Kalman filter based processor for track reconstruction in MarlinTPC

– Kalman filter, structure, prelimnary results –

Bo LI(Tsinghua U.)
Keisuke FUJII(KEK)
Yuanning GAO(Tsinghua U.)
Katsumasa IKEMATSU(KEK)
Yulan LI(Tsinghua U.)
Yukihiro KATO(Kinki U.)

Track reconstruction and Kalman filter

Track reconstruction usually includes two parts:

- Track finding
  - Hough transform
  - Neural network
  - Track following(track model, track seed, hit selection criterion)
  - …
- Track fitting
  - Least squares estimation
  - Maximum likelihood estimation

Kalman filter
Invented by R.E. kalman in 1960, and introduced in the HEP community in 1980's.

The motivation of using Kalman filter for TPC tracking:

- Relatively easily deal with inhomogeneous magnetic field(crucial requirement now!), multiple scattering, and energy loss;
- Carry out track finding and track fitting simultaneously;
- The computation time is proportional to site number, which makes computation expense less.

Kalman filter algorithm for tracking

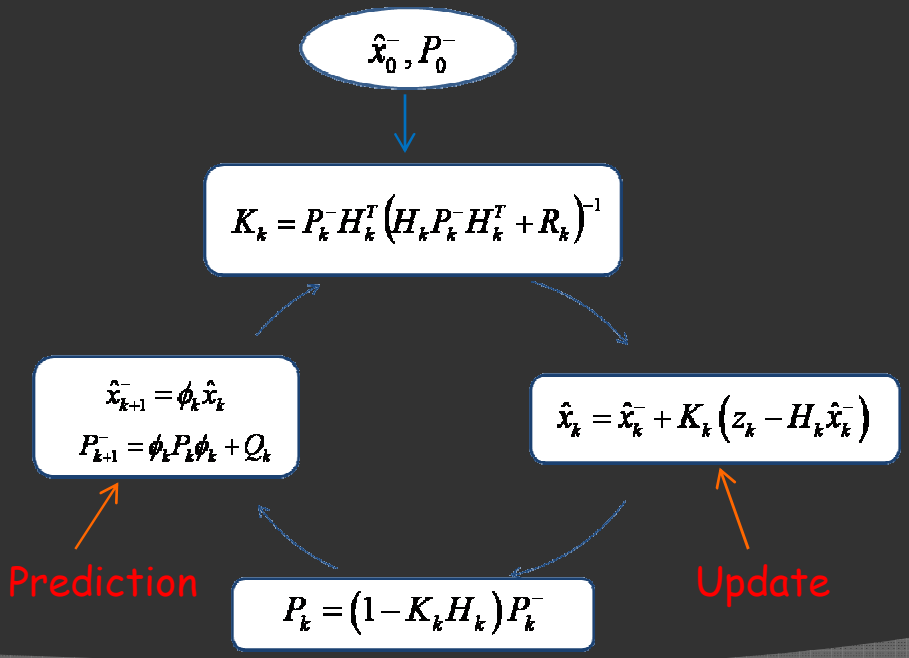
System equation:

$$x_{k+1} = \phi_k x_k + w_k$$

Meas. equation:

$$z_k = H_k x_k + v_k$$

x: state vector
φ: transfer matrix
z: measurement
H: projection matrix
K : gain matrix
P/Q: covariance of w/v



- Kalman filter accumulates hit information and calculates the track parameters;
- Kalman filter prediction provides a good criterion for hit selection.

Track models

See KalTest reference by K. Fujii

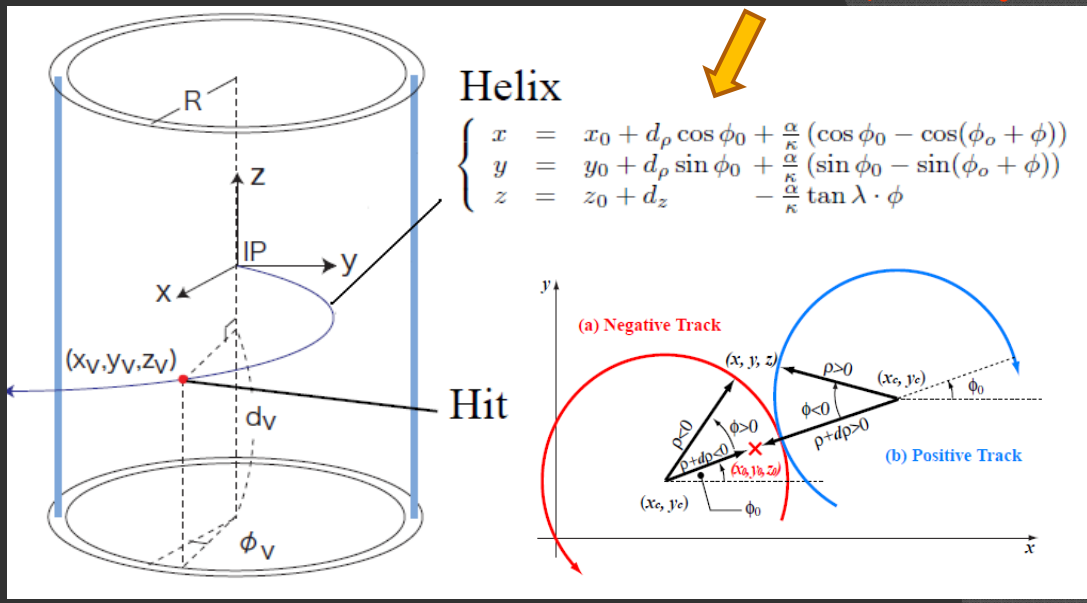
2D straight track $y = ax + b$

state vector $X = \begin{bmatrix} a \\ b \end{bmatrix}$

transfer matrix $\phi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

projection matrix $H = \begin{bmatrix} x & 1 \end{bmatrix}$

- Despite of different track model, the track following method is nearly the same for straight line and helix.
- For helix model, the Kalman filter is not linear, it is usually called as extended Kalman filter.



