III. Physikalisches
Institut A

RWTH AACHEN UNIVERSITY

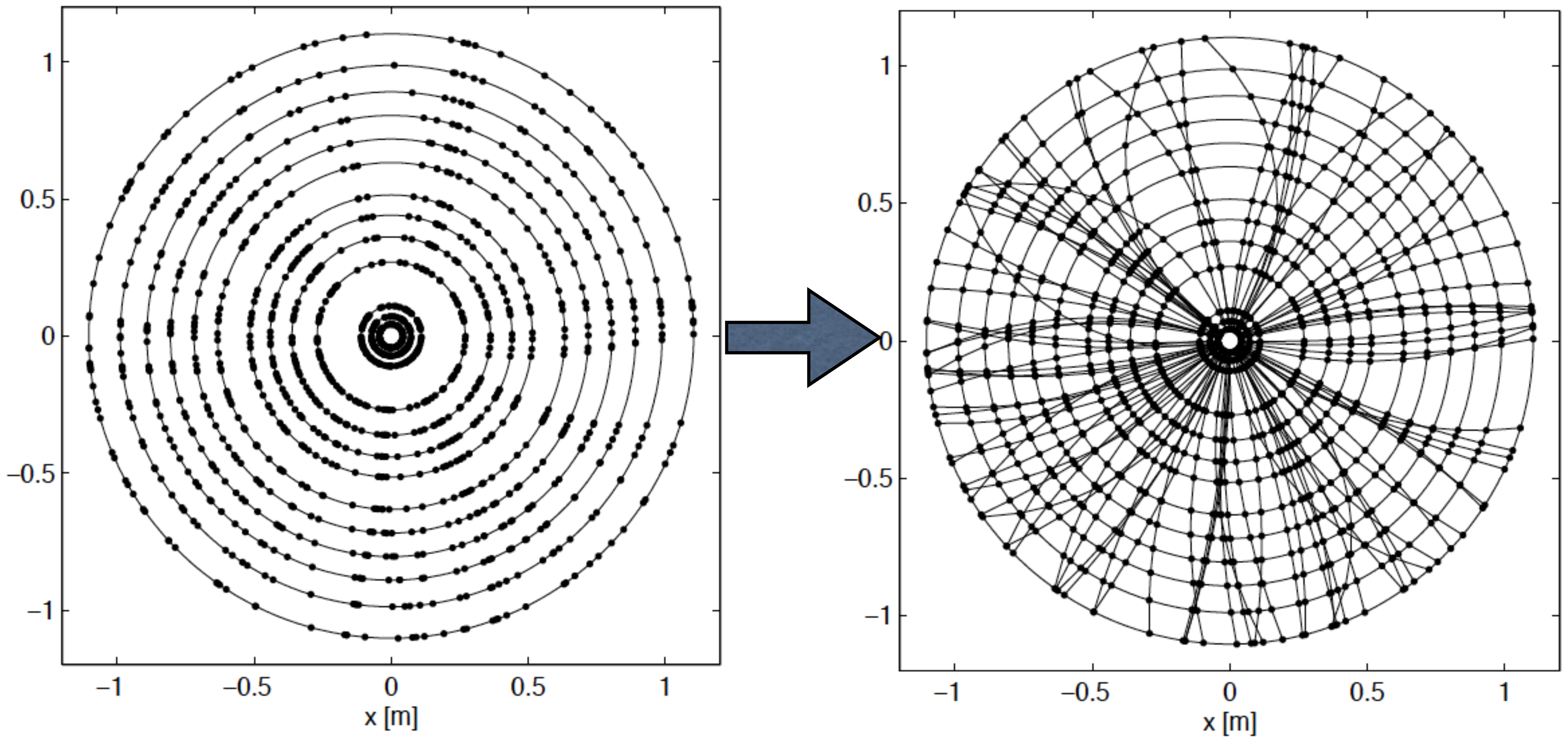
Experimental Techniques in Particle Physics (WS 2020/2021)

