NA 568 - Winter 2022

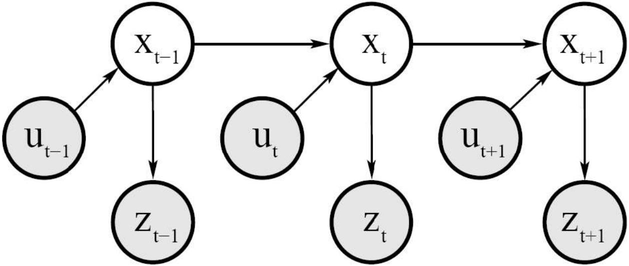
# Nonlinear Kalman Filtering

**Maani Ghaffari**

January 13, 2021

UNIVERSITY OF MICHIGAN

# Dynamic Bayesian Network for Controls, States, and Sensations



# State Estimation

- Estimate the state $x$ of a system given observations $z$ and controls $u$
- **Goal:**

$$p(x \mid z, u)$$

Courtesy: C. Stachniss

# Prediction and Correction Step

- Bayes filter can be written as a two step process
- **Prediction step**

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\; bel(x_{t-1})\, dx_{t-1}$$

- **Correction step**

$$bel(x_t) = \eta\, p(z_t \mid x_t)\; \overline{bel}(x_t)$$

Courtesy: C. Stachniss

# Motion and Observation Model

- Prediction step

$$\overline{bel}(x_t) = \int \underbrace{p(x_t \mid u_t, x_{t-1})}_{\text{motion model}} \, bel(x_{t-1}) \, dx_{t-1}$$

- Correction step

$$bel(x_t) = \eta \, \underbrace{p(z_t \mid x_t)}_{\text{sensor or observation model}} \, \overline{bel}(x_t)$$

Courtesy: C. Stachniss

- Given:
  - Stream of observations $z_{1:t}$ and action data $u_{1:t}$
  - Sensor/measurement model $p(z_t|x_t)$
  - Action/motion/transition model $p(x_t|x_{t-1},u_t)$
- Wanted:
  - The state $x_t$ of dynamical system
  - The posterior of state is called belief $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$

**Algorithm 1** `Bayes-filter`

**Require:** Belief $bel(x_{t-1}) = p(x_{t-1}|z_{1:t-1}, u_{1:t-1})$, action $u_t$, measurement $z_t$;

1: **for** all state variables **do**

2: $\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)bel(x_{t-1})\mathrm{d}x_{t-1}$ // Predict using action/control input $u_t$

3: $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$ // Update using perceptual data $z_t$

4: **return** $bel(x_t)$

Nonlinear dynamic system with additive noise.

- Nonlinear process model:

$$x_k = f(u_k, x_{k-1}) + w_k$$

- Nonlinear measurement model:

$$z_k = h(x_k) + v_k$$

Nonlinear dynamic system with multiplicative noise.

- Nonlinear process model:

$$x_k = f(u_k, x_{k-1}, w_k)$$

- Nonlinear measurement model:

$$z_k = h(x_k, v_k)$$

**Q.** How to map belief (a probability distribution) through a nonlinear function?

**Q.** How to map belief (a probability distribution) through a nonlinear function?

Key ideas:

- Linearization via Taylor expansion

**Q.** How to map belief (a probability distribution) through a nonlinear function?

Key ideas:

- Linearization via Taylor expansion
- Unscented Transform (deterministic sampling)

**Q.** How to map belief (a probability distribution) through a nonlinear function?

Key ideas:

- Linearization via Taylor expansion
- Unscented Transform (deterministic sampling)
- Monte-Carlo methods (random sampling)

**Q.** How to map belief (a probability distribution) through a nonlinear function?

Key ideas:

- Linearization via Taylor expansion
  $\longrightarrow$ Extended Kalman Filter (EKF)

- Unscented Transform (deterministic sampling)
  $\longrightarrow$ Unscented Kalman Filter (UKF)

- Monte-Carlo methods (random sampling)
  $\longrightarrow$ Sequential Monte-Carlo methods (Particle Filters)

# Linearization via Taylor Expansion

Linearization of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ around point $a$ is

$$\begin{aligned} f(x) &\approx f(a) + \frac{\partial f}{\partial x}\Big|_{x=a}(x-a) \\ &= (f(a) - \frac{\partial f}{\partial x}\Big|_{x=a} a) + \frac{\partial f}{\partial x}\Big|_{x=a} x \\ &:= x_0 + Fx \end{aligned}$$

Affine! We know how to propagate a Gaussian through an affine map.