d_ρ: the distance beteen the helix and the hit in x-y plane
φ_0: the azimuthal angle of the hit with respect to the helix center
κ: Q/P_t
d_z: the distance between the helix and the hit in the z direction
$\tan\lambda$: the angle of the helix to the x-y plane

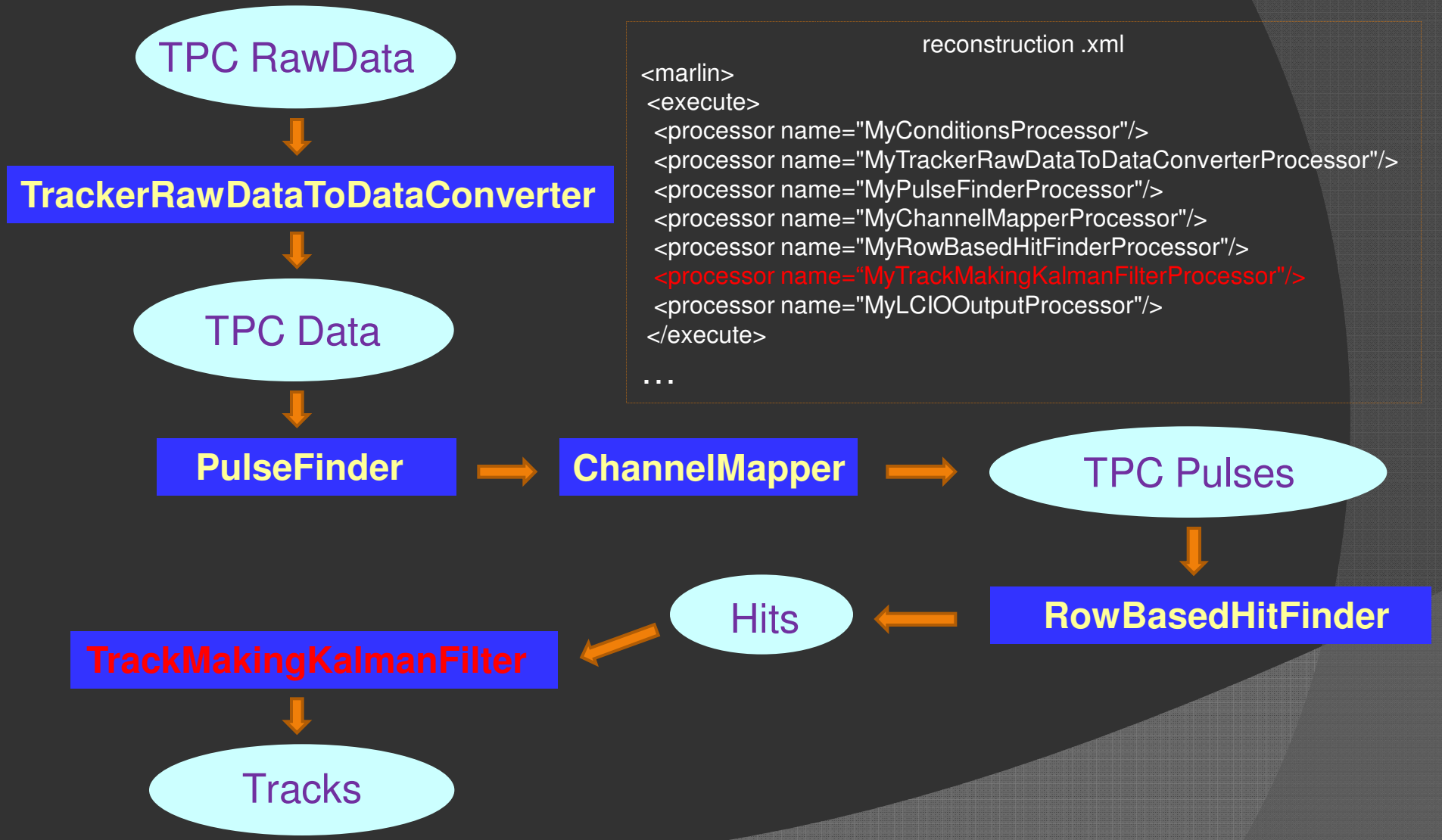
state vector $X_k = (d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$

transfer matrix $F_k = \left(\frac{\partial X_k}{\partial X_{k-1}}\right)$

KalTest

- KalTest is a Kalman filter package written in C++. It provides basic libraries for track fitting with Kalman filter technique.
- KalTest contains 3 subpackages:
  - Kallib: implements the generic algorithm of the Kalman filter
  - KalTrackLib: inherits from the generic base classes in KalLib, and implements their pure virtual methods for track fitting
  - GeomLib: provides track model and detector surfaces
- To get KalTest package and more information about helix track model, please visit KalTest homepage: http://www-jlc.kek.jp/subg/offl/kaltest/

