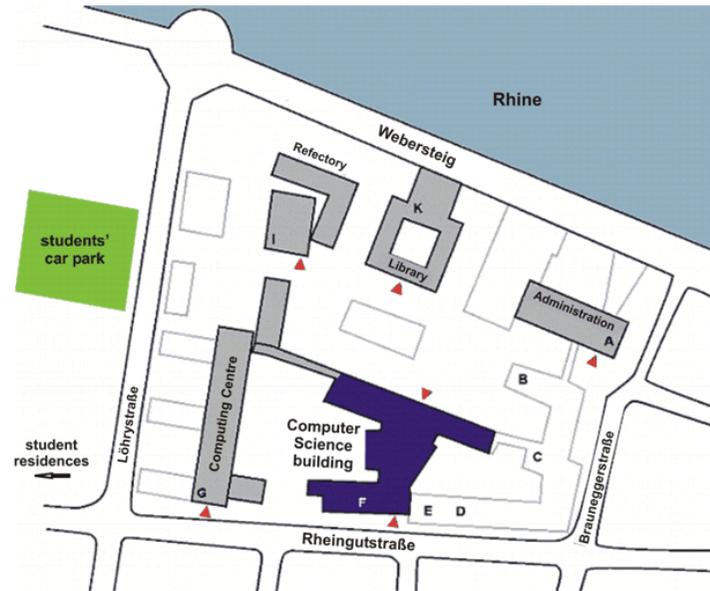


# Lokalisierung

- Einleitung
  - Problemstellung, Varianten,  
Sensorik, Umgebungskarten, Verfahren
- Laterationsverfahren
- Koppelnavigation
  - Idee, Navigationsgleichungen,  
Fehlerfortpflanzung
- Lokalisierung mit einem Kalman-Filter
  - Idee, Algorithmus für linearen und erweiterten Kalmanfilter,  
Beispiele, Selbstlokalisierung mit einem Kalmanfilter
- Monte-Carlo-Lokalisierung
  - Idee, Algorithmus, Beispiele

# Allgemeine Problemstellung



- Roboter bewegt sich in der Umgebung (Steuerbefehle bekannt)
- Roboter nimmt seine Umgebung mit seiner Sensorik wahr.
- Evtl. Umgebungskarte verfügbar.
- Ziel:  
schätze die Position des Roboters  $\mathbf{x}_k = (x, y, \theta)$  zu jedem Zeitpunkt  $t_k$   
in einem globalen Koordinatensystem (in der Umgebung fixiert).

# Varianten

---

- **Lokale Selbstlokalisierung**

- Die initiale Position des Roboters ist ungefähr bekannt.
- Ziel ist es, die Position neu zu berechnen, sobald sich der Roboter bewegt (**position tracking**).

- **Globale Selbstlokalisierung**

- Die initiale Position des Roboters ist nicht bekannt.
- Ziel ist es, die Position aufgrund von Roboterbewegungen und neuen Sensordaten zu finden.
- Schwieriger als lokale Selbstlokalisierung.
- Als zusätzliche Problemstellung kann dazukommen, dass der Roboter während der Lokalisierungsphase willkürlich an eine andere Position platziert werden kann (**kidnapped robot problem**)

# Sensorik

## ▪ Propriozeptive Sensoren

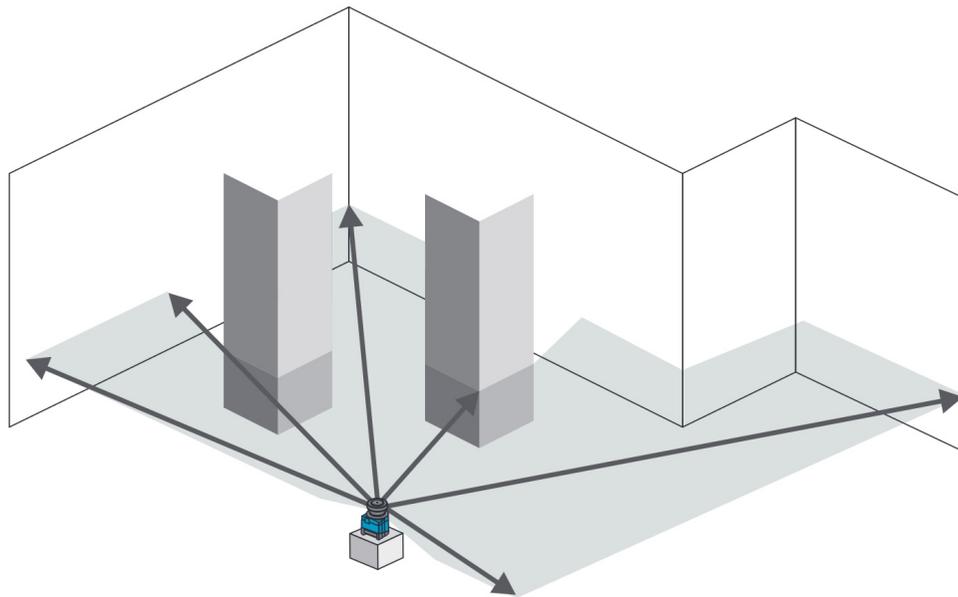
Sensor	Einsatz
Radsensor	Messung der Drehzahl von Rädern zur Bewegungsschätzung.
Drehgeber	Messung der Winkel bei Lenkeinschlag und Armgelenken.
Beschleunigungssensor	Zur Bewegungsschätzung.
Gyroskop	Messung von Drehraten zur Bewegungsschätzung.
Kompass	Messung der Orientierung zur Bewegungsschätzung.

## ▪ Exteriozeptive Sensoren

Sensor	Einsatz
Kamera	Lokalisierung, Bewegungsschätzung, Objekterkennung.
RGBD-Kamera	Kamera mit Tiefeninformation. Lokalisierung, Bewegungsschätzung, Objekterkennung, Hinderniserkennung.
Laser	Lokalisierung, Bewegungsschätzung, Objekterkennung, Hinderniserkennung.
Radar	Hinderniserkennung.
GPS	Lokalisierung bei Outdoor-Anwendungen.

# 2D-Laser-Scanner

- Entfernungsmesser durch Laufzeitmessung eines reflektierten Laserstrahls
- Durch drehenden Spiegel wird Entfernung in verschiedene Richtungen gemessen
- Typ. Scanbereich:  $180^\circ$  –  $270^\circ$
- Typ. Winkelauflösung:  $0.25^\circ$ ,  $0.5^\circ$  oder  $1^\circ$



Messprinzip  
Bilder von [www.sick.com](http://www.sick.com)



Sick LMS 100  
Winkelauflösung:  $0.25^\circ$  /  $0.5^\circ$   
Scanbereich:  $270^\circ$   
Scanfrequenz 25/50 Hz  
Reichweite: 18 m

# RGBD-Kamera

---

- Wertet Verzerrung eines projizierten Infrarot-Musters aus und berechnet daraus Tiefeninformation für jeden Pixel.
- Liefert 3D-Punktwolke mit RGB-Farbwerten.
- Im Vergleich zum Laser-Scanner geringerer Öffnungswinkel.



Asus (Xtion PRO)

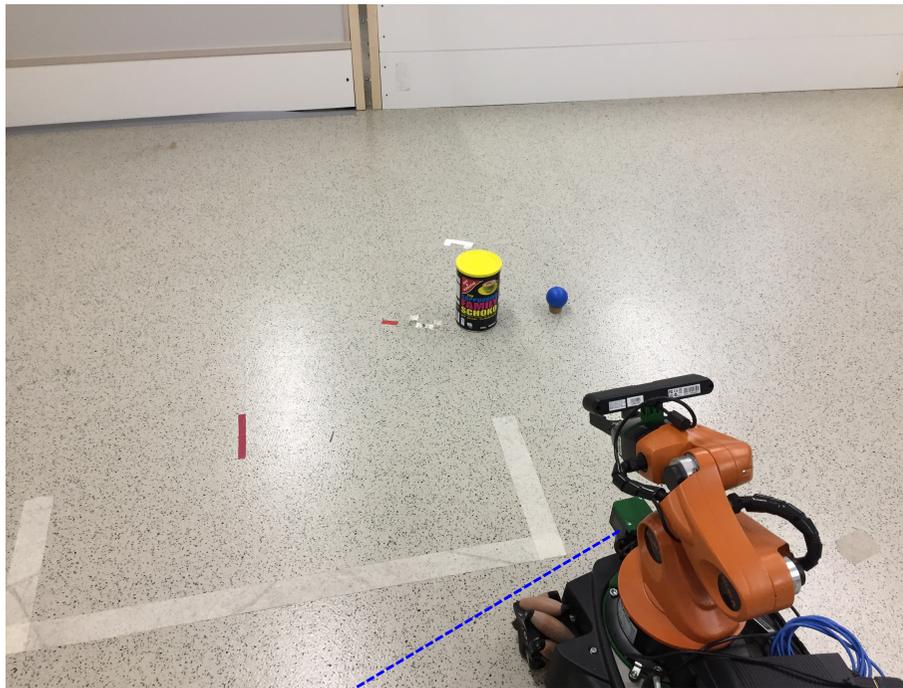
Auflösung: 1280x1024 Pixel

Tiefeninformation: 640x480 Pixel bei 30 fps

Tiefenbereich: 0,8 - 3,5m

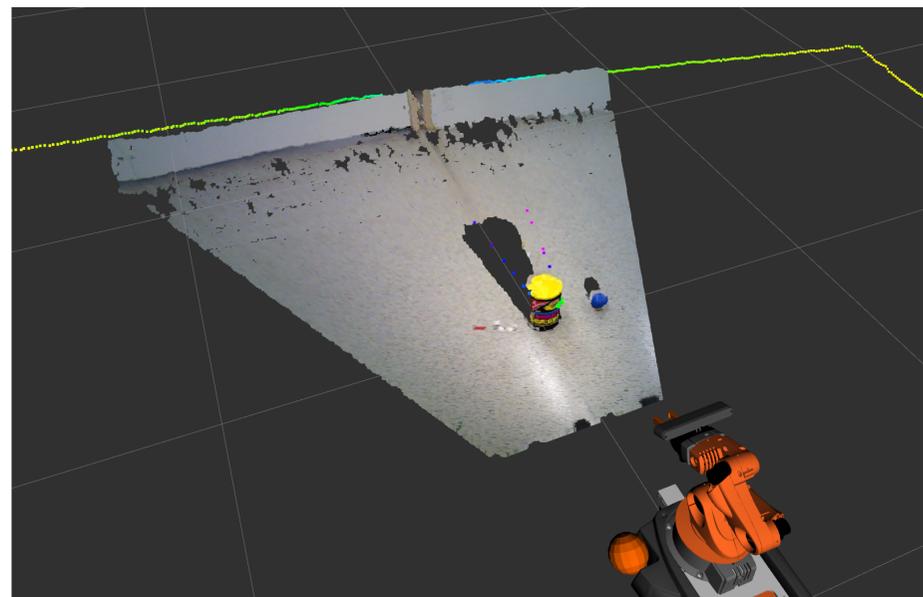
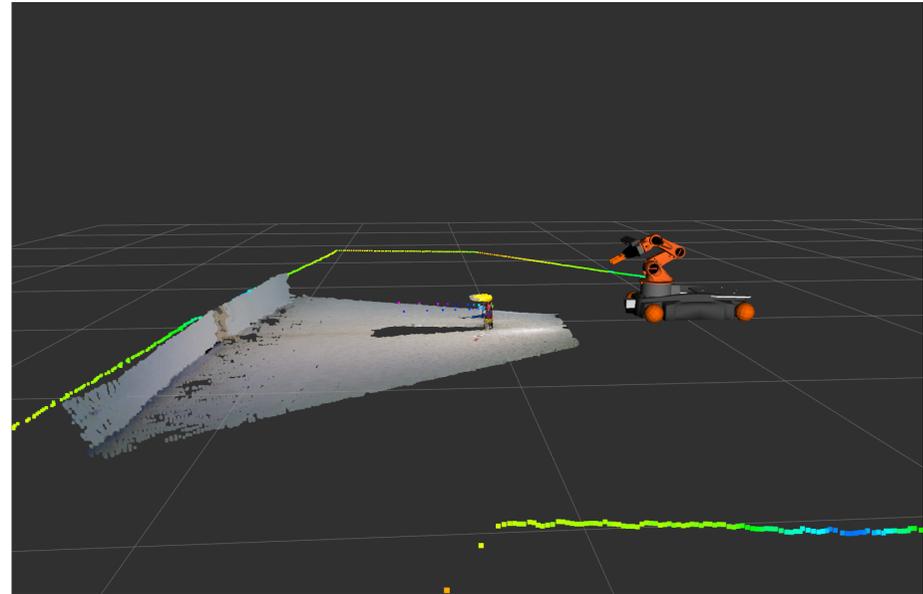
Öffnungswinkel: 58° H, 45° V

# Beispiel für Laser- und RGBD-Daten



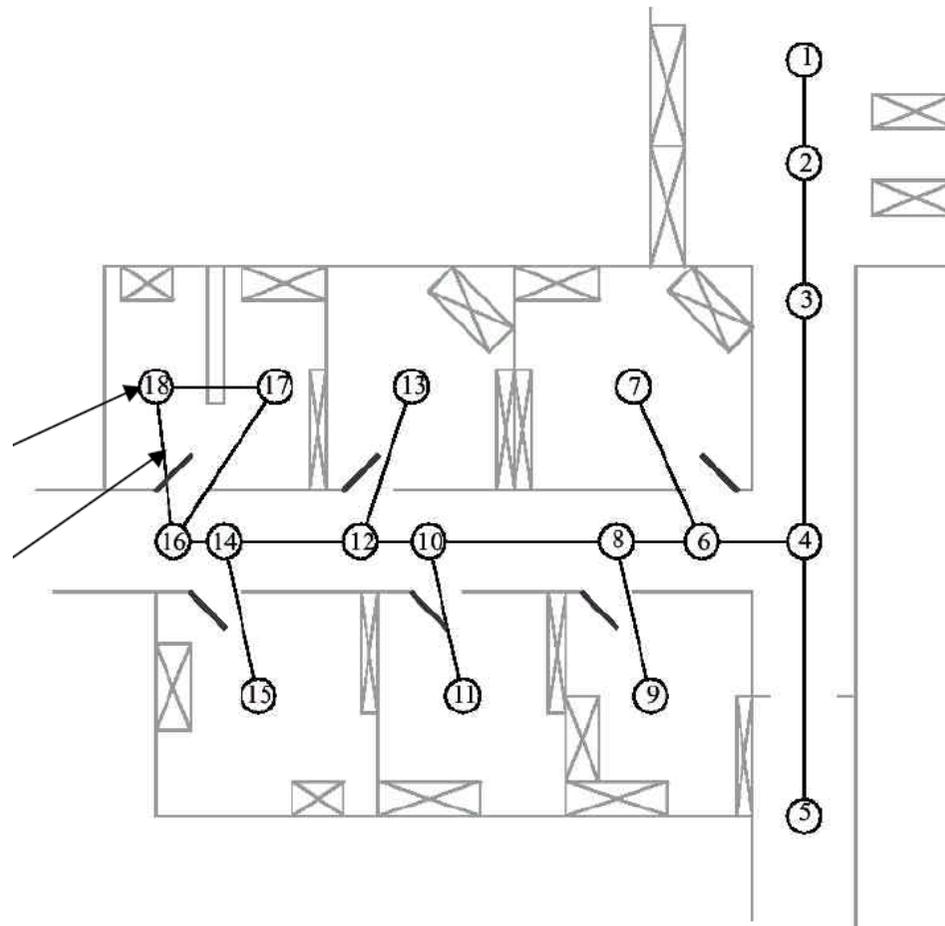
HOKUYO UST-10LX Laser

Laserdaten und RGBD-Daten  
mit ROS-Werkzeug Rviz  
visualisiert.



