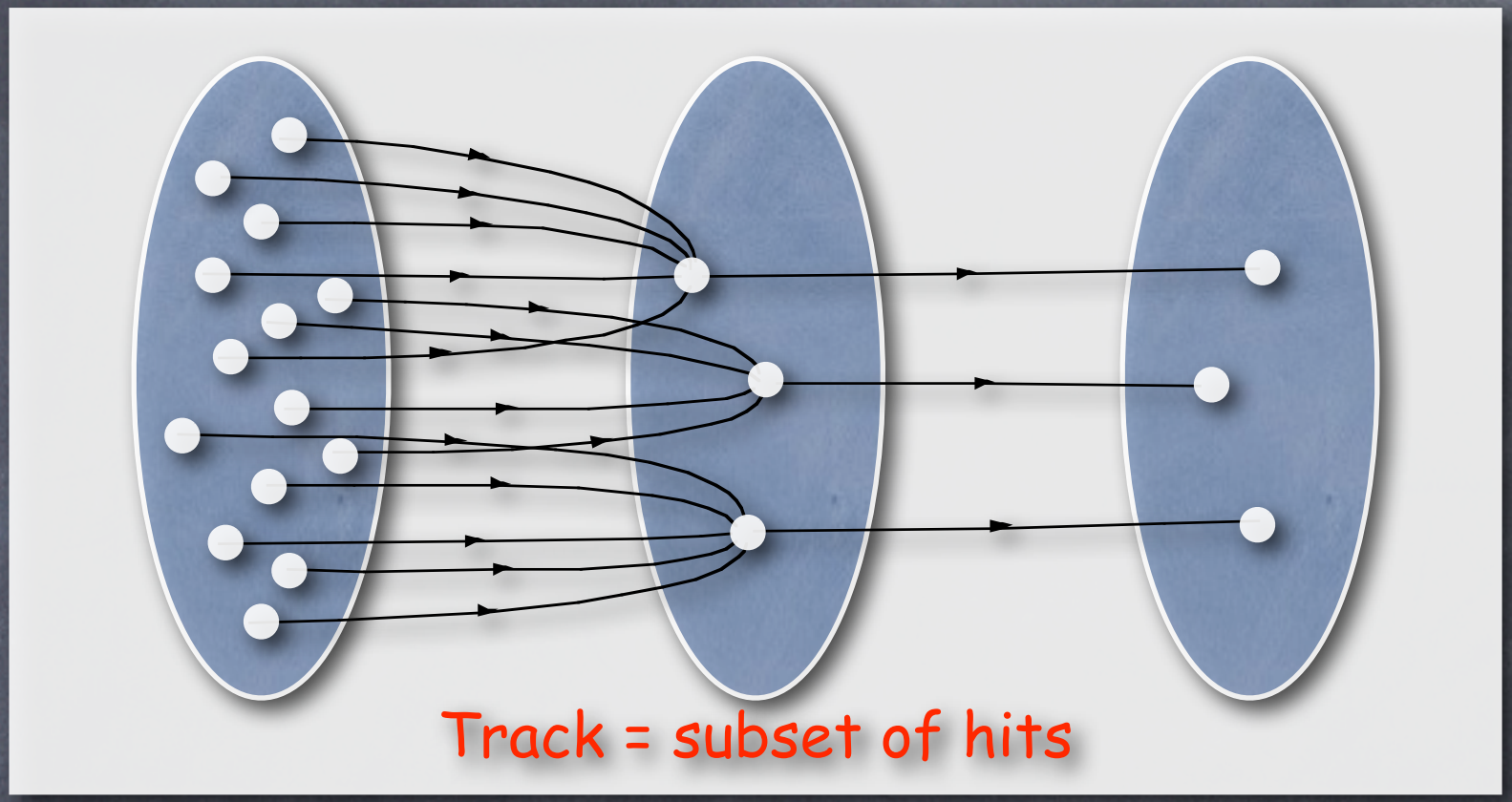
Tracking Code in Uranus/Satellites

-- Extended Kalman Filter --

Kiesuke Fujii, KEK

April 16, 2009

Tracking = Track Fitting x Track Finding



Set of Hits → Set of Tracks → Set of Track Parm. Vectors

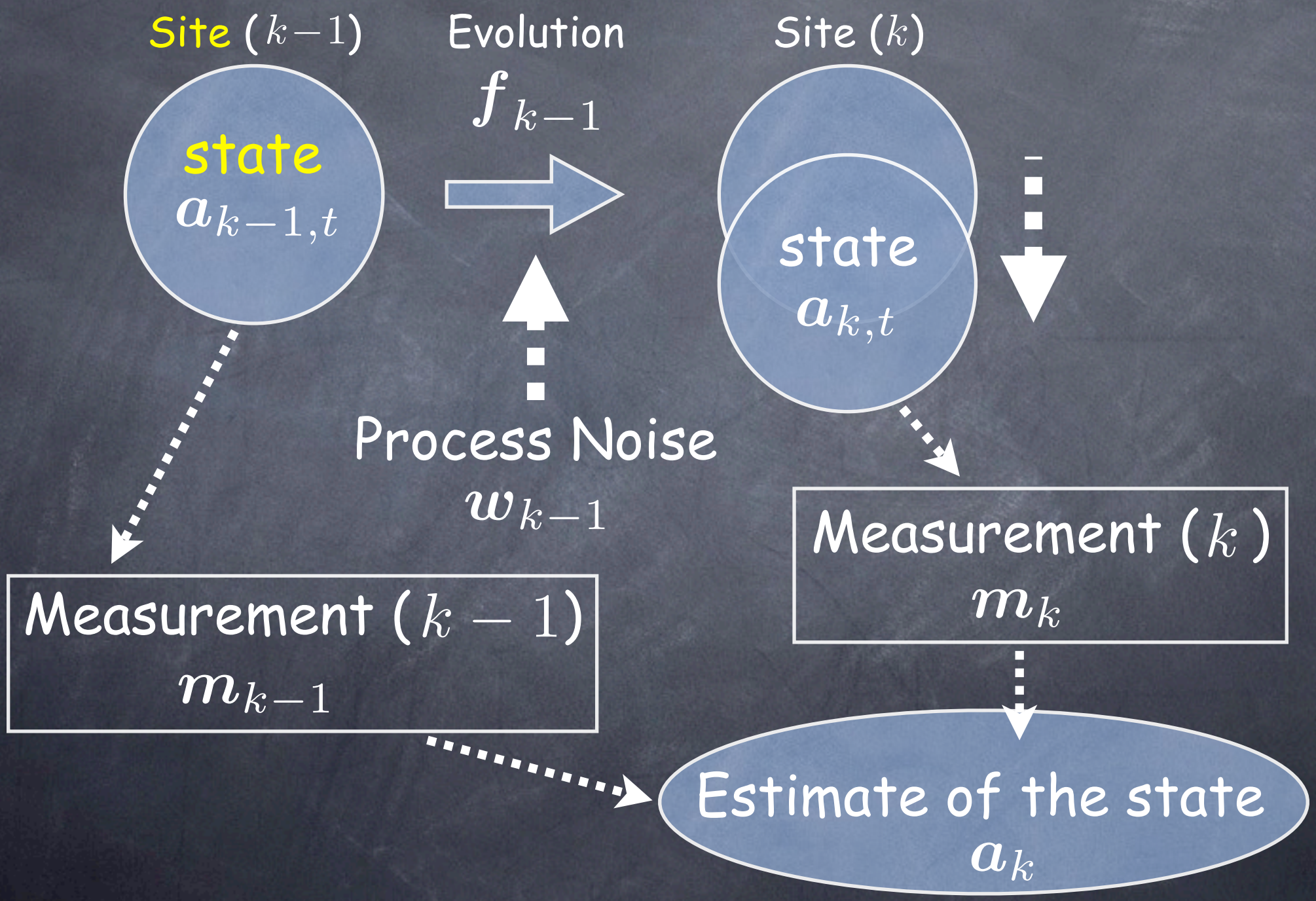
Track Finding Track Fitting

Today's menu

Kalman Filter Formulation	Its C++ Implementation	Application to ILC Trackers

Statement of the Problem

System


Site $(k-1)$
Evolution
f_{k-1}
Site (k)
state
$a_{k-1,t}$
state
$a_{k,t}$
Process Noise
w_{k-1}
Measurement (k)
m_k
Measurement $(k-1)$
m_{k-1}
Estimate of the state
a_k

Track = as a Kalman System consisting of Sites (Hits)

state

$$\boldsymbol{a}_k = \begin{pmatrix} d_\rho \\ \phi_0 \\ \kappa \\ d_z \\ \tan\lambda \end{pmatrix}_k$$

Helix parameter vector at (k)

$$\boldsymbol{w}_{k-1}$$

Multiple scattering between $(k-1)$ and (k)

site

$$\boldsymbol{m}_k$$

Measured hit point at (k)

$$\boldsymbol{\epsilon}_k$$

random detector noise

System Equation (Equation of Motion)

$$\boldsymbol{a}_{k,t} = \boldsymbol{f}_{k-1}(\boldsymbol{a}_{k-1,t}) + \boldsymbol{w}_{k-1}$$