Reconstruction chain in MarlinTPC

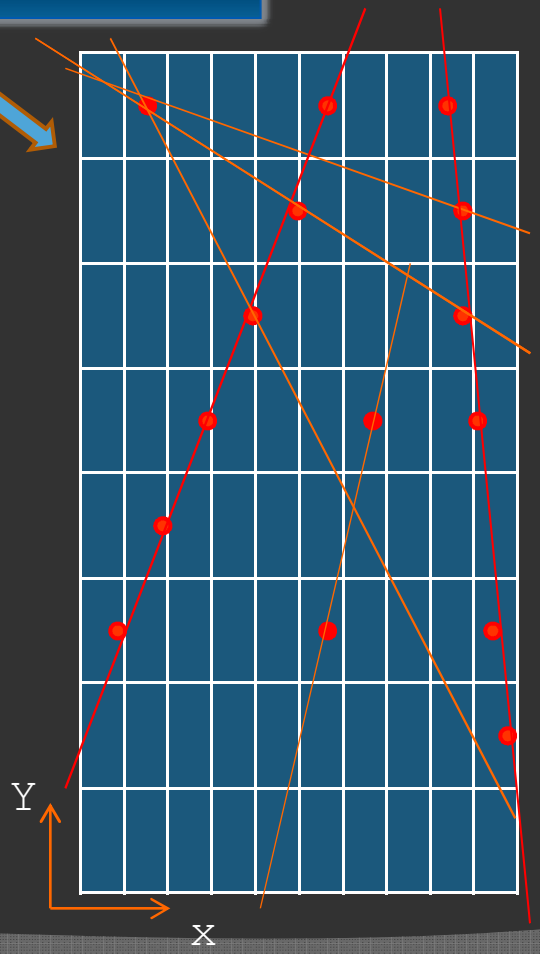


reconstruction .xml

```
<marlin>
 <execute>
  <processor name="MyConditionsProcessor"/>
  <processor name="MyTrackerRawDataToDataConverterProcessor"/>
  <processor name="MyPulseFinderProcessor"/>
  <processor name="MyChannelMapperProcessor"/>
  <processor name="MyRowBasedHitFinderProcessor"/>
  <processor name="MyTrackMakingKalmanFilterProcessor"/>
  <processor name="MyLCIOOutputProcessor"/>
 </execute>
 …
```