# Metrische und topologische Karten

- Metrische Karten sind in einem KS eingezeichnet
- Topologische Karten beschreiben Nachbarschaftbeziehungen von relevanten Orten (Raum, Tür, Flur, ...) in der Umgebung als Graph



# Gitterbasierte Karten

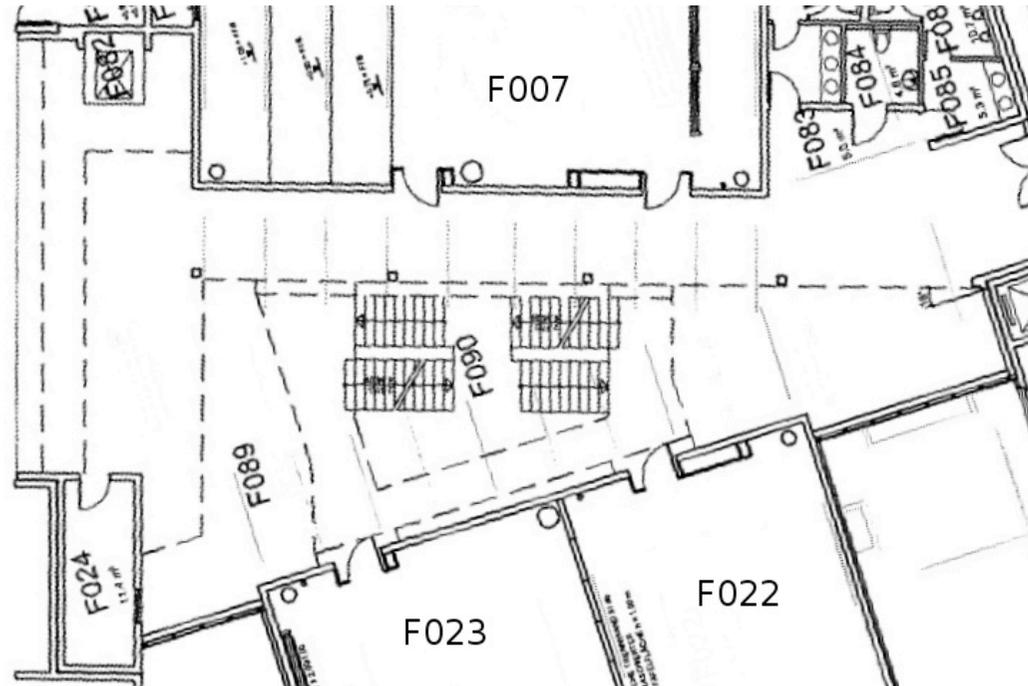
---



- Belegtheitsgitter  
schwarz = belegt, weiss = frei, grau = unbekannt.
- Je nach Auflösung sehr detailliert und speicheraufwendig.
- Typisches Ergebnis bei autonomer Kartenerstellung.
- Oft verwendet bei Planungs- und Navigationsverfahren.

# Linienbasierte Karten

---



- Vektorgraphik.
- Wenig speicheraufwendig.
- Autonome Erstellung aufwendig.

# Punktbasierte Karten

---



- Karte besteht aus einer Menge von Hindernispunkten.
- Wenig speicheraufwendig.
- Typisches Ergebnis bei autonomer Kartenerstellung.
- Hier: aus Laserscans zusammengesetzt (Scan Map).

# Übersicht über Lokalisierungsverfahren

---

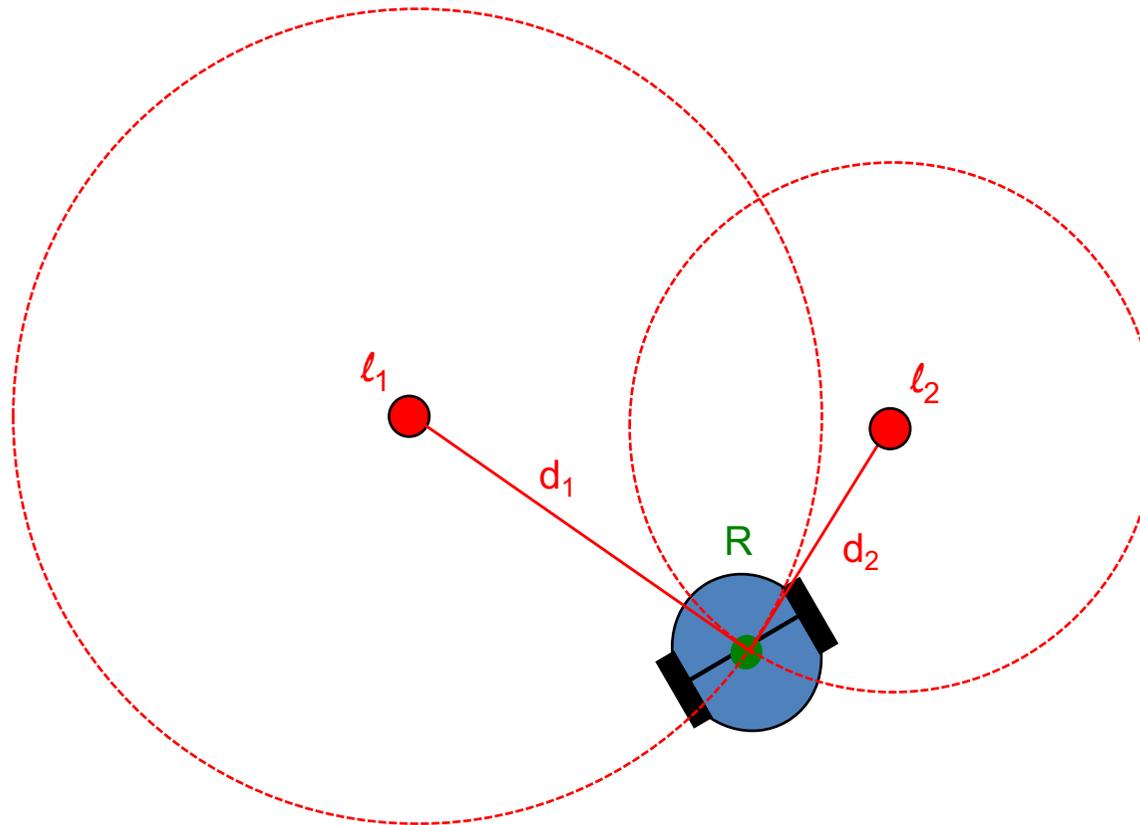
Verfahren	Lokalisierungs- variante	Lokalisierungs- genauigkeit	Typische Sensorik	Umgebungs- karte
Laterations- verfahren	lokal und global	sehr genau	Abstands- sensor	Landmarken
Koppel- navigation	lokal	Drift	Odometrie	nein
Kalman-Filter	lokal	sehr genau	Odometrie + Abstands- sensor	ja
Partikel-Filter	lokal und global	genau	Odometrie + Abstands- sensor	ja

# Lokalisierung

- Einleitung
  - Problemstellung, Varianten,  
Sensorik, Umgebungskarten, Verfahren
- Laterationsverfahren
- Koppelnavigation
  - Idee, Navigationsgleichungen,  
Fehlerfortpflanzung
- Lokalisierung mit einem Kalman-Filter
  - Idee, Algorithmus für linearen und erweiterten Kalmanfilter,  
Beispiele, Selbstlokalisierung mit einem Kalmanfilter
- Monte-Carlo-Lokalisierung
  - Idee, Algorithmus, Beispiele

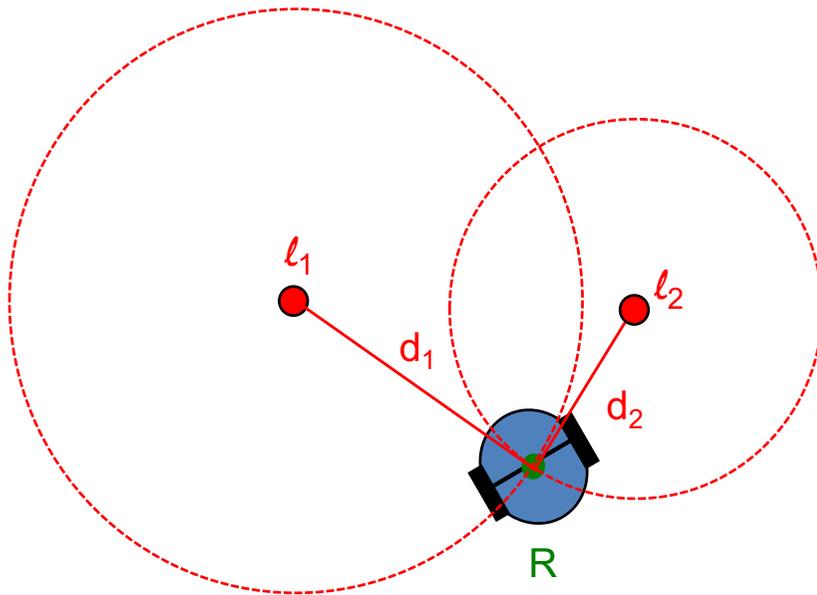
# Laterationsverfahren

---



- Messe Entfernung  $d_i$  zu Landmarken  $l_i$  mit bekannten Positionen
- Berechne daraus Position  $\mathbf{x}_R$  (ohne Orientierung) des Roboters R.
- Bei 2D-Position: 2 Landmarken mit Entfernungsmessungen
- Bei 3D-Position: 3 Landmarken mit Entfernungsmessungen

# Algorithmus für 2D-Lateration



- Gleichungssystem mit unbekannter Roboterposition  $\mathbf{x}_R = (x_R, y_R)$  und bekannten Landmarkenpositionen  $(x_1, y_1)$ ,  $(x_2, y_2)$ .

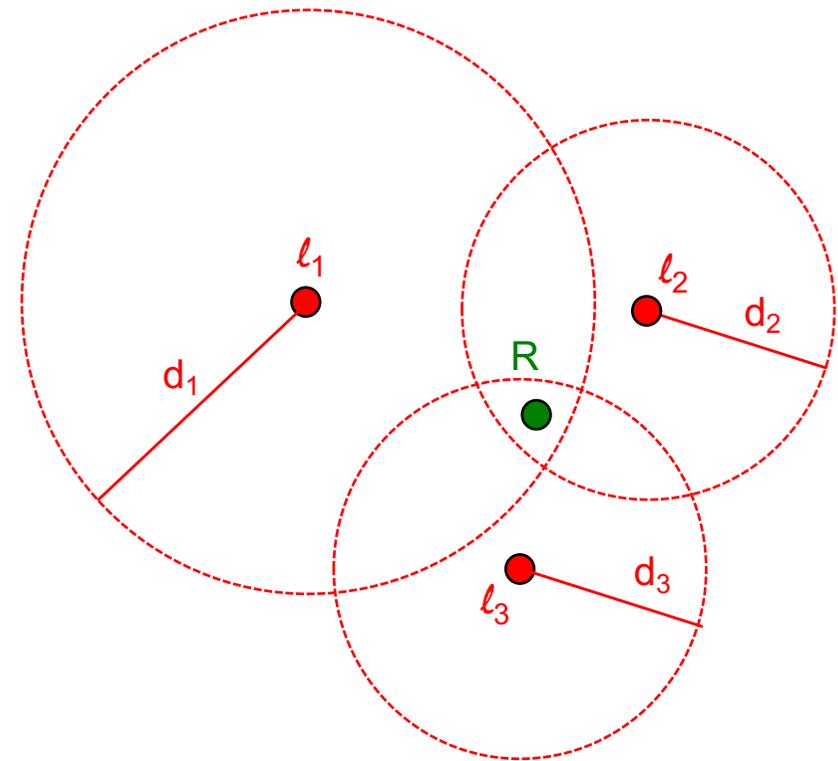
$$\begin{aligned}(x_1 - x_R)^2 + (y_1 - y_R)^2 - d_1^2 &= 0 \\(x_2 - x_R)^2 + (y_2 - y_R)^2 - d_2^2 &= 0\end{aligned}$$

- Iteratives Verfahren statt explizites Auflösen der Gleichungen.
- Fasse Gleichungssystem auf als Nullstellenproblem: finde  $\mathbf{x}_R = (x_R, y_R)$ , so dass Gleichungen erfüllt sind.
- Wende dazu Newton-Verfahren an.
- Problem: Es gibt i.a. 2 Nullstellen (Kreise schneiden sich an 2 Stellen). Starte daher das Verfahren in der Nähe der Roboterposition.

# Least-Square-Verfahren bei mehr als 2 Landmarken

- Bei mehr als 2 Landmarken überbestimmtes Gleichungssystem.

$$\begin{aligned}(x_1 - x_R)^2 + (y_1 - y_R)^2 - d_1^2 &= 0 \\(x_2 - x_R)^2 + (y_2 - y_R)^2 - d_2^2 &= 0 \\(x_3 - x_R)^2 + (y_3 - y_R)^2 - d_3^2 &= 0 \\&\dots\end{aligned}$$

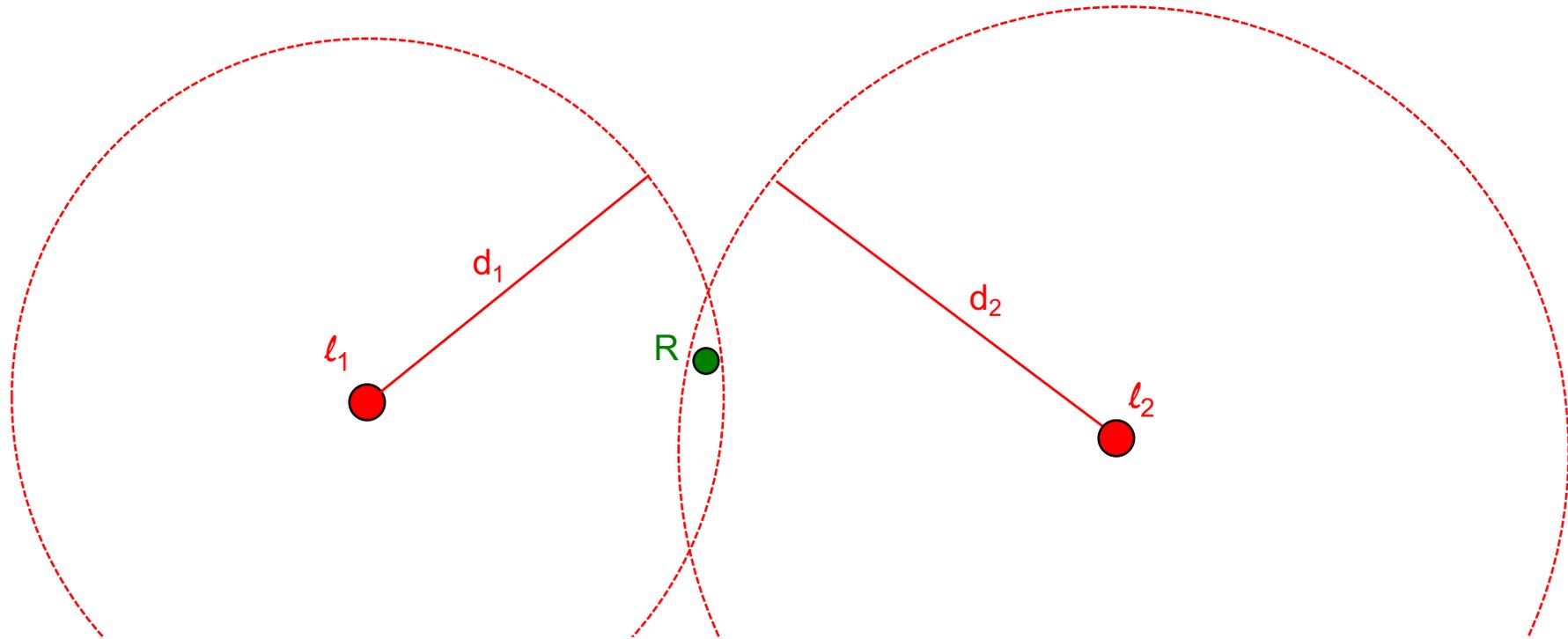


- I.a. keine Lösung, da sich Kreise nicht in einem Punkt schneiden.
- Least-Square-Ansatz: finde Roboterposition  $\mathbf{x}_R = (x_R, y_R)$ , so dass Summe der Fehler-Quadrate SSE minimal wird:

$$SSE = \sum_i [(x_i - x_R)^2 + (y_i - y_R)^2 - d_i^2]^2$$

# Probleme bei ungünstiger Geometrie

---



- Bei ungünstiger Geometrie (z.B. 2 Landmarken mit kleinem Schnittwinkel der Kreise) ist die Mehrdeutigkeit schwer auflösbar.
- Außerdem sind die Entfernungsmessungen fehlerbehaftet, so dass sich ein großer Lokalisierungsfehler ergeben kann.
- Abhilfe: Sensordaten bei sich bewegendem Roboter integrieren. Genau das leisten Kalman- und Partikelfilter (später).

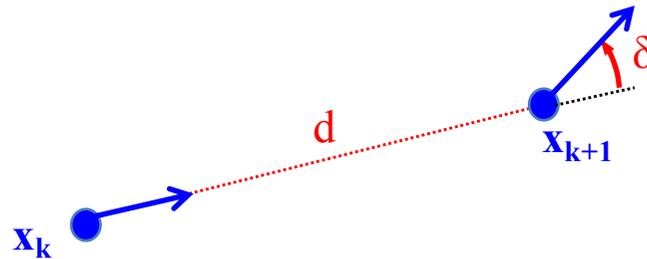
# Lokalisierung

- Einleitung
  - Problemstellung, Varianten,  
Sensorik, Umgebungskarten, Verfahren
- Laterationsverfahren
- Koppelnavigation
  - Idee, Navigationsgleichungen,  
Fehlerfortpflanzung
- Lokalisierung mit einem Kalman-Filter
  - Idee, Algorithmus für linearen und erweiterten Kalmanfilter,  
Beispiele, Selbstlokalisierung mit einem Kalmanfilter
- Monte-Carlo-Lokalisierung
  - Idee, Algorithmus, Beispiele

# Idee der Koppelnavigation

---

- Schätze die neue Pose  $\mathbf{x}_{k+1}$  aufgrund der alten Pose  $\mathbf{x}_k$ , der zurückgelegten Strecke  $d$  und der Richtungsänderung  $\delta$ .



