MPD Collaboration **Version 1.1**

November 24, 2022

MpdTpcKalmanTrack tutorial for dummies and intermediate users or Some things you always wanted to know about MpdTpcKalmanTrack but were afraid to ask

A. Zinchenko*

VBLHEP JINR, Dubna, Russia

1 Introduction

Currently, the MpdTpcKalmanTrack object represents a track reconstructed in the MPD TPC using the Kalman filter - based track reconstruction procedure MpdTpcKalmanFilter.

Some information on the object is presented in Sect. 2, while Sect. 3 gives some examples of how one can work it.

2 Main track parameters

As often accepted, the reconstructed track can be parameterized as a vector of 5 parameters at some position along the trajectory. Outside of the TPC active volume, i.e. outside of the innner and outer borders of the TPC readout chambers, it corresponds to the vector $(r\phi_p, z, \phi_m, \alpha, \mathrm{sign}(q)/p_T)$ at a radius r, where ϕ_p is the azimuthal angular position of the given point on the track ($r\phi_p$ is a circle arc length), z is the longitudinal coordinate, ϕ_m is the track momentum direction in the transverse plane at the point, α is the track momentum direction with respect to the transverse plane ($\alpha = \pi/2 - \theta$ with θ being the polar angle), $\mathrm{sign}(q)/p_T$ is the signed inverse transverse momentum of the track. Inside the TPC active volume, some track parameters in the Kalman filter procedure are changed to the local ones, namely, the transverse coordinate $r\phi_p$ becomes the coordinate x along the padrow direction in the readout sector and the radius r becomes the coordinate y across the padrow direction.

Track parameters are computed at the point of the closest approach to the OZ-axis (or to the nominal beam line location), although some parameters are invariant with respect to the exact point choice. There are several functions to access track parameters of the object *track*:

*Alexander.Zinchenko@jinr.ru

`int nHits = track.GetNofTrHits(); // number of hits attached to the track`
`Double_t chi2 = track.GetChi2() / (nHits * 2 - 5); //` χ^2/NDF
`Double_t r = track.GetPosNew(); //` DCA (distance of the closest approach to OZ-axis)
`TVector3 mom3 = track.Momentum3(); //` momentum vector at PCA
`etc.`

To distinguish between primary and secondary tracks it is possible to use the function `GetChi2Vertex()` which returns the χ^2-value of the track with respect to the primary vertex (with $NDF = 2$), computed in the primary vertex reconstruction package. It should be noted here that track reconstruction package assumes the pion mass hypothesis, resulting in somewhat higher χ^2-values (from the track fitting and with respect to the primary vertex) for heavier particles (protons, for example).

One can also access the dE/dx - value, computed for the track:

`Double_t dedx = track.GetDedx(Double_t coef = 6.036e-3); //` dE/dx converted from ADC counts to keV/cm using the conversion factor $coef$

3 Working with MpdTpcKalmanTrack

To illustrate how to work with MpdTpcKalmanTrack objects, this Section gives some examples. In the first one, it is demonstrated how to obtain track parameters at the point of the closest approach to the reconstructed primary vertex and the DCA at this point, which could be used to make another way of separating primary from secondary tracks than explained above. The computation can be done using the MpdHelix functionality as shown below:

```
//_____________________________________________________________________
MpdHelix MakeHelix(const MpdKalmanTrack *tr)
{
   Double_t r = tr->GetPosNew();
   Double_t phi = tr->GetParam(0) / r;
   Double_t x = r * TMath::Cos(phi);
   Double_t y = r * TMath::Sin(phi);
   Double_t dip = tr->GetParam(3);
   Double_t cur = 0.3 * 0.01 * 5.0 / 10; // magnetic field of 5 kG
   cur *= TMath::Abs (tr->GetParam(4));
   TVector3 o(x, y, tr->GetParam(1));
   Int_t h = (Int_t) TMath::Sign(1.1,tr->GetParam(4));
   MpdHelix helix(cur, dip, tr->GetParam(2)-TMath::PiOver2()*h, o, h);
   return helix;
}

//_____________________________________________________________________

   ...
   TVector3 primVert; // vertex position
   MpdTpcKalmanTrack *tr; // pointer to reconstructed track
```

```
MpdHelix helix = MakeHelix(tr);
TVector3 pca;
Double_t s = helix.pathLength(primVert);
pca = helix.at(s);
pca -= primVert;
Double_t dca = pca.Mag();
...
```

The second example shows how to refit `track` (pointer) with certain `mass` and `charge` hypotheses. To do this, it is necessary to instantiate and initialize the main TPC tracking engine `MptTpcKalmanFilter`. A "minimalistic" version is shown below (i.e. the one with the default value of the magnetic field of 5 kG):

```
FairRunAna ana;
MpdKalmanFilter::Instance("KF")->Init();
MpdTpcKalmanFilter *recoTpc = new MpdTpcKalmanFilter("TPC Kalman filter");
recoTpc->SetSectorGeo(MpdTpcSectorGeo::Instance());
recoTpc->FillGeoScheme();
...
MpdTpcKalmanTrack trCor = *track; // make a copy of the original track
int ok = recoTpc->Refit(&trCor, mass, TMath::Abs(charge));
MpdTpcKalmanTrack *tr = &trCor;
if (!ok) tr = track; // failed to refit - take original track
...
```

The third example shows how to perform a vertex-constrained fit of a `track` (pointer):

```
TChain *dst = ...; // input chain
TClonesArray* vtxs = nullptr;
dst->SetBranchAddress("Vertex",&vtxs);
...
MpdVertex *mpdVert = (MpdVertex*) vtxs->First(); // reconstructed vertex
...
MpdKfPrimaryVertexFinder vertFind; // create vertex finder
TClonesArray *smoothTracks = new TClonesArray("MpdTpcKalmanTrack",9); // track
container
smoothTracks->Delete();

new ((*smoothTracks)[0]) MpdTpcKalmanTrack(*track);
mpdVert->GetIndices()->Set(1);
(*mpdVert->GetIndices())[0] = 0;
vertFind.SetVertices(vtxs);
vertFind.SetTracks(smoothTracks);
vertFind.SetSmoothSame(1);

vertFind.Smooth();
MpdTpcKalmanTrack *tr1 = (MpdTpcKalmanTrack*) smoothTracks->UncheckedAt(0);
TVector3 mom3 = tr1->Momentum3(); // vertex-constrained momentum
```

4 Outlook

This tutorial will be being updated from time to time.