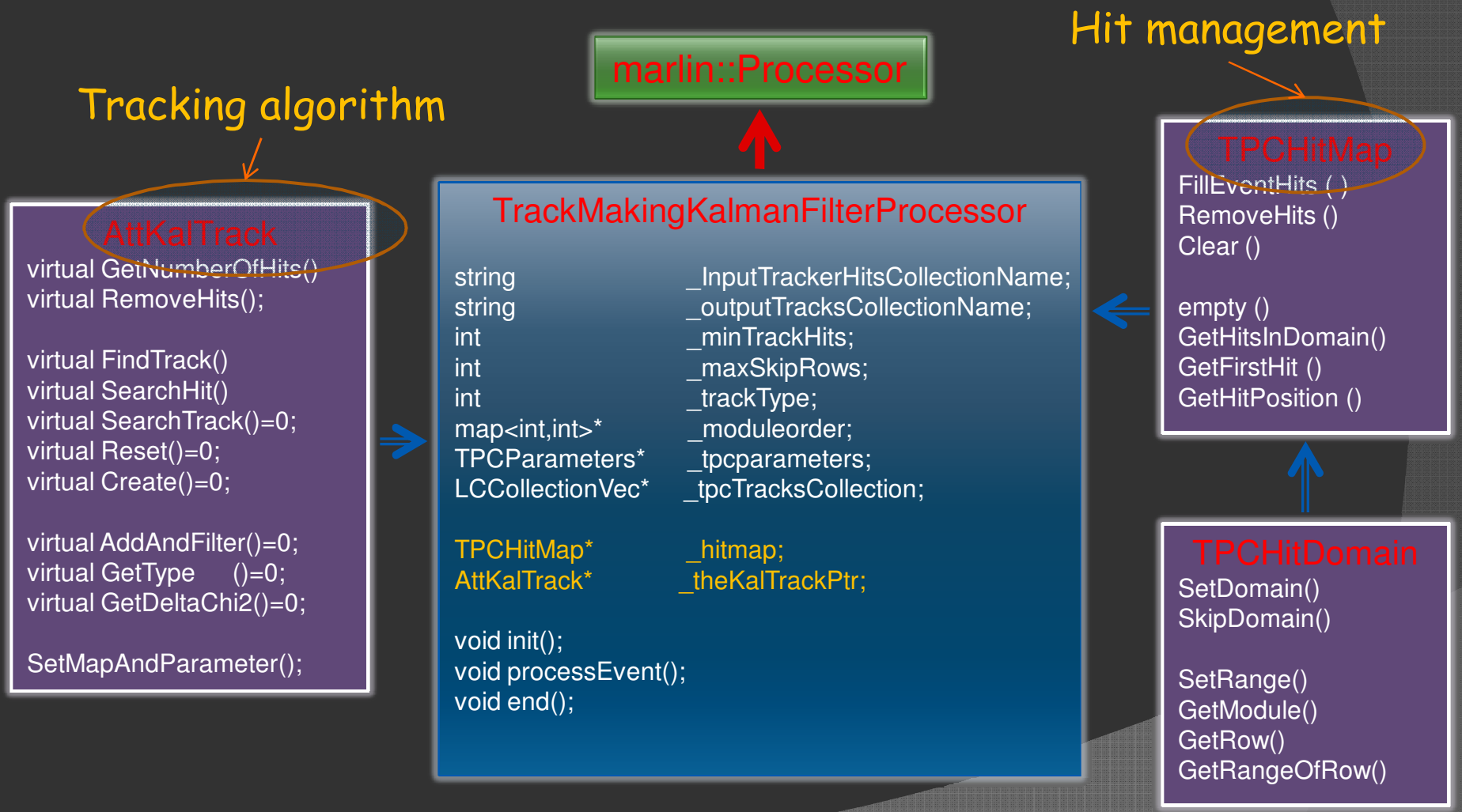
Tracking process

Track criteria
MaxSkipRows = 1
MinTrackHits = 5

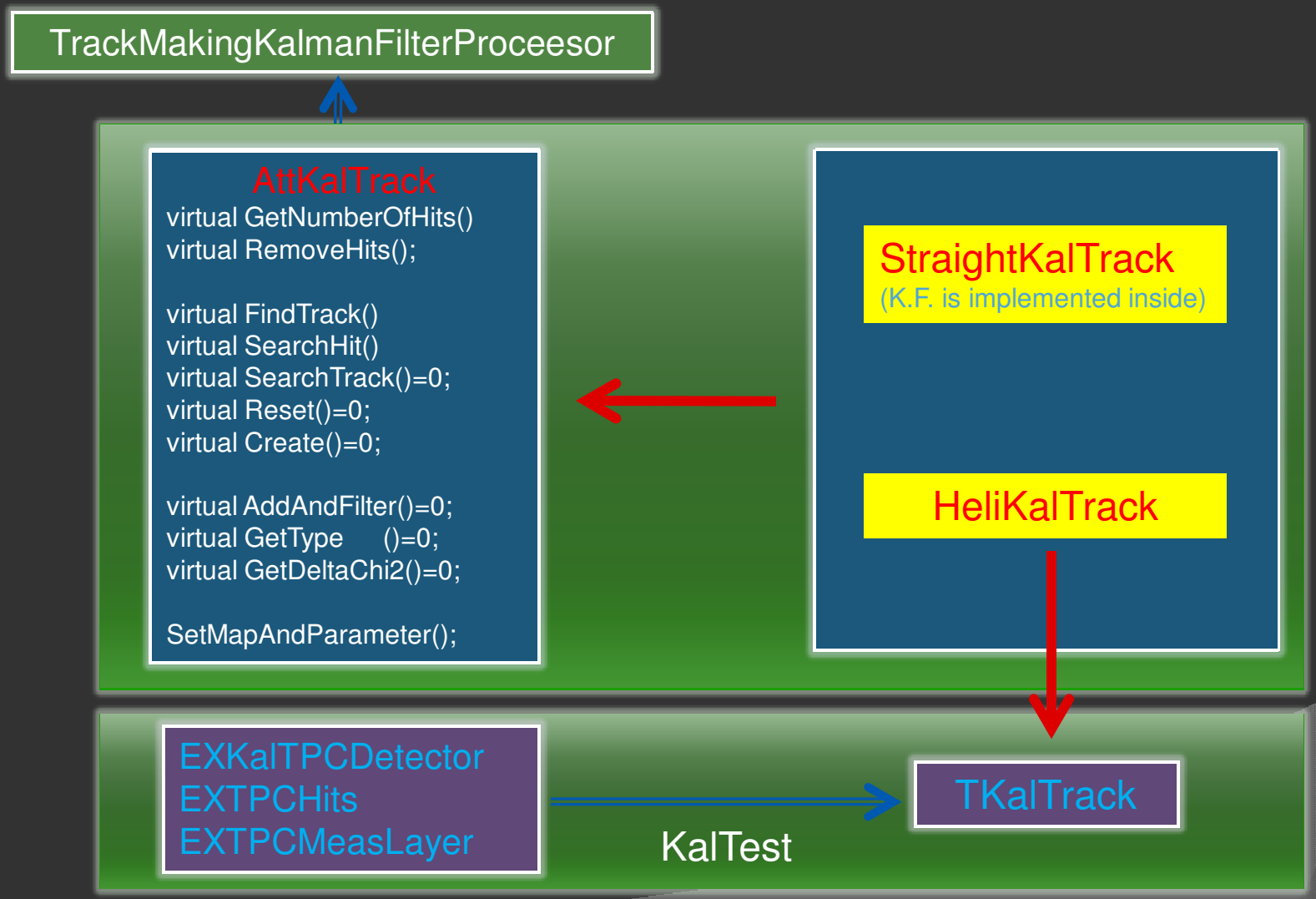


If Process Oriented → Object Oriented, the whole work will be distributed to different objects, making the code flexible.

The class organization in processor



Track classes



Implementation in the processor

Then, the code in the processor becomes very neat:

```
void TrackMakingKalmanFilterProcessor::init()
{
     const TPCParameters* tpcparameters =& Global::GEAR->getTPCParameters();
     _moduleorder=new map<int,int>;

     _hitmap=new TPCHitMap(tpcparameters, _InputTrackerHitsCollectionName, _moduleorder);
     _theKalTrackPtr = new StraightKalTrack;
     _theKalTrackPtr->SetMapAndParameter(_tpcparameters,_hitmap);
}

void TrackMakingKalmanFilterProcessor::processEvent(LCEvent *evt)
{
    _tpcTracksCollection = new LCCollectionVec(LCIO::TRACK);

    _hitmap->FillEventHits(evt);

    while(!_hitmap->empty())
       if( TrackImpl* atrack=_theKalTrackPtr->FindTrack(_hitmap->GetFirstHit()) )
          _tpcTracksCollection->addElement(atrack);

    if(int tracknum = _tpcTracksCollection->getNumberOfElements())
       evt->addCollection( _tpcTracksCollection,_outputTracksCollectionName );
}

void TrackMakingKalmanFilterProcessor::end()
{
   delete _theKalTrackPtr;
   delete _hitmap;

   cout << "End of TrackMakingKalmanFilterProcessor" << endl;
}
```