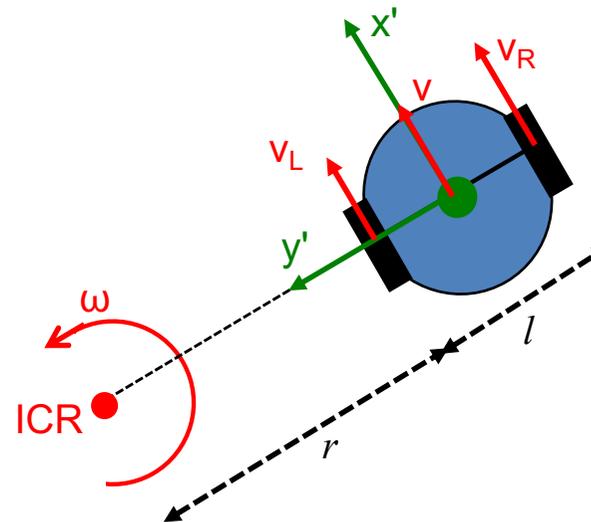
- kommt aus der Seefahrt; engl. [dead reckoning](#).
- Die Messung des zurückgelegten Wegs wird auch Odometrie genannt. [Odometrie](#) = Hodometrie = hodos (griech. Weg) + metron (griech. Maß).
- Bei Radfahrzeugen lässt sich die Odometrie einfach über Messung der Radumdrehungen durchführen.
- Eine Richtungsänderung kann auch über Radumdrehungen gemessen werden.

# Wiederholung Differentialantrieb

- Durch Radsensoren lassen sich die Geschwindigkeiten der beiden Räder  $v_L$ ,  $v_R$  messen.
- Aus  $v_L$ ,  $v_R$  und der Achslänge  $l$  können  $v$  und  $\omega$  unmittelbar hergeleitet werden.

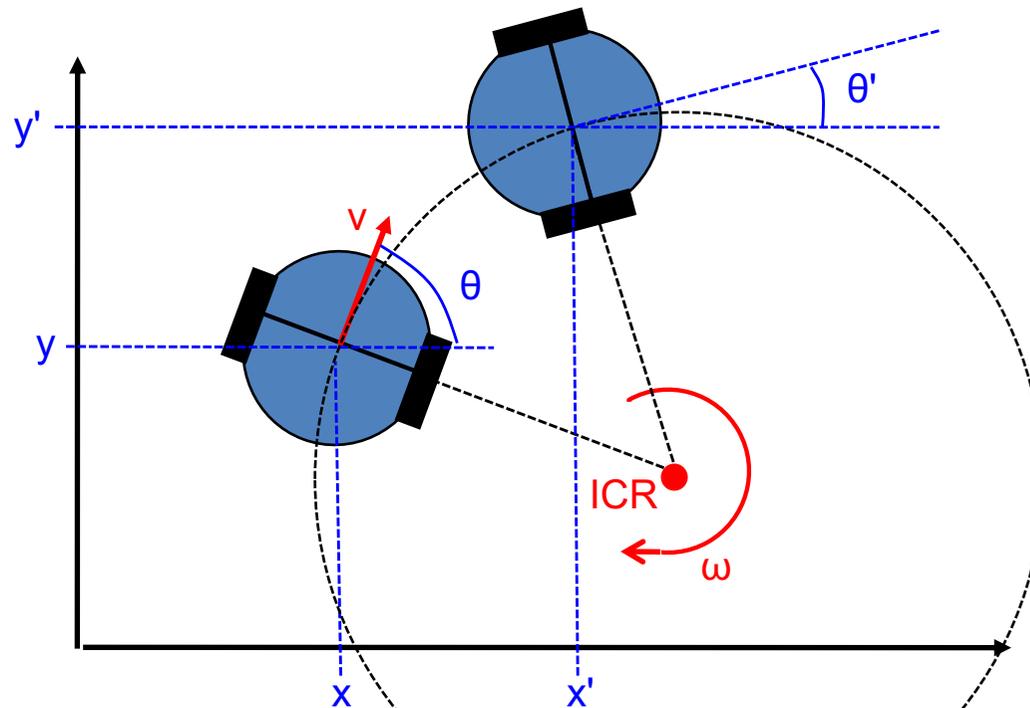
$$v = \frac{v_R + v_L}{2}$$

$$\omega = \frac{v_R - v_L}{l}$$



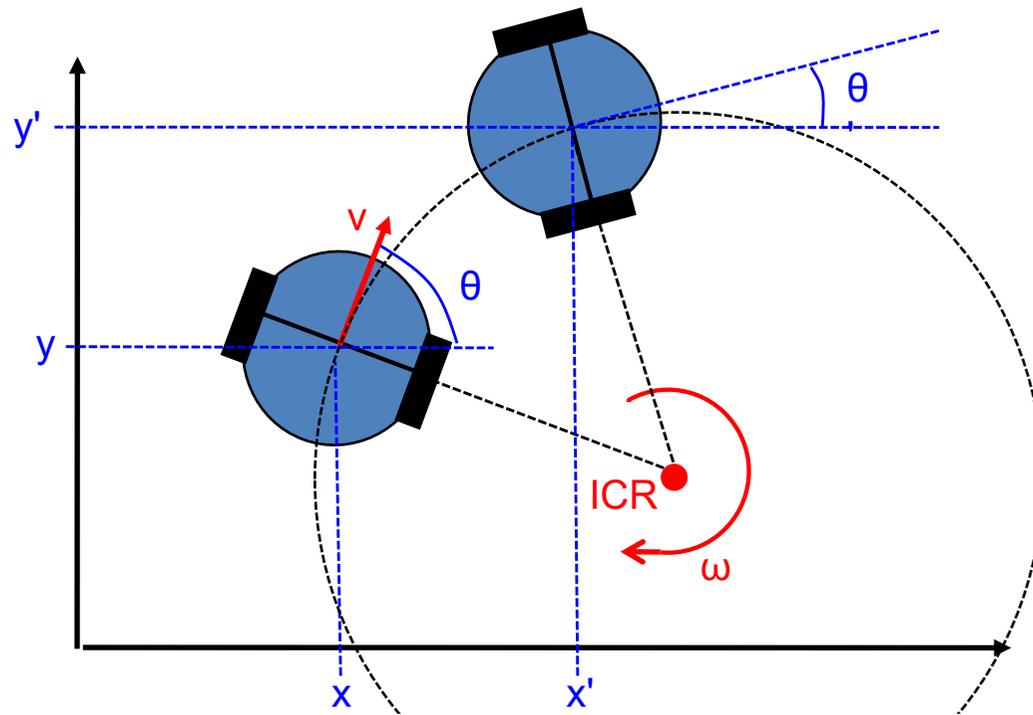
# Koppelnavigation für Radfahrzeuge (1)

- Annahme: aktuelle Geschwindigkeit  $v$  und Winkelgeschwindigkeit  $\omega$  sind bekannt (entweder als Messwert oder als Steuerbefehl) und sind im aktuellen Zeitabschnitt  $T$  (z.B.  $T = 0.01$  sec) konstant.
- Damit: Roboter fährt einen Kreisbogen mit bekanntem Radius und bekannter Länge.
- Hier: Differentialantrieb.  
Ackermann- und Mecanumantrieb werden analog behandelt.



- Pose zum Zeitpunkt  $t_k$ :  
$$\mathbf{x}_k = (x, y, \theta)$$
- Pose zum Zeitpunkt  $t_{k+1}$ :  
$$\mathbf{x}_{k+1} = (x', y', \theta')$$

# Koppelnavigation für Radfahrzeuge (2)



- Pose zum Zeitpunkt  $t_k$ :  
$$\mathbf{x}_k = (x, y, \theta)$$
- Pose zum Zeitpunkt  $t_{k+1}$ :  
$$\mathbf{x}_{k+1} = (x', y', \theta')$$

- Approximation: Roboter fährt im Zeitschritt  $T$  in Richtung  $\theta + \omega T/2$  (halbe Richtungsänderung) die Strecke  $vT$ :

$$x' = x + vT * \cos(\theta + \omega T/2)$$

$$y' = y + vT * \sin(\theta + \omega T/2)$$

$$\theta' = \theta + \omega T$$

# Bemerkungen

---

- In der Literatur wird der Term  $\omega T/2$  oft weggelassen. Die Approximation ist dann weniger genau:

$$x' = x + vT * \cos(\theta)$$

$$y' = y + vT * \sin(\theta)$$

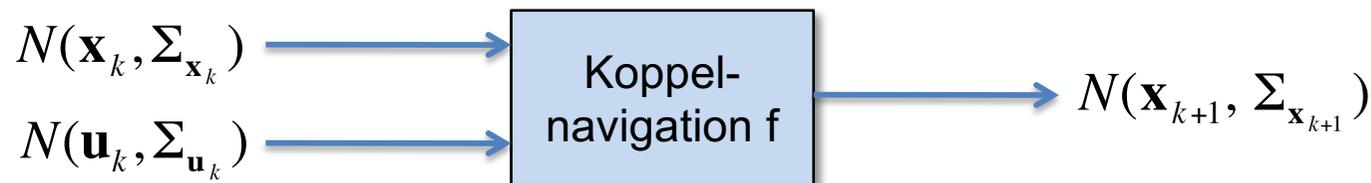
$$\theta' = \theta + \omega T$$

- Die Kreisbewegung kann auch exakt berechnet werden. Die Approximation wird dadurch genauer:
  - aus  $v$ ,  $\omega$  wird der Radius  $r$  der Kreisbewegung berechnet.
  - aus alter Pose  $x$ ,  $y$ ,  $\theta$  und Radius  $r$  lässt sich ICR bestimmen.
  - aus ICR,  $\theta'$  und Radius  $r$  lässt sich neue Position  $x'$ ,  $y'$  bestimmen.

# Fehlerfortpflanzung bei Koppelnavigation (1)

- Es soll nun untersucht werden, wie sich eine normalverteilte Unsicherheit in der Position des Roboters und im Steuerbefehl in eine Unsicherheit des Folgezustands fortpflanzt.
- Es wird von den vereinfachten Koppelnavigationsgleichungen von S. 5-23 ausgegangen.

$$\underbrace{\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{pmatrix}}_{\mathbf{x}_{k+1}} = \underbrace{\begin{pmatrix} x_k \\ y_k \\ \theta_k \end{pmatrix}}_{\mathbf{x}_k} + \underbrace{\begin{pmatrix} T \cos \theta_k & 0 \\ T \sin \theta_k & 0 \\ 0 & T \end{pmatrix}}_{f(\mathbf{x}_k, \mathbf{u}_k)} \underbrace{\begin{pmatrix} v_k \\ \omega_k \end{pmatrix}}_{\mathbf{u}_k}$$



# Fehlerfortpflanzung bei Koppelnavigation (2)

- Aus Kapitel 4 folgt:

$$\Sigma_{x_{k+1}} = \mathbf{F} \Sigma_{x u_k} \mathbf{F}^T$$

$$\mathbf{F} = \frac{\partial f}{\partial (\mathbf{x}_k, \mathbf{u}_k)} (\mathbf{x}_k, \mathbf{u}_k)$$

} Jacobi-Matrix

$$\Sigma_{x u_k} = \begin{pmatrix} \Sigma_{x_k} & \mathbf{0}_{3 \times 2} \\ \mathbf{0}_{2 \times 3} & \Sigma_{u_k} \end{pmatrix}$$

Position  $\mathbf{x}_k$  und Steuerbefehl  $\mathbf{u}_k$  sind nicht korreliert.

- Formel lässt sich umschreiben in:

$$\Sigma_{x_{k+1}} = \mathbf{F}_{x_k} \Sigma_{x_k} \mathbf{F}_{x_k}^T + \mathbf{F}_{u_k} \Sigma_{u_k} \mathbf{F}_{u_k}^T$$

$$\mathbf{F}_{x_k} = \frac{\partial f}{\partial \mathbf{x}_k} (\mathbf{x}_k)$$

$$\mathbf{F}_{u_k} = \frac{\partial f}{\partial \mathbf{u}_k} (\mathbf{u}_k)$$

Teile der Jacobi-Matrix  $\mathbf{F}$

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_{x_k} & \mathbf{F}_{u_k} \end{pmatrix}$$

# Berechnung der Jacobi-Matrizen

- Koppelnavigation:

$$\underbrace{\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{pmatrix}}_{\mathbf{x}_{k+1}} = \underbrace{\begin{pmatrix} x_k \\ y_k \\ \theta_k \end{pmatrix}}_{\mathbf{x}_k} + \underbrace{\begin{pmatrix} T \cos \theta_k & 0 \\ T \sin \theta_k & 0 \\ 0 & T \end{pmatrix}}_{f(\mathbf{x}_k, \mathbf{u}_k)} \underbrace{\begin{pmatrix} v_k \\ \omega_k \end{pmatrix}}_{\mathbf{u}_k}$$

- Jacobi-Matrizen:

$$\mathbf{F}_{\mathbf{x}_k} = \frac{\partial f}{\partial \mathbf{x}_k} = \begin{pmatrix} 1 & 0 & -T v_k \sin \theta_k \\ 0 & 1 & T v_k \cos \theta_k \\ 0 & 0 & 1 \end{pmatrix}$$

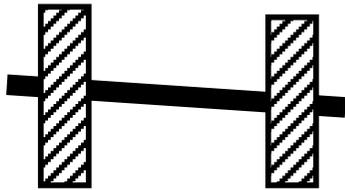
$$\mathbf{F}_{\mathbf{u}_k} = \frac{\partial f}{\partial \mathbf{u}_k} = \begin{pmatrix} T \cos \theta_k & 0 \\ T \sin \theta_k & 0 \\ 0 & T \end{pmatrix}$$

- Fehlerfortpflanzung:

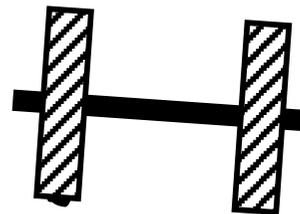
$$\Sigma_{\mathbf{x}_{k+1}} = \mathbf{F}_{\mathbf{x}_k} \Sigma_{\mathbf{x}_k} \mathbf{F}_{\mathbf{x}_k}^T + \mathbf{F}_{\mathbf{u}_k} \Sigma_{\mathbf{u}_k} \mathbf{F}_{\mathbf{u}_k}^T$$

# Unsicherheit des Steuerbefehls

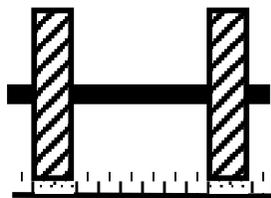
- Der Steuerbefehl  $\mathbf{u} = (v, \omega)$  ergibt sich aus der Messung der Geschwindigkeit des linken und des rechten Rads und der Achslänge  $l$ .
- Für die Ungenauigkeit gibt es verschiedene Ursachen.



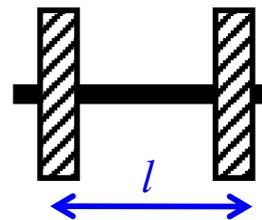
Unterschiedliche  
Raddurchmesser



Unebenheiten



Untergrund



Auflagepunkt nicht genau in Radmitte,  
daher ist auch  $l$  fehlerbehaftet

# Fehlermodell für Steuerbefehl (1)

---

- Die Geschwindigkeitsmessung besteht im wesentlichen aus der Messung der im k-ten Zeitintervall zurückgelegten Strecke  $d_k$ . Hierbei ist der Fehler:

$$\sigma_{d_k}^2 = k_d * d_k$$

- Dabei ist  $k_d$  eine Konstante und beschreibt, welche Fehler (Varianz) bei der Messung von 1 Meter gemacht wird (z.B.  $k_d = (0.05\text{m})^2/1\text{m}$ ).  $k_d$  kann durch eine statistische Untersuchung ermittelt werden.
- Für die Fehler der Geschwindigkeit  $v_k = d_k/T$  ergibt sich dann:

$$\begin{aligned}\sigma_{v_k}^2 &= \frac{1}{T^2} * \sigma_{d_k}^2 \\ &= \frac{k_d}{T^2} * d_k \\ &= \frac{k_d}{T} * v_k\end{aligned}$$

# Fehlermodell für Steuerbefehl (2)

---

- Eine ähnliche Überlegung für die Winkelgeschwindigkeit  $\omega$  führt auf:

$$\sigma_{\omega_k}^2 = \frac{k_\theta}{T} * \omega_k$$

- Dabei ist  $k_\theta$  eine Konstante, die angibt, welcher Fehler (Varianz) bei einer 360-Drehung gemacht wird (z.B.  $(5 \text{ Grad})^2/360 \text{ Grad}$ ).
- Zusätzlich gehen wir von einer weiteren Dreh-Fehler (Drift) aus, der von der zurückgelegten Strecke abhängt:

$$\sigma_{\omega_k}^2 = \frac{k_{\text{Drift}}}{T} * v_k$$

- Dabei ist  $k_{\text{Drift}}$  eine Konstante, die angibt, welcher Winkelgeschwindigkeitsfehler (Varianz) bei einer 1 Meter langen Fahrt gemacht wird (z.B.  $(2 \text{ Grad})^2/1\text{m}$ ).