process noise from $(k-1)$ to (k)

true state vector at $(k-1)$

true state vector at (k)

Assume that process noise is random and unbiased

$$\begin{cases} \langle \boldsymbol{w}_k \rangle & = & \boldsymbol{0} \\ \langle \boldsymbol{w}_k \boldsymbol{w}_k^T \rangle & \equiv & \boldsymbol{Q}_k \end{cases}$$

Measurement Equation

$$\boldsymbol{m}_k = \boldsymbol{h}_k(\boldsymbol{a}_{k,t}) + \boldsymbol{\epsilon}_k$$

measurement noise

true measurement vector at Site (k)

measurement vector at Site (k)

Assume that measurement noise is random and unbiased

$$\begin{cases} \langle \boldsymbol{\epsilon}_k \rangle & = & \boldsymbol{0} \\ \langle \boldsymbol{\epsilon}_k \boldsymbol{\epsilon}_k^T \rangle & \equiv & \boldsymbol{V}_k \equiv \boldsymbol{G}_k^{-1} \end{cases}$$

What We Need = Recurrence Formulae

Machineary to do:

(i) Prediction

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''>k} \quad : \text{ future}$$

(ii) Filtering

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''=k} \quad : \text{ present}$$

(iii) Smoothing

$$\{\boldsymbol{m}_{k'}; k' \leq k\} \mapsto \boldsymbol{a}_{k''<k} \quad : \text{ past}$$

