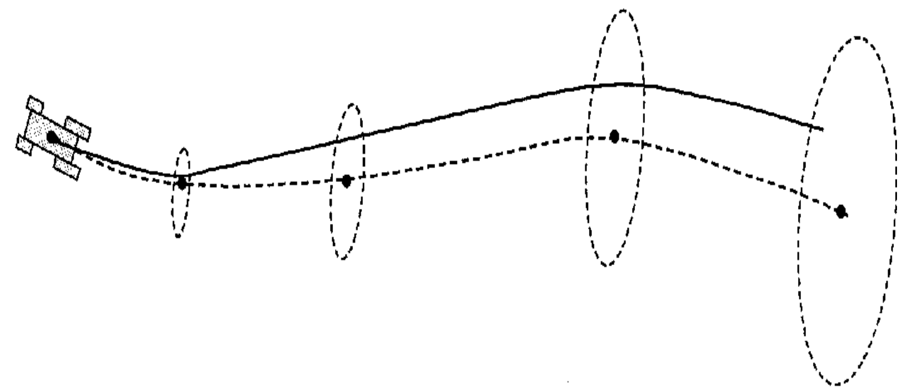
Probabilistic Localization

Relative Localization

- dead-reckoning / odometry, etc.
- accumulation of error



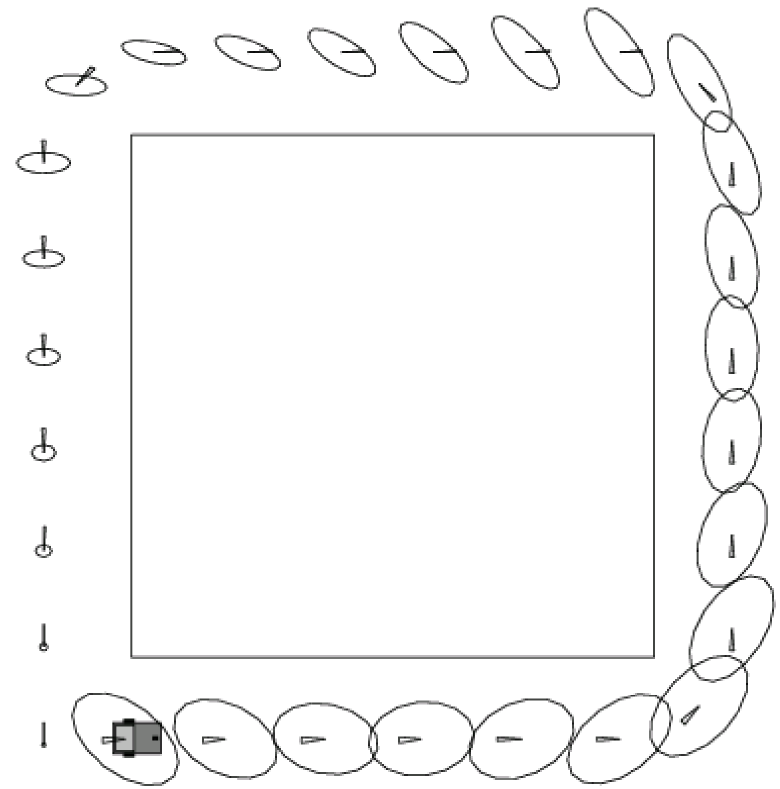
⇒how to
- keep track?
- correct when possible?
  (using global localization feedback)

Localization Error Representation

- 3-dim Gaussian for pose
  - expected x, y, theta
  - plus covariances
- noisy motion estimates increase uncertainty

note: approximate model

- ignores non-Gaussian noise,
- e.g., bump noise



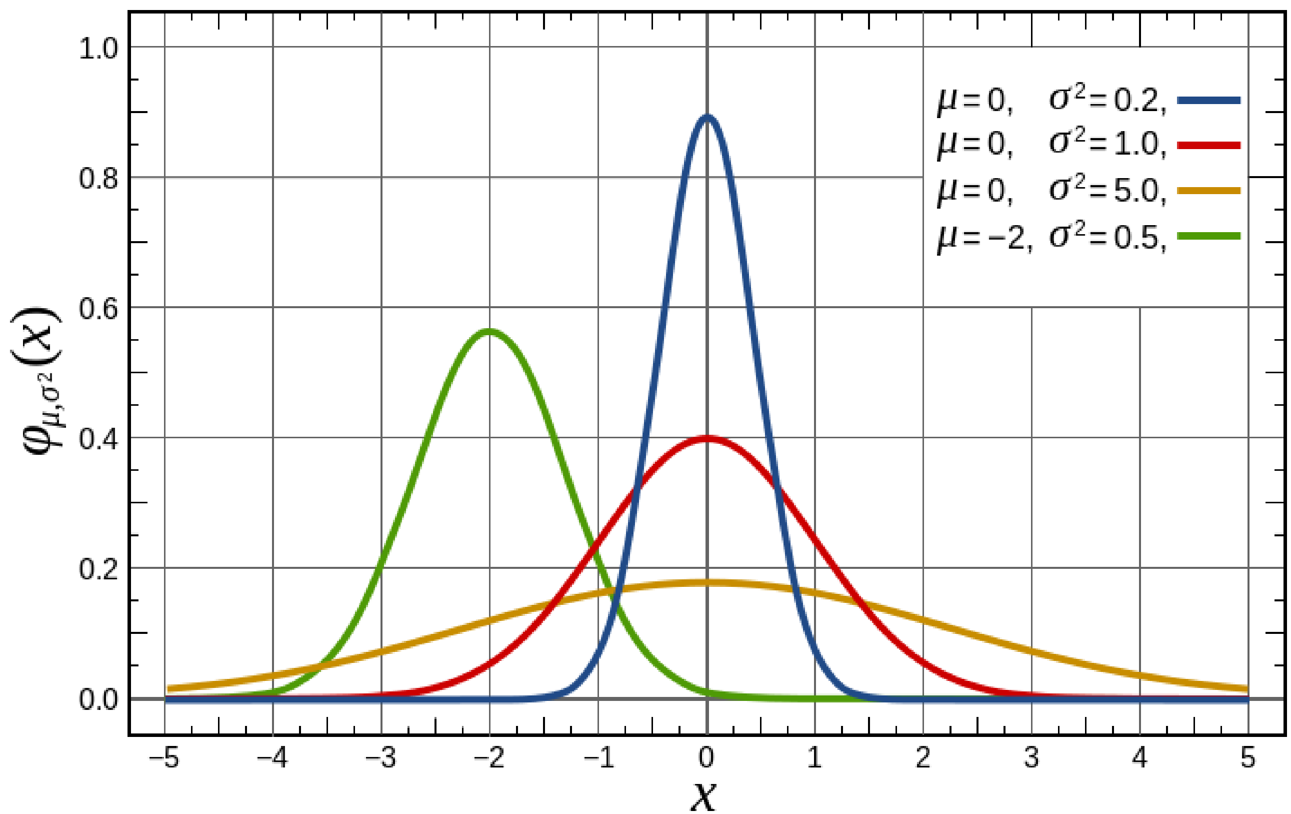
example for x,y

Gaussian

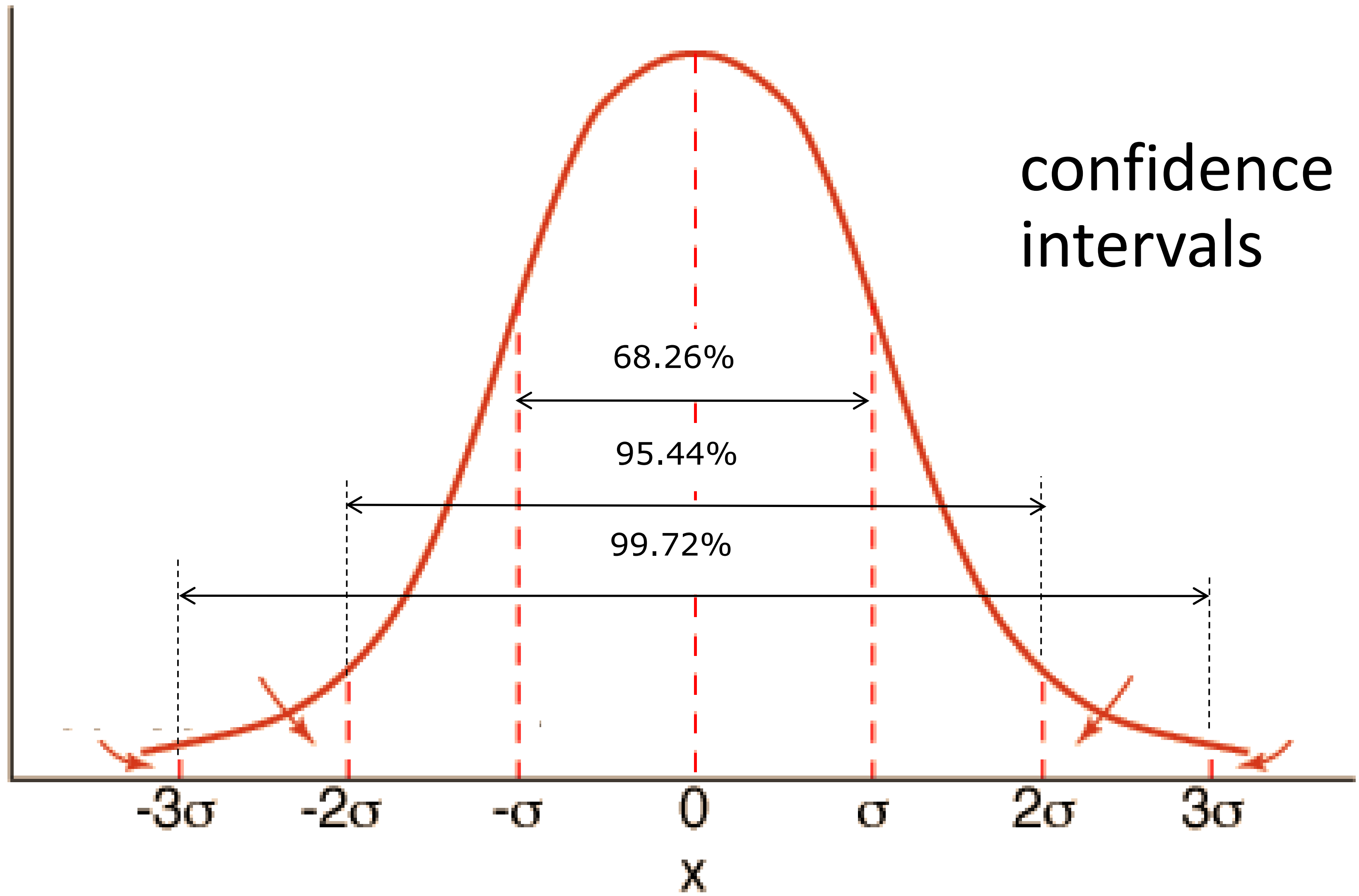
Gaussian (aka Normal) distribution

- mean μ (aka expectation)
- variance σ^2 (standard deviation σ)

$$f_{Gaussian}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$
$$= N(\mu, \sigma)$$