# Recall: Affine Transformation of a Multivariate Gaussian

Suppose $x \sim \mathcal{N}(\mu, \Sigma)$ and $y = Ax + b$.

Then $y \sim \mathcal{N}(A\mu + b, A\Sigma A^\mathsf{T})$.

# EKF Algorithm

$$F_k = \frac{\partial f}{\partial x}\bigg|_{x=\mu_{k-1}},\ W_k = \frac{\partial f}{\partial w}\bigg|_{x=\mu_{k-1}},\ H_k = \frac{\partial h}{\partial x}\bigg|_{x=\mu_k^-},\ V_k = \frac{\partial h}{\partial \mathbf{v}}\bigg|_{x=\mu_k^-}$$

**Algorithm 2** `Extended-Kalman-filter`

**Require:** belief mean $\mu_{k-1}$, belief covariance $\Sigma_{k-1}$, action $u_k$, measurement $z_k$;

1: $\mu_k^- \leftarrow f(u_k, \mu_{k-1})$ ▷ predicted mean
2: $\Sigma_k^- \leftarrow F_k \Sigma_{k-1} F_k^\top + W_k Q_k W_k^\top$ ▷ predicted covariance
3: $\nu_k \leftarrow z_k - h(\mu_k^-)$ ▷ innovation
4: $S_k \leftarrow H_k \Sigma_k^- H_k^\top + V_k R_k V_k^\top$ ▷ innovation covariance
5: $K_k \leftarrow \Sigma_k^- H_k^\top S_k^{-1}$ ▷ filter gain
6: $\mu_k \leftarrow \mu_k^- + K_k \nu_k$ ▷ corrected mean
7: $\Sigma_k \leftarrow (I - K_k H_k)\Sigma_k^-$ ▷ corrected covariance
8: $//\Sigma_k \leftarrow (I - K_k H_k)\Sigma_k^-(I - K_k H_k)^\top + K_k R_k K_k^\top$ ▷ numerically stable form
9: **return** $\mu_k$, $\Sigma_k$

A target is moving in a 2D plane. The ownship position is known and fixed at the origin. We have access to relative noisy range and bearing measurements of the target position at any time step.

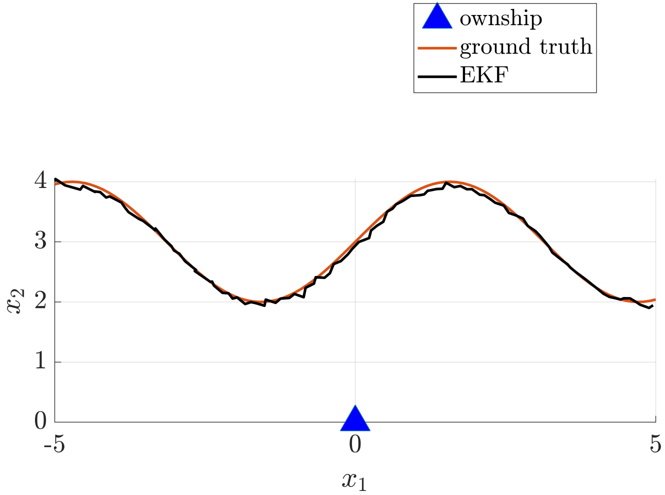
$$x_k = f(u_k, x_{k-1}) + w_k = x_{k-1} + w_k$$