Typical Usage of Kalman Filter in Tracking

Typical Procedure for Tracking

outermost innermost

$$(1) \to (2) \to \cdots \to (k-1) \to (k) \to \cdots (n)$$

filtering

$$\begin{array}{ccccccccccc} & \boldsymbol{a}_2^1 & & & & \boldsymbol{a}_{k-1}^{k-2} & & \boldsymbol{a}_k^{k-1} & & & \boldsymbol{a}_n \\ \nearrow & \Downarrow & \nearrow & \cdots & \nearrow & \Downarrow & \nearrow & \Downarrow & \nearrow & \cdots & \nearrow \\ \boldsymbol{a}_1 & \boldsymbol{a}_2 & & & & \boldsymbol{a}_{k-1} & & \boldsymbol{a}_k & & & \end{array}$$

prediction

$$(1) \cdots \leftarrow (k) \leftarrow (k+1) \leftarrow \cdots \leftarrow (n-1) \leftarrow (n)$$

$$\boldsymbol{a}_1^n \cdots \Leftarrow \boldsymbol{a}_k^n \Leftarrow \boldsymbol{a}_{k+1}^n \Leftarrow \cdots \Leftarrow \boldsymbol{a}_{n-1}^n \Leftarrow \boldsymbol{a}_n$$

$$\boldsymbol{a}_n \equiv \boldsymbol{a}_n^n$$

smoothing

Extrapolation

$$\boldsymbol{a}_n \Rightarrow \boldsymbol{a}_{n+1}^n = \boldsymbol{a}_{IP} \qquad \boldsymbol{a}_0^n = \boldsymbol{a}_{\mathrm{cal}} \Leftarrow \boldsymbol{a}_1^n$$

prediction prediction

Alignment, Resolution Study, etc.

Need to eliminate point (k)

$$(1)\cdots\cdots\cdots(k-1) \qquad (k) \qquad (k+1)\cdots\cdots\cdots(n)$$

$\Downarrow$ Inverse Kalman Filter

$\boldsymbol{a}_k^{n*}$ Reference Track Param.

$\Downarrow$

$\boldsymbol{h}_k(\boldsymbol{a}_k^{n*})$ Expected Hit Position

$\Downarrow$

$\boldsymbol{r}_k^{n*} = \boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{a}_k^{n*})$ Residual to Look At