Gaussian

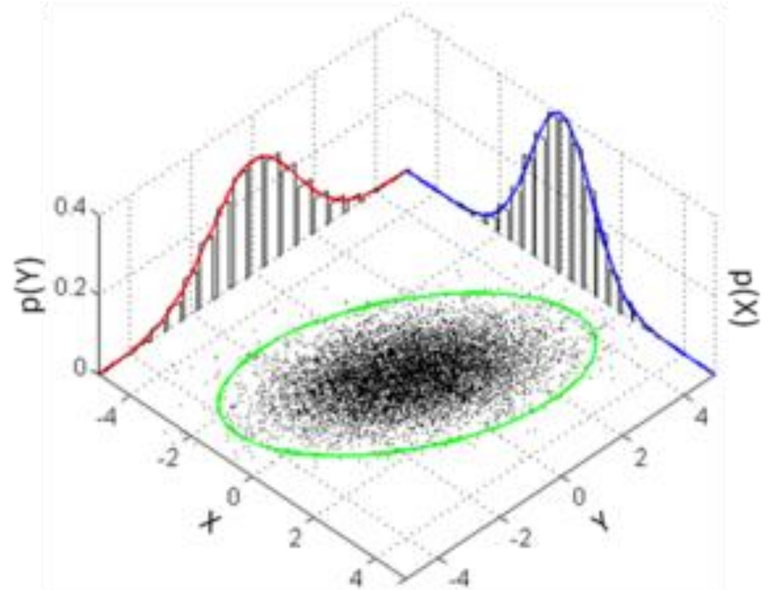


confidence intervals

Multivariate Gaussian

- distribution over *k* random variables $x = (x_1, \ldots, x_k)$
- mean vector μ, covariance matrix Σ

$$f_{Gaussian}(x) = (2\pi)^{-\frac{k}{2}} |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)} = N(\mu, \Sigma)$$



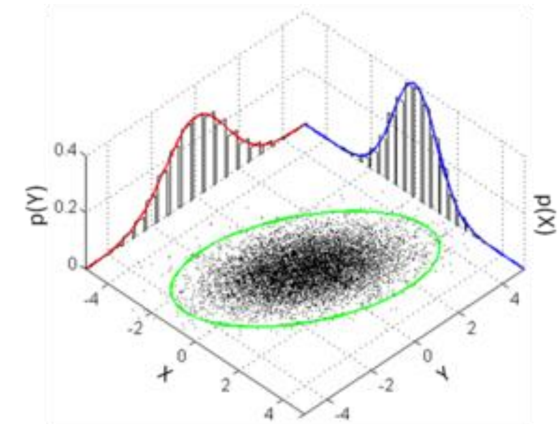
notes:

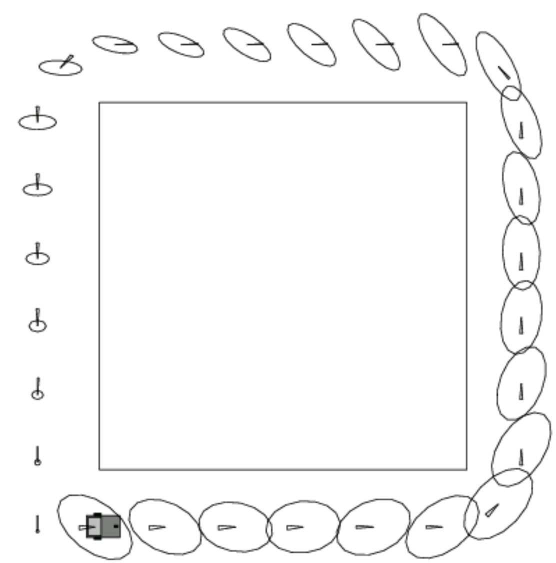
- |A| = det(A) (determinant)
- robotics, often covariance as *C* (not *Σ*)
- confidence intervals become confidence regions in form of (hyper)ellipsoids

Localization Error Visualization

(2D) localization error
= Gaussian in (x, y, theta)

- just consider x,y for display
- equidensity contours
  - contours with equal prob. mass
  - are ellipsoids for Gaussian
- principal axes
  - directions = eigenvectors of Σ
  - squared (relative) lengths = corresponding eigenvalues





Error Propagation

more precisely, uncertainty propagation, i.e.,

- development of covariance *C*
- of random vector x under function f()

case 1: linear fct f(), i.e., f(x)=Ax $$C^f = ACA^T$$

case 2: arbitrary fct f() with Jacobian J_f $$C^f = J_f C J_f^T$$

(note: J_A = A)

Error Propagation

covariance C, fct f() with Jacobian J_f

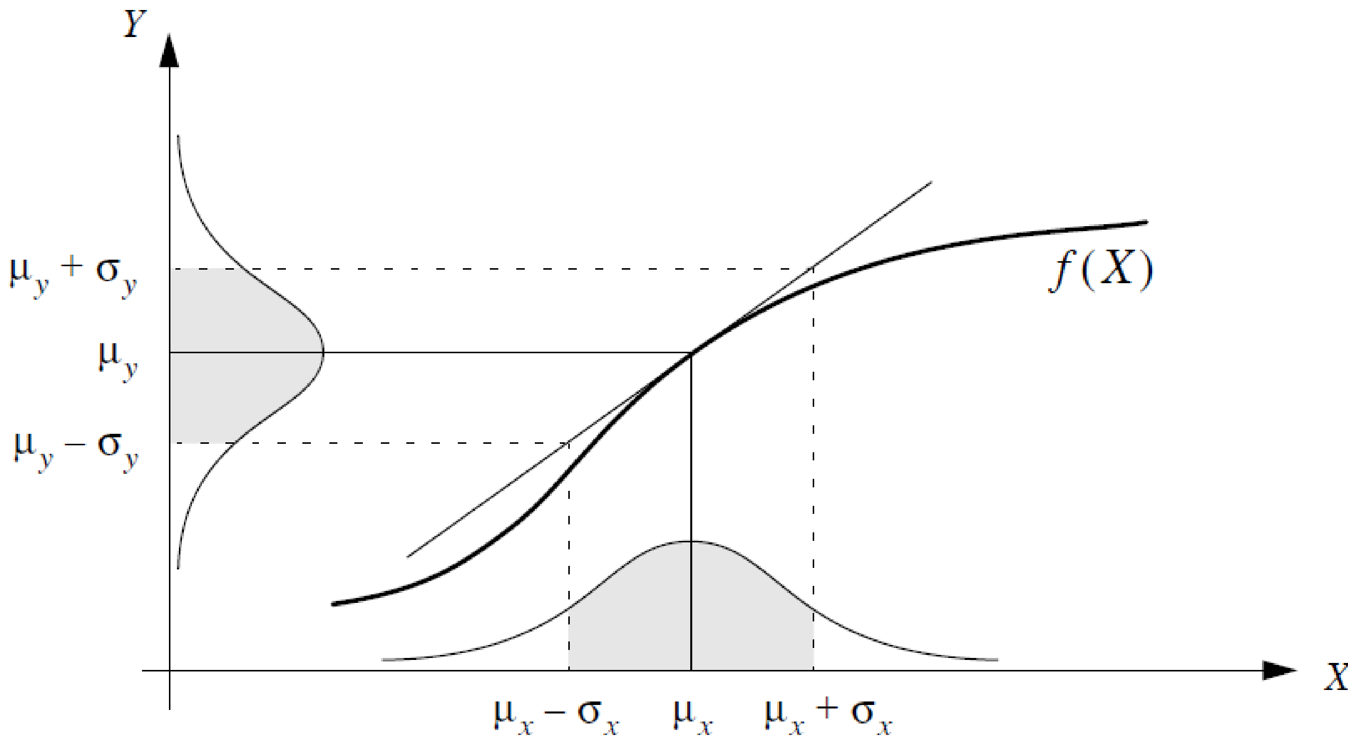
$$C^f = J_f C J_f^T$$

- aka *error propagation rule*
  or even *error propagation law*
  (though more "dirty hack" than "law" ☺)
- based on first order Taylor approximation
- i.e., linearization at point c $f(x) \approx f(c) + J_f (x - c)$

Error Propagation

e.g., 1-dim: consider [μ-σ, μ+σ]

$$Y = f(X) \approx f(\mu_x) + \left.\frac{\partial f}{\partial X}\right|_{X=\mu_x} (X - \mu_x)$$