$$z_k = h(x_k) + v_k = \left[ \begin{array}{c} \sqrt{{x_k^1}^2 + {x_k^2}^2} \\ \mathrm{atan2}(x_k^1, x_k^2) \end{array} \right] + v_k$$

$$F_k = I_2,\ G_k = 0_2,\ Q_k = 0.001\ I_2,\ R_k = \mathrm{diag}(0.05^2, 0.01^2)$$

$$H_k = \left[ \begin{array}{cc} \frac{x_k^1}{\sqrt{{x_k^1}^2 + {x_k^2}^2}} & \frac{x_k^2}{\sqrt{{x_k^1}^2 + {x_k^2}^2}} \\ \frac{x_k^2}{{x_k^1}^2 + {x_k^2}^2} & \frac{-x_k^1}{{x_k^1}^2 + {x_k^2}^2} \end{array} \right]$$

See `ekf_single_target.m` for code.



# EKF Summary

- Highly efficient; polynomial time in measurement dimensionality $n_z$ and state dimensionality $n_x$.
- Not optimal.
- Can diverge if nonlinearities are large.
- Can work well in practice for many problems despite violating all the underlying assumptions.

# Unscented Transform Overview

- Compute a set of sigma points (samples) $\mathcal{X}$;
- Each sigma point has a weight $w$;
- Transform the point through the nonlinear function $g(x)$;
- Compute a Gaussian distribution from weighted points using weighted sample mean and covariance;

The first sigma point is the mean.

$$x_0 = \mu$$

$$x_i = \mu + \ell'_i \quad i = 1, \ldots, n$$

$$x_i = \mu - \ell'_{i-n} \quad i = n+1, \ldots, 2n$$

$\ell'_i$ is the $i$-th column of $L'$ where $L' = \sqrt{(n+\kappa)}L$ and $\Sigma = LL^\mathsf{T}$ can be computed using Cholesky decomposition. $n$ is the dimension of the state and $\kappa$ is a user-definable parameter.

Let $g : \mathbb{R}^n \to \mathbb{R}^m$. Compute a Gaussian from weighted and transformed points

$$\mu' = \sum_{i=0}^{2n} w_i g(x_i)$$

$$\Sigma' = \sum_{i=0}^{2n} w_i (g(x_i) - \mu')(g(x_i) - \mu')^\mathsf{T}$$

# Unscented Transform: Weights

$$w_0 = \frac{\kappa}{n+\kappa}$$

$$w_i = \frac{1}{2(n+\kappa)} \quad i = 1, \ldots, 2n$$

The user-defined parameter, $\kappa$, can be tuned to adjust the weight for a particular transformation. For instance $\kappa = 2$ (see *State Estimation for Robotics*, Timothy D. Barfoot, 2018, Ch. 4.2.7.).

**Remark**

*If the noise is additive, we simply add the noise covariance to the propagated sample covariance. If the noise is multiplicative, we augment the state with noise by adding zeros of appropriate dimension to the mean and constructing a block-diagonal covariance matrix of the state and noise covariances. Note that in the latter case the dimension of the augmented state is increased to the sum of the dimensions of the state and noise vectors; hence, more samples need to be drawn. See State Estimation for Robotics, Timothy D. Barfoot, 2018, Ch. 4.2.9.*

## Example: Unscented Transform

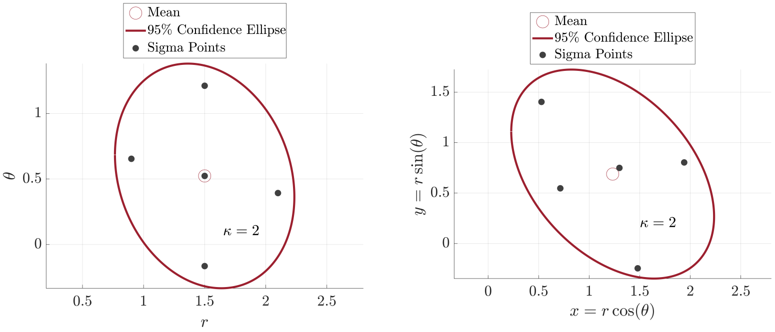
See `unscented_transform_example.m` for code.