C++ Implementation Kalman Filter Library

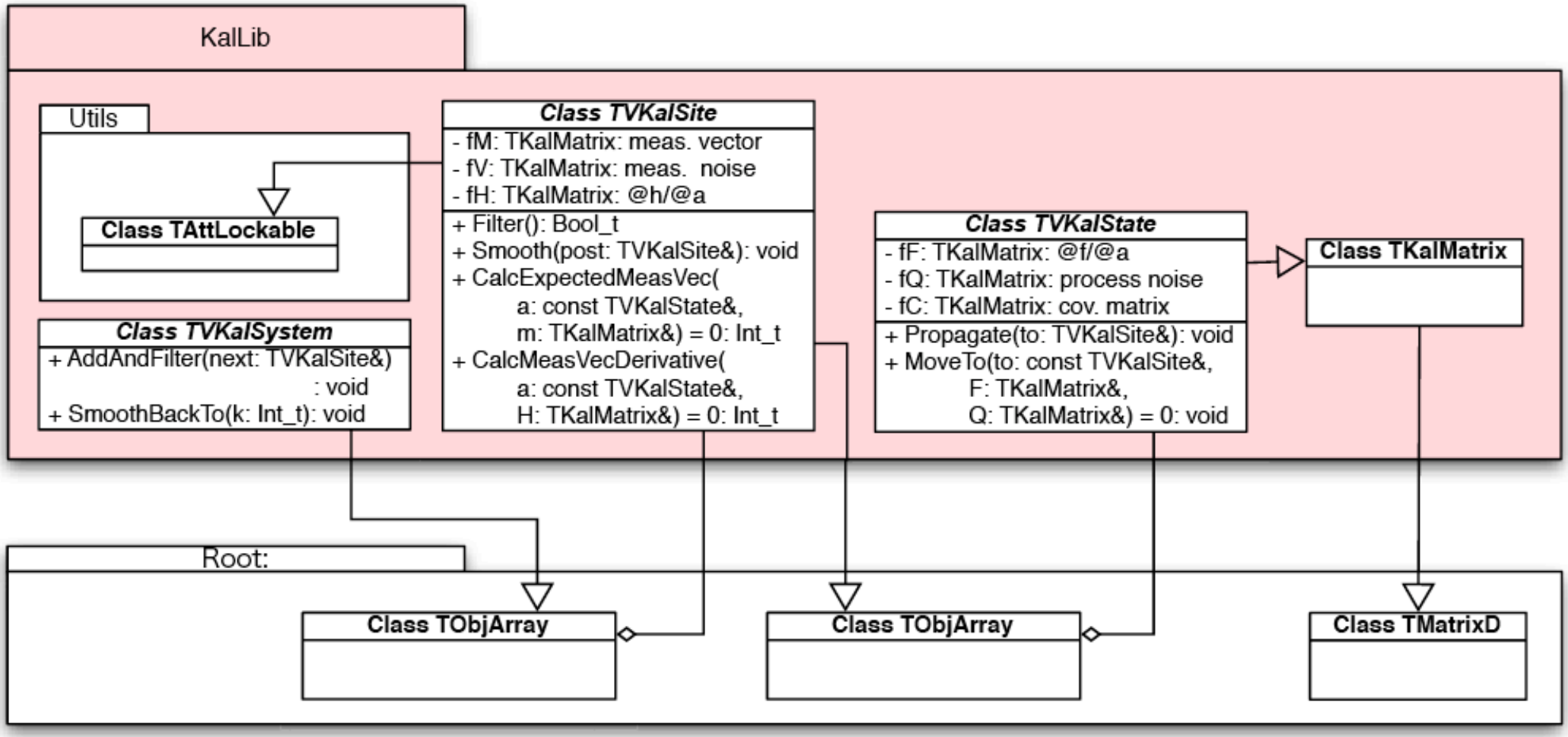
KF, Y.Nakashima, and A.Yamaguchi

Kalman Filter Library Features

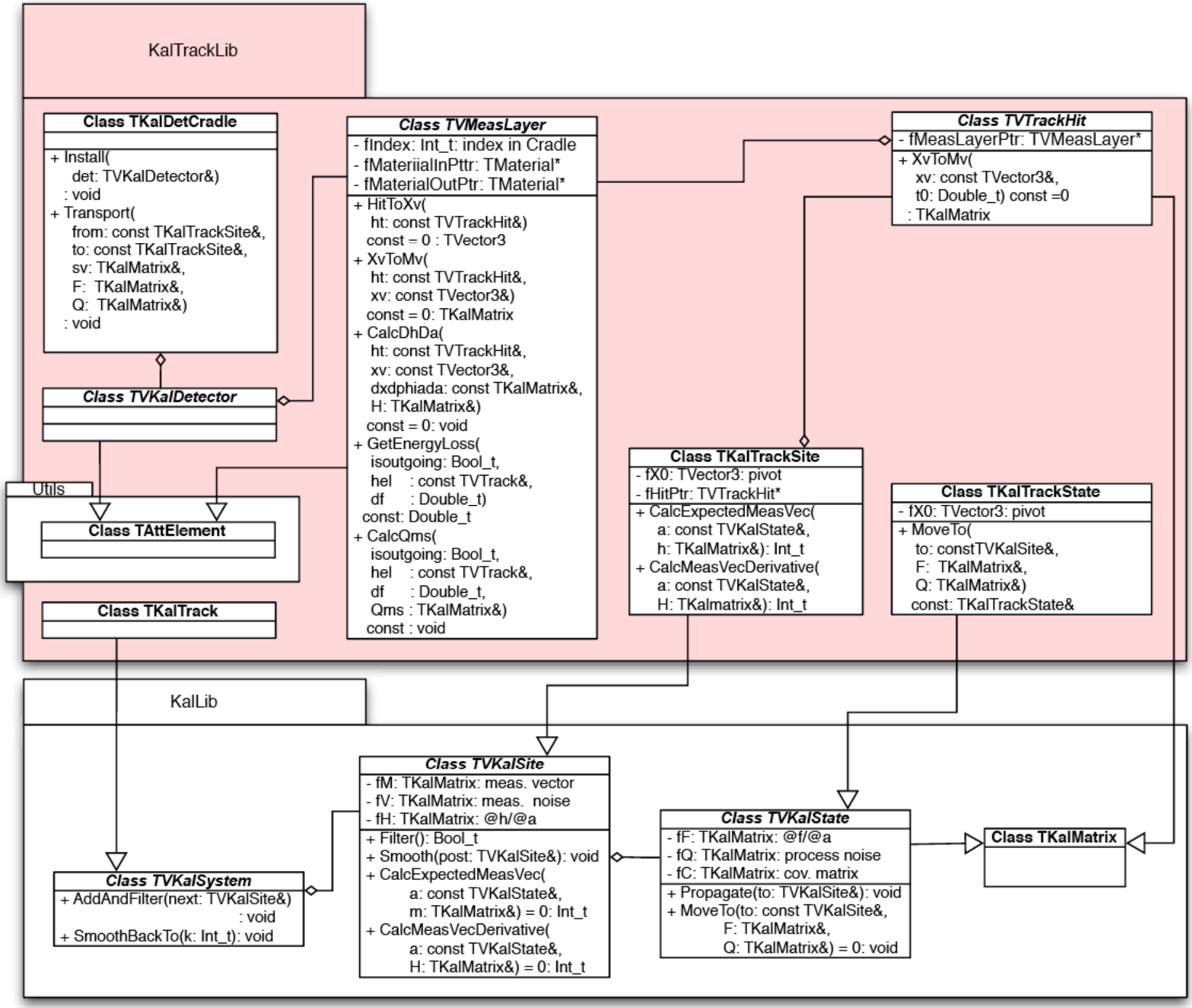
- KalLib: general base classes that implement algorithm
  - TVKalSystem, TVKalSite, TVKalState
- KalTrackLib: that implements pure virtuals of KalLib for track fitting purpose
- GeomLib: geometry classes that provide
  - track models (helix, straight line, ...)
  - surfaces (cylinder, hyperboloid, flat plane, ...)