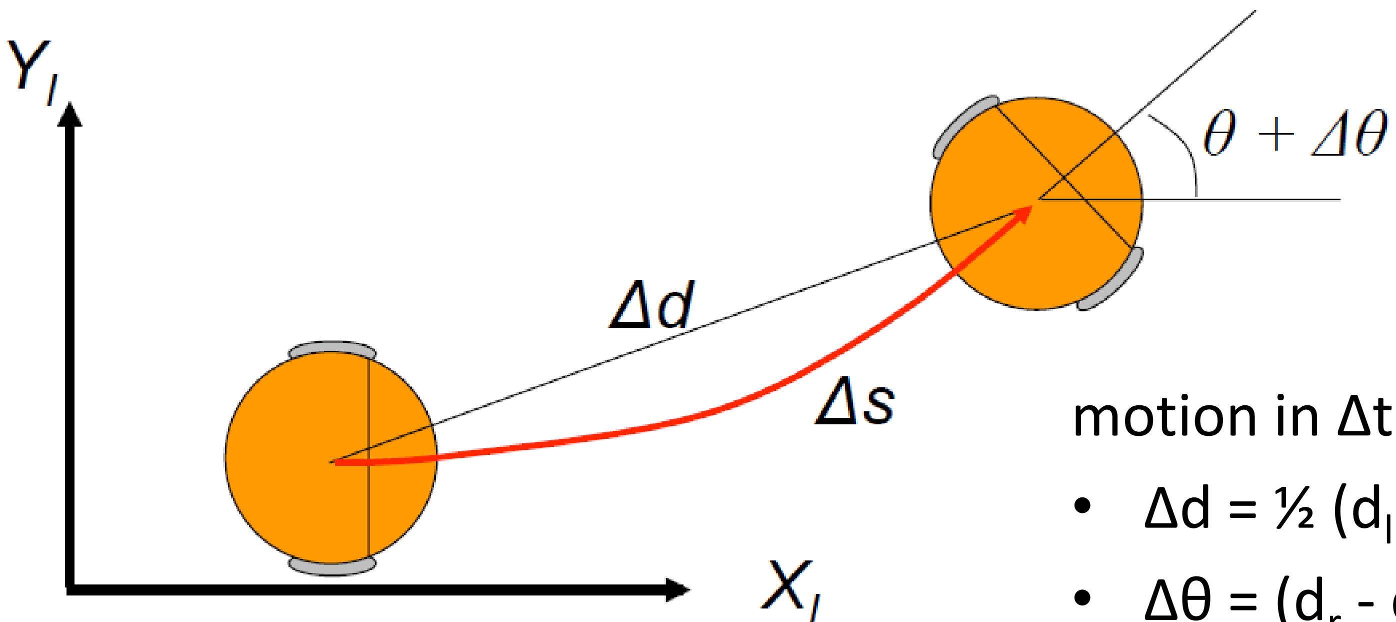
$$\mu_Y = f(\mu_X)$$

$$\sigma_Y = \left.\frac{\partial f}{\partial X}\right|_{X=\mu_x} \sigma_X$$

Example: Differential Drive Odometry

recap: incremental time updates Δt

- $d_{l/r}$: distance by left/right wheel in Δt ($d_{l/r} = v_{l/r}\,\Delta t$)
- robot drives arc, but Δt small: $\Delta d \approx \Delta s$



motion in Δt

- $\Delta d = ½\,(d_l + d_r)$
- $\Delta\theta = (d_r - d_l)/D$

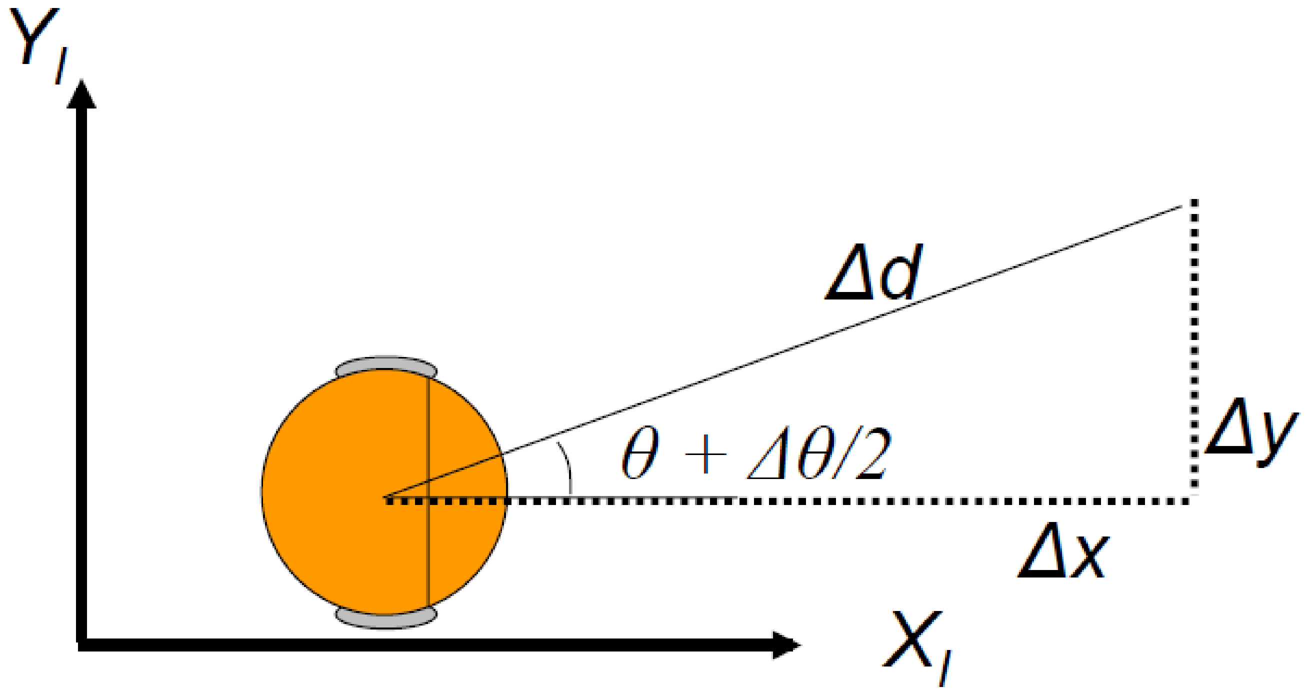
(D = distance of wheels)

Example: Differential Drive Odometry

pose update:

- $x_{t+1} = x_t + \Delta x$
- $y_{t+1} = y_t + \Delta y$
- $\theta_{t+1} = \theta_t + \Delta\theta$

- $\Delta x = \cos(\theta+\Delta\theta/2)\ \Delta d$
- $\Delta y = \sin(\theta+\Delta\theta/2)\ \Delta d$
- $\Delta\theta = (d_l - d_r)/D$



Example: Differential Drive Odometry

- $x_{t+1} = x_t + \cos(\theta+\Delta\theta/2)\ \Delta d$
- $y_{t+1} = y_t + \sin(\theta+\Delta\theta/2)\ \Delta d$
- $\theta_{t+1} = \theta_t + \Delta\theta$

- $d_{l/r}$: dist. by left/right wheel
- $\Delta d = ½\ (d_l + d_r)$
- $\Delta\theta = (d_l - d_r)/D$

$$f(x, y, \theta, d_l, d_r) = (x, y, \theta)^T + (\Delta x, \Delta y, \Delta\theta)^T$$

$$= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \frac{d_r + d_l}{2} \cos\left(\theta + \frac{d_r - d_l}{D}\right) \\ \frac{d_r + d_l}{2} \sin\left(\theta + \frac{d_r - d_l}{D}\right) \\ \frac{d_r - d_l}{D} \end{pmatrix}$$

Example: Differential Drive Odometry

$$f(x, y, \theta, d_l, d_r) = \begin{pmatrix} x + \frac{d_r + d_l}{2}\cos\left(\theta + \frac{d_r - d_l}{2D}\right) \\ y + \frac{d_r + d_l}{2}\sin\left(\theta + \frac{d_r - d_l}{2D}\right) \\ \theta + \frac{d_r - d_l}{D} \end{pmatrix}$$

uncertainty update:

$$C_{t+1} = \underbrace{J_{f(x,y,\theta)} C_t^{(x,y,\theta)} J^T{}_{f(x,y,\theta)}}_{\text{motion component}} + \underbrace{J_{f(d_l,d_r)} C_t^{(\Delta x,\Delta y,\Delta\theta)} J^T{}_{f(d_l,d_r)}}_{\text{wheel-slip}}$$

Example: Differential Drive Odometry

$$f(x, y, \theta, d_l, d_r) = \begin{pmatrix} x + \dfrac{d_r + d_l}{2} \cos\left(\theta + \dfrac{d_r - d_l}{2D}\right) \\ y + \dfrac{d_r + d_l}{2} \sin\left(\theta + \dfrac{d_r - d_l}{2D}\right) \\ \theta + \dfrac{d_r - d_l}{D} \end{pmatrix}$$

$$J_{f(x,y,\theta)} = \begin{pmatrix} \dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} & \dfrac{\partial f}{\partial \theta} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & -\dfrac{d_r + d_l}{2} \sin\left(\theta + \dfrac{d_r - d_l}{2D}\right) \\ 0 & 1 & \dfrac{d_r + d_l}{2} \cos\left(\theta + \dfrac{d_r - d_l}{2D}\right) \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & -\Delta d \sin(\theta + \Delta\theta / 2) \\ 0 & 1 & \Delta d \cos(\theta + \Delta\theta / 2) \\ 0 & 0 & 1 \end{pmatrix}$$

