Status of the Track Event Model
Classes and Tools

E. Rodrigues, NIKHEF

- event classes
- tools and “helping” algorithms

How to use the new TEM best
- using the event classes
- using the “core” tools
- extending the functionality

Status of the Track Fitting
- the TrackFitter package
- ongoing tests

In preparation for DC’06
- what is still needed

Outlook
Status of the Track Event Model

Event classes

- **TrackEvent**: Track, State, Measurement, Node, TrackFunctor

Tr/

- **TrackFitEvent**: XxxMeasurement, FitNode

Kernel/

- **LHCbKernel**: LHCbID

Status

- *Rather stable & tested since a while now*
- *Your comments / questions / feedback are welcome*
  - *First (positive!) feedback from RICH & Calo given @ T-Rec meeting on 24th Oct.*
- *Extra functionality can be incorporated if needed*
  - *Useful / necessary functionality will be considered …*
Status of the Track Event Model
Tools and “helping” Algorithms

Tr/
- TrackInterfaces
- TrackExtrapolators
- TrackProjectors
- TrackTools
- TrackMCTools
- TrConverters

- Too many tools and algorithms to list here
- Full list given many times in the past
- Documentation is there for you … ;-)

Trg/
- TrgConverter

Status
- Full list (adapted from “old” TEM) rather stable & tested since a while now
- Little direct interaction with user – mainly e.g. TrackMasterExtrapolator
Status of the Track Fitting

The TrackFitter package

Package Tr/TrackFitter/

- **TrackEventFitter algo**
- **TrackKalmanFilter tool**

Status :

- All functionality adapted to the “new” TEM, e.g
  - iteration of filtering-smoothing sequence
  - outliers removal & update of “reference trajectories” available
  - Upstream/downstream fitting…

- First public version of TrackFitter available in CVS

- Results of ongoing tests follow …
In what follows:

- All initial tracks produced with the ideal pattern recognition of the old TEM
- Tracks selection - loose quality cuts:
  - *Only long tracks*
  - *Momentum > 1 GeV*
  - *# of hits on track > 20*

OLD

- TrFitTracks fitted *upstream* with old KF code
- states predicted at each meas. position
- TrFitTracks converted to Tracks

NEW

- non-fitted TrFitTracks converted to Tracks
- Tracks fitted *upstream* with new TEM KF package
- States predicted at each meas. position

Plots:

- Separate for OT, IT, Velo-R and Velo-Φ measurements

⇒ plots produced looping over all pairs of (state,measurement), all at same z-positions, by construction
Status of the Track Fitting

Ongoing Tests

Small differences still exist … under investigation ….
Status of the Track Fitting
Ongoing Tests

Small differences still exist … under investigation ….
Some side remarks:

- All TEM was performed in “blind development”
  - *New code timed for the first time only ~2 weeks ago!*
- Speed of “new” fitting code same as in “old” TEM
  - *Fair speed comparison*
  - *Fitting with the new TEM seems ~2-3% faster than with old TEM*
- This does not mean we will not try and improve further …!
- We keep testing and trying to understand possible features
Preliminary remarks:

- Design choices have been discussed at length elsewhere
- Best usage of current event model reflects those choices AND conventions

Some basic guidelines:

- Tools that provide info typically have references as arguments
  - Means these tools do not get ownership of objects passed to them
  - E.g. extrapolators, projectors
- Some methods return pointers
  - In this case the user gets the ownership – is responsible for deletion
  - E.g. clone methods
How to use the new TEM best
Using the Event Classes

// .h file
// from TrackEvent
#include "Event/Track.h"

// .cpp file

Tracks* tracksCont = get<Tracks>(TrackLocation::Default);

dbg() << "Tracks container contains " << tracksCont->size() << " tracks" << endreq;

for (Tracks::const_iterator iTrk = tracksCont->begin(); tracksCont->end() != iTrk; ++iTrk) {
  Track& track = *(*iTrk);
  dbg() << "-> Track # " << track.key() << endreq
        << " * charge = " << track.charge() << endreq
        << " * is of type = " << track.type() << endreq
        << " * # states = " << track.nStates() << endreq
        << " * # measurements = " << track.nMeasurements() << endreq;
}
// .cpp file

