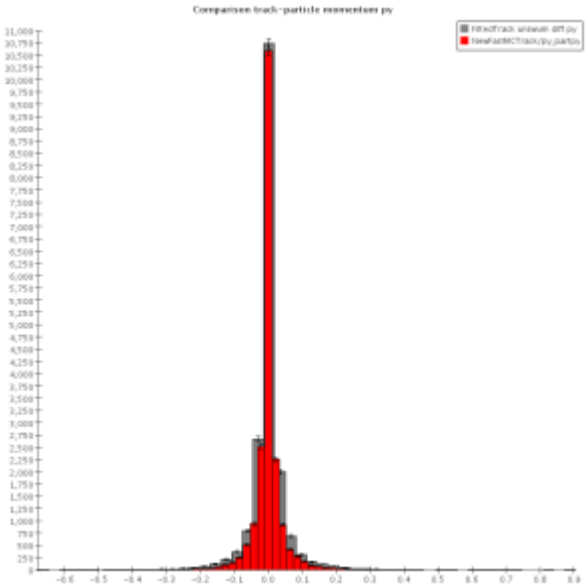


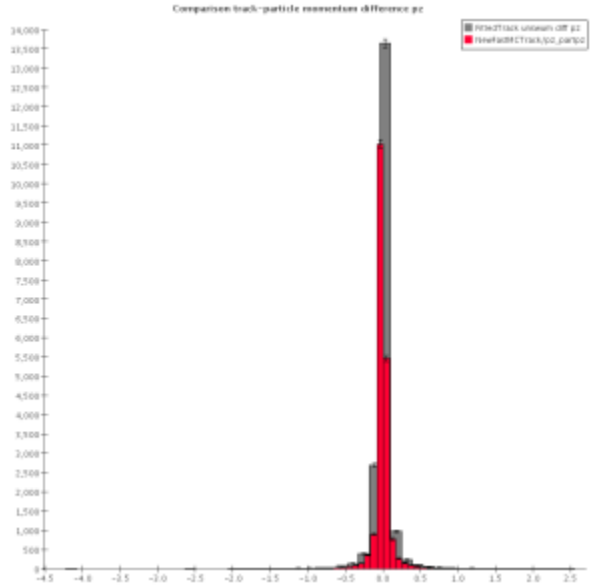




OK, that looks pretty bad. Now, remember that the old Track has the wrong reference point, and that it is instantiated at the origin of the MCParticle. Let's compare this then to a new fast mc track that is properly smeared and swum to the origin of the particle.