# Fehlermodell für Steuerbefehl (3)

---

- Wir gehen davon aus, dass die Fehler bei der Geschwindigkeit und der Winkelgeschwindigkeit nicht korreliert sind und erhalten damit:

$$\Sigma_{\mathbf{u}_k} = \begin{pmatrix} \sigma_{v_k}^2 & 0 \\ 0 & \sigma_{\omega_k}^2 \end{pmatrix}$$

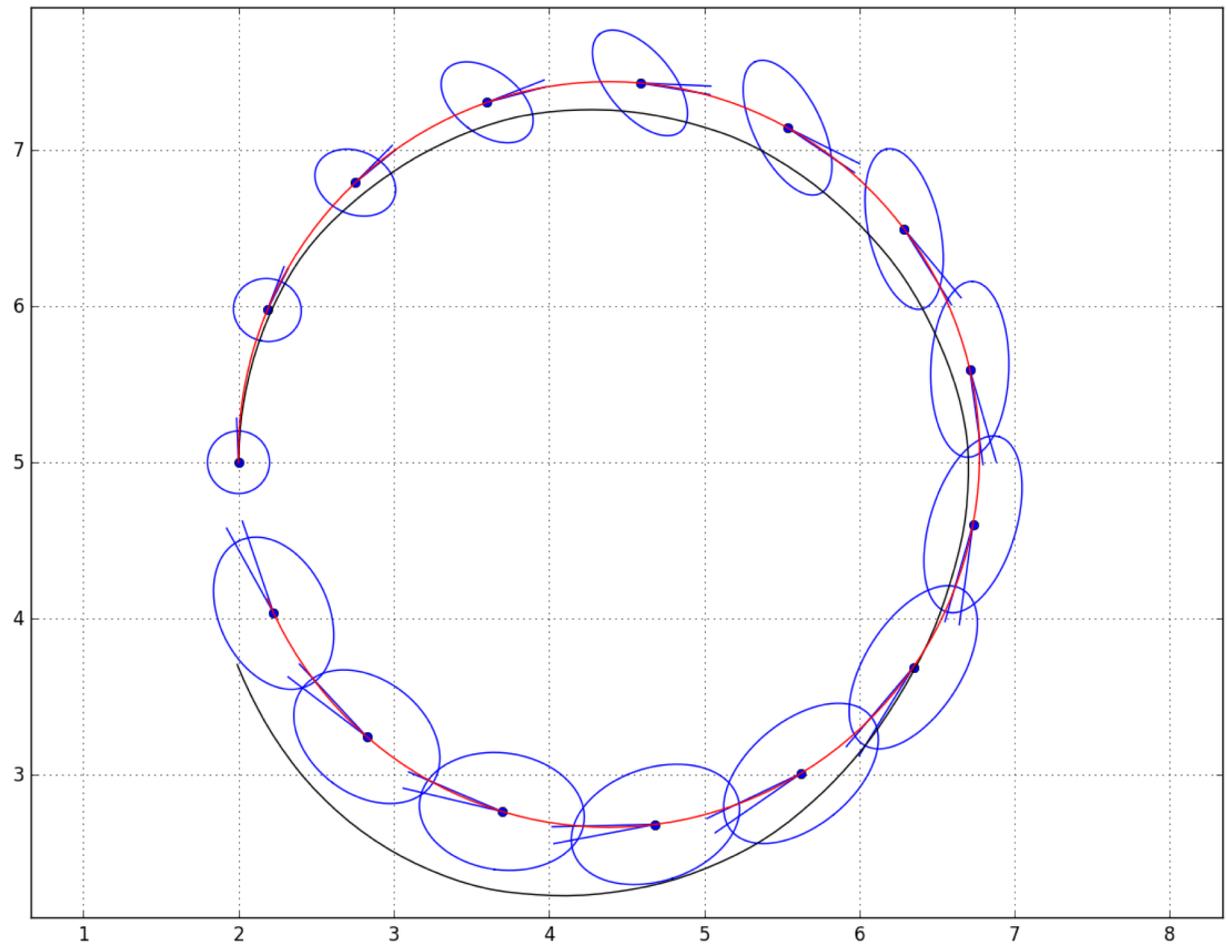
$$\sigma_{v_k}^2 = \frac{k_d}{T} * v_k$$

$$\sigma_{\omega_k}^2 = \frac{k_\theta}{T} * \omega_k + \frac{k_{\text{Drift}}}{T} * v_k$$

- Beachte: bei negativer Winkelgeschwindigkeit (Rechtskurve) bzw. negativer Geschwindigkeit (Rückwärtsfahrt) sind Beträge zu nehmen.

# Simulation

- tatsächliche Strecke in schwarz
- Kreisfahrt mit konstanter Geschwindigkeit  $v = 1$  m/sec und  $\omega = 24$  Grad/sec.
- mit Koppelnavigation geschätzte Strecke in rot, mit Kovarianzen in blau
- Abtastintervall  $T = 0.1$  sec und  $n = 140$  Zeitschritte.
- Fehlermodell:
  - $k_d = (0.02\text{m})^2/1\text{m}$
  - $k_\theta = (5 \text{ Grad})^2/360 \text{ Grad}$
  - $k_{\text{Drift}} = (2 \text{ Grad})^2/1\text{m}$



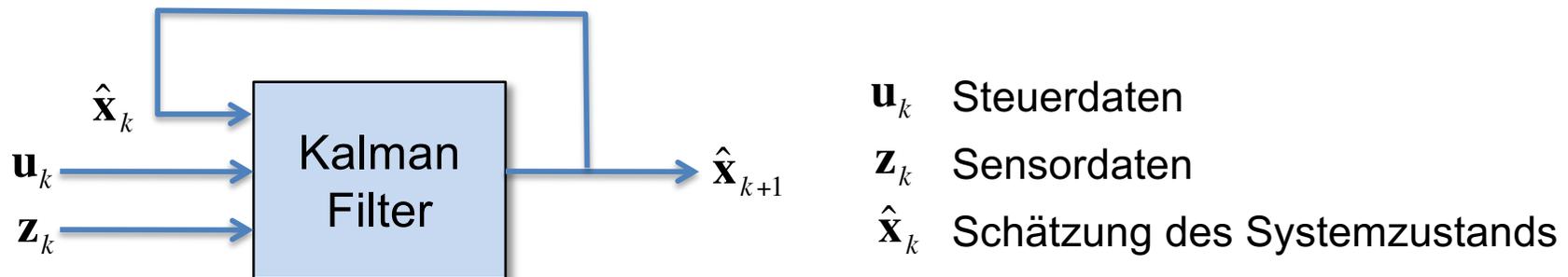
Einheiten in m

# Lokalisierung

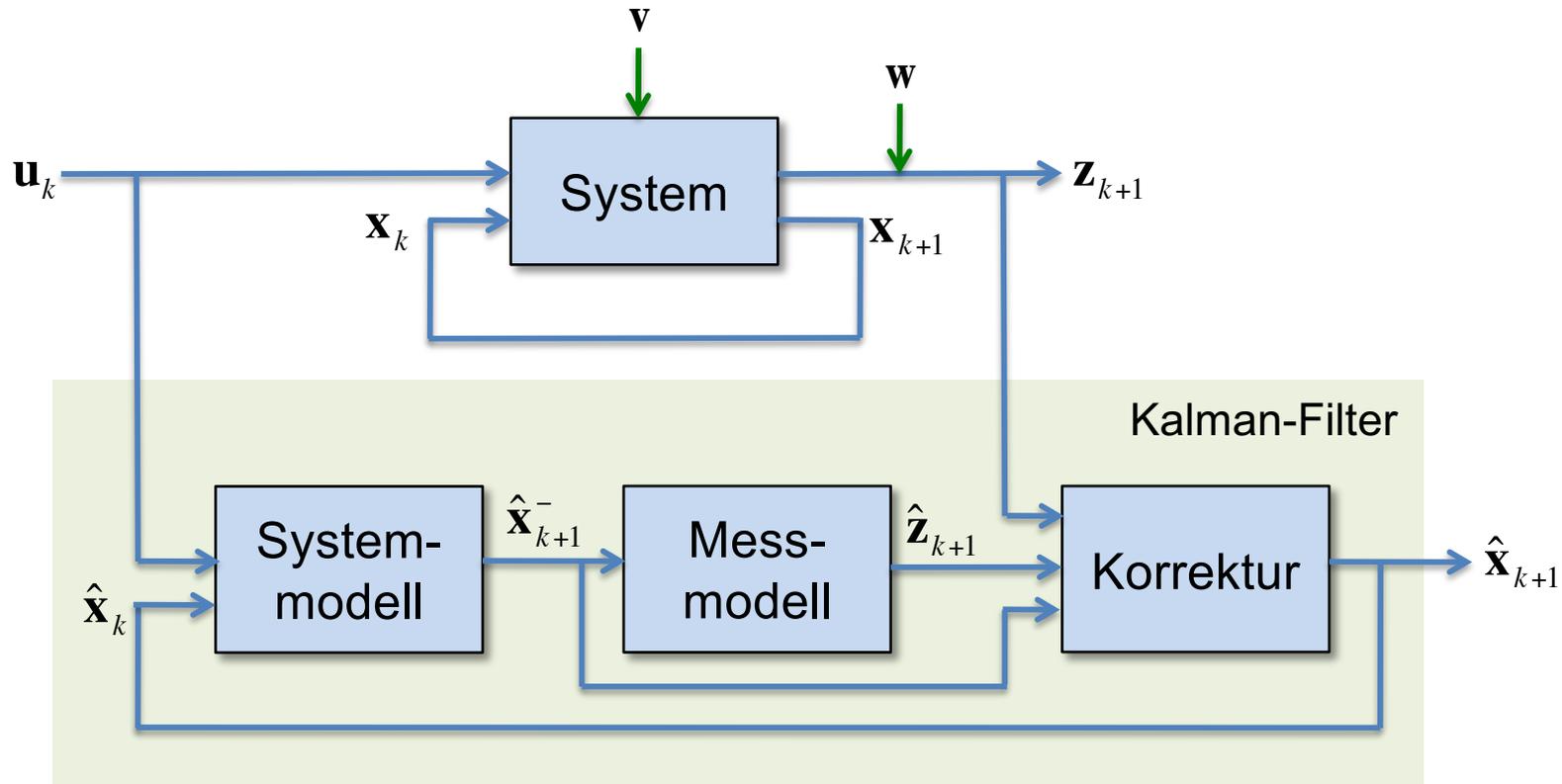
- Einleitung
  - Problemstellung, Varianten,  
Sensorik, Umgebungskarten, Verfahren
- Laterationsverfahren
- Koppelnavigation
  - Idee, Navigationsgleichungen,  
Fehlerfortpflanzung
- Lokalisierung mit einem Kalman-Filter
  - Idee, Algorithmus für linearen und erweiterten Kalmanfilter,  
Beispiele, Selbstlokalisierung mit einem Kalmanfilter
- Monte-Carlo-Lokalisierung
  - Idee, Algorithmus, Beispiele

# Ziel des Kalman-Filters

- Die Koppelnavigation liefert eine Schätzung der Position. Die Schätzung driftet jedoch sehr stark.
- Es liegt nun nahe, zusätzlich Sensorwerte (z.B. Abstandswerte zu Landmarken) zu verwenden, um die Positionsschätzung zu verbessern.
- Die Integration von Sensorwerten kann mit einem Kalmanfilter erreicht werden.
- Der Kalmanfilter berechnet aufgrund von Steuerdaten, Messdaten und einer alten Schätzung des Systemzustands einen neuen verbesserten Schätzwert.



# Funktionsweise des Kalman-Filters



$\mathbf{u}_k$  Steuerdaten

$\mathbf{z}_k$  Sensordaten

$\mathbf{x}_k$  Systemzustand

$\mathbf{v}$  Systemrauschen

$\mathbf{w}$  Messrauschen

$\hat{\mathbf{x}}_k$  Schätzung des Systemzustands zum Zeitpunkt  $t_k$

$\hat{\mathbf{x}}_{k+1}^-$  Vorhersage des Systemzustands zum Zeitpunkt  $t_{k+1}$   
(Messwert noch nicht berücksichtigt!)

$\hat{\mathbf{z}}_{k+1}$  Schätzung des Messwerts aufgrund des vorhergesagten  
Systemzustands

$\hat{\mathbf{x}}_{k+1}$  Neuer Schätzwert des Systemzustands zum Zeitpunkt  $t_{k+1}$

# Lineares System- und Messmodell

---

- Das lineare Systemmodell ist mit einem normalverteilten und mittelwertfreien Systemrauschen  $\mathbf{v}_k$  versehen.

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{v}_k$$
$$\mathbf{v}_k \sim N(\mathbf{0}, \Sigma_{\mathbf{v}})$$

- Wir gehen davon aus, dass das Systemrauschen im wesentlichen von einem verrauschten Steuerbefehl  $\mathbf{u}$  mit Kovarianz  $\Sigma_{\mathbf{u}}$  verursacht wird:

$$\Sigma_{\mathbf{v}} = \mathbf{B}\Sigma_{\mathbf{u}}\mathbf{B}^T$$

- Das lineare Messmodell beschreibt, wie sich Messwerte aus dem Systemzustand ergeben. Auch hier ist das Messmodell mit einem normalverteilten und mittelwertfreien Messrauschen  $\mathbf{w}_k$  versehen.

$$\mathbf{z}_k = \mathbf{C}\mathbf{x}_k + \mathbf{w}_k$$
$$\mathbf{w}_k \sim N(\mathbf{0}, \Sigma_{\mathbf{z}})$$

# Algorithmus für linearen Kalman-Filter

**KalmanFilter**(  $\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{z}_{k+1}$  ):

1. Vorhersage:

$$2. \quad \hat{\mathbf{x}}_{k+1}^- = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k$$

$$3. \quad \Sigma_{k+1}^- = \mathbf{A}\Sigma_k\mathbf{A}^T + \mathbf{B}\Sigma_u\mathbf{B}^T$$

4. Korrektur:

$$5. \quad \hat{\mathbf{z}}_{k+1} = \mathbf{C}\hat{\mathbf{x}}_{k+1}^-$$

$$6. \quad \mathbf{K} = \Sigma_{k+1}^- \mathbf{C}^T (\mathbf{C}\Sigma_{k+1}^- \mathbf{C}^T + \Sigma_z)^{-1}$$

$$7. \quad \hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_{k+1}^- + \mathbf{K}(\mathbf{z}_{k+1} - \hat{\mathbf{z}}_{k+1})$$

$$8. \quad \Sigma_{k+1} = (\mathbf{I} - \mathbf{K}\mathbf{C})\Sigma_{k+1}^-$$

9. **return**  $\hat{\mathbf{x}}_{k+1}$

Neuer Systemzustand  $\hat{\mathbf{x}}_{k+1}^-$  mit Kovarianz  $\Sigma_{k+1}^-$  wird vorhergesagt.

Aus dem vorhergesagten Systemzustand  $\hat{\mathbf{x}}_{k+1}^-$  wird ein erwarteter Messwert  $\hat{\mathbf{z}}_{k+1}$  berechnet

Mithilfe des Kalman-Gain  $\mathbf{K}$  wird der vorhergesagte Systemzustand korrigiert.

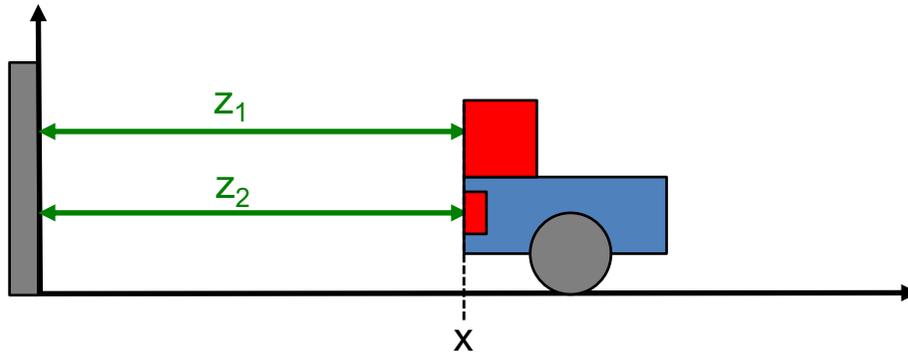
- Liegt in einem Zeitschritt kein Messwert vor, dann entfällt der Korrekturschritt, und es gilt:

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_{k+1}^-$$

$$\Sigma_{k+1} = \Sigma_{k+1}^-$$

# Beispiel: Fusionierung von zwei Messwerten

- Roboter ist stationär (bewegt sich nicht) und misst mit zwei unterschiedlichen Sensoren seine Position (= Abstand zur Wand).