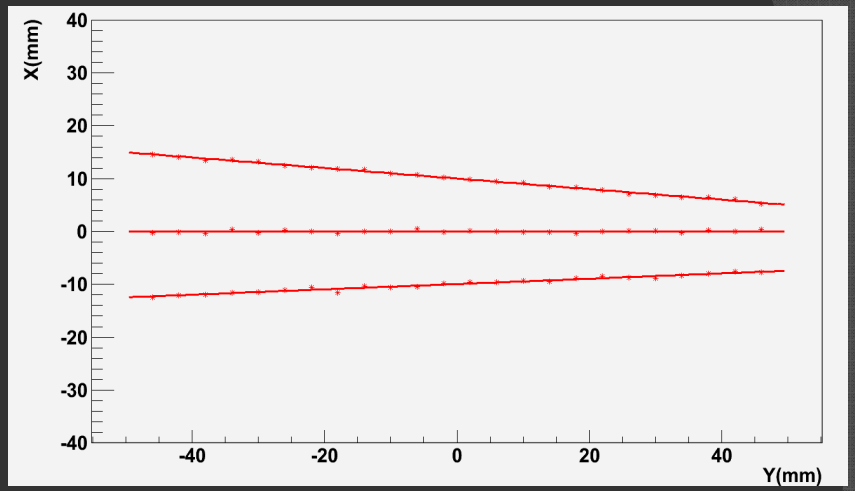
Configuration of processor

This is an configuration file of TrackMakingKalmanFilterProcessor:

```
<processor name="MyTrackMakingKalmanFilterProcessor" type="TrackMakingKalmanFilterProcessor">
<!--For simple track finding and fitting by Kalman filter!-->
 <!--Name of the Input TrackerHits collection-->
 <parameter name="InputTrackerHits" type="string" lcioInType="TrackerHit"> TPCHits </parameter>
 <!--Name of the output Tracks collection-->
 <parameter name="OutputTracks" type="string" lcioOutType="Track"> TPCTracks </parameter>
 <!--Maximum number of subsequnetly missing hits (default: 1)-->
 <parameter name="MaxSkipRows" type="int" value="1 "/>
 <!--Minimum number of hits on track (default: 15)-->
 <parameter name="MinTrackHits" type="int" value="15"/>
 <!--if not 0 the output hits collection is set transient (default: 0)-->
 <parameter name="SetOutputHitsTransient" type="int" value="0"/>
 <!--if not 0 the output trackscollection is set transient (default: 0)-->
 <parameter name="SetOutputTrackCandidatesTransient" type="int" value="0"/>
 <!--Track type (default: 1 0=straight line 1=helix)-->
 <parameter name="TrackType" type="int" value="0"/>
</processor>
```

Straight line reconstruction

```
<PadRowLayout2D

    type="RectangularPadRowLayout"
    xMin="-32." xMax="32." yMin="-48.">
    <row repeat="24" nPad="64" padHeight="3.8"
padWidth="0.8" rowHeight="4." />

</PadRowLayout2D>
```



Single module
Pad Width: 0.8mm
Pad height: 3.8mm
Pad Pitch: 1mm
Row Height: 4mm

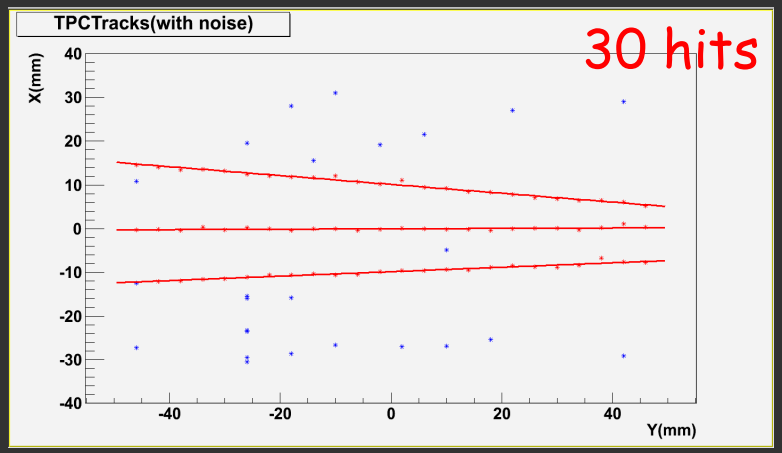
Hit position error: 0.2mm

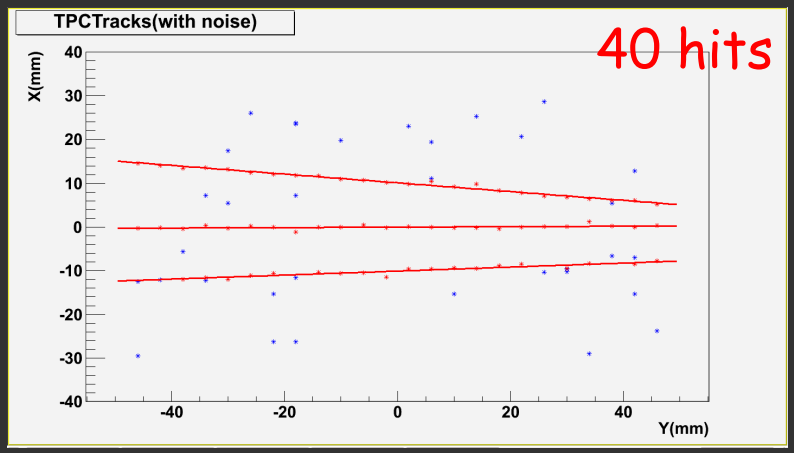
Real	
Slope	Intercept
0.05	-10.00
0.00	0.00
-0.01	10.00