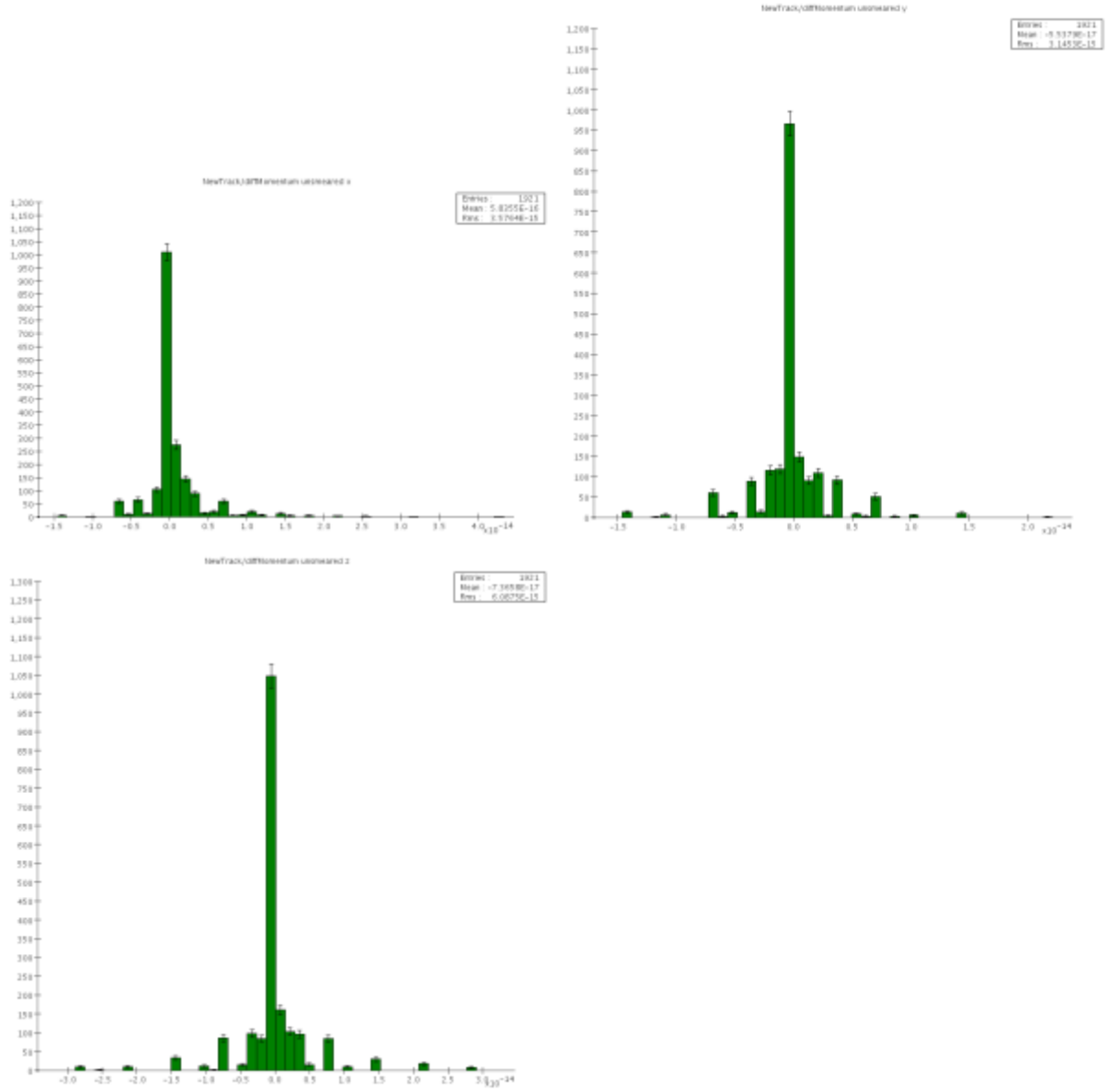






Aha ! So as long as we use Tracks from the IP (which admittedly *is* the majority of the tracks) or know exactly where in the detector the Track was created from the particle (...) this seems to do OK.

Just for kicks: The new interface can of course turn off smearing. So let's make some unsmeared tracks and swim them.



A new Track

Interface Design goals:

- The Track should make sense independent of the detector, it should be self-contained
- You should be able to reliably swim the parameters to a given point
- You should be able to reliably translate the track parameters to space and momentum and back

The interface to the FastMC should be flexible enough to allow switching the smearing on/off.

For this to work many of the existing classes had to be refactored to use common methods to ensure consistency. Furthermore the current HelixSwimmer has to rely on the implementation of Track being broken in the same way as ReconTrack...

Code Examples

If you want the new Tracks in the event, it's as easy as

```
import org.lcsim.contrib.JanStrube.tracking.NewMCFastTrackDriver;
...
loop.add(NewMCFastTrackDriver());
```

For more control, you will need to instantiate a Factory that can give you the Tracks. (The Track doesn't know anything about the B field, so it can't translate from MCParticle parameters to LCIO Track parameters by itself)

```
NewFastMCTrackFactory factory = new NewFastMCTrackFactory(event, false); // beamConstraint when smearing ?
for (MCParticle daughter : part.getDaughters()) {
                  Track unsmearedTrack = tf.getTrack(
                      new SpaceVector(daughter.getMomentum()), // momentum
                      new SpacePoint(daughter.getOrigin()), // position
                      new SpacePoint(part.getEndPoint()), // reference point
                      (int)daughter.getCharge(), // charge
                      rand, // random generator for smearing
                      false); // smearing ? default is true

                  Track smearedTrack = tf.getTrack(
                      new SpaceVector(daughter.getMomentum()),
                      new SpacePoint(daughter.getOrigin()),
                      new SpacePoint(part.getEndPoint()),
                      (int)daughter.getCharge(),
                      rand,
                      true);
}
```

For more information please consult the source code. I will try to expand on the examples.