- Systemmodell: stationärer Roboter ohne Steuerbefehl und Rauschen:

$$x_{k+1} = x_k$$

- Messmodell

$$\mathbf{z}_k = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} x_k + \mathbf{w}_k$$

# Anwendung des Kalmanfilters

Initiale Schätzung der Position ist sehr unsicher.

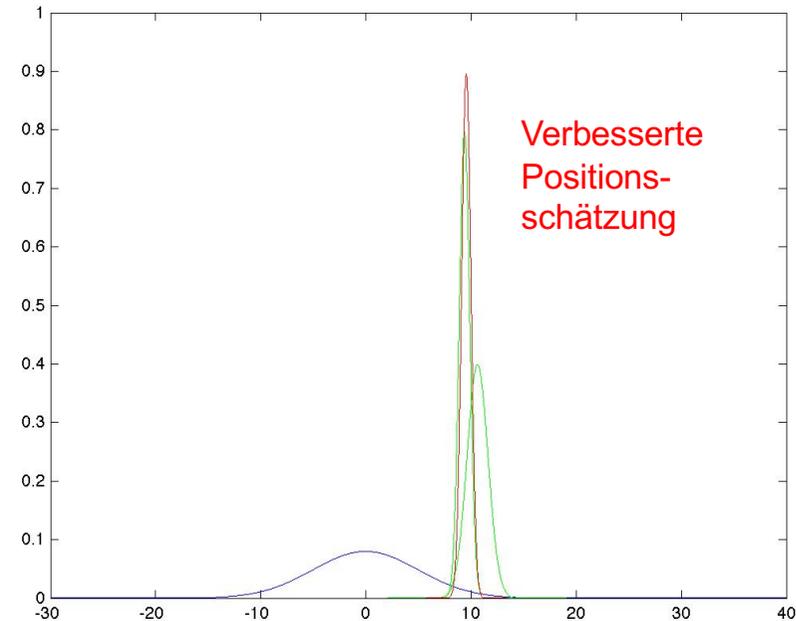
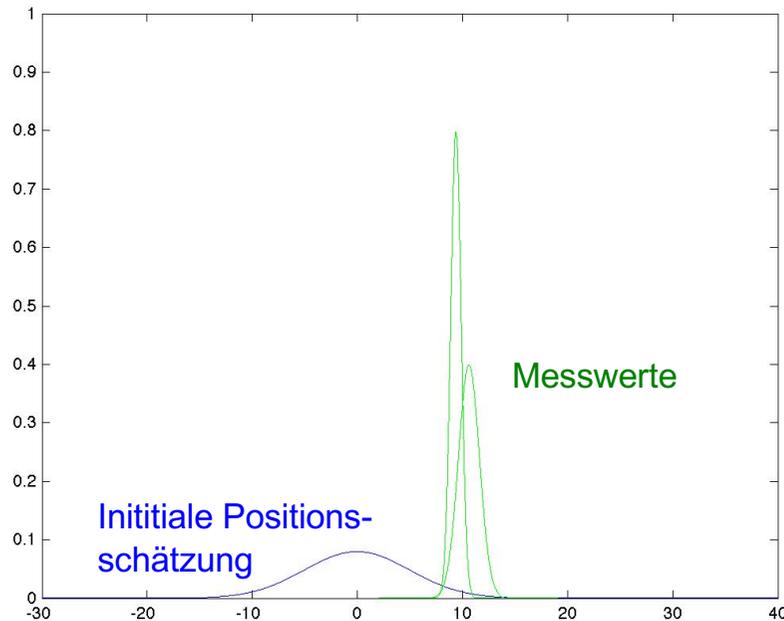
$$\hat{x}_0 = 0$$
$$\Sigma_0 = 20$$

Zwei Messwerte gegeben

$$\mathbf{z}_1 = \begin{pmatrix} 9.4 \\ 10.6 \end{pmatrix}$$
$$\Sigma_z = \begin{pmatrix} 0.5^2 & 0 \\ 0 & 1^2 \end{pmatrix}$$

Kalman Filter

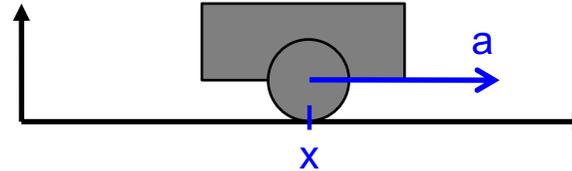
$$\hat{x}_1 = 9.6$$
$$\Sigma_0 = 0.2$$



Tatsächliche Position  $x = 10$

# Beispiel: bewegtes Fahrzeug (1)

- Roboter wird horizontal beschleunigt mit Steuerbefehl  $a$ .



- Systemzustand besteht aus Position und Geschwindigkeit:

$$\mathbf{x}_k = \begin{pmatrix} x_k \\ v_k \end{pmatrix}$$

- Steuervektor ist die Beschleunigung  $a$ :

$$u_k = a$$

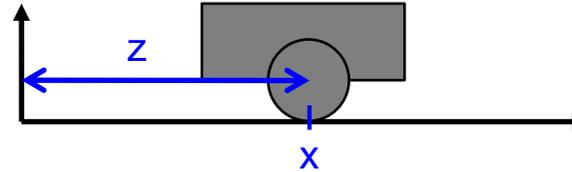
- Systemmodell

$$\underbrace{\begin{pmatrix} x_{k+1} \\ v_{k+1} \end{pmatrix}}_{\mathbf{x}_{k+1}} = \underbrace{\begin{pmatrix} 1 & T \\ 0 & 1 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} x_k \\ v_k \end{pmatrix}}_{\mathbf{x}_k} + \underbrace{\begin{pmatrix} 0 \\ T \end{pmatrix}}_{\mathbf{B}} \underbrace{a}_{u_k}$$

# Beispiel: bewegtes Fahrzeug (2)

---

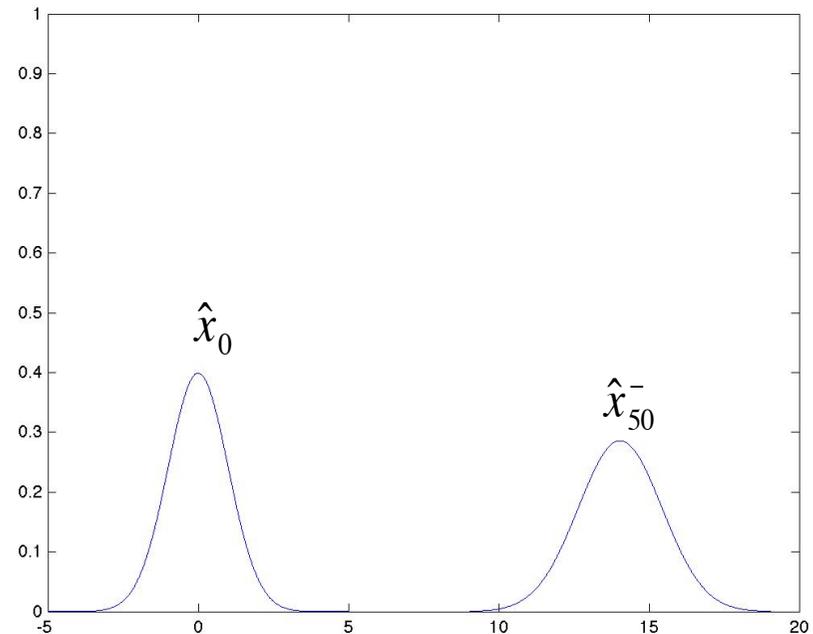
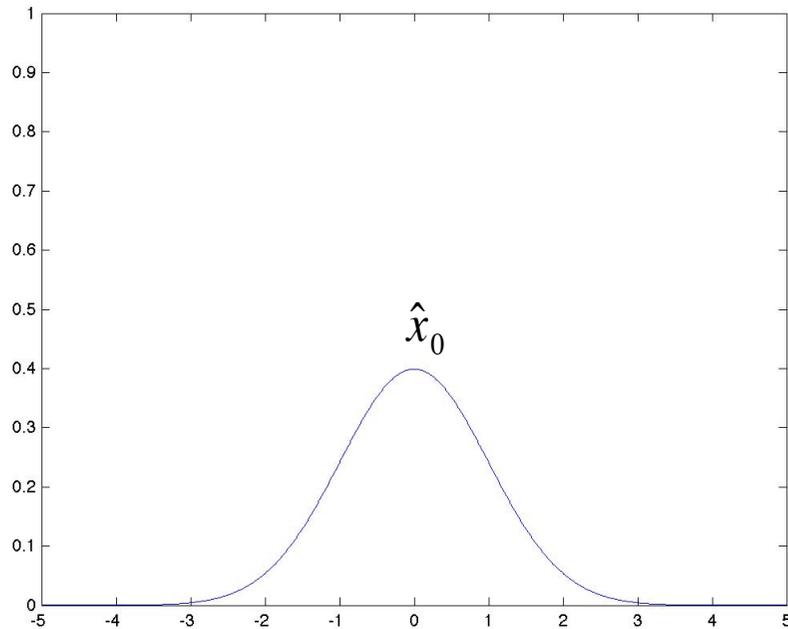
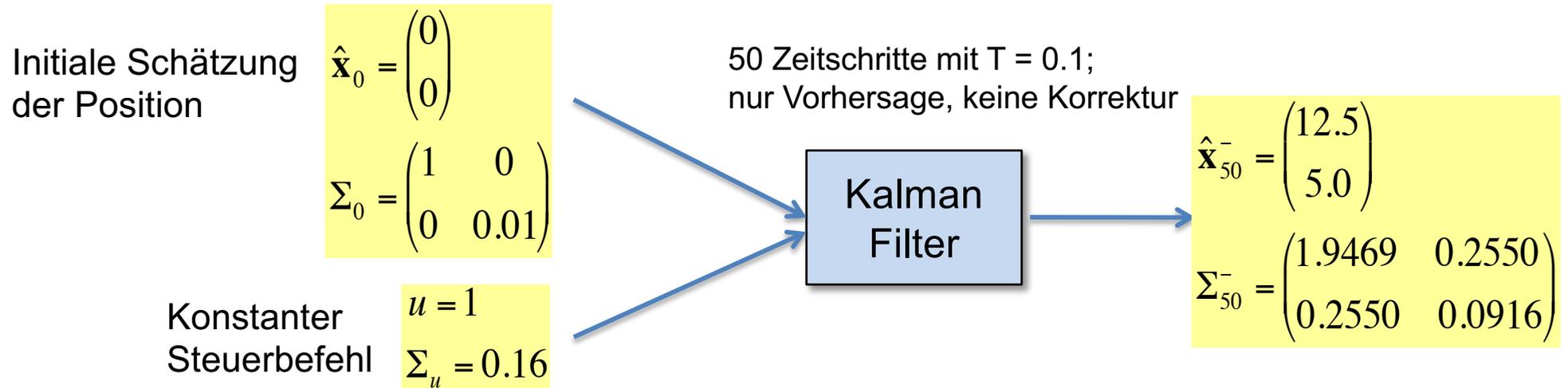
- Roboter misst über Abstandssensor direkt seine Position.



- Messmodell

$$z_k = \underbrace{\begin{pmatrix} 1 & 0 \end{pmatrix}}_{\mathbf{C}} \underbrace{\begin{pmatrix} x_k \\ v_k \end{pmatrix}}_{\mathbf{x}_k}$$

# Simulation des Kalmanfilters (1)



Darstellung der Positionsschätzung

# Simulation des Kalmanfilters (2)

$$\hat{\mathbf{x}}_{50}^- = \begin{pmatrix} 12.5 \\ 5.0 \end{pmatrix}$$

$$\Sigma_{50}^- = \begin{pmatrix} 1.9469 & 0.2550 \\ 0.2550 & 0.0916 \end{pmatrix}$$

Messwert

$$z_{50} = 11.79$$

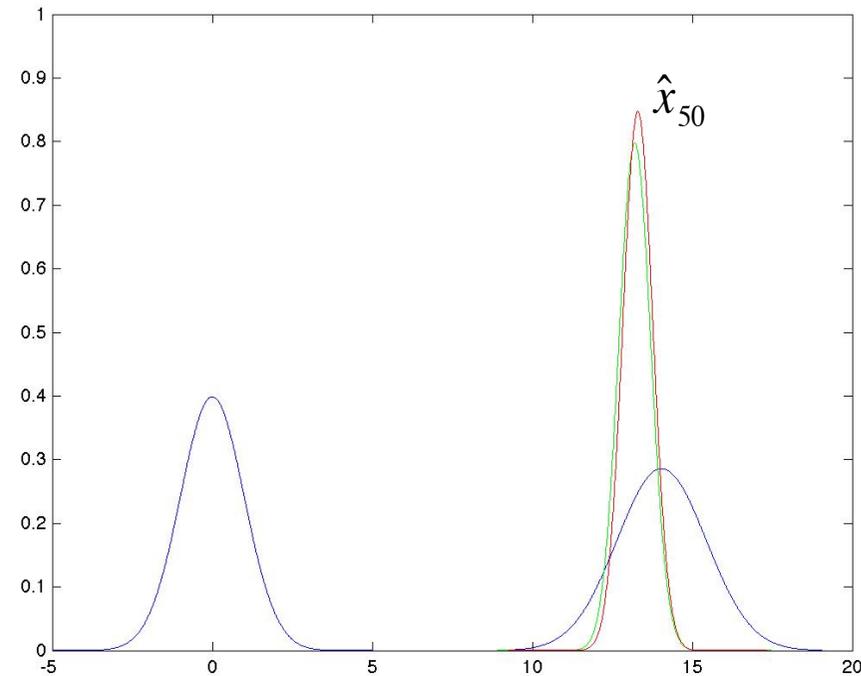
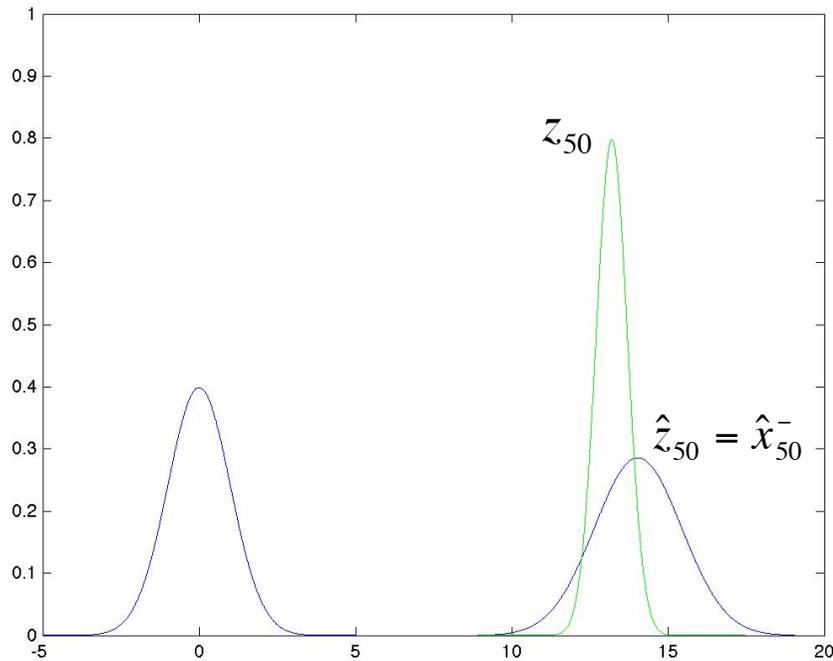
$$\Sigma_z = 0.25$$