Example: Differential Drive Odometry

$$f(x,y,\theta,d_l,d_r) = \begin{pmatrix} x + \frac{d_r + d_l}{2}\cos\left(\theta + \frac{d_r - d_l}{2D}\right) \\ y + \frac{d_r + d_l}{2}\sin\left(\theta + \frac{d_r - d_l}{2D}\right) \\ \theta + \frac{d_r - d_l}{D} \end{pmatrix}$$

$$J_{f(d_l,d_r)} = \begin{pmatrix} \frac{\partial f}{\partial d_l} & \frac{\partial f}{\partial d_r} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{2}\cos(\theta + \frac{d_r - d_l}{2D}) + \frac{d_r + d_l}{4D}\sin(\theta + \frac{d_r - d_l}{2D}) & \frac{1}{2}\cos(\theta + \frac{d_r - d_l}{2D}) - \frac{d_r + d_l}{4D}\sin(\theta + \frac{d_r - d_l}{2D}) \\ -\frac{1}{2}\sin(\theta + \frac{d_r - d_l}{2D}) - \frac{d_r + d_l}{4D}\cos(\theta + \frac{d_r - d_l}{2D}) & \frac{1}{2}\sin(\theta + \frac{d_r - d_l}{2D}) + \frac{d_r + d_l}{4D}\cos(\theta + \frac{d_r - d_l}{2D}) \\ -\frac{1}{D} & \frac{1}{D} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{2}\cos(\theta + \Delta\theta/2) + \frac{\Delta d}{2D}\sin(\theta + \Delta\theta/2) & \frac{1}{2}\cos(\theta + \Delta\theta/2) - \frac{\Delta d}{2D}\sin(\theta + \Delta\theta/2) \\ -\frac{1}{2}\sin(\theta + \Delta\theta/2) - \frac{\Delta d}{2D}\cos(\theta + \Delta\theta/2) & \frac{1}{2}\sin(\theta + \Delta\theta/2) + \frac{\Delta d}{2D}\cos(\theta + \Delta\theta/2) \\ -\frac{1}{D} & \frac{1}{D} \end{pmatrix}$$

Example: Differential Drive Odometry

$$C_{t+1} = J_{f(x,y,\theta)} C_t^{(x,y,\theta)} J^T{}_{f(x,y,\theta)} + J_{f(d_l,d_r)} C_t^{(\Delta x,\Delta y,\Delta\theta)} J^T{}_{f(d_l,d_r)}$$

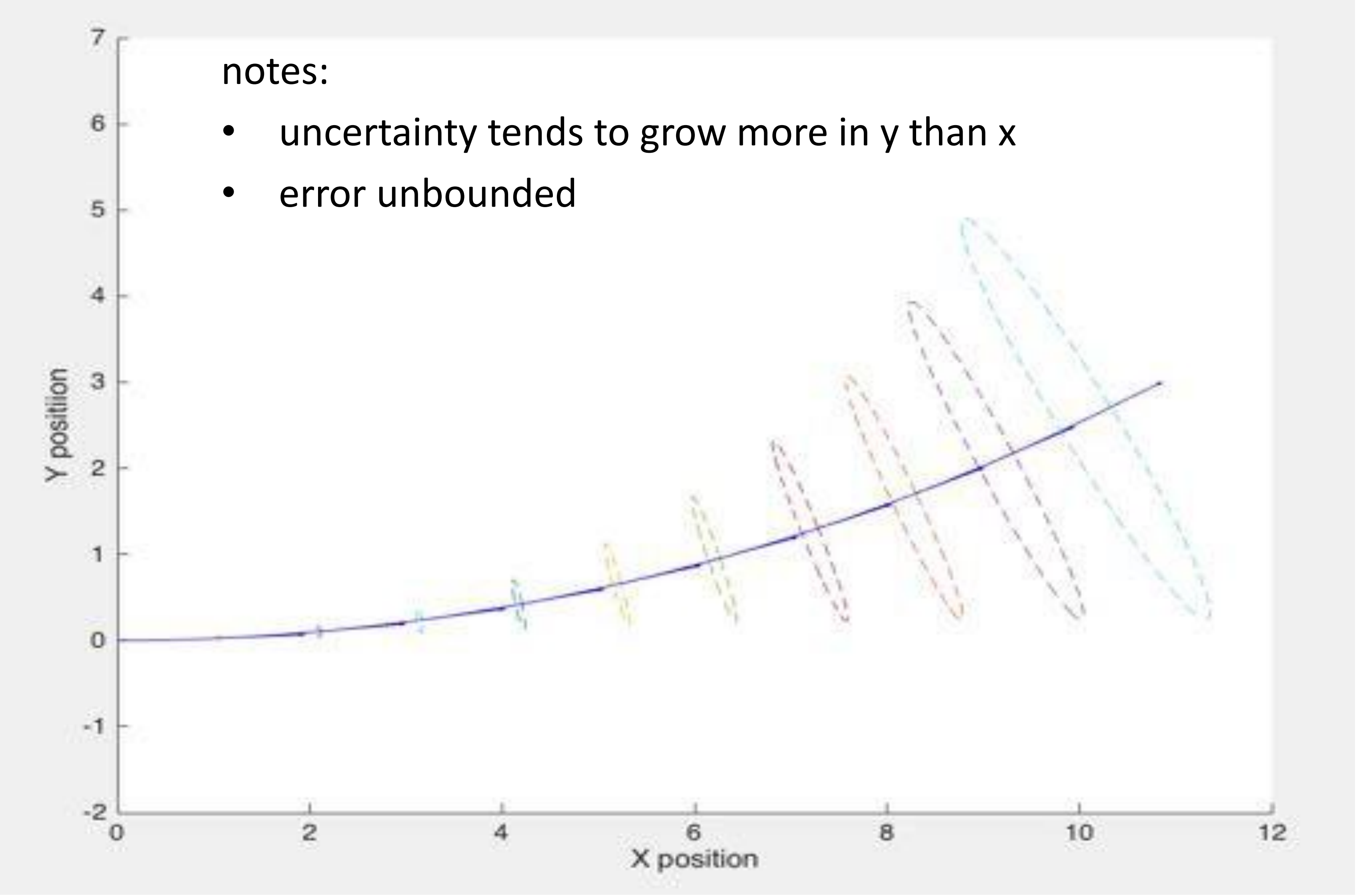
- start
  - just start pose at $(0,0,0)^T$
  - and perfect localization

$$C_0^{(x,y,\theta)} = 0$$

- modeling of slip
  - proportional to left/right wheel distance
  - with constant(s), e.g., found via calibration

$$C_t^{(\Delta x,\Delta y,\Delta\theta)} = \begin{pmatrix} c_l|d_l| & 0 \\ 0 & c_r|d_r| \end{pmatrix}$$

Example: Differential Drive Odometry



Kalman Filter

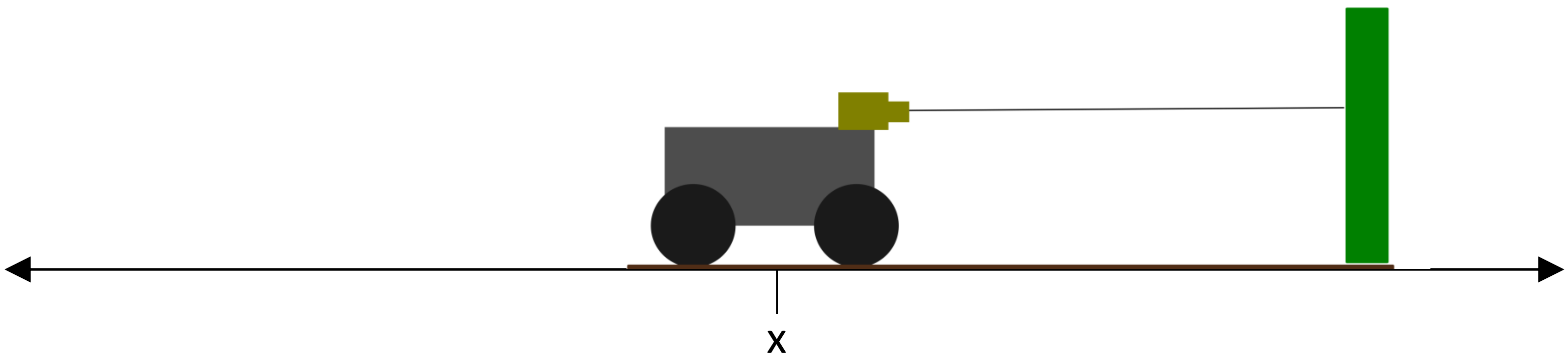
Kalman Filter

- **recursive** data processing algorithm
  - no need to store all previous measurements
  - and to reprocess all data at each time step
- to generate **optimal** estimate from measurements
  - for linear system (but also widely used for non-linear, too)
  - and white Gaussian noise (i.e., uncorrelated in time)

can e.g., be used for localization

- e.g., by incorporating (noisy) absolute measurements
- in (noisy) relative motion estimates (by e.g., odometry)

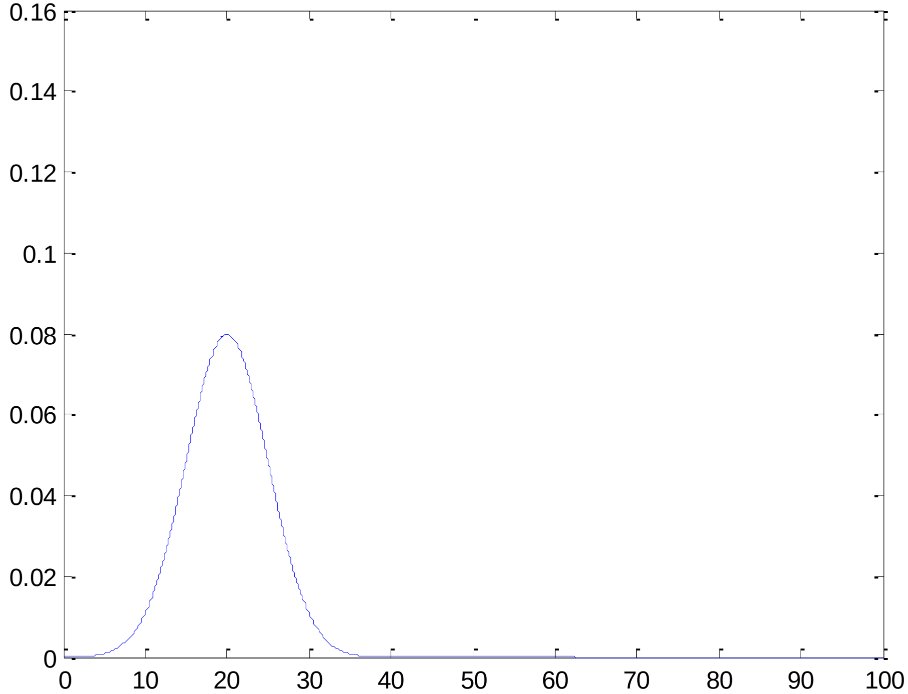
Simple Example: 1D Motion



- 1-dimensional localization: position x(t)
- odometry & perception of a landmark
- assume Gaussian distributed measurements