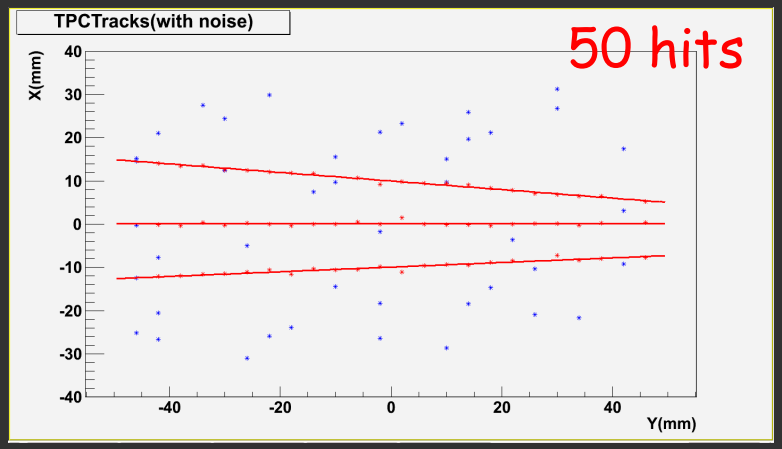
Reconstructed	
Slope	Intercept
0.0502	-9.9354
0.0008	-0.0010
-0.0997	10.0189

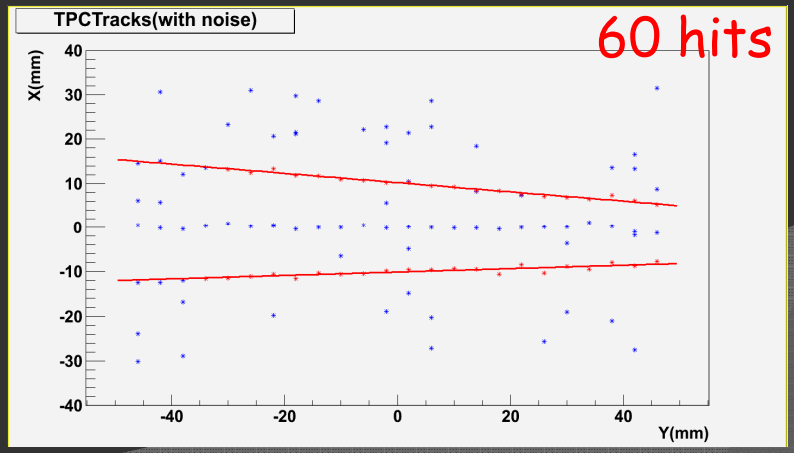
Straight line reconstruction(Cont.)