Track Reconstruction, Kalman Filter

Prof. Alexander Schmidt

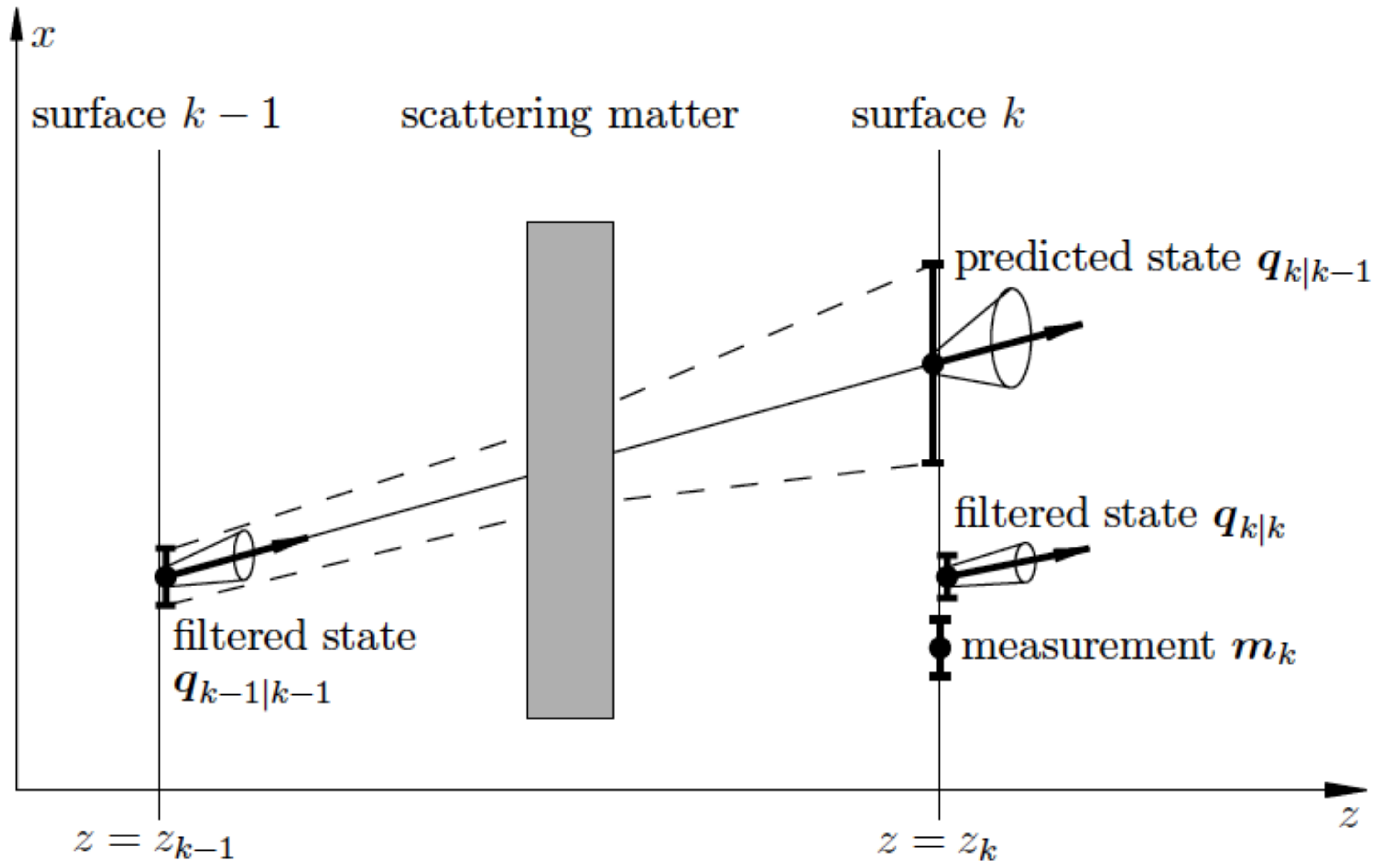
22.12.2020

Track reconstruction



- classification or pattern recognition problem
- multiple ambiguous hypotheses possible

Kalman filter algorithm, see details in Exercise



x
surface $k-1$
scattering matter
surface k
predicted state $\boldsymbol{q}_{k|k-1}$
filtered state $\boldsymbol{q}_{k|k}$
measurement $\boldsymbol{m}_k$
filtered state $\boldsymbol{q}_{k-1|k-1}$
$z = z_{k-1}$
$z = z_k$
z

Introduction to Kalman filter

"**Kalman filtering**, also known as **linear quadratic estimation** (**LQE**), is an algorithm that uses a series of measurements observed over time (or space), containing statistical noise and other inaccuracies, and produces **estimates** of unknown variables that are more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe."