Simple Example: 1D Motion

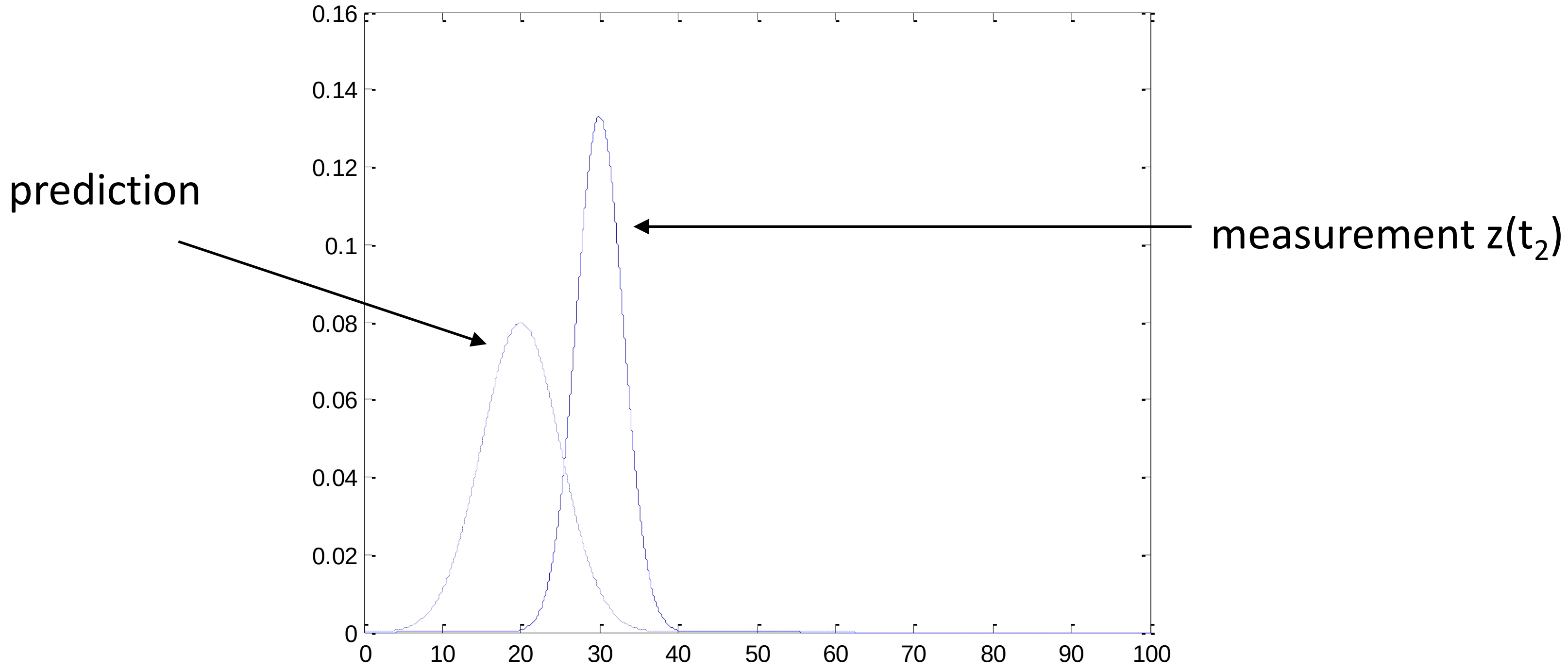
1st: **standing still** and multiple, noisy measurements



- location estimate at t_1: mean $\mu_1 = z_1$ and variance = σ^2_1
- first estimate of position: $\hat{x}(t_1) = \mu_1 = z_1$
- error variance of estimate: $\sigma^2(t_1) = \sigma^2_1$
- robot in same position at time t_2 : <u>*predicted*</u> position is z_1

Simple Example: 1D Motion

1st: **standing still** and multiple, noisy measurements



- prediction $\hat{y}^{-}(t_2)$
- landmark measurement at t_2: mean = z_2 and variance = σ^2_2
- *correct* prediction with this measurement to get $\hat{x}(t_2)$
- via linear interpolation with variances as weights

Fusing the data

- interpolation is "best" combination
  - based on statistical criteria, namely
  - maximum likelihood estimate and
  - minimum variance of all possible linear combinations

$$\mu = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} z_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} z_2 \qquad \sigma^2 = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right)^{-1}$$

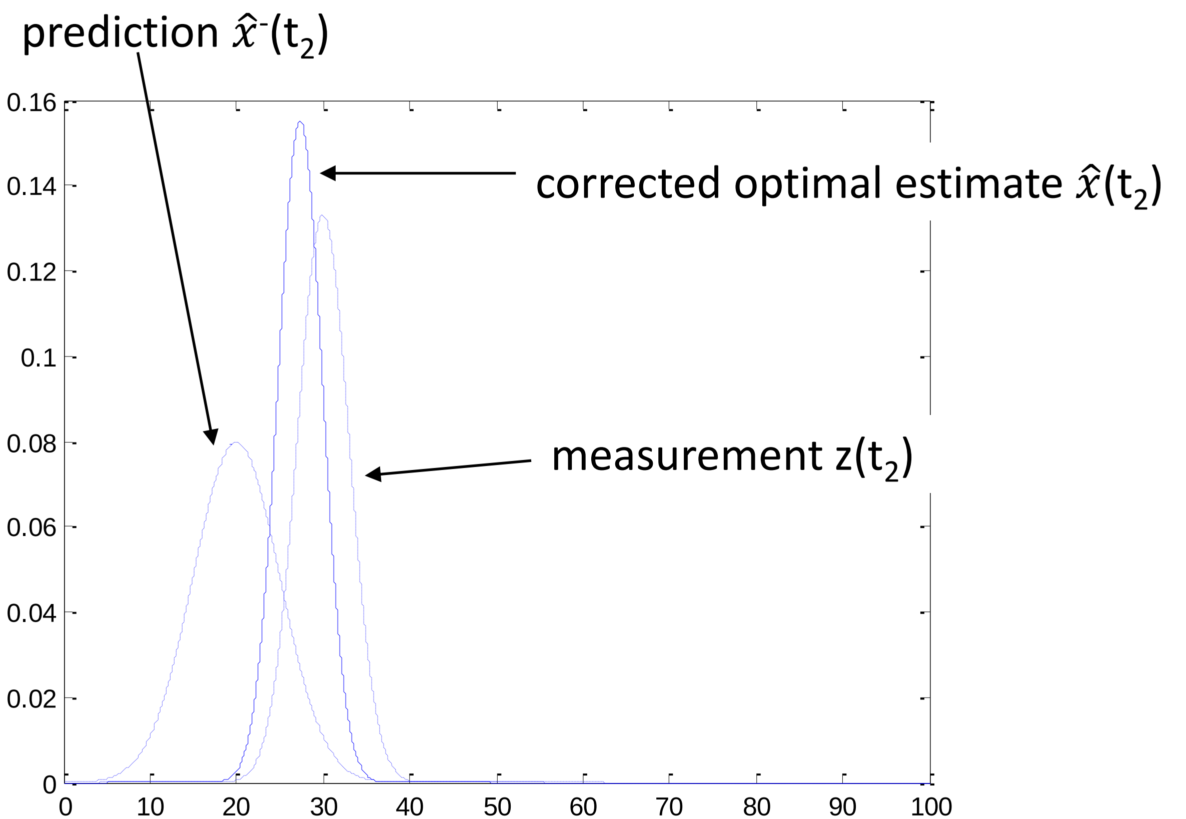
- new state estimate then $\hat{x}(t_2) = \mu$

The influence of the variances

- variances σ^2_i determine
  - how much to trust the measurements
- e.g., variances are equal

  => sensor measurements are averaged
- one is larger, e.g., σ^2_1
  - there is more uncertainty in the measurement z_1
  - and it is weighted less heavily than z_2
- so, even poor quality data contains information
  - it is included without "spoiling" the better quality data
  - and it improves the output of the filter

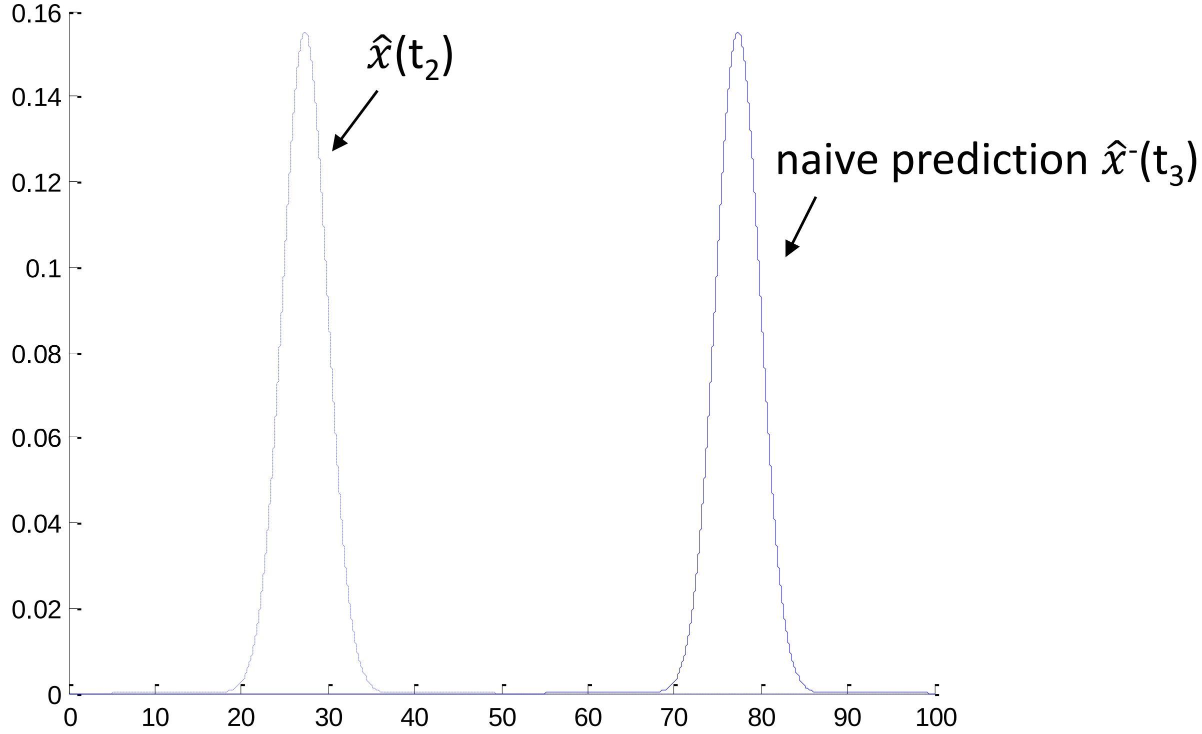
Simple Example: 1D Motion



- corrected mean: new optimal estimate of position
- new variance is smaller than either of the previous two variances