Noise hits are added into…





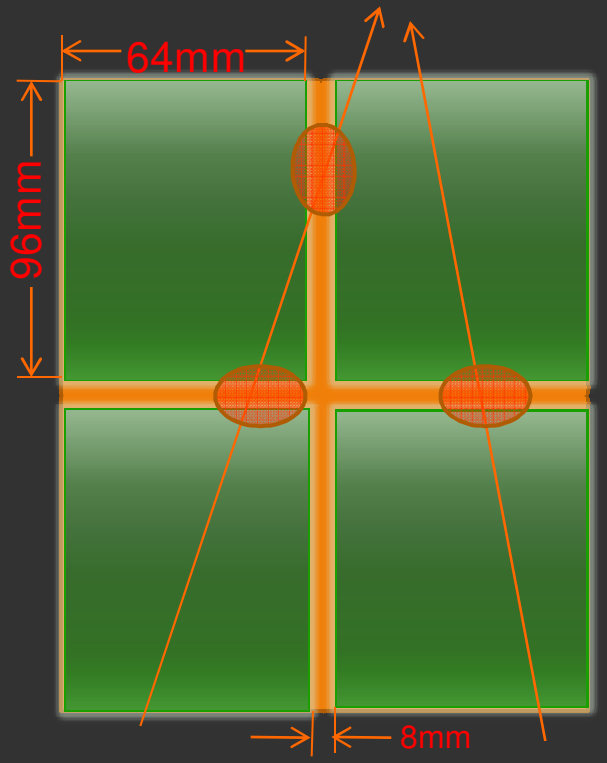




Multi-module

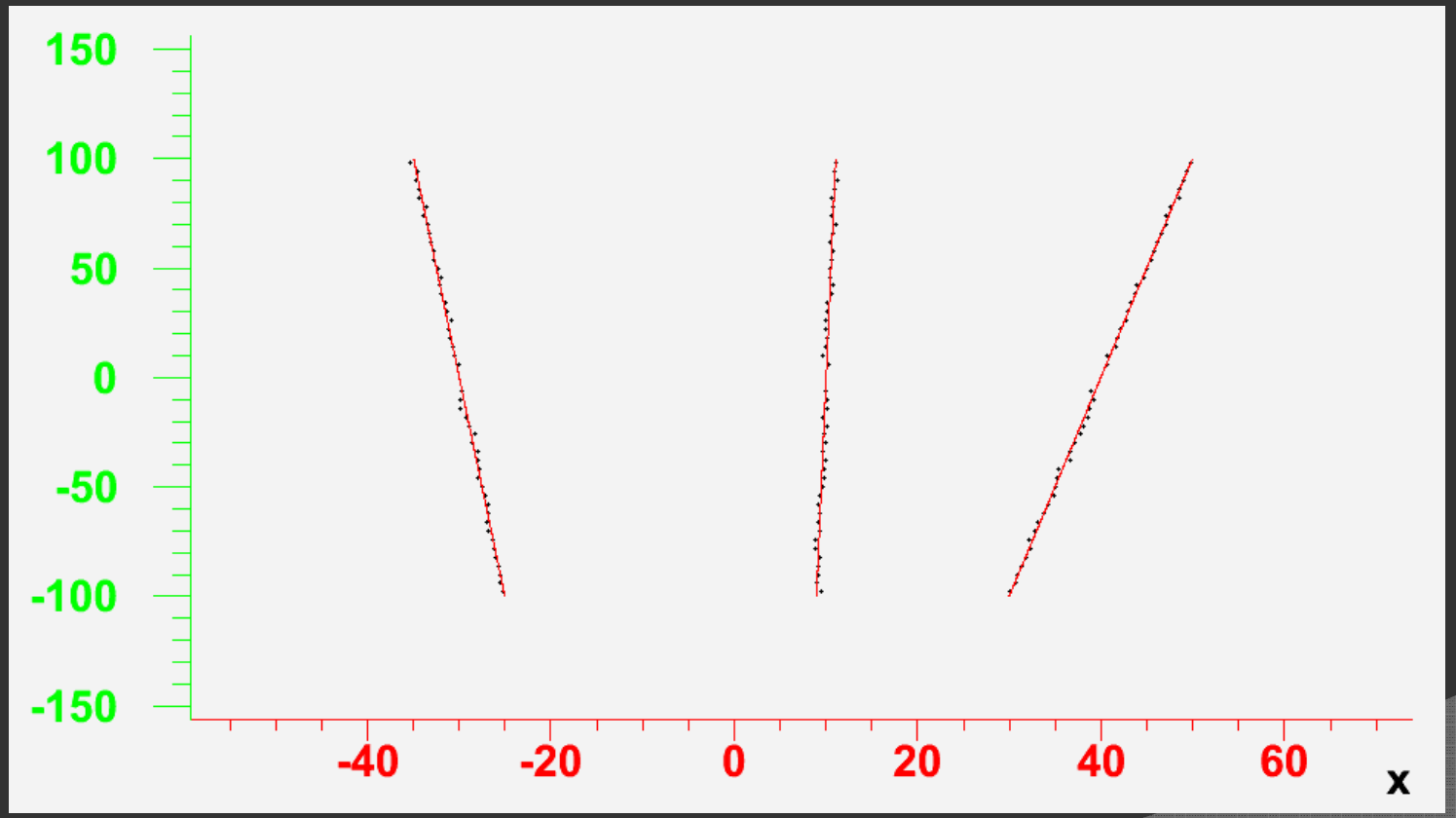
```
<modules moduleIDStartCount="0">
  <default>
    <PadRowLayout2D    type="RectangularPadRowLayout"
       xMin="-32." xMax="32." yMin="-48.">
       <row repeat="24" nPad="64" padHeight="3.8"
        padWidth="0.8" rowHeight="4." />
    </PadRowLayout2D>
  </default>

  <module>
    <offset x_r="36." y_phi="-52." />
  </module>

  <module>
    <offset x_r="-36." y_phi="-52." />
  </module>

  <module>
    <offset x_r="-36." y_phi="52." />
  </module>

  <module>
    <offset x_r="36." y_phi="52." />
  </module>
</modules>
```