nur Korrektur-  
schritt

Kalman  
Filter

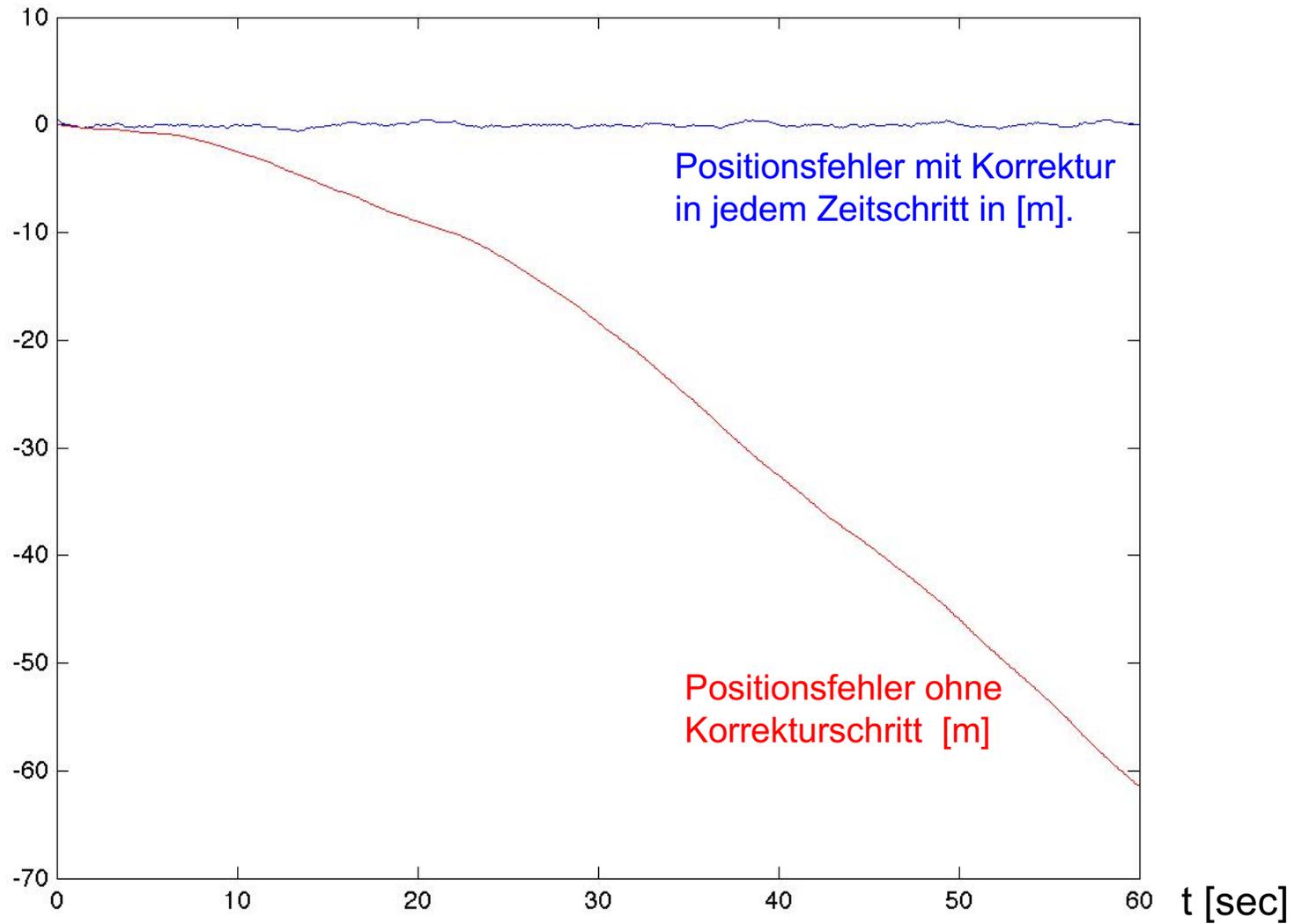
$$\hat{\mathbf{x}}_{50} = \begin{pmatrix} 11.90 \\ 4.98 \end{pmatrix}$$

$$\Sigma_{50} = \begin{pmatrix} 0.2216 & 0.0290 \\ 0.0290 & 0.0620 \end{pmatrix}$$

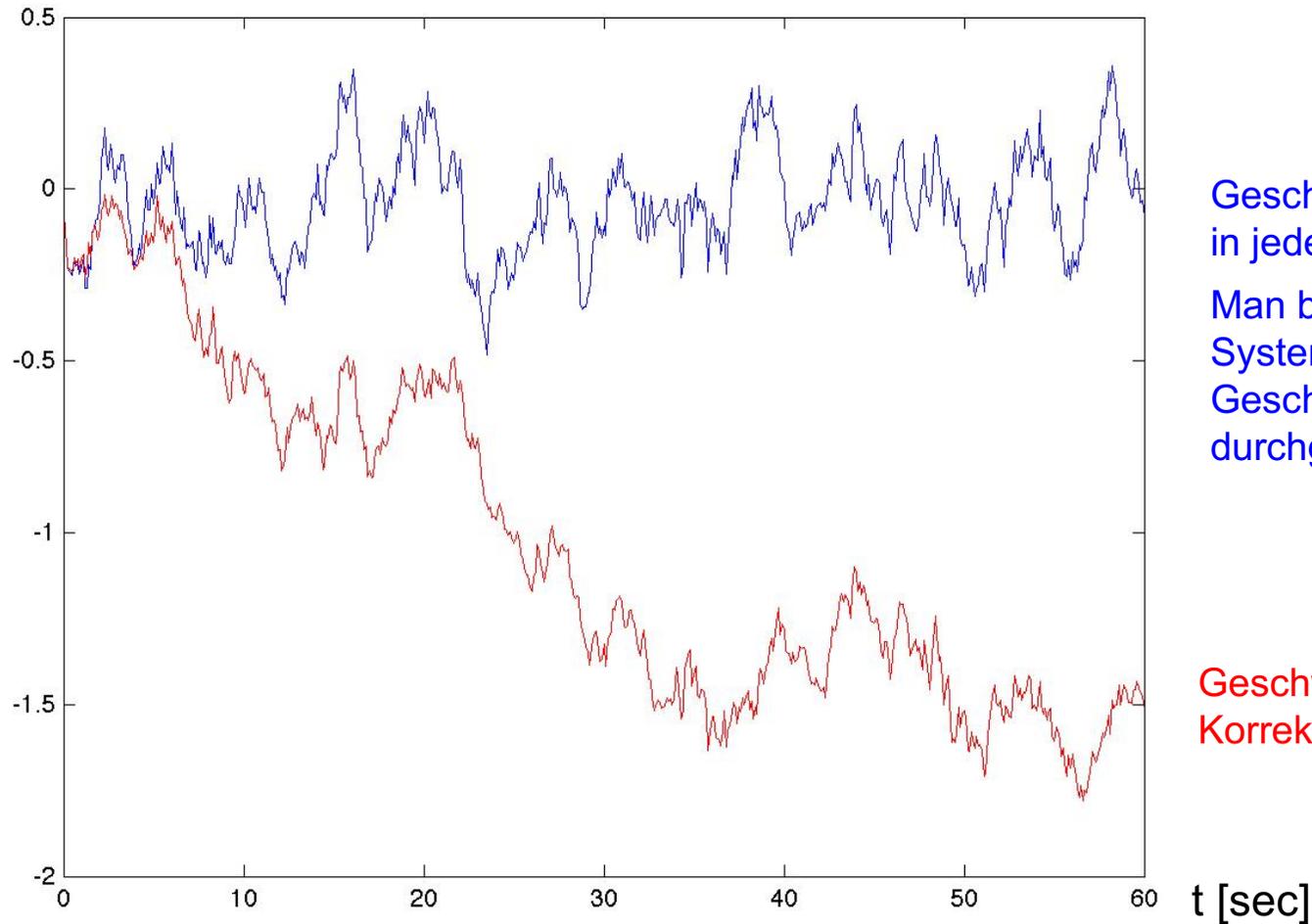


Darstellung der Positionsschätzung

# Positionsgenauigkeit



# Geschwindigkeitsgenauigkeit



Geschwindigkeitsfehler mit Korrektur  
in jedem Zeitschritt in [m/sec].

Man beachte, dass im gegebenen  
System nur Positions- und keine  
Geschwindigkeitsmessungen  
durchgeführt werden.

Geschwindigkeitsfehler ohne  
Korrekturschritt [m/sec]

# Nicht-lineares System- und Messmodell

---

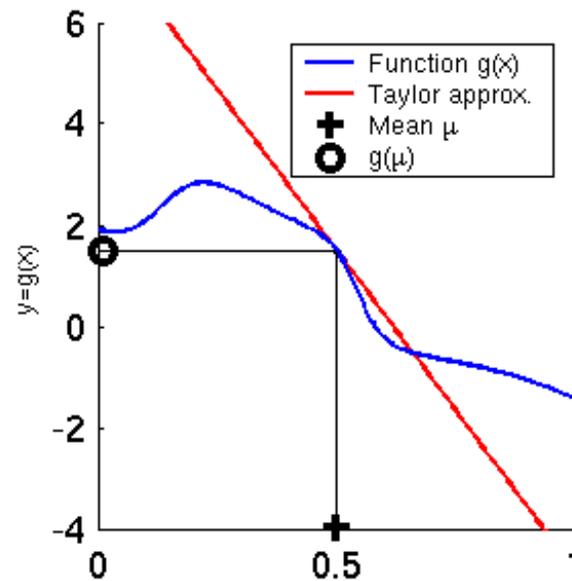
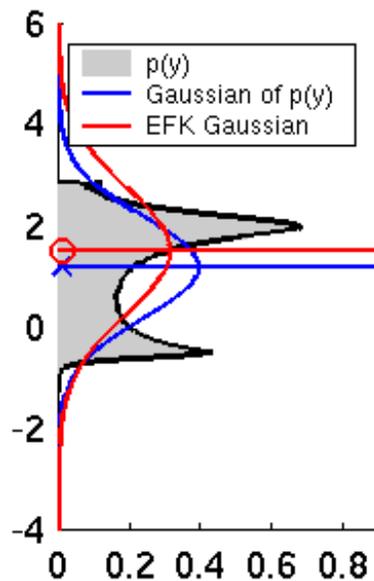
- System- und Messmodell sind nun nicht-linear:

$$\begin{array}{ll} \mathbf{x}_{k+1} = g(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{v}_k & \mathbf{v}_k \sim N(\mathbf{0}, \Sigma_{\mathbf{v}}) \\ \mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{w}_k & \mathbf{w}_k \sim N(\mathbf{0}, \Sigma_{\mathbf{z}}) \end{array}$$

- Wir gehen davon aus, dass das Systemrauschen  $\mathbf{v}_k$  im wesentlichen von einem verrauschten Steuerbefehl  $\mathbf{u}_k$  mit Kovarianz  $\Sigma_{\mathbf{u}}$  verursacht wird.

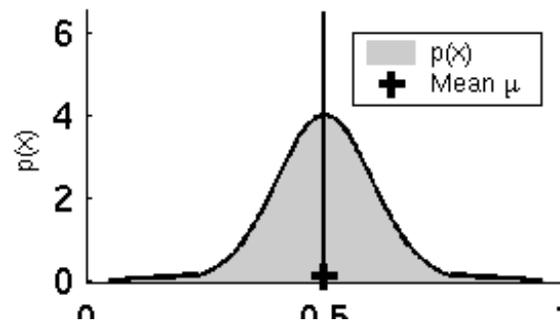
# Erweiterter Kalman-Filter (EKF)

- Linearisiere  $g$  und  $h$  mit Hilfe ihrer Jacobi-Matrizen
- Benutze Jacobi-Matrizen zur Berechnung der Kovarianzen (wie in Kap. 4)



Linearisierung von  $g$ ,  
wobei  $G$  die Jacobi-Matrix  
von  $g$  ist:

$$g(\mathbf{x}) \approx G(\hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}}) + g(\hat{\mathbf{x}})$$



# Algorithmus für erweiterten Kalman-Filter

**KalmanFilter**( $\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{z}_{k+1}$ ):

1. Vorhersage:

$$2. \quad \hat{\mathbf{x}}_{k+1}^- = g(\hat{\mathbf{x}}_k, \mathbf{u}_k)$$

$$3. \quad \Sigma_{k+1}^- = \mathbf{G}_x \Sigma_k \mathbf{G}_x^T + \mathbf{G}_u \Sigma_u \mathbf{G}_u^T$$

4. Korrektur:

$$5. \quad \hat{\mathbf{z}}_{k+1} = h(\hat{\mathbf{x}}_{k+1}^-)$$

$$6. \quad \mathbf{K} = \Sigma_{k+1}^- \mathbf{H}^T (\mathbf{H} \Sigma_{k+1}^- \mathbf{H}^T + \Sigma_z)^{-1}$$

$$7. \quad \hat{\mathbf{x}}_{k+1} = \hat{\mathbf{x}}_{k+1}^- + \mathbf{K}(\mathbf{z}_{k+1} - \hat{\mathbf{z}}_{k+1})$$

$$8. \quad \Sigma_{k+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\Sigma_{k+1}^-$$

9. **return**  $\hat{\mathbf{x}}_{k+1}$

$\mathbf{G}_x$ ,  $\mathbf{G}_u$  und  $\mathbf{H}$  sind Jacobi-Matrizen:

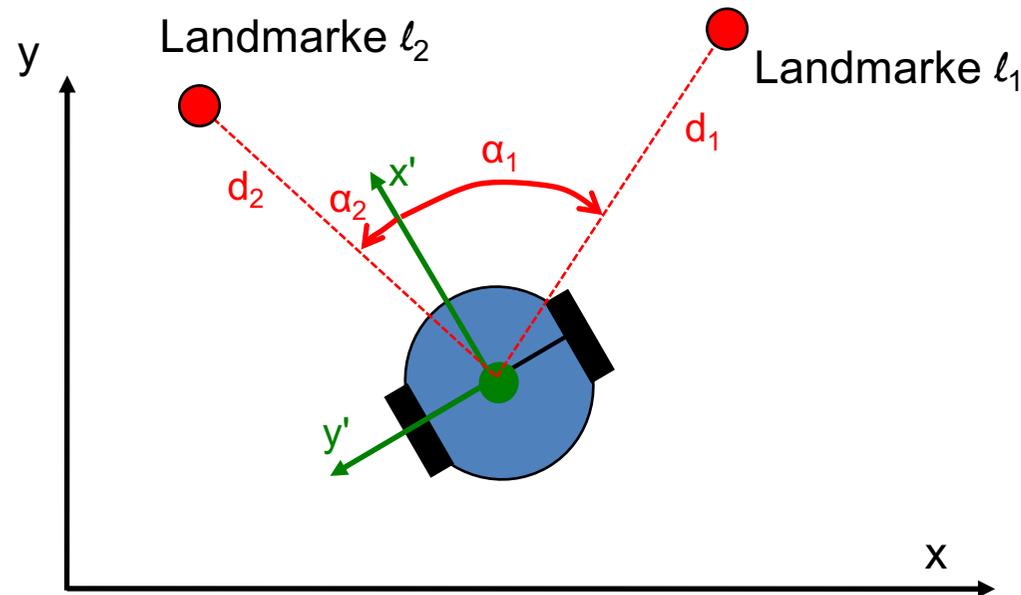
$$\mathbf{G}_x = \frac{\partial g}{\partial \mathbf{x}_k}(\hat{\mathbf{x}}_k, \mathbf{u}_k)$$

$$\mathbf{G}_u = \frac{\partial g}{\partial \mathbf{u}_k}(\hat{\mathbf{x}}_k, \mathbf{u}_k)$$

$$\mathbf{H} = \frac{\partial h}{\partial \mathbf{x}_k}(\hat{\mathbf{x}}_{k+1}^-)$$

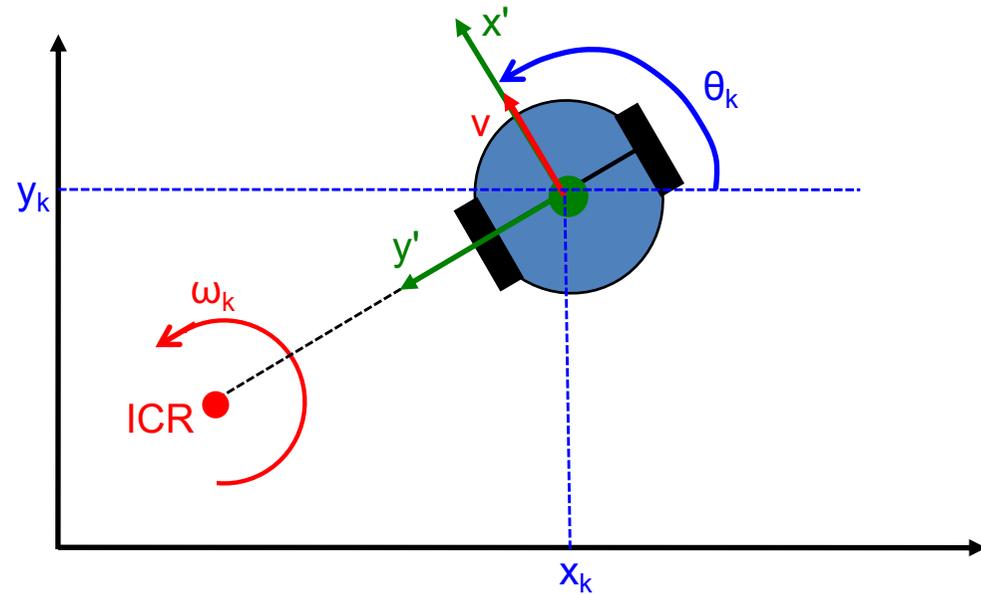
# Selbstlokalisierung mit Landmarken

- Roboter benutzt zur Selbstlokalisierung Abstands- und Orientierungsmessungen zu Landmarken.
- Positionen der beiden Landmarken sind bekannt:  $(x_{11}, y_{11})$  und  $(x_{12}, y_{12})$