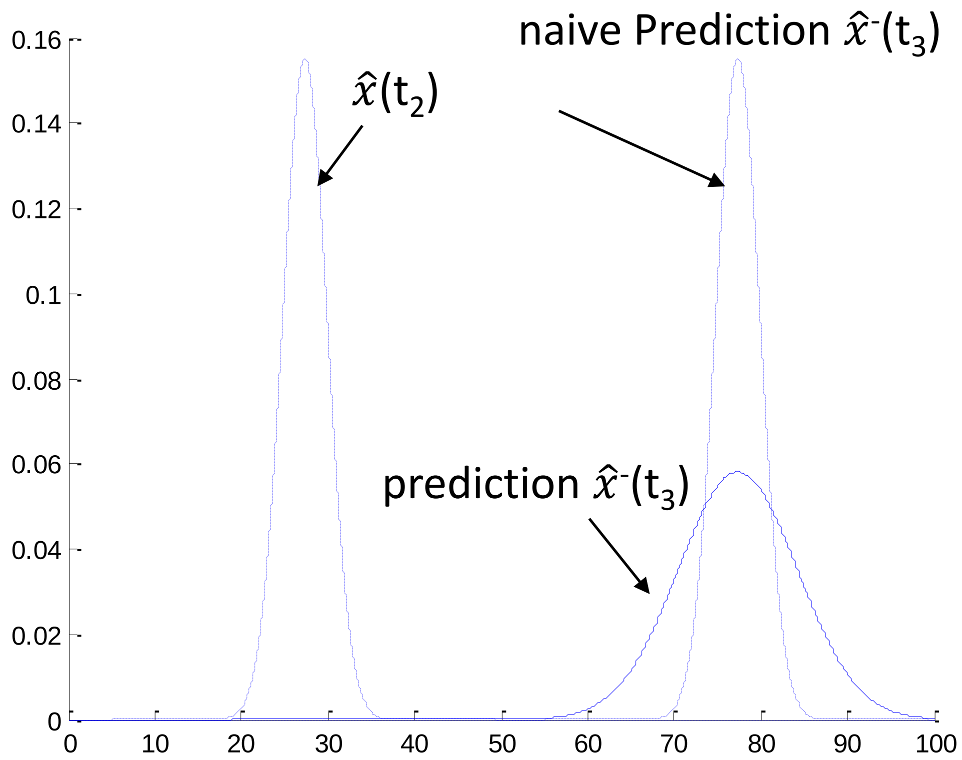
Simple Example: 1D Motion

2nd: **including noisy motion** (and related uncertainty based model)



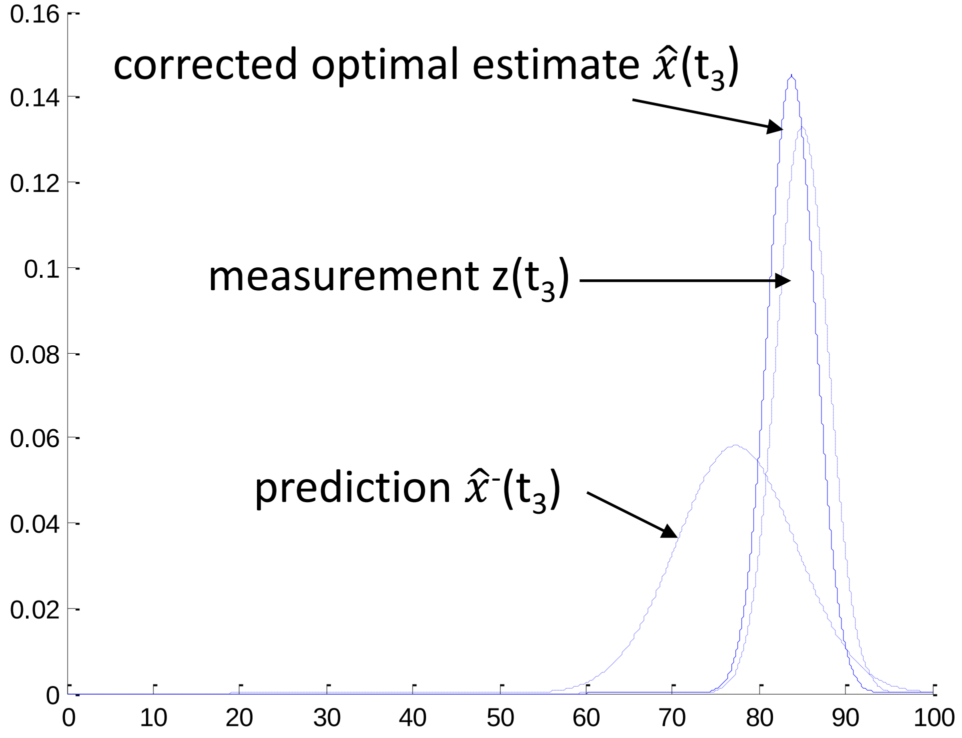
- at time t_3, robot moves with velocity dx/dt=u
- naive approach: shift probability to the right, i.e., add *ut* to mean
- but motion estimates (odometry) are noisy

Simple Example: 1D Motion



- use proper model by adding Gaussian noise *w*
- dx/dt = u + *w*
- distribution for prediction “moves” and “spreads out”

Simple Example: 1D Motion



- new measurement of the landmark at t_3
- correct again the prediction
- using difference between prediction and measurement aka ***residual***

and so on...

Overview Kalman

- initial conditions ($\hat{x}_{k-1}$ and σ_{k-1})
- prediction ($\hat{x}^{-}_{k}$, σ^{-}_{k})
  - use of initial conditions and model (e.g., odometry)
  - to make prediction
- measurement (z_k)
  - take measurement
- correction ($\hat{x}_k$, σ_k)
  - use measurement to correct prediction
  - by 'fusing' prediction and residual
  - i.e, by merging two Gaussians
  - result: optimal estimate with smaller variance

Kalman Filter

given: linear system with white Gaussian noise

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}$$
$$z_k = Hx_k + v_k$$

Zero-mean Gaussians with covariance matrices Q, R

$$p(w) = N(0, Q)$$
$$p(v) = N(0, R)$$

Kalman Filter

- a priori state estimate $\hat{x}_k^-$
  - at step k
  - includes all knowledge of the process prior to step k
- a posteriori state estimate $\hat{x}_k$
  - at step k
  - given the current measurement
- a priori and a posteriori estimate errors

$$e_k^- = x_k - \hat{x}_k^- \qquad\qquad e_k = x_k - \hat{x}_k$$

- a priori and a posteriori estimate error covariances

$$P_k^- = E(e_k^- e_k^{-T}) \qquad\qquad P_k = E(e_k e_k^T)$$

Kalman Filter

blending factor

aka (Kalman) gain

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

(measurement) innovation

aka residual

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

Kalman Filter

Time Update ("Predict")

(1) Project the state ahead

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1}$$

(2) Project the error covariance ahead

$$P_k^- = AP_{k-1}A^T + Q$$

Measurement Update ("Correct")

(1) Compute the Kalman gain

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

(2) Update estimate with measurement z_k

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

(3) Update the error covariance

$$P_k = (I - K_k H)P_k^-$$

Initial estimates for $\hat{x}_{k-1}$ and P_{k-1}

Extended Kalman Filter

Extended Kalman Filter

state-evolution and measurement equations are often non-linear

$$\cancel{x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}}$$
$$\cancel{z_k = Hx_{k-1} + v_{k-1}}$$

$\Rightarrow$Extended Kalman Filter (EKF)

Extended Kalman Filter

state-evolution and measurement
fct's *f* and *h* are non-linear:

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1}$$
$$z_k = h(x_k) + v_k$$

Zero-mean Gaussians with
covariance matrices Q, R

$$p(w) = N(0, Q)$$
$$p(v) = N(0, R)$$

Extended Kalman Filter

linearization of update equations   Jacobian of

- $f : J_f$
- $h : J_h$

- predictor step:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1})$$

$$P_k^- = J_f P_{k-1} J_f^{\,T} + Q$$

- Kalman gain:

$$K_k = P_k^- J_h^{\,T} (J_h P_k^- J_h^{\,T} + R)^{-1}$$

- corrector step:

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-))$$

$$P_k = (I - K_k J_h) P_k^-$$

Extended Kalman Filter

- Extended Kalman Filter straightforward to use
- but unfortunately known to be often not stable,
- i.e., to diverge due to the linearization