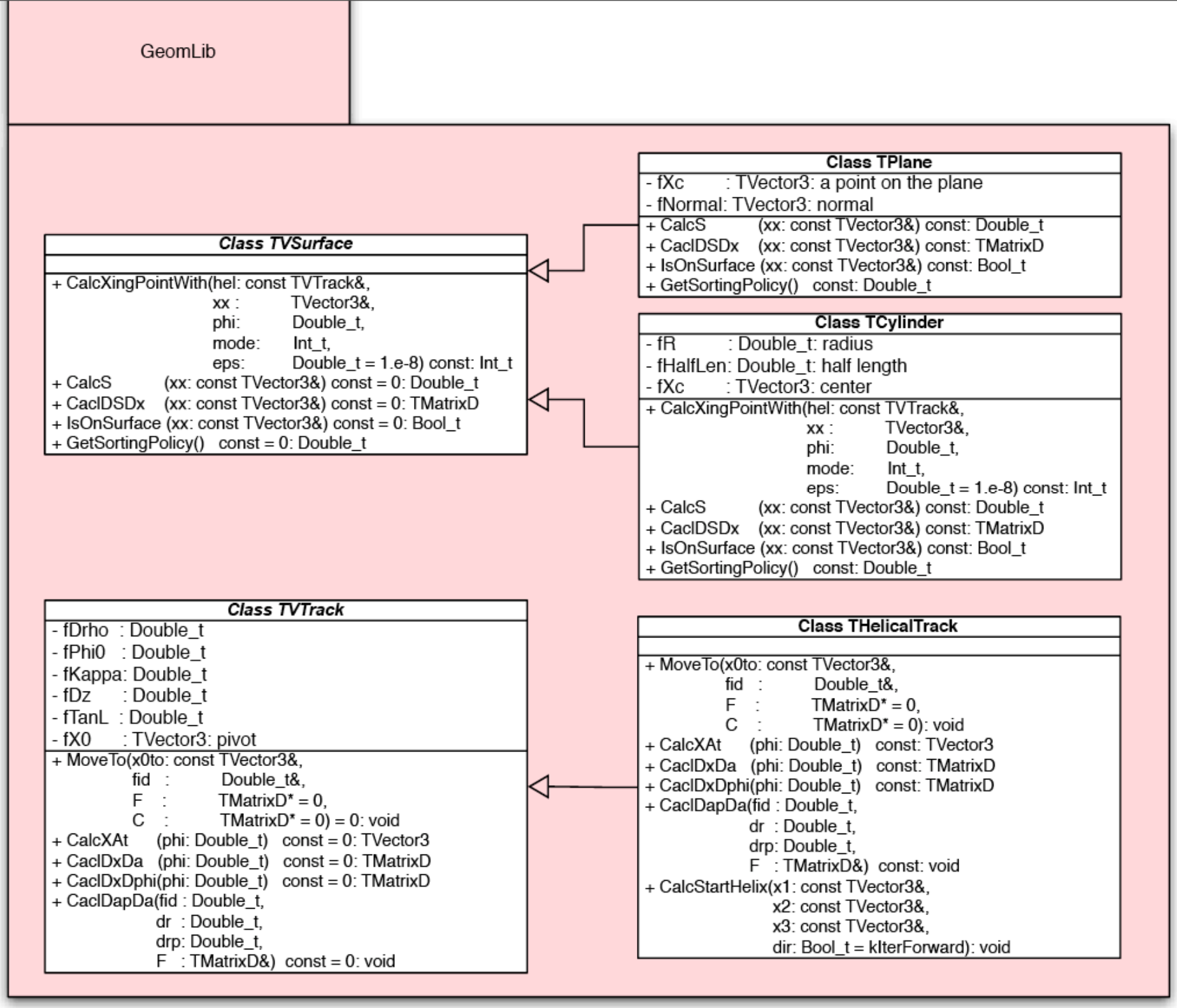
- Minimum number of user-implemented classes
  - MeasLayer : measurement layer
  - KalDetector : an array containing MeasLayers
    - You can put different kinds of MeasLayers
  - Hit : coordinate vector as defined by the MeasLayer
- Track model can change site to site which allows B-field variation along a particle trajectory