R. Kalman (2006)

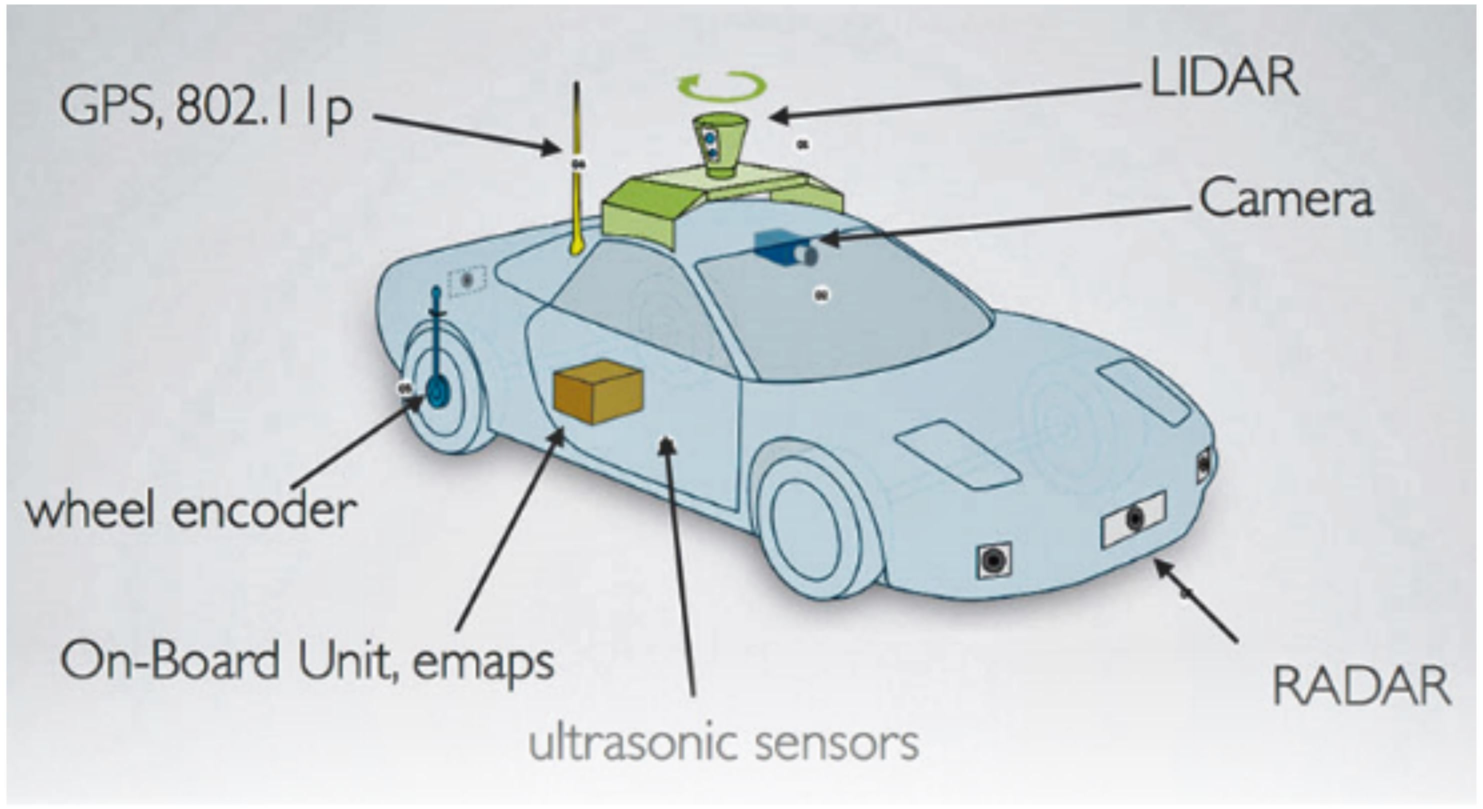
- The algorithm was discovered in 1960.
- One of the first spectacular applications was in the Apollo programme. (Trajectory estimation of the spacecraft with the Apollo computer)

- Countless applications in industry, research, medicine and military.
- Many derivatives have been developed

- State of the art in track finding and track fitting in particle physics experiments.

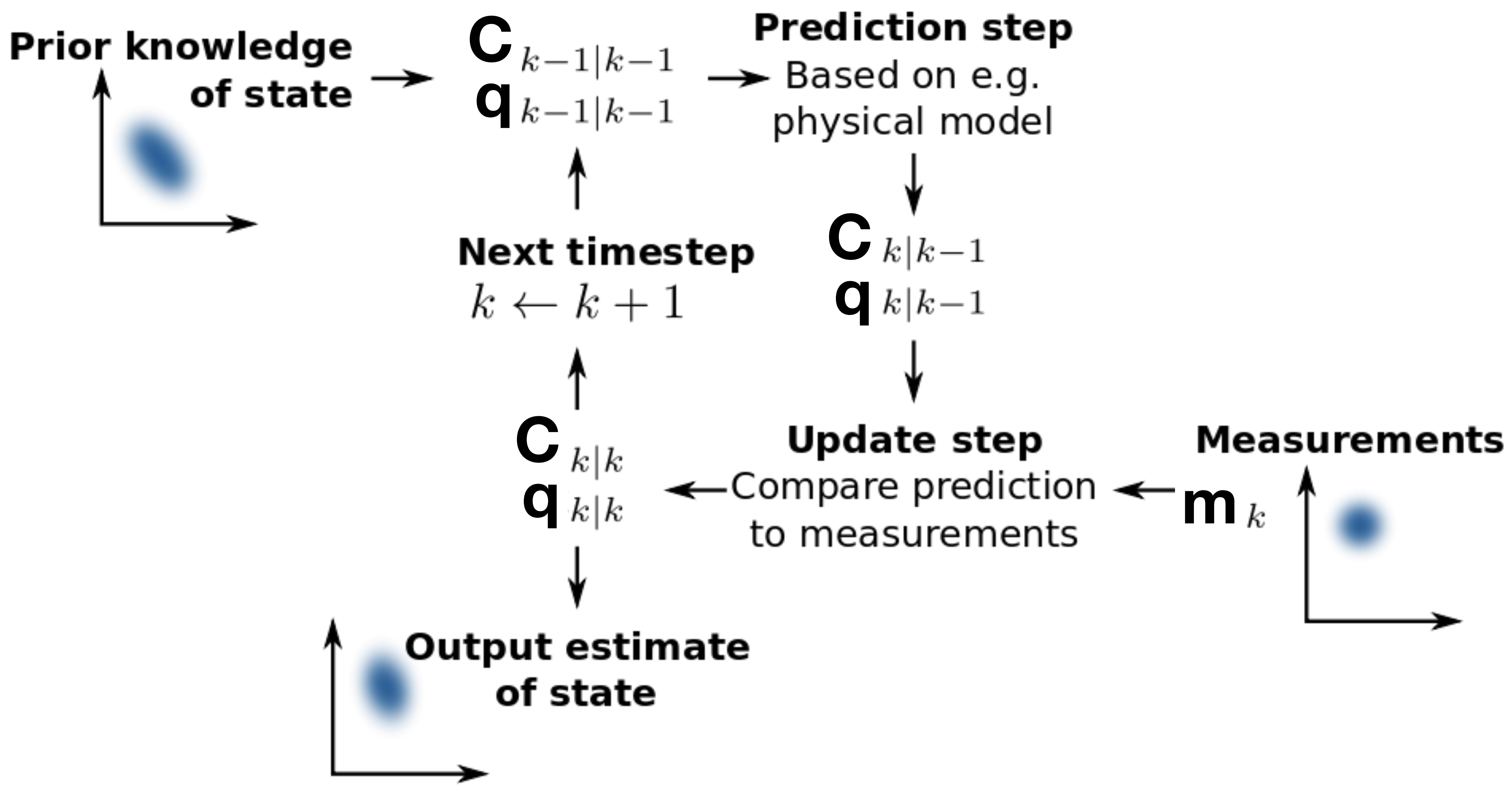
- Despite it's widespread applications, the fundamental mathematical description is rather simple.