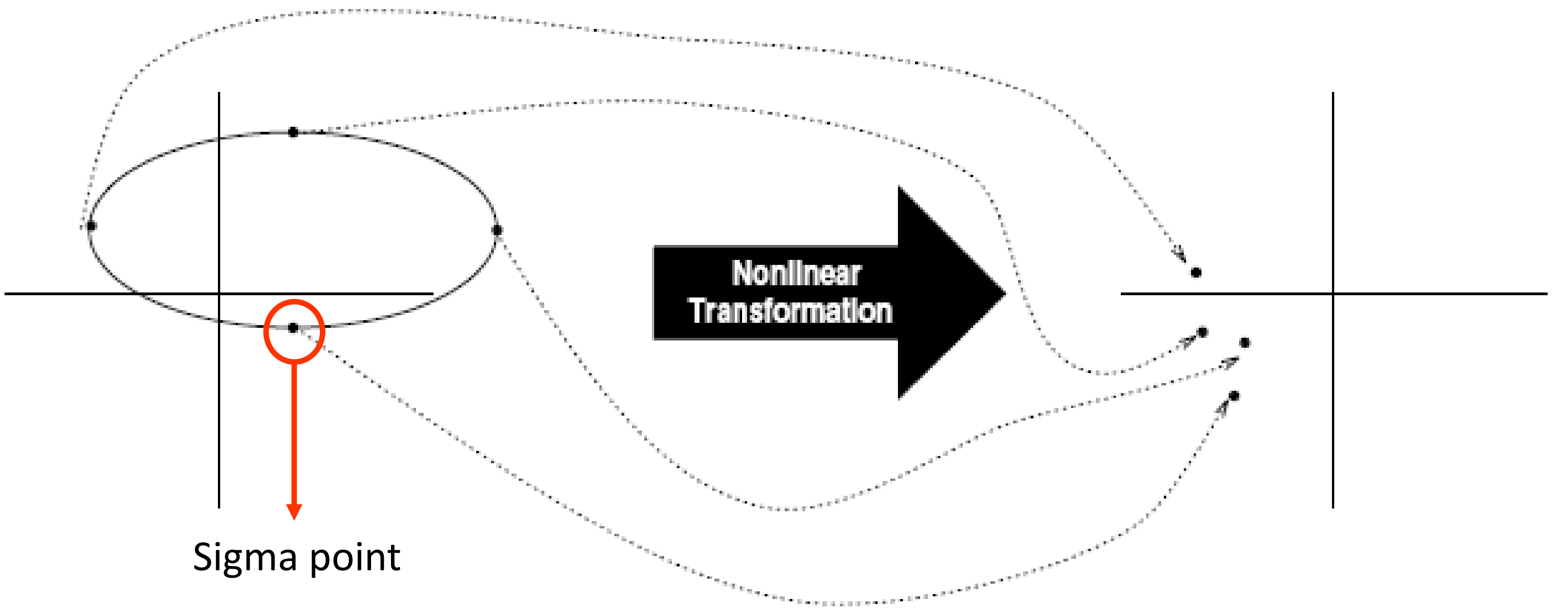
=> Unscented Kalman Filter (UKF)

Unscented Kalman Filter

Unscented Kalman Filter (UKF)

basic idea:

- do not linearize transformation
- but choose (few) sample points
- to represent mean and covariance

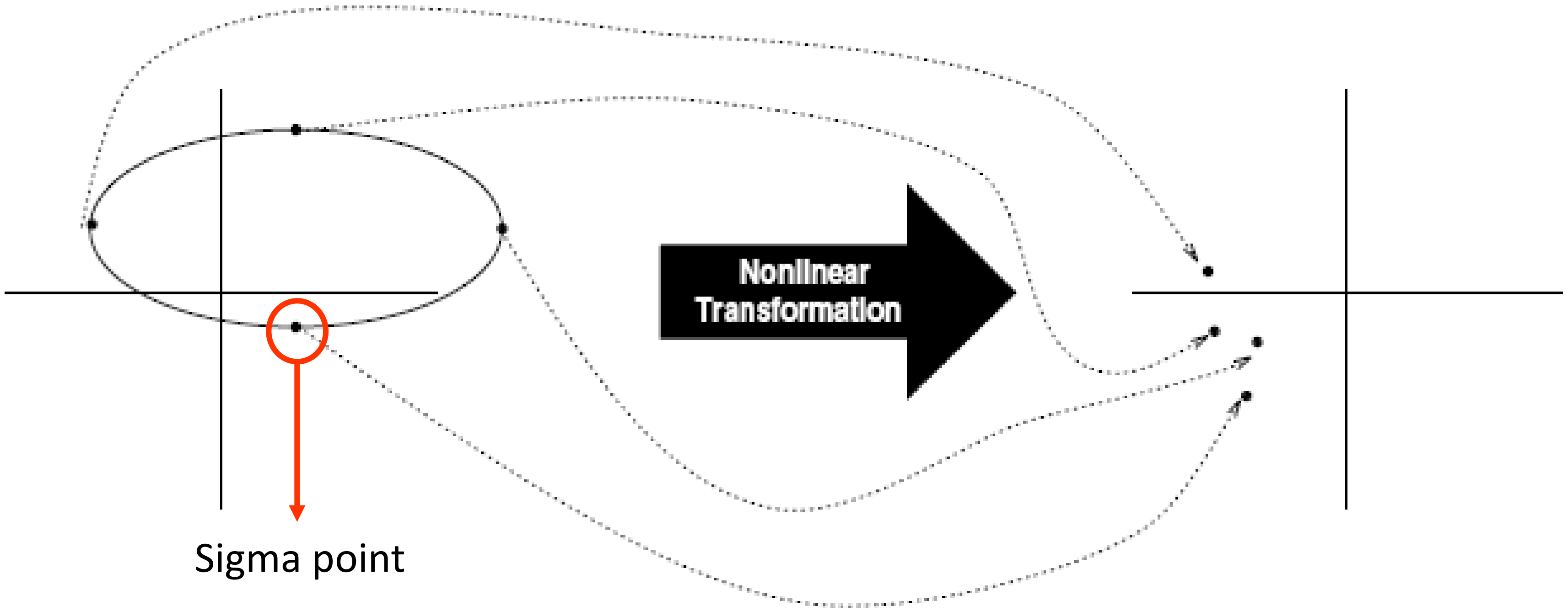


Unscented Kalman Filter (UKF)

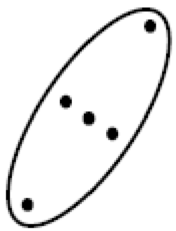
basic idea: choose (few) sample points for mean and covariance

advantages

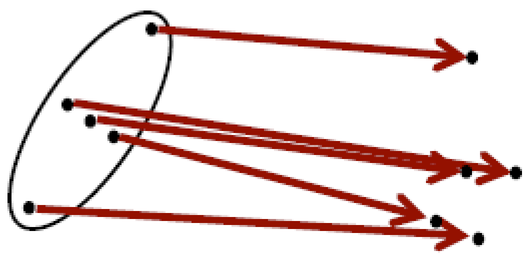
- (can be) more accurate than EKF
- no need for Jacobians



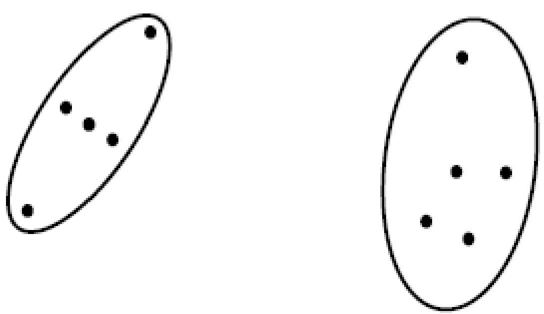
Unscented Kalman Filter (UKF)



set of sigma points

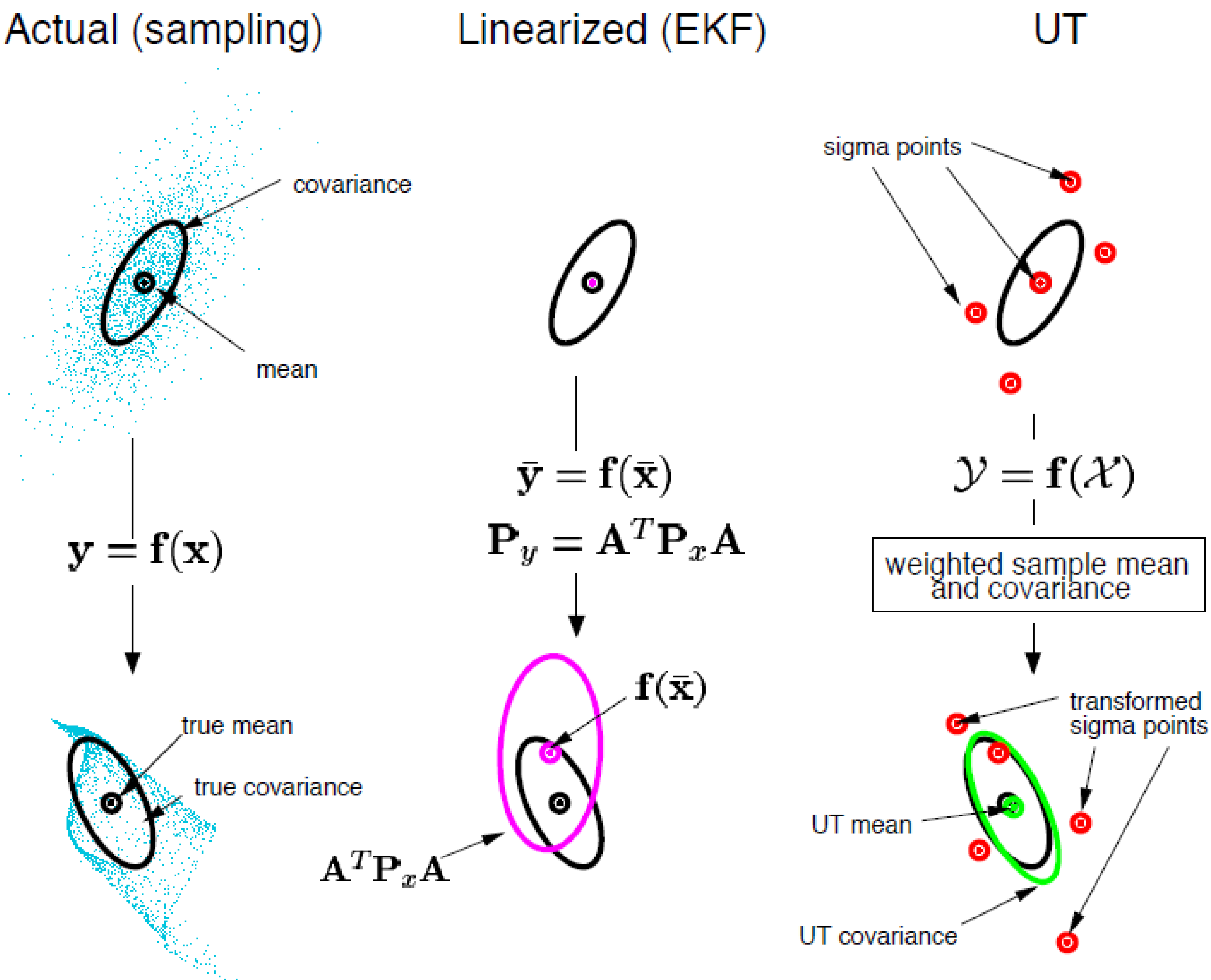


transform each one
with non-linear fct



compute new Gaussian from
transformed and weighted points

Comparison EKF and UKF



Unscented Kalman Filter

- sigma points χ_i, weights w_i
- choose so that

$$\sum_i w_i = 1$$

$$\mu = \sum_i w_i \chi_i$$

$$\Sigma = \sum_i w_i (\chi_i - \mu)(\chi_i - \mu)^T$$

note: there is no unique solution for χ_i, w_i
(there are several versions of UKF)