Kalman Filter Class Organization

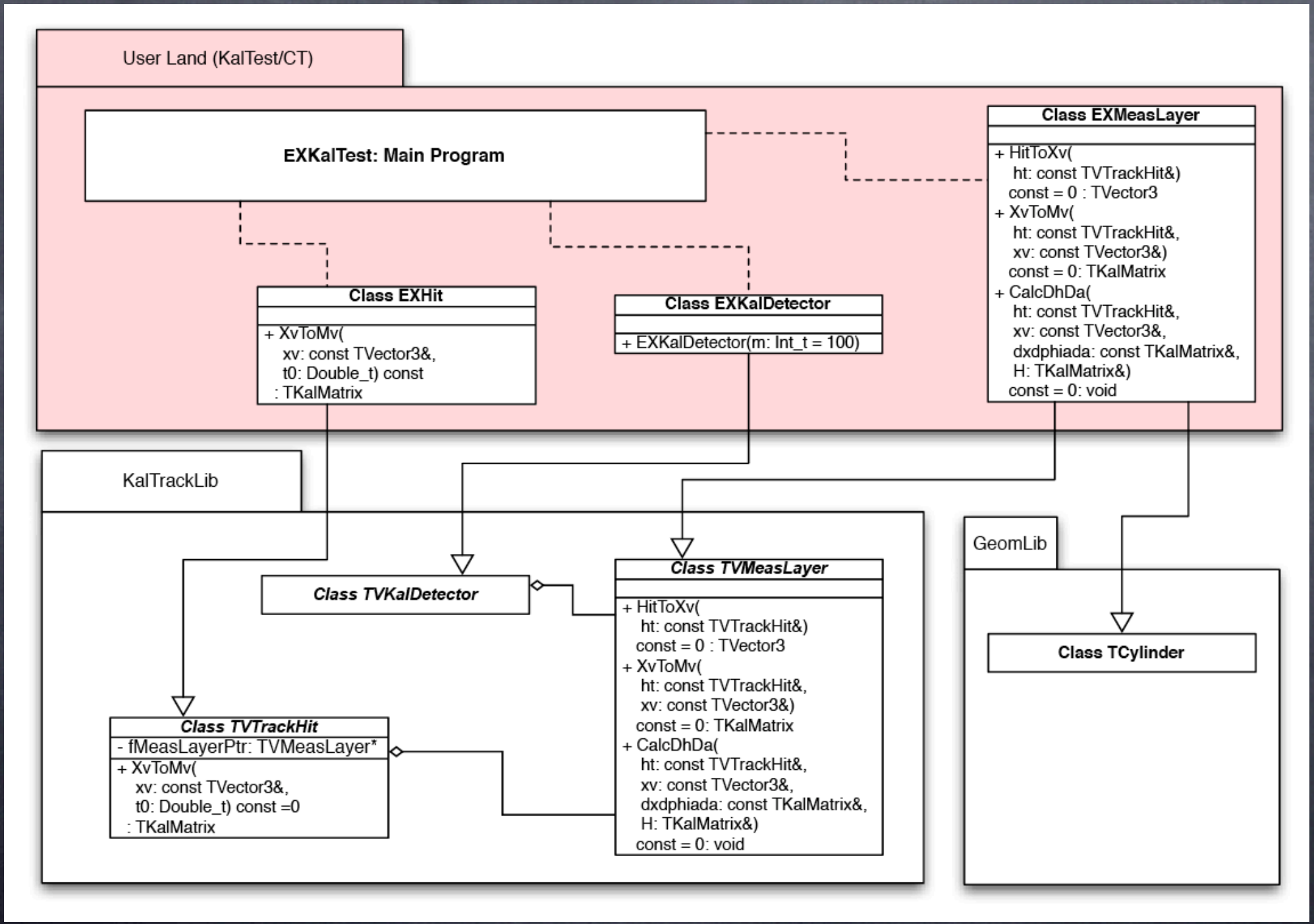


A TVKalSite carries predicted, filtered, and smoothed TVKalState's
Application-specific functions are pure virtual and to be implemented in a derived class


KalTrackLib
Class TKalDetCradle
+ Install(
det: TVKalDetector&)
: void
+ Transport(
from: const TKalTrackSite&,
to: const TKalTrackSite&,
sv: TKalMatrix&,
F: TKalMatrix&,
Q: TKalMatrix&)
: void
Class TVKalDetector
Class TVMeasLayer
- fIndex: Int_t: index in Cradle
- fMateriialInPttr: TMaterial*
- fMaterialOutPtr: TMaterial*
+ HitToXv(
ht: const TVTrackHit&)
const = 0 : TVector3
+ XvToMv(
ht: const TVTrackHit&,
xv: const TVector3&)
const = 0: TKalMatrix
+ CalcDhDa(
ht: const TVTrackHit&,
xv: const TVector3&,
dxdphiada: const TKalMatrix&,
H: TKalMatrix&)
const = 0: void
+ GetEnergyLoss(
isoutgoing: Bool_t,
hel : const TVTrack&,
df : Double_t)
const: Double_t
+ CalcQms(
isoutgoing: Bool_t,
hel : const TVTrack&,
df : Double_t,
Qms : TKalMatrix&)
const : void
Class TVTrackHit
- fMeasLayerPtr: TVMeasLayer*
+ XvToMv(
xv: const TVector3&,
t0: Double_t) const =0
: TKalMatrix
Class TKalTrackSite
- fX0: TVector3: pivot
- fHitPtr: TVTrackHit*
+ CalcExpectedMeasVec(
a: const TVKalState&,
h: TKalMatrix&): Int_t
+ CalcMeasVecDerivative(
a: const TVKalState&,
H: TKalmatrix&): Int_t
Class TKalTrackState
- fX0: TVector3: pivot
+ MoveTo(
to: constTVKalSite&,
F: TKalMatrix&,
Q: TKalMatrix&)
const: TKalTrackState&
Utils
Class TAttElement
Class TKalTrack
KalLib
Class TVKalSite
- fM: TKalMatrix: meas. vector
- fV: TKalMatrix: meas. noise
- fH: TKalMatrix: @h/@a
+ Filter(): Bool_t
+ Smooth(post: TVKalSite&): void
+ CalcExpectedMeasVec(
a: const TVKalState&,
m: TKalMatrix&) = 0: Int_t
+ CalcMeasVecDerivative(
a: const TVKalState&,
H: TKalMatrix&) = 0: Int_t
Class TVKalSystem
+ AddAndFilter(next: TVKalSite&)
: void
+ SmoothBackTo(k: Int_t): void
Class TVKalState
- fF: TKalMatrix: @f/@a
- fQ: TKalMatrix: process noise
- fC: TKalMatrix: cov. matrix
+ Propagate(to: TVKalSite&): void
+ MoveTo(to: const TVKalSite&,
F: TKalMatrix&,
Q: TKalMatrix&) = 0: void
Class TKalMatrix