Transform a Gaussian distribution from polar to Cartesian coordinates.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos(\theta) \\ r\sin(\theta) \end{bmatrix}, \qquad \begin{bmatrix} r \\ \theta \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 1.5 \\ \pi/6 \end{bmatrix}, \begin{bmatrix} 0.3^2 & -0.14^2 \\ -0.14^2 & 0.35^2 \end{bmatrix})$$

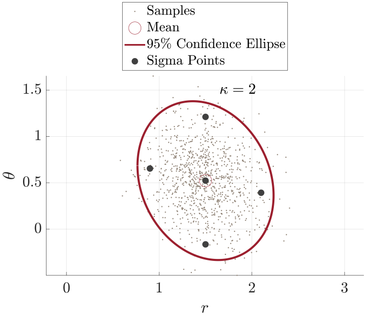
# Example: Unscented Transform

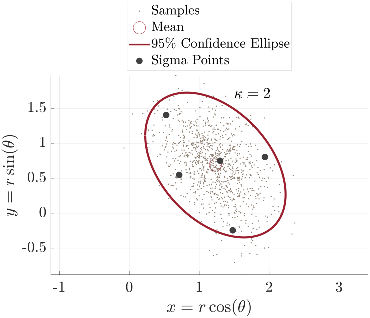
$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos(\theta) \\ r\sin(\theta) \end{bmatrix}, \qquad \begin{bmatrix} r \\ \theta \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 1.5 \\ \pi/6 \end{bmatrix}, \begin{bmatrix} 0.3^2 & -0.14^2 \\ -0.14^2 & 0.35^2 \end{bmatrix}\right)$$



# Example: Unscented Transform

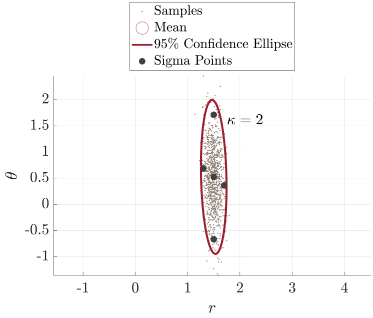
$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos(\theta) \\ r\sin(\theta) \end{bmatrix}, \qquad \begin{bmatrix} r \\ \theta \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 1.5 \\ \pi/6 \end{bmatrix}, \begin{bmatrix} 0.3^2 & -0.14^2 \\ -0.14^2 & 0.35^2 \end{bmatrix}\right)$$

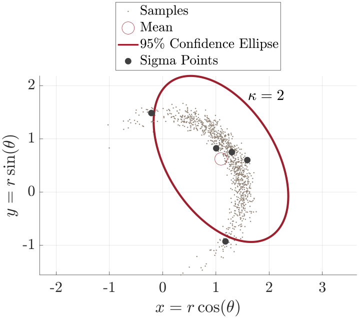


# Example: Unscented Transform

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} r\cos(\theta) \\ r\sin(\theta) \end{bmatrix}, \qquad \begin{bmatrix} r \\ \theta \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 1.5 \\ \pi/6 \end{bmatrix}, \begin{bmatrix} 0.1^2 & -0.09^2 \\ -0.09^2 & 0.6^2 \end{bmatrix})$$