4 modules

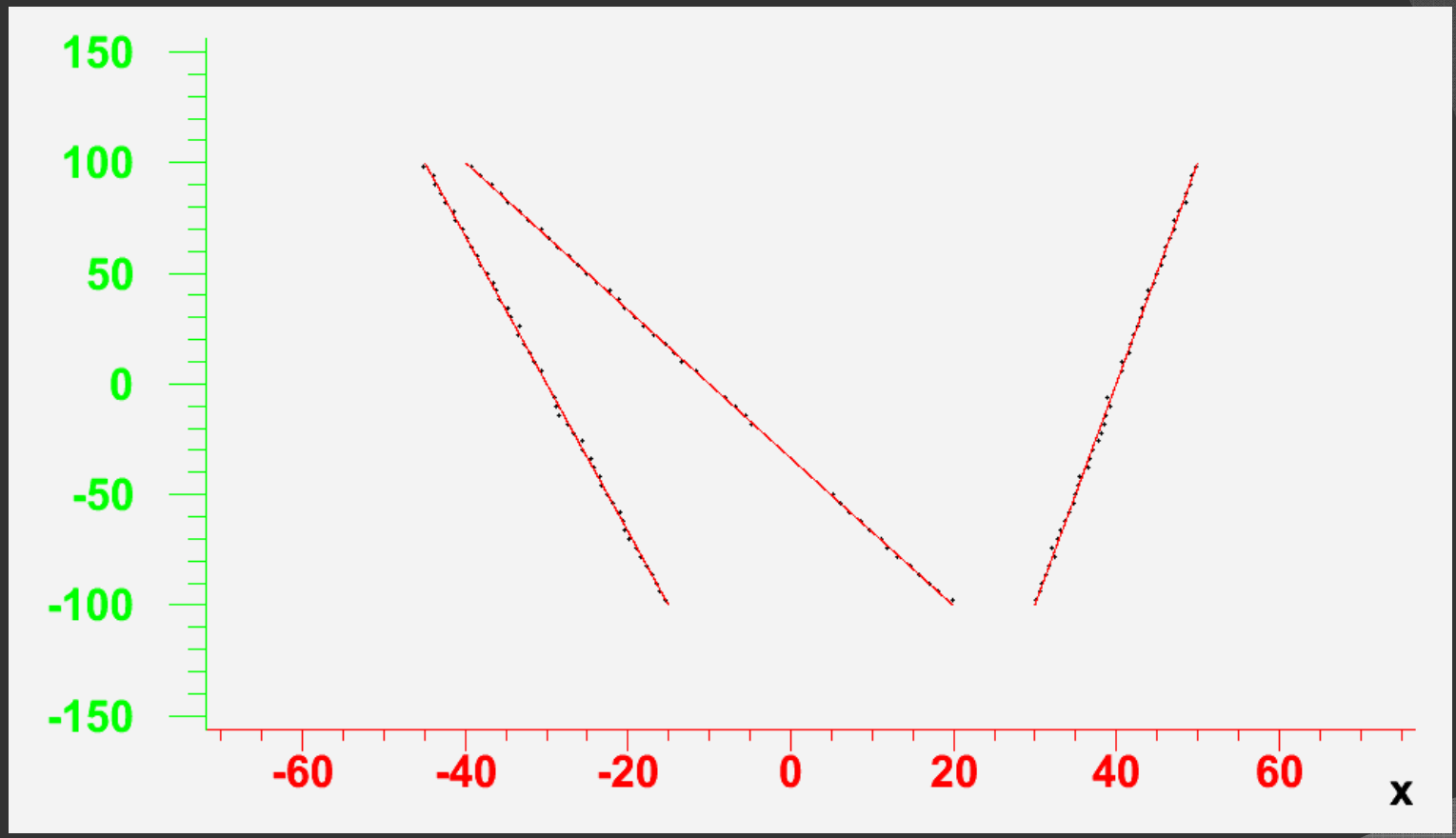


The main problem is how to cross boundaries and transfer modules. This work is done by TPCHitDomain.

Multi-module result

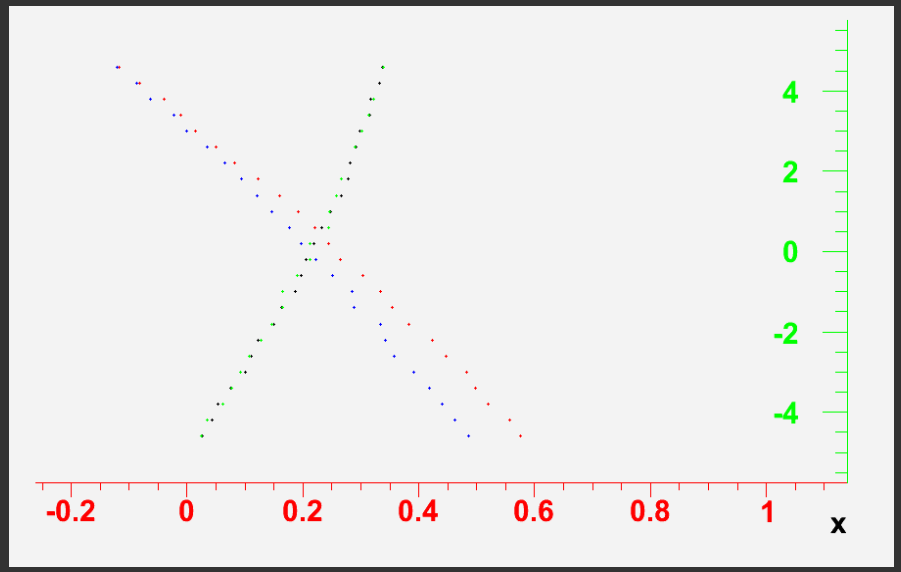


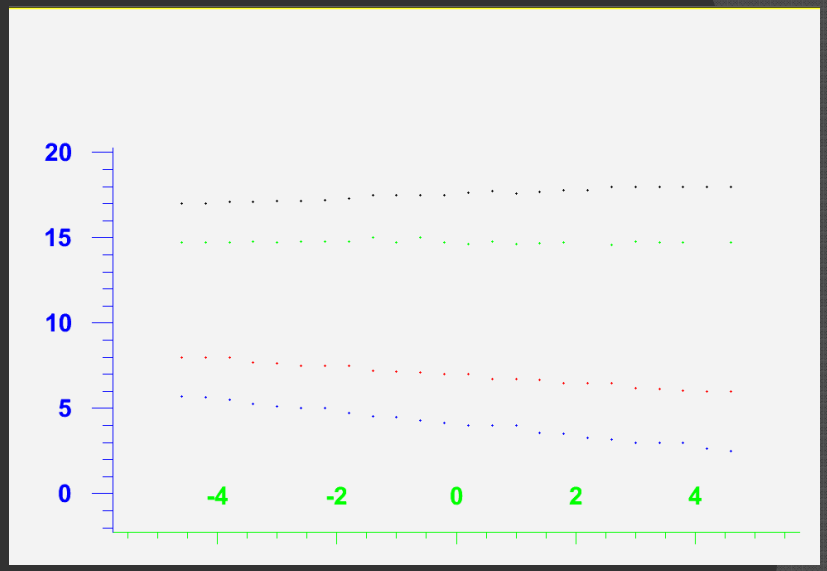
Multi-module result(cont.)



- The result for straight line tracking is proper. ☺
- Framework for multi-module tracking also works. ☺

Helix result





- Track finding result for helix is reasonable .☺
- But at present, the fitting result seems to be dependent on the track seed ☹, which need to be improved.

Summary and plan

- Kalman filter based processor has been developed in MarlinTPC;
- Processor was used to reconstruct straight line and helix with toy MC data;
- Multi-module framework of processor has been setup.

- Check and improve the helix track reconstruction result;
- Implement multi-module helix reconstruction code, and apply it to test beam data;
- In the long run, we can feed back to MarlinReco.

Thanks for your attention!