GeomLib
Class TPlane
- fXc : TVector3: a point on the plane
- fNormal: TVector3: normal
+ CalcS (xx: const TVector3&) const: Double_t
+ CaclDSDx (xx: const TVector3&) const: TMatrixD
+ IsOnSurface (xx: const TVector3&) const: Bool_t
+ GetSortingPolicy() const: Double_t
Class TVSurface
+ CalcXingPointWith(hel: const TVTrack&,
xx : TVector3&,
phi: Double_t,
mode: Int_t,
eps: Double_t = 1.e-8) const: Int_t
+ CalcS (xx: const TVector3&) const = 0: Double_t
+ CaclDSDx (xx: const TVector3&) const = 0: TMatrixD
+ IsOnSurface (xx: const TVector3&) const = 0: Bool_t
+ GetSortingPolicy() const = 0: Double_t
Class TCylinder
- fR : Double_t: radius
- fHalfLen: Double_t: half length
- fXc : TVector3: center
+ CalcXingPointWith(hel: const TVTrack&,
xx : TVector3&,
phi: Double_t,
mode: Int_t,
eps: Double_t = 1.e-8) const: Int_t
+ CalcS (xx: const TVector3&) const: Double_t
+ CaclDSDx (xx: const TVector3&) const: TMatrixD
+ IsOnSurface (xx: const TVector3&) const: Bool_t
+ GetSortingPolicy() const: Double_t
Class TVTrack
- fDrho : Double_t
- fPhi0 : Double_t
- fKappa: Double_t
- fDz : Double_t
- fTanL : Double_t
- fX0 : TVector3: pivot
+ MoveTo(x0to: const TVector3&,
fid : Double_t&,
F : TMatrixD* = 0,
C : TMatrixD* = 0) = 0: void
+ CalcXAt (phi: Double_t) const = 0: TVector3
+ CaclDxDa (phi: Double_t) const = 0: TMatrixD
+ CaclDxDphi(phi: Double_t) const = 0: TMatrixD
+ CaclDapDa(fid : Double_t,
dr : Double_t,
drp: Double_t,
F : TMatrixD&) const = 0: void
Class THelicalTrack
+ MoveTo(x0to: const TVector3&,
fid : Double_t&,
F : TMatrixD* = 0,
C : TMatrixD* = 0): void
+ CalcXAt (phi: Double_t) const: TVector3
+ CaclDxDa (phi: Double_t) const: TMatrixD
+ CaclDxDphi(phi: Double_t) const: TMatrixD
+ CaclDapDa(fid : Double_t,
dr : Double_t,
drp: Double_t,
F : TMatrixD&) const: void
+ CalcStartHelix(x1: const TVector3&,
x2: const TVector3&,
x3: const TVector3&,
dir: Bool_t = kIterForward): void