// ...
Track& track = *(iTrk);
double debug()
{
    << "-> Track # " << track.key() << endreq
    << "  * is valid          = " << !track.checkFlag( Track::Invalid ) << endreq
    << "  * is unique        = " << track.checkFlag( Track::Unique ) << endreq
    << "  * is a long track = " << track.checkType( Track::Long ) << endreq
    << "  * is Backward      = " << track.checkFlag( Track::Backward ) << endreq
    << "  * has been fitted OK = " << track.checkStatus( Track::Fitted ) << endreq
    << "  * $\chi^2$ from fit = " << track.chi2() << endreq;
}

// ...
How to use the new TEM best
Using the Event Classes

// .cpp file

// …
Track& track = *(*iTrk);
// position and momentum of the “first state” (i.e. the one stored by default on the DST)
HepPoint3D pos;
HepVector3D mom;
HepSymMatrix cov6D;
track.positionAndMomentum( pos, mom, cov );
// …
HepVector3D slp;
HepSymMatrix errSlp;
track.slopes( slp, errSlp );
HepVector3D slp2 = track.slopes();
// …
ddebug()
  << "-> track momentum = " << track.p() << endreq
  << " * transverse momentum = " << track.pt() << endreq;
// …
}
How to use the new TEM best
Using the “core” Tools

// .h file
// from TrackEvent
#include "Event/Track.h"
// from TrackInterfaces
#include "TrackInterfaces/ITrackExtrapolator.h"
// …
ITrackExtrapolator* m_extrapolator;

// .cpp file
// … ( e.g. m_particleID = 211 for a pion )
Track& track = *(iTTrk);
double z = 1000.;
State state;
StatusCode sc = m_extrapolator -> propagate( track, z, state, m_particleID );
// after extrapolation the state will have its state vector and covariance matrix updated …

// …
}
// from TrackEvent
#include "Event/TrackFunctor.h"
#include "Event/Measurement.h"

// counting the number of VELO measurements!
TrackFunctor::HasKey<Measurement> isVeloR (&Measurement::checkType, Measurement::VeloR );
TrackFunctor::HasKey<Measurement> isVeloPhi (&Measurement::checkType, Measurement::VeloPhi );
unsigned int nVeloMeas = TrackFunctor::nMeasurements( track, isVeloR )
 + TrackFunctor::nMeasurements( track, isVeloPhi );

// can be exploited to fabricate functions:
TrackFunctor::HasKey<Track> isBackward (&Track::checkFlag, Track::Backward );
if ( isBackward(track) ) { // … }
// …
}
How to use the new TEM best
Extending the Functionality

Extra Flags for private usage:
  • Flags enum in Track.h has some dedicated “specific bits” for this

Use cases not yet considered:
  • Your comments / questions / feedback are welcome
  • Useful / necessary functionality will be considered for implementation …
THE goal:

• Fully operational tracking working with a realistic geometry!

What is still needed, “mainly”:

• Event classes and tools:
  • Extend / adapt code for realistic geometries – see Edwin’s talk
  • Improvements/bugs are not impossible/unrealistic …

• Pattern recognition:
  • All PR packages in accordance with new TEM – see Matt’s talks

• Setting-up of a tracking reconstruction sequence for Brunel
  • Visit the Brunel’s tracking sequence and in particular the “clone killer”

• Integration of B-field map in the software
Outlook

• We are commissioning the new Track Event Model!

• The Track Event Model is in good shape!

• Most of the classes & tools have been tested

• Adapted code with new TEM seems to be as fast as old code

• Time to start profiling the classes / tools / algorithms in more detail

• A lot still ahead of us to cope with real geometries

• Time is tight but progress is steady …