# Systemmodell

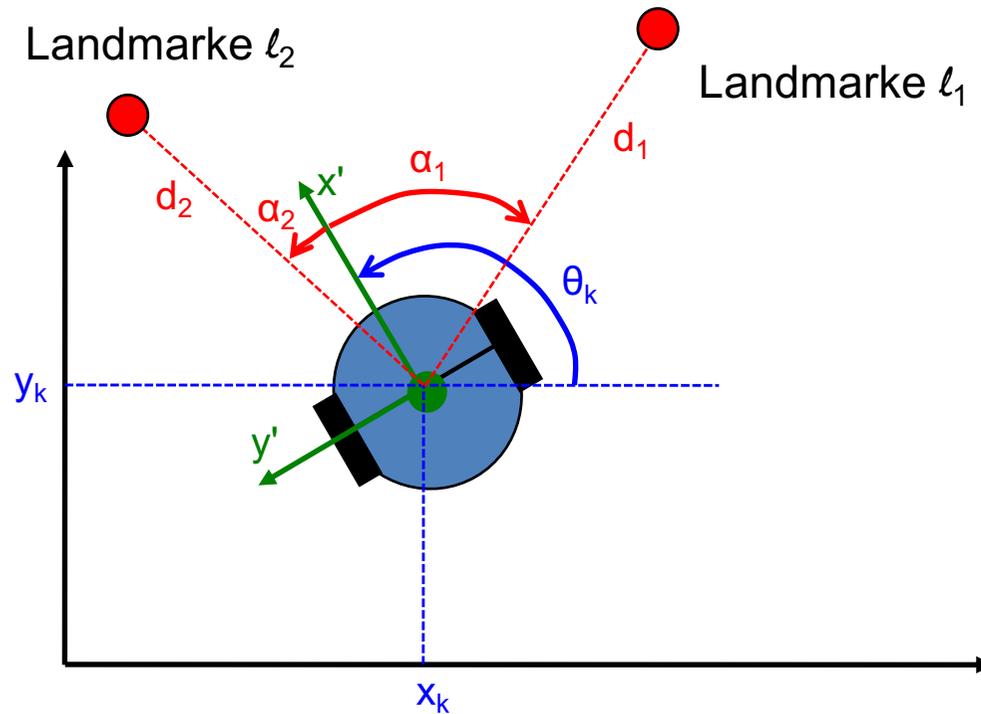
- Der Roboter wird zum Zeitpunkt  $t_k$  mit dem Steuerbefehl  $\mathbf{u}_k$  bewegt.
- Steuerbefehl besteht aus Geschwindigkeit  $v_k$  und Winkelgeschwindigkeit  $\omega_k$ .
- Der Systemzustand  $(x_k, y_k, \theta_k)$  beschreibt die Position des Roboters zum Zeitpunkt  $t_k$ .



$$\underbrace{\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{pmatrix}}_{\mathbf{x}_{k+1}} = \underbrace{\begin{pmatrix} x_k + T v_k \cos \theta_k \\ y_k + T v_k \sin \theta_k \\ \theta_k + T \omega_k \end{pmatrix}}_{g(\mathbf{x}_k, \mathbf{u}_k)}$$

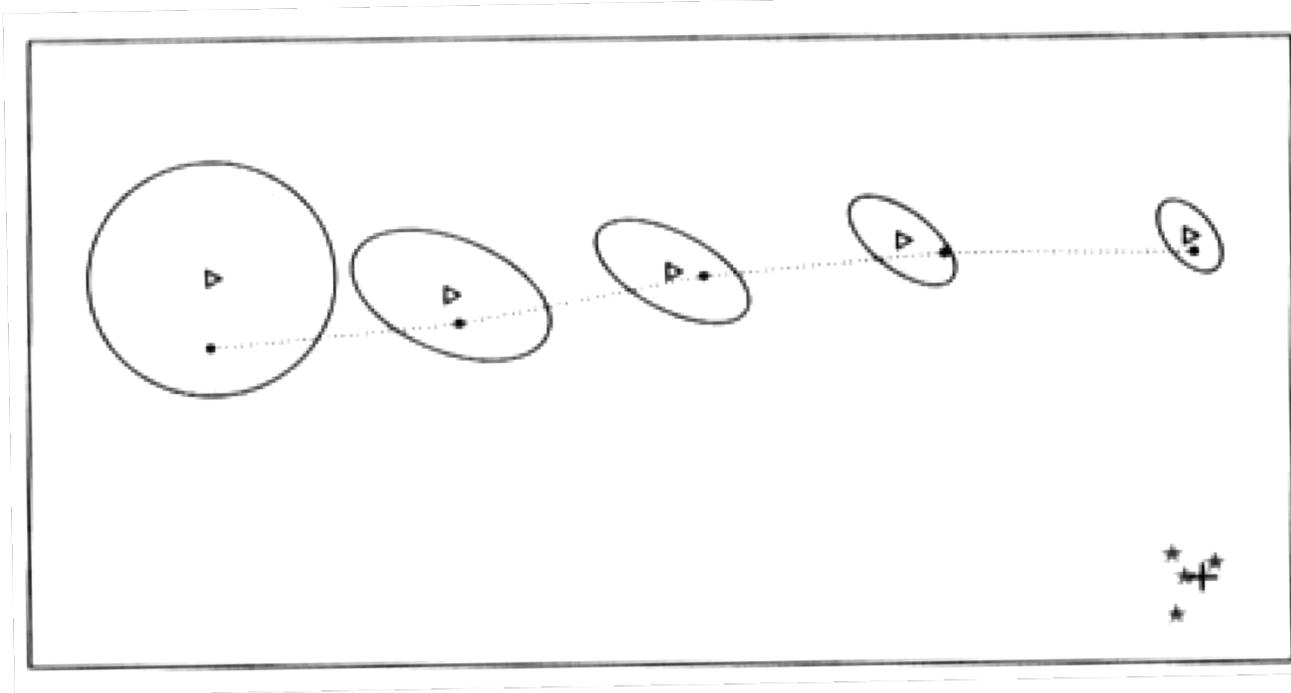
$$\text{mit } \mathbf{u}_k = \begin{pmatrix} v_k \\ \omega_k \end{pmatrix} \quad \text{und } \Sigma_u = \begin{pmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{pmatrix}$$

# Messmodell



$$\underbrace{\begin{pmatrix} d_1 \\ \alpha_1 \\ d_2 \\ \alpha_2 \end{pmatrix}}_{\mathbf{z}_k} = \underbrace{\begin{pmatrix} \sqrt{(x_k - x_{l1})^2 + (y_k - y_{l1})^2} \\ \text{atan2}(y_{l1} - y_k, x_{l1} - x_k) - \theta_k \\ \sqrt{(x_k - x_{l2})^2 + (y_k - y_{l2})^2} \\ \text{atan2}(y_{l2} - y_k, x_{l2} - x_k) - \theta_k \end{pmatrix}}_{h(\mathbf{x}_k)} + \mathbf{w}_k \quad \text{mit } \Sigma_{\mathbf{z}} = \begin{pmatrix} \sigma_d^2 & 0 & 0 & 0 \\ 0 & \sigma_\alpha^2 & 0 & 0 \\ 0 & 0 & \sigma_d^2 & 0 \\ 0 & 0 & 0 & \sigma_\alpha^2 \end{pmatrix}$$

# Beispielfahrt mit einer Landmarke



- Die gepunktete Linie ist die tatsächliche Wegstrecke.
- Die  $\sigma$ -Ellipsen und die Dreiecke stellen die geschätzte Lage bzw. Orientierung dar.
- Das Kreuz ist die Landmarke. Die Sterne geben die gemessene Position der Landmarke an. Es ist deutlich das Messrauschen zu sehen.
- Die Anwendung kann auf  $n$  Landmarken erweitert werden. Zusätzlich kann noch das Problem der Zuordnung der Messdaten zu den Landmarken-Nummern dazukommen. (siehe auch [Choset et al])

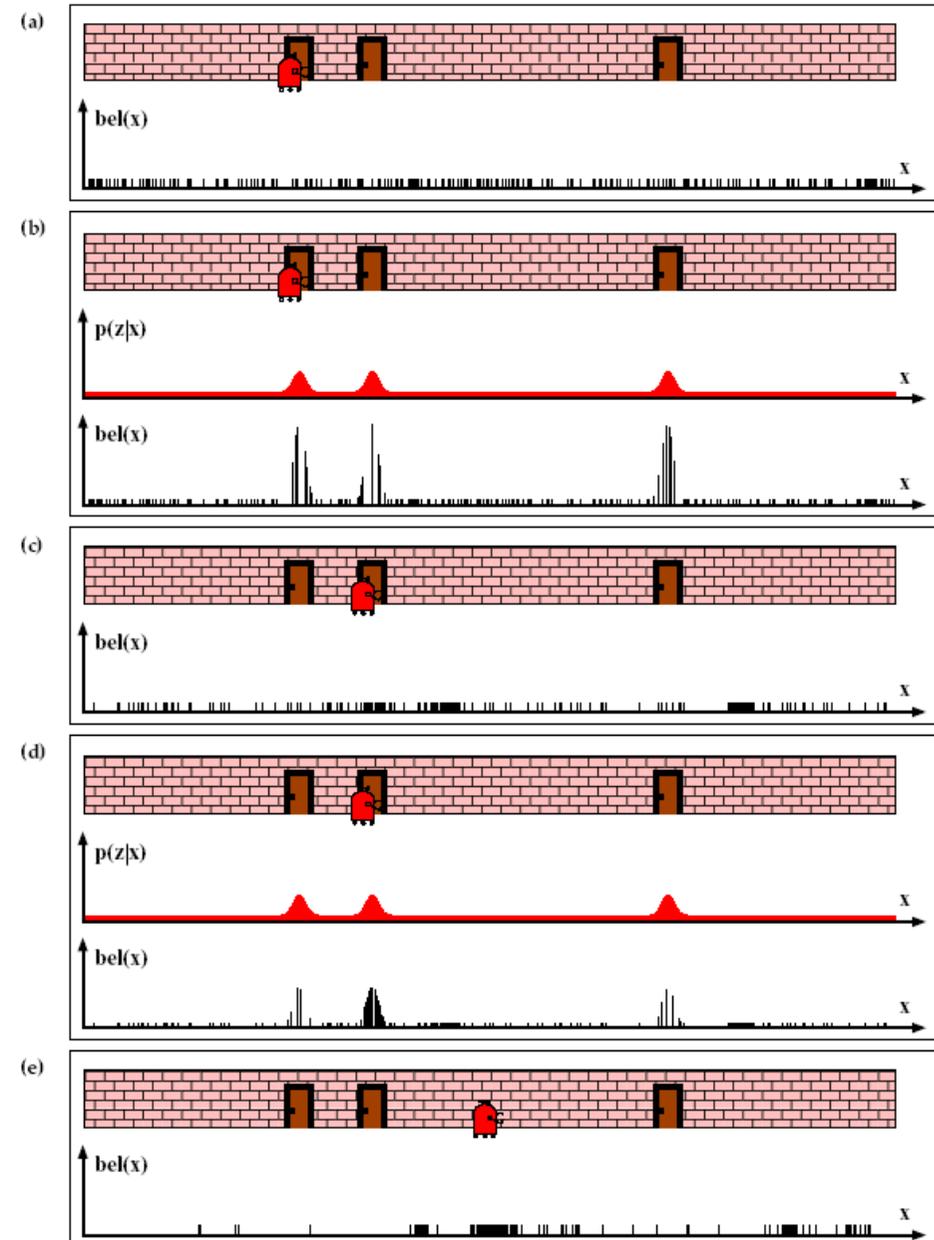
# Lokalisierung

- Einleitung
  - Problemstellung, Varianten,  
Sensorik, Umgebungskarten, Verfahren
- Laterationsverfahren
- Koppelnavigation
  - Idee, Navigationsgleichungen,  
Fehlerfortpflanzung
- Lokalisierung mit einem Kalman-Filter
  - Idee, Algorithmus für linearen und erweiterten Kalmanfilter,  
Beispiele, Selbstlokalisierung mit einem Kalmanfilter
- Monte-Carlo-Lokalisierung
  - Idee, Algorithmus, Beispiele

# Idee

Positionsschätzung wird durch ein Menge von **Partikeln** (vertikale Striche) dargestellt.

- (a) Keine Information über die Anfangsposition; Partikel sind über alle  $x$ -Werte zufällig verteilt.
- (b) **Gewichtung**:  
Durch eine Sensormessung  $z$  werden die **Gewichte** (Strichhöhe) verändert.
- (c) **Resampling**:  
Aus der Partikelmenge werden zufällig aber entsprechend ihrem Gewicht Partikeln gezogen.  
  
Anschließend wird der **Steuerbefehl** (Bewegung) **integriert**.
- (d) erneute **Gewichtung** mit neuem Sensorwert
- (e) erneutes **Resampling** und **Integrierung des Steuerbefehls**



# Monte-Carlo-Lokalisierung (MCL)

**Algorithm MCL**( $\chi_k, u_k, z_{k+1}$ )

$\chi_k' = \chi_k = \emptyset;$

Integration des Steuerbefehls  $u_k$  und  
Gewichtung mit Sensorwert  $z_{k+1}$ :

for  $i = 1$  to  $M$  do

$x_{k+1}[i] = \text{sampleMotionModel}(u_k, x_k[i]);$

$w_i = \text{measurementModel}(z_{k+1}, x_{k+1}[i], m);$

$\chi_{k+1}' = \chi_{k+1}' \cup \{(x_{k+1}[i], w_i)\};$

endfor

Resampling:

$\chi_{k+1} = \emptyset;$

for  $i = 1$  to  $M$  do

ziehe  $i$  zufällig mit Wahrscheinlichkeit  $w_i$ ;

$\chi_{k+1} = \chi_{k+1} \cup \{x_{k+1}[i]\};$

endfor

return  $\chi_{k+1};$

Positionsschätzung dargestellt  
durch Partikelmenge  
 $\chi_k = \{x_k[1], x_k[2], \dots, x_k[M]\}.$

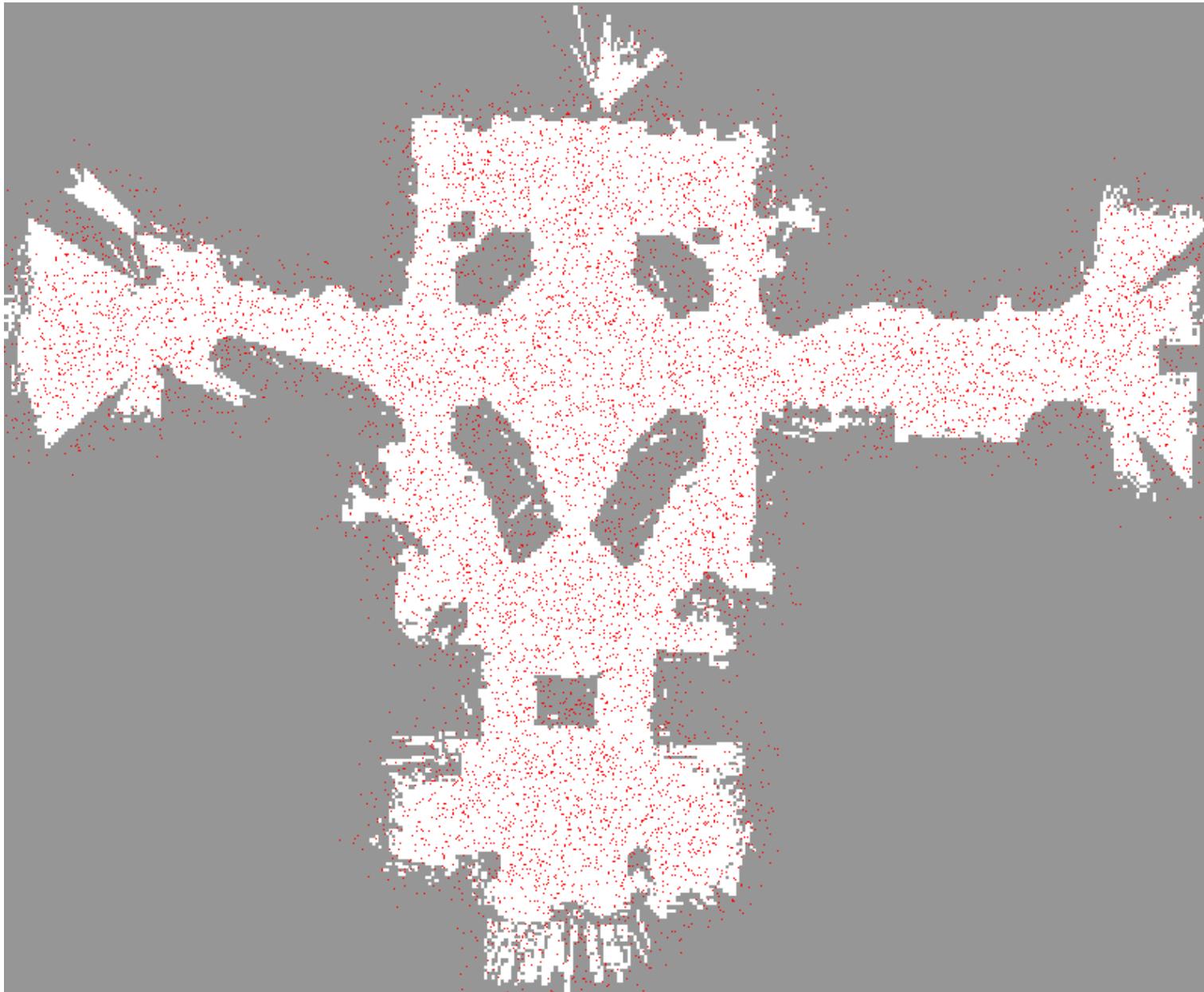
**Bewegungsmodell:**  
Hier wird auf Partikel  $x_k[i]$   
ein zufällig generierter  
Steuerbefehl angewendet.