Sample User Program

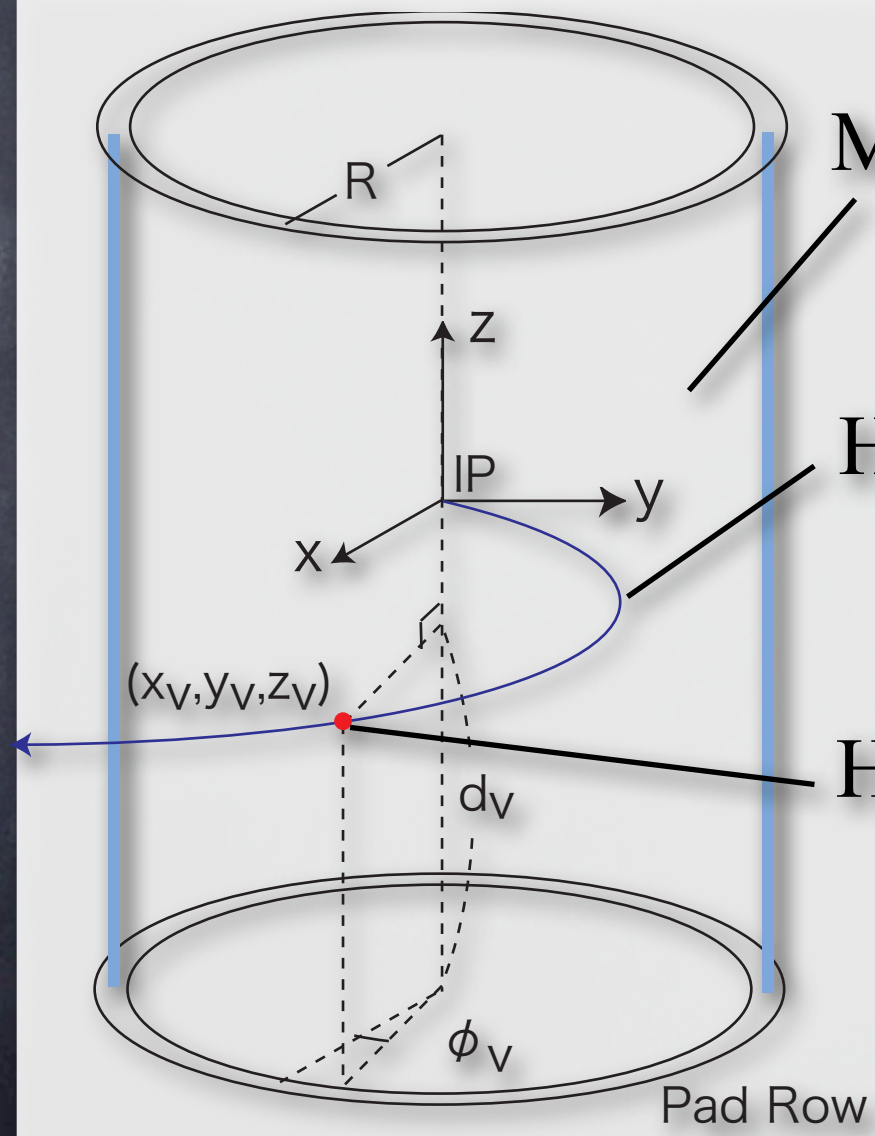


Application to ILC Track Fitting

Example of Detector Implementation

TPC Implementation

Define a KalDetector (TPCKalDetector) inheriting TVKalDetector



Measurement Layer

$$S(x,y,z) = x^2 + y^2 - R^2 = 0$$

Helix

$$\begin{cases} x = x_0 + d_\rho \cos\phi_0 + \frac{\alpha}{\kappa}(\cos\phi_0 - \cos(\phi_o + \phi)) \\ y = y_0 + d_\rho \sin\phi_0 + \frac{\alpha}{\kappa}(\sin\phi_0 - \sin(\phi_o + \phi)) \\ z = z_0 + d_z - \frac{\alpha}{\kappa}\tan\lambda \cdot \phi \end{cases}$$

Hit

$$\begin{pmatrix} R \cdot \phi_v \\ d_v \end{pmatrix} = \begin{pmatrix} R \cdot \tan^{-1}\left(\frac{y_v}{x_v}\right) \\ L/2 \mp z_v \end{pmatrix}$$

Define TPCMeasLayer by inheriting TVMeasLayer and implement its pure virtual methods:

HitToXv

$$\begin{pmatrix} x_v \\ y_v \\ z_v \end{pmatrix} = \begin{pmatrix} R \cdot \cos\phi_v \\ R \cdot \sin\phi_v \\ \pm(L/2 - d_v) \end{pmatrix}$$

XvToMv

$$\begin{pmatrix} R \cdot \phi_v \\ d_v \end{pmatrix} = \begin{pmatrix} R \cdot \tan^{-1}\left(\frac{y_v}{x_v}\right) \\ L/2 \mp z_v \end{pmatrix}$$

CalcDhDa

Meas.Vector Derivative w.r.t. Track Parameter Vector

$$\begin{pmatrix} \frac{\partial(R\cdot\phi_v)}{\partial \mathrm{a}} \\ \frac{\partial d_v}{\partial \mathrm{a}} \end{pmatrix} = \begin{pmatrix} -\frac{y_v}{R}\left(\frac{\partial x_v}{\partial \mathrm{a}}\right) + \frac{x_v}{R}\left(\frac{\partial y_v}{\partial \mathrm{a}}\right) \\ \mp \frac{\partial z_v}{\partial \mathrm{a}} \end{pmatrix}$$

$$\frac{\partial \mathrm{X}(\phi(\mathrm{a}),\mathrm{a})}{\partial \mathrm{a}} = \frac{\partial \mathrm{X}}{\partial \phi}\cdot\frac{\partial \phi}{\partial \mathrm{a}} + \frac{\partial \mathrm{X}}{\partial \mathrm{a}}$$

$$\frac{\partial \phi}{\partial \mathrm{a}} = -\frac{1}{\left(\frac{\partial S}{\partial \mathrm{X}}\cdot\frac{\partial \mathrm{X}}{\partial \phi}\right)}\frac{\partial S}{\partial \mathrm{X}}\cdot\frac{\partial \mathrm{X}}{\partial \mathrm{a}}$$

Notice that some TPCMeasLayer's may be implemented as dummy representing just boundaries of different materials

Add TPCMeasLayer's to TPCKalDetector with Add(..) method to complete TPC implementation

Integration of different trackers into a single tracking system

Define other trackers such as IT and VTX in a similar way and install them into TKalDetCradle as

```
TKalDetCradle    toygld;
VTXKalDetector   vtxdet;  toygld.Indtall(vtxdet);
ITKalDetector    itdet;   toygld.Install(itdet);
TPCKalDetector   tpcdet;  toygld.Install(tpcdet);
toygld.Sort();
```

Upon installation of each detector, its shell evaporates and only its MeasLayer's remain flatly expanded in the cradle
The last line sorts out the flatly expanded MeasLayer's from inside to outside

```
// ---------------------------
//  Add sited to the kaltrack
// ---------------------------

EXHYBTrack kaltrack;   // a track is a kal system
kaltrack.SetOwner();   // kaltrack owns sites
kaltrack.Add(&sited);  // add the dummy site to this track

// ---------------------------
//  Prepare hit iterrator
// ---------------------------

TIter nextsite(&kalhits, gkDir); // come in to IP, if gkDir = kIterBackward

// ---------------------------
//  Start Kalman Filter
// ---------------------------

TVTrackHit *hitp = 0;
while ((hitp = dynamic_cast<TVTrackHit *>(nextsite()))) {
  TKalTrackSite  &site = *new TKalTrackSite(*hitp); // new site
  if (!kaltrack.AddAndFilter(site)) {               // filter it
    cerr << " site discarded!" << endl;
    delete &site;                          // delete it if failed
  }
} // end of Kalman filter

// ---------------------------
//  Smooth the track
// ---------------------------

kaltrack.SmoothBackTo(0);
```

More information available from the following URL:

http://www-jlc.kek.jp/subg/offl/kaltest/

where you can find a reference manual for the KalTest package and some other useful documents.

The reference manual contains full derivations of relevant formulae for extended Kalman filter technique.