# UKF Algorithm

**Algorithm 3** `Unscented-Kalman-filter`

**Require:** belief mean $\mu_{k-1}$, belief covariance $\Sigma_{k-1}$, action $u_k$, measurement $z_k$;

1: $\mathcal{X}_{k-1} \leftarrow$ compute the set of $2n+1$ sigma points using $\mu_{k-1}$ and $\Sigma_{k-1}$
2: $w^- \leftarrow$ compute the set of $2n+1$ weights
3: $\mu_k^- = \sum_{i=0}^{2n} w_i^- f(u_k, x_{k-1,i})$ ▷ predicted mean
4: $\Sigma_k^- \leftarrow \sum_{i=0}^{2n} w_i^- (f(u_k, x_{k-1,i}) - \mu_k^-)(f(u_k, x_{k-1,i}) - \mu_k^-)^\mathsf{T} + Q_k$ ▷ predicted covariance
5: $\mathcal{X}_k^- \leftarrow$ compute the set of $2n+1$ sigma points using $\mu_k^-$ and $\Sigma_k^-$
6: $w \leftarrow$ compute the set of $2n+1$ weights
7: $z_k^- = \sum_{i=0}^{2n} w_i h(x_{k,i}^-)$ ▷ predicted measurement
8: $\nu_k \leftarrow z_k - z_k^-$ ▷ innovation
9: $S_k \leftarrow \sum_{i=0}^{2n} w_i (h(x_{k,i}^-) - z_k^-)(h(x_{k,i}^-) - z_k^-)^\mathsf{T} + R_k$ ▷ innovation covariance
10: $\Sigma_k^{xz} \leftarrow \sum_{i=0}^{2n} w_k^{[i]} (x_{k,i}^- - \mu_k^-)(h(x_{k,i}^-) - z_k^-)^\mathsf{T}$ ▷ state and measurement cross covariance
11: $K_k \leftarrow \Sigma_k^{xz} S_k^{-1}$ ▷ filter gain
12: $\mu_k \leftarrow \mu_k^- + K_k \nu_k$ ▷ corrected mean
13: $\Sigma_k \leftarrow \Sigma_k^- - K_k S_k K_k^\mathsf{T}$ ▷ corrected covariance
14: **return** $\mu_k$, $\Sigma_k$

## Example: UKF Target Tracking

A target is moving in a 2D plane. The ownship position is known and fixed at the origin. We have access to relative noisy range and bearing measurements of the target position at any time step.

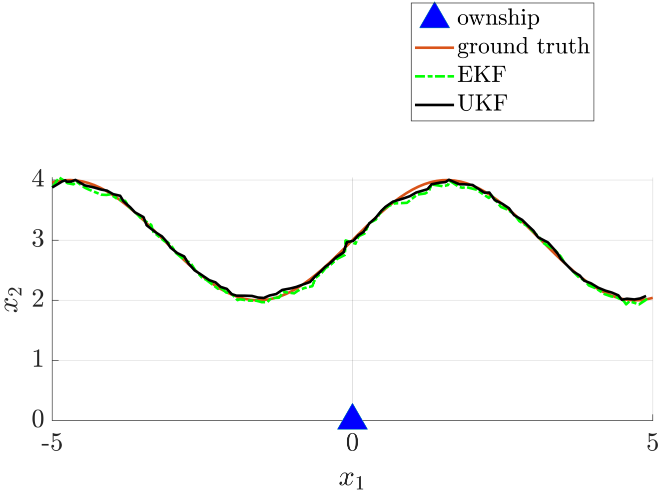
$$x_k = f(u_k, x_{k-1}) + w_k = x_{k-1} + w_k$$

$$z_k = h(x_k) + v_k = \begin{bmatrix} \sqrt{{x_k^1}^2 + {x_k^2}^2} \\ \mathrm{atan2}(x_k^1, x_k^2) \end{bmatrix} + v_k$$

$$F_k = I_2,\ G_k = 0_2,\ Q_k = 0.001\ I_2,\ R_k = \mathrm{diag}(0.05^2, 0.01^2)$$

# Example: UKF Target Tracking

See `ukf_single_target.m` for code.



# UKF Summary

- Highly efficient: same complexity as EKF, with a constant factor slower in typical practical applications;
- Better linearization than EKF;
- Derivative-free: no Jacobians needed.
- Not optimal.

## UKF vs. EKF

- Same results as EKF for linear models;
- Better approximation than EKF for non-linear models;
- Differences often “somewhat small”;
- No Jacobians needed for the UKF;
- Same complexity class;
- Slightly slower than the EKF

# Readings

- Probabilistic Robotics: Ch. 3
- State Estimation for Robotics: Ch. 3 and 4