Unscented Kalman Filter

n : dimension

λ : scaling parameter

$$\chi_0 = \mu$$

$$\chi_i = \mu + \left(\sqrt{(n+\lambda)\Sigma}\right)_i \qquad \text{for } i = 1,\ldots,n$$

$$\chi_i = \mu - \left(\sqrt{(n+\lambda)\Sigma}\right)_{i-n} \quad \text{for } i = n+1,\ldots,2n$$

note: $(A)_i$ = column vector i of A

Matrix Square Root

matrix square root: S with $SS^T = \Sigma$

recap:

- Σ symmetric (hence, $c\Sigma$ symmetric)
- SVD of symmetric matrix $\Sigma = VLV^T$

$$\Sigma = SS^T = VLV^T$$

$$\Rightarrow S = VL^{(1/2)}$$

Matrix Square Root

matrix square root: S with $SS^T = \Sigma$

Cholesky decomposition

- $\Sigma = LDL^T = LD^{1/2}(D^{1/2}L)^T = GG^T$
  - symmetric Σ
  - D : positive diagonal matrix
  - L : normed lower triangular matrix (normed: diagonal is all 1's)
  - G : lower triangular matrix
- often used in UKF implementations

Unscented Kalman Filter

computing the weights

- w^m : for mean
- w^c : for covariance

$$w_0^m = \frac{\lambda}{n+\lambda}$$

$$w_0^c = w_0^m + (1 - \alpha^2 + \beta)$$

$$w_i^m = w_i^c = \frac{1}{2(n+\lambda)} \quad \text{for } i = 1,\ldots,2n$$

α, β : parameters

Unscented Kalman Filter

$$\chi_0 = \mu$$

$$\chi_i = \mu + \left(\sqrt{(n+\lambda)\Sigma}\right)_i \quad (\mathrm{i} \le n)$$

$$\chi_i = \mu - \left(\sqrt{(n+\lambda)\Sigma}\right)_{i-n} \quad (\mathrm{i} > n)$$

$$w_0^m = \frac{\lambda}{n+\lambda}$$

$$w_0^c = w_0^m + (1 - \alpha^2 + \beta)$$

$$w_i^m = w_i^c = \frac{1}{2(n+\lambda)}$$

parameters: $\alpha \in]0,1]$

$\beta = 2$ (optimal for Gaussians)

$\lambda = \alpha^2 (n+\kappa) - n$

$\kappa \ge 0$

Unscented Kalman Filter

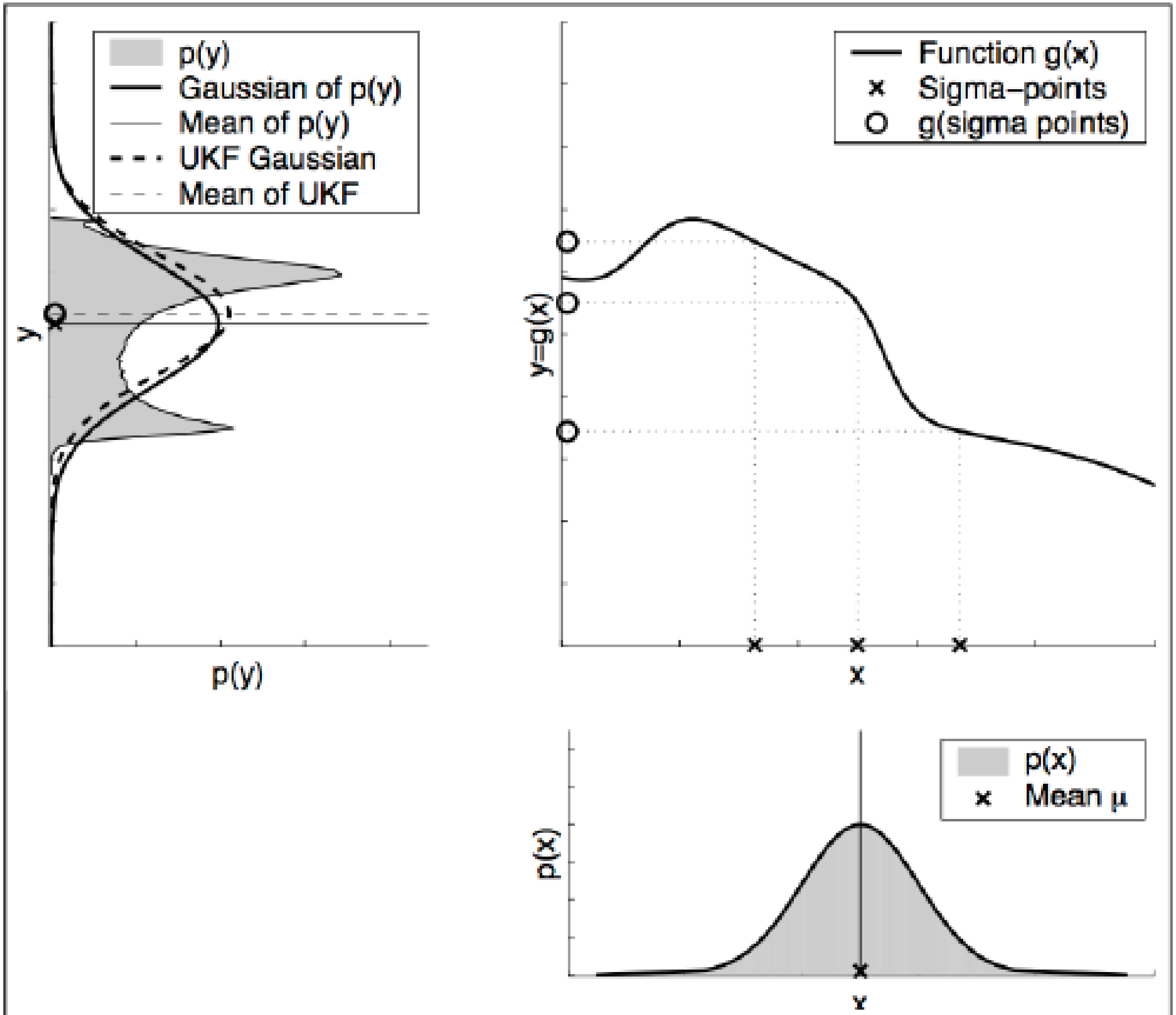
new Gaussian N(μ’,Σ’)

from sigma points transformed by non-linear f()

$$\mu' = \sum_{i=0}^{2n} w_i^m f(\chi_i)$$

$$\Sigma' = \sum_{i=0}^{2n} w_i^c (f(\chi_i) - \mu')(f(\chi_i) - \mu')^T$$

Example



p(y)
Gaussian of p(y)
Mean of p(y)
UKF Gaussian
Mean of UKF
y
p(y)
Function g(x)
Sigma–points
g(sigma points)
y=g(x)
x
p(x)
Mean μ
p(x)
x

Unscented Kalman Filter

predict:

$$\chi_{i,k} = f(\chi_{i,k-1})$$

$$\hat{x}_k^- = \sum_{i=0}^{2n} w_i^m \chi_{i,k-1}$$

$$z_{i,k-1} = h(\chi_{i,k-1})$$

$$\hat{z}_k^- = \sum_{i=0}^{2n} w_i^m z_{i,k-1}$$

$$P_k^- = \sum_{i=0}^{2n} w_i^c (\chi_{i,k} - \hat{x}_k^-)(\chi_{i,k} - \hat{x}_k^-)^T$$

correct:

$$P_{z_k z_k} = \sum_{i=0}^{2n} w_i^c (z_{i,k} - \hat{z}_k^-)(z_{i,k} - \hat{z}_k^-)^T$$

$$P_{x_k z_k} = \sum_{i=0}^{2n} w_i^c (\chi_{i,k} - \hat{z}_k^-)(z_{i,k} - \hat{z}_k^-)^T$$

$$K_k = P_{x_k z_k} P_{z_k z_k}^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - \hat{z}_k^-)$$

$$P_k = P_k^- - K_k P_{z_k z_k} K_k^T$$

Particle Filter

Particle Filter (PF)

- represent distribution
  - by **randomly chosen weighted samples** (particles)
  - population based somewhat similar to Evolutionary Algorithm
- particles are transformed under systems dynamics (model)
- test predicted states (particles) with observation
- do a selection
  - multiply or discard particles
  - i.e., survival of the fittest

- hence „Monte Carlo“ filter (aka „condensing“)
- (likely) convergence depends on #samples, i.e., particles

Particle Filter

example: localization

(from Dieter Fox, UWash)

- red dot
  - particle
  - estimated robot pose
  - init: random
- 24 sonar sensors
  - match range with given map
  - basis for selection

more about PF soon...