**Messmodell:**  
Hier wird geprüft, wie gut die  
gemessenen Sensorwerte  $z_{k+1}$  zur  
Position  $x_{k+1}[i]$  in einer  
Umgebungskarte (map)  $m$  passen.

Neue Positionsschätzung

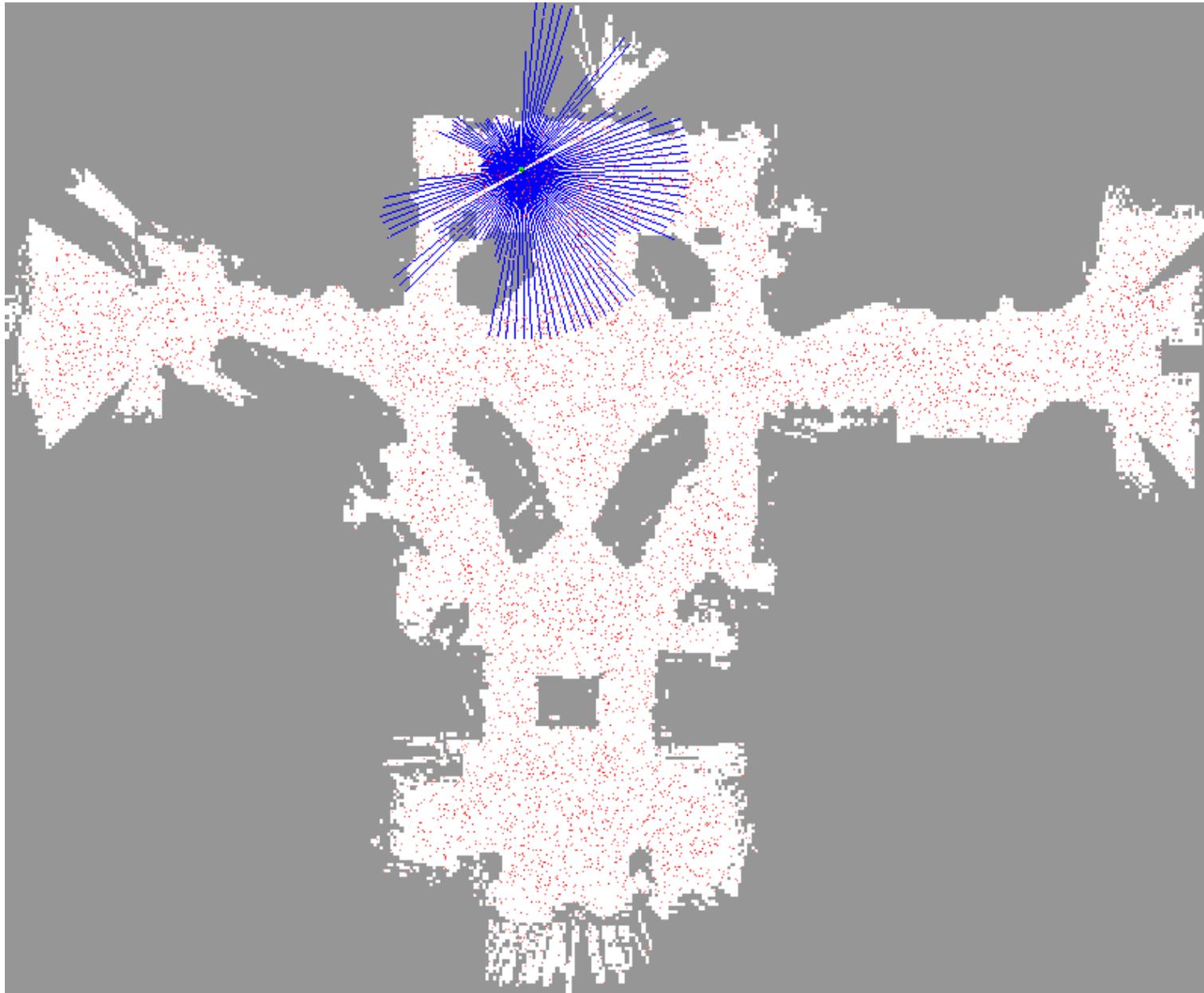
# Beispiel (1)

---



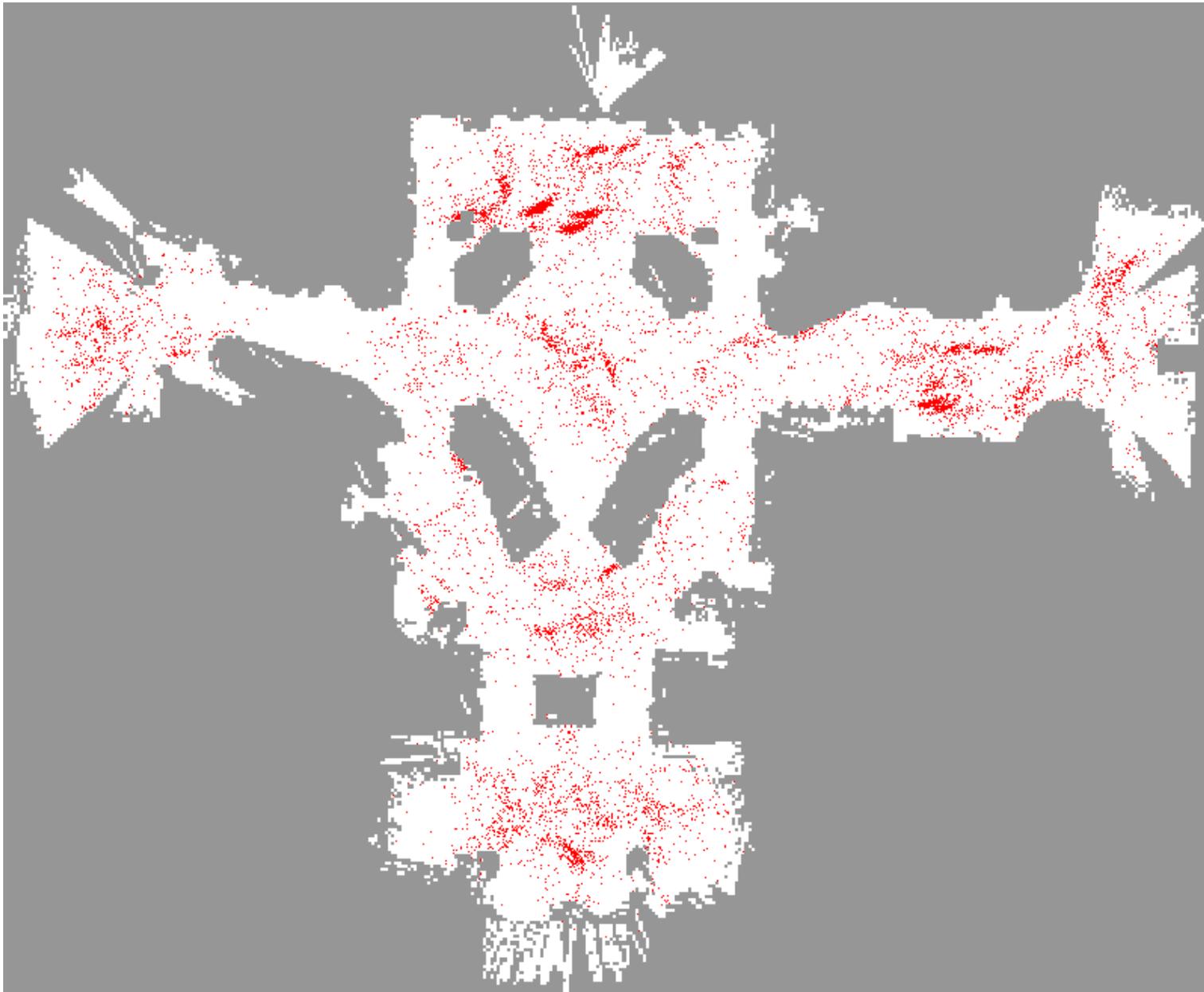
# Beispiel (2)

---



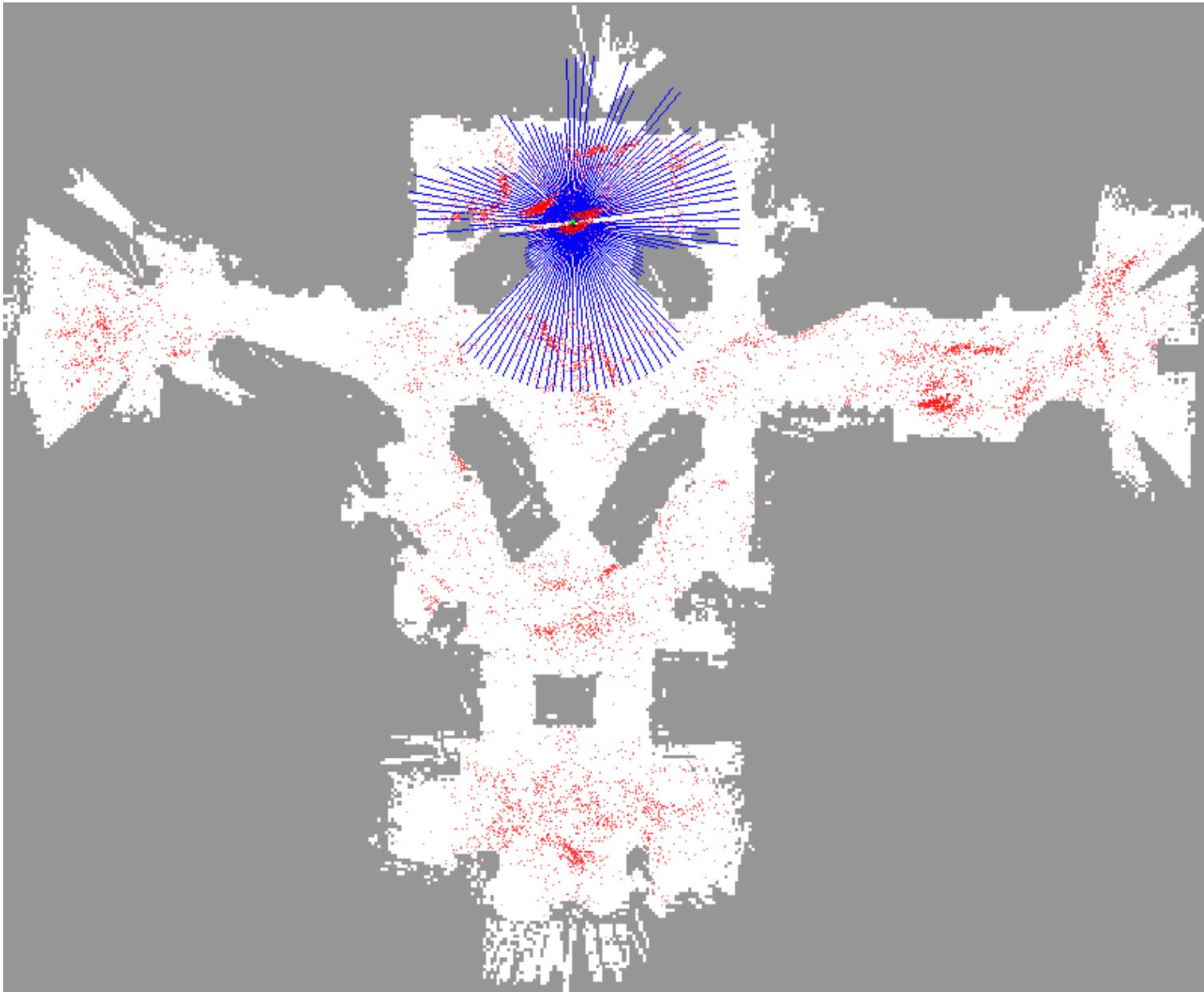
# Beispiel (3)

---



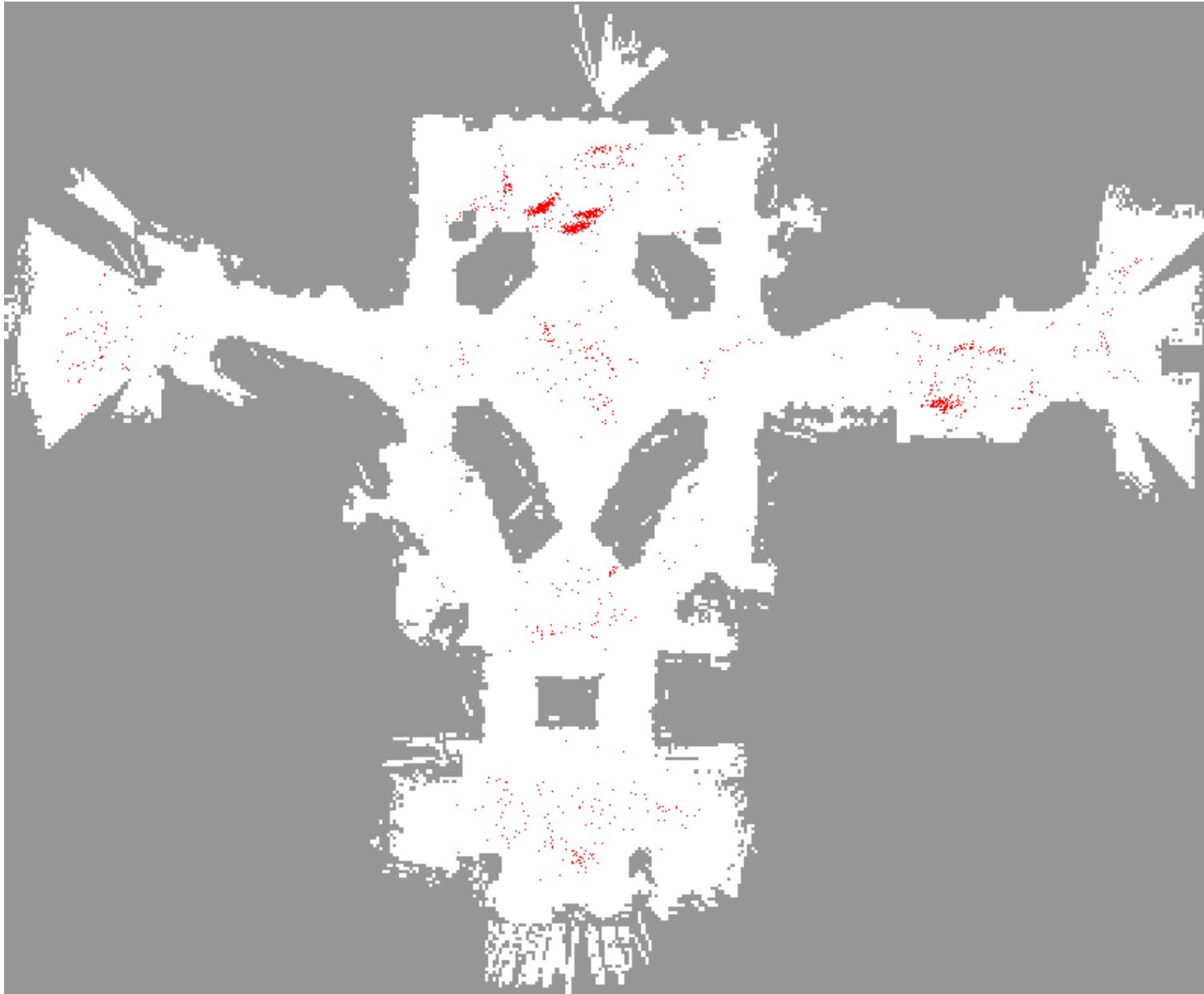
# Beispiel (4)

---