Modern example application



- combine information from multiple sources to understand the environment around the car
- predict the next set of actions of the car in-front   (or person crossing the street)
- update the prediction whenever more information is available
- updating the prediction does NOT mean discarding information from the previous iteration (from a single very precise position measurement, we don't know the direction and velocity of an object)
- the uncertainties of the prediction are equally important

principle



- $\mathbf{q}$ is the state vector (e.g. track parameters)
- $\mathbf{C}$ is the uncertainty (covariance matrix)
- $\mathbf{m}$ is a measurement, which is usually not in perfect agreement with the prediction

ingredients

model of the state *q_i* at surface *i*:

- trajectory state described by analytical representation of a helix
- parameters **can** change from surface to surface (but they don't have to change, e.g. they are constant for a perfect helix in a homogeneous B Field)
- but the helix may change because of material interactions
- a helix has 5 parameters (the choice of parameters is experiment dependent)
- a helix is the solution of the equations of motion of a charged particle in a magnetic field

uncertainty of the state

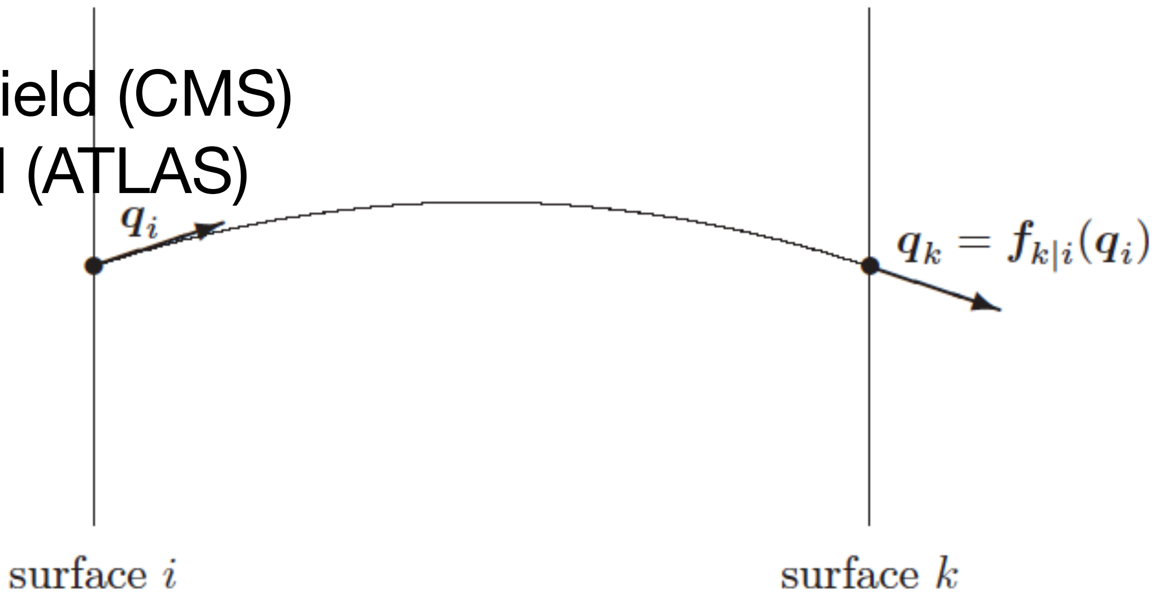
- 5x5 covariance matrix ***C_i***

propagator *f*:

- propagates the state from one surface *i* to the next surface *k*

$$\boldsymbol{q}_k = \boldsymbol{f}_{k|i}(\boldsymbol{q}_i)$$

- trivial in case of a homogeneous magnetic field (CMS)
- not trivial for inhomogeneous magnetic field (ATLAS) (may require numerical approximations)



ingredients

material effects

- trajectory distorted through material interactions
- ionisation (Bethe-Bloch), multiple scattering, …
- represented as additional uncertainty of the state
- covariance matrix $\boldsymbol{Q_{ik}}$
- “process noise” in the language of control theory

measurement model

- a measurement $\boldsymbol{m_k}$ of a hit in the tracker coordinate system
- note that the hit coordinates (e.g. xy) are NOT identical to the trajectory state parameters
- the connection between measurement and trajectory state is given through the measurement model $\boldsymbol{h_k}$

$$\boldsymbol{m}_k = \boldsymbol{h}_k(\boldsymbol{q}_k)$$

measurement uncertainty

- the hit positions in the tracker are associated with uncertainties
- covariance matrix $\boldsymbol{V_k}$
- this is often called measurement noise

some math

state transition model *$F_{k|i}$*:

- propagates the state from one surface *i* to the next surface *k* (it is the propagator in linearized matrix form)

- Jacobian Matrix calculated from propagator: $\boldsymbol{F}_{k|i} = \frac{\partial \boldsymbol{q}_k}{\partial \boldsymbol{q}_i}$

- note that the state of the helix can be identical from layer to layer for a perfect helix. The changes are ideally very small. (validity of linear approximation is assumed)

analogous treatment for measurement model:

- Jacobian Matrix: $\boldsymbol{H}_k = \frac{\partial \boldsymbol{m}_k}{\partial \boldsymbol{q}_k}$

propagation of the uncertainties:

- as the state, the covariance matrix is propagated from layer to layer

$$\boldsymbol{C}_k = \boldsymbol{F}_{k|i}\boldsymbol{C}_i\boldsymbol{F}_{k|i}^{\mathrm{T}} + \boldsymbol{Q}_k$$

Q is the process noise

some math

the core of the Kalman filter now consists of three lines

- the measurement and state prediction are combined into an updated state using the Kalman gain

$$\boldsymbol{q}_{k|k} = \boldsymbol{q}_{k|k-1} + \boldsymbol{K}_k[\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{q}_{k|k-1})]$$

- the Kalman gain matrix is given by

$$\boldsymbol{K}_k = \boldsymbol{C}_{k|k-1}\boldsymbol{H}_k^{\mathrm{T}}(\boldsymbol{V}_k + \boldsymbol{H}_k\boldsymbol{C}_{k|k-1}\boldsymbol{H}_k^{\mathrm{T}})^{-1}$$

- also the covariance matrix must be updated:

$$\boldsymbol{C}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{C}_{k|k-1}$$

comments

There is a mathematical proof that the Kalman formalism is optimal in case the model perfectly matches the real system, and the uncertainties are uncorrelated, and the covariances are exactly known. The proof uses a minimisation of the difference between true state and the estimated state (refer to literature).

comments to understand the expression:

$$\boldsymbol{q}_{k|k} = \boldsymbol{q}_{k|k-1} + \boldsymbol{K}_k[\boldsymbol{m}_k - \boldsymbol{h}_k(\boldsymbol{q}_{k|k-1})]$$

- if the projection is at the **identical** location as the measurement, there is no update to the predicted state, because ***m-h=0***
- otherwise, there is a linear update of the state proportional to ***m-h***
- the matrix ***K*** weights the difference between projection and measurement, based on their uncertainties
- note that ***m*** and ***h*** are in measurement coordinates, while ***K(m-h)*** is in trajectory state coordinates (helix parameters)

comments

comments to understand the expression:

$$\boldsymbol{K}_k = \boldsymbol{C}_{k|k-1}\boldsymbol{H}_k^{\mathrm{T}}(\boldsymbol{V}_k + \boldsymbol{H}_k\boldsymbol{C}_{k|k-1}\boldsymbol{H}_k^{\mathrm{T}})^{-1},$$

- ***HCH*** takes the projected state uncertainties into measurement coordinates
- ***V+HCH*** is the total uncertainty in measurement coordinates
- we can rewrite the expression:

$$K_k = H_k^{-1}\frac{H_k C_{k|k-1} H_k^T}{V_k + H_k C_{k|k-1} H_k^T}$$

- for small ***C***, or large **V** the expression goes to 0
  - → small uncertainties of the prediction and/or large uncertainties of the measurement means that the prediction has larger weight

- for large ***C***, or small **V** the expression goes to 1
  - → large uncertainties of the prediction and/or small uncertainties of the measurement means that the prediction has smaller weight

concrete implementation

we choose the trajectory parameterisation:

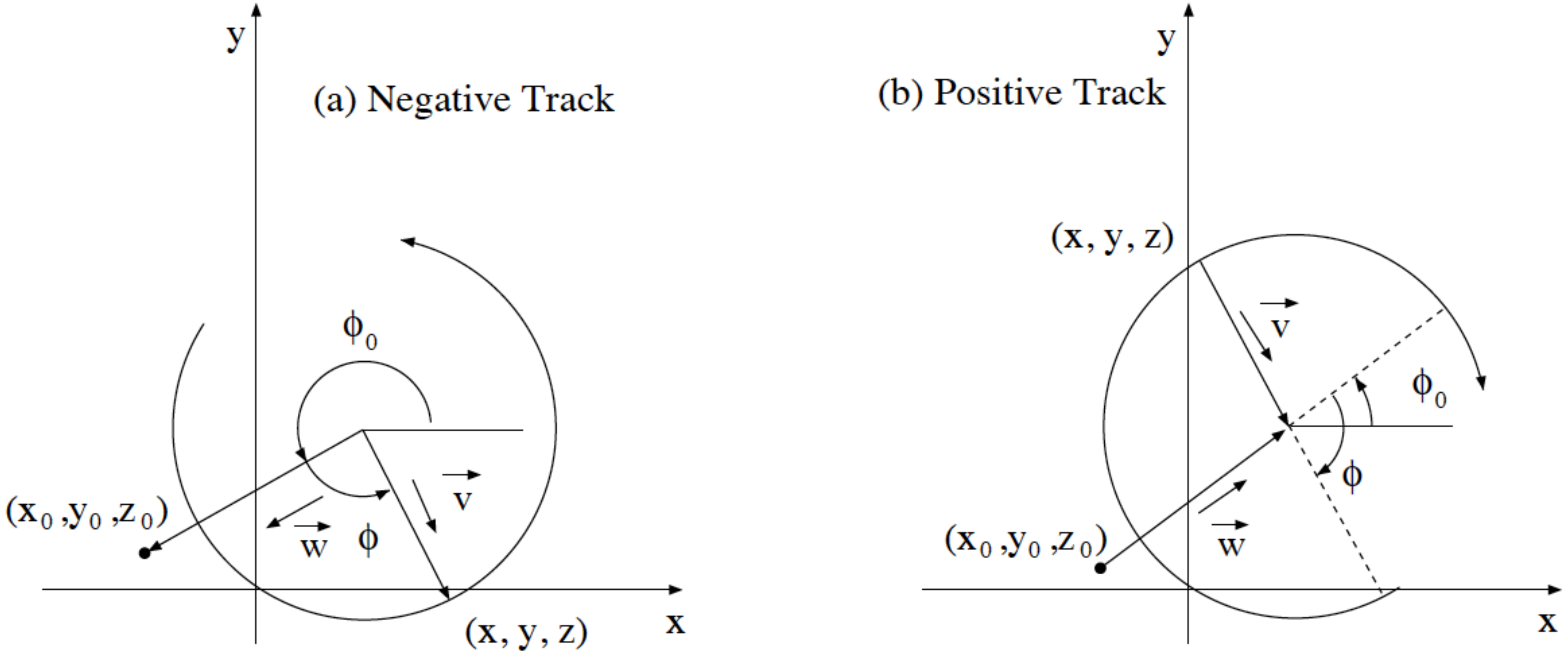
$$q = \mathbf{a} = (d_\rho, \phi_0, \kappa, d_z, \tan\lambda)^T$$

- ϕ_0 is the azimuthal angle to specify the pivot with respect to the helix center
- d_ρ is the signed distance of the helix from the pivot in x-y plane,
- κ is $1/P_t$ (reciprocal of the transverse momentum) and the sign of κ represents the charge of the track assigned by the track fitting,
- d_z is the signed distance of the helix from the pivot in the z direction,
- $\tan\lambda$ is the slope of the track, tangent of the dip angle.

if we choose (0,0,0) as the reference point and use 2D only, and assume a perfect helix going through (0,0,0) then:

- **$d_\rho = 0$**
- **$d_z = 0$**
- **$\tan\lambda = 0$**

concrete implementation



concrete implementation

the analytical position along the helix is given by:

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

where alpha is the magnetic-field-constant, $\alpha = 1/cB =$

The helix center[1] in x-y plane is

$$\begin{aligned} x_c &= x_0 + \left(d_\rho + \frac{\alpha}{\kappa}\right)\cos\phi_0 \\ y_c &= y_0 + \left(d_\rho + \frac{\alpha}{\kappa}\right)\sin\phi_0, \end{aligned}$$

the signed radius of the circle is

$$\rho = \frac{\alpha}{\kappa}$$

the track momentum is given by

$$P_t = \frac{1}{|\kappa|}$$

concrete implementation

state transition model $F_{k|i}$:

$$\boldsymbol{F} = \left(\frac{\partial \mathbf{a}'}{\partial \mathbf{a}}\right) = \begin{pmatrix} \frac{\partial d'_\rho}{\partial d_\rho} & \frac{\partial d'_\rho}{\partial \phi_0} & \frac{\partial d'_\rho}{\partial \kappa} & \frac{\partial d'_\rho}{\partial d_z} & \frac{\partial d'_\rho}{\partial \tan\lambda} \\ \frac{\partial \phi'_0}{\partial d_\rho} & \frac{\partial \phi'_0}{\partial \phi_0} & \frac{\partial \phi'_0}{\partial \kappa} & \frac{\partial \phi'_0}{\partial d_z} & \frac{\partial \phi'_0}{\partial \tan\lambda} \\ \frac{\partial \kappa'}{\partial d_\rho} & \frac{\partial \kappa'}{\partial \phi_0} & \frac{\partial \kappa'}{\partial \kappa} & \frac{\partial \kappa'}{\partial d_z} & \frac{\partial \kappa'}{\partial \tan\lambda} \\ \frac{\partial d'_z}{\partial d_\rho} & \frac{\partial d'_z}{\partial \phi_0} & \frac{\partial d'_z}{\partial \kappa} & \frac{\partial d'_z}{\partial d_z} & \frac{\partial d'_z}{\partial \tan\lambda} \\ \frac{\partial \tan\lambda'}{\partial d_\rho} & \frac{\partial \tan\lambda'}{\partial \phi_0} & \frac{\partial \tan\lambda'}{\partial \kappa} & \frac{\partial \tan\lambda'}{\partial d_z} & \frac{\partial \tan\lambda'}{\partial \tan\lambda} \end{pmatrix}$$

as discussed before, analytical expressions for a helix can be calculated this way

concrete implementation

$$
\begin{aligned}
\frac{\partial d'_\rho}{\partial d_\rho} &= \cos\phi'_0 \cos\phi_0 + \sin\phi'_0 \sin\phi_0 \\
\frac{\partial d'_\rho}{\partial \phi_0} &= \left(d_\rho + \frac{\alpha}{\kappa}\right)(\sin\phi'_0 cos\phi_0 - \cos\phi'_0 \sin\phi_0) \\
\frac{\partial d'_\rho}{\partial \kappa} &= \left(\frac{\alpha}{\kappa^2}\right)\{1 - (\cos\phi'_0 \cos\phi_0 + \sin\phi'_0 \sin\phi_0)\} \\
\frac{\partial d'_\rho}{\partial d_z} &= \frac{\partial d'_\rho}{\partial \tan\lambda} = 0 \\
\frac{\partial \phi'_0}{\partial d_\rho} &= -\frac{1}{d'_\rho + \frac{\alpha}{\kappa}}(\sin\phi'_0 \cos\phi_0 - \cos\phi'_0 \sin\phi_0) \\
\frac{\partial \phi'_0}{\partial \phi_0} &= \left(\frac{d_\rho + \frac{\alpha}{\kappa}}{d'_\rho + \frac{\alpha}{\kappa}}\right)(\cos\phi'_0 \cos\phi_0 + \sin\phi'_0 \sin\phi_0) \\
\frac{\partial \phi'_0}{\partial \kappa} &= \left(\frac{\alpha}{\kappa^2}\right)\frac{1}{d'_\rho + \frac{\alpha}{\kappa}}(\sin\phi'_0 \cos\phi_0 - \cos\phi'_0 \sin\phi_0) \\
\frac{\partial \phi'_0}{\partial d_z} &= \frac{\partial \phi'_0}{\partial \tan\lambda} = 0 \\
\frac{\partial \kappa'}{\partial \kappa} &= 1
\end{aligned}
$$

concrete implementation

$$
\begin{aligned}
\frac{\partial \kappa'}{\partial d_\rho} &= \frac{\partial \kappa'}{\partial \phi_0} = \frac{\partial \kappa'}{\partial d_z} = \frac{\partial \kappa'}{\partial \tan\lambda} = 0 \\
\frac{\partial d'_z}{\partial d_\rho} &= \left(\frac{\alpha}{\kappa}\right)\left(\frac{\tan\lambda}{d'_\rho + \frac{\alpha}{\kappa}}\right)(\sin\phi'_0 \cos\phi_0 - \cos\phi'_0 \sin\phi_0) \\
\frac{\partial d'_z}{\partial \phi_0} &= \left(\frac{\alpha}{\kappa}\right)\tan\lambda\left(1 - \left(\frac{d_\rho + \frac{\alpha}{\kappa}}{d'_\rho + \frac{\alpha}{\kappa}}\right)(\cos\phi'_0 \cos\phi_0 + \sin\phi'_0 \sin\phi_0)\right)
\end{aligned}
$$

note that only a few of these derivatives are needed for our concrete case

most of them are even trivial

concrete implementation

measurement model:

$$H = \left(\frac{\partial \mathbf{x}}{\partial \mathbf{a}}\right) = \begin{pmatrix} \frac{\partial x}{\partial d_\rho} & \frac{\partial x}{\partial \phi_0} & \frac{\partial x}{\partial \kappa} & \frac{\partial x}{\partial d_z} & \frac{\partial x}{\partial \tan\lambda} \\ \frac{\partial y}{\partial d_\rho} & \frac{\partial y}{\partial \phi_0} & \frac{\partial y}{\partial \kappa} & \frac{\partial y}{\partial d_z} & \frac{\partial y}{\partial \tan\lambda} \\ \frac{\partial z}{\partial d_\rho} & \frac{\partial z}{\partial \phi_0} & \frac{\partial z}{\partial \kappa} & \frac{\partial z}{\partial d_z} & \frac{\partial z}{\partial \tan\lambda} \end{pmatrix}$$

$$\begin{aligned} \frac{\partial x}{\partial d_\rho} &= \cos\phi_0 \\ \frac{\partial x}{\partial \phi_0} &= -d_\rho \sin\phi_0 + \left(\frac{\alpha}{\kappa}\right)\{-\sin\phi_0 + \sin(\phi_0 + \phi)\} \end{aligned}$$

concrete implementation

$$\begin{aligned}
\frac{\partial x}{\partial \kappa} &= -\left(\frac{\alpha}{\kappa^2}\right)\{\cos\phi_0 - \cos(\phi_0+\phi)\} \\
\frac{\partial x}{\partial d_z} &= \frac{\partial x}{\partial \tan\lambda} = 0 \\
\frac{\partial y}{\partial d_\rho} &= \sin\phi_0 \\
\frac{\partial y}{\partial \phi_0} &= d_\rho\cos\phi_0 + \left(\frac{\alpha}{\kappa}\right)\{\cos\phi_0 - \cos(\phi_0+\phi)\} \\
\frac{\partial y}{\partial \kappa} &= -\left(\frac{\alpha}{\kappa^2}\right)\{\sin\phi_0 - \sin(\phi_0+\phi)\} \\
\frac{\partial y}{\partial d_z} &= \frac{\partial y}{\partial \tan\lambda} = 0 \\
\frac{\partial z}{\partial d_\rho} &= \frac{\partial z}{\partial \phi_0} = 0 \\
\frac{\partial z}{\partial \kappa} &= \left(\frac{\alpha}{\kappa^2}\right)\tan\lambda\phi \\
\frac{\partial z}{\partial d_z} &= 1 \\
\frac{\partial z}{\partial \tan\lambda} &= -\left(\frac{\alpha}{\kappa}\right)\phi.
\end{aligned}$$

comments

- this form of the linear Kalman filter can suffer from numerical instabilities (convergence problems in case of matrix multiplication with very small floating point numbers)
- can add some stabilisation noise

- it can also diverge or become unstable if
    - the model does not describe the true dynamics of the system well enough
    - the initial state (“seed”) does not cover the true state within uncertainties
    - the measurement is far away from the predicted state (not compatible within uncertainties)
    - the state is not observable based on the measurement information

comments

- for the exercise we need linear algebra (matrix multiplications and inversions)

- we could also use `TMatrixD` from root
- however, there are numerical problems with `TMatrixD`

- instead we will use the Eigen library for matrix algebra

http://eigen.tuxfamily.org

- simply download it and include it in your Jupyter code with

```
#include “eigen-3.3.9/Eigen/Dense"
```

in a separate cell on top of everything

Installation

- installation instructions for Eigen:

- open a terminal in jupyter, go to your working directory
- dowload it:

```
wget https://gitlab.com/libeigen/eigen/-/archive/3.3.9/eigen-3.3.9.tar.gz
```

- upack it:

```
tar xvzf eigen-3.3.9.tar.gz
```

- use it by putting the following line at the top of your jupyter code in a separate cell:

```
#include “eigen-3.3.9/Eigen/Dense"
```

comments

- Matrix algebra with Eigen:

`Matrix2d a;`	creates a 2-dimensional matrix with double precision entries
`a << 1, 2, 3, 4;`	fills the matrix elements (can also be accessed with a(0,0),…
`a+b;`	sum of two matrices (must be of same size)
`a.transpose()`	transpose of matrix a

exercise

- we start with track **fitting** (not track finding)
  (this means we know which hit belongs to which track, and we want to get the track parameters from the hits)

- follow the trajectory of one single track with known hits
- we start with the same original file as last week (`TrackerHits1.root`)

- start with a well guessed “seed”
  (in reality a seed is calculated from all hit doublet/triplets in the inner layers)

- assume some approximate initial covariance matrix (note the comments about this in the previous slides)

- propagate the trajectory from layer to layer, updating it with the next hits
- you can use 1 µm as position uncertainties for the hits

- use the skeleton with pre-defined trajectory state classes and propagation

- we will need more than 1 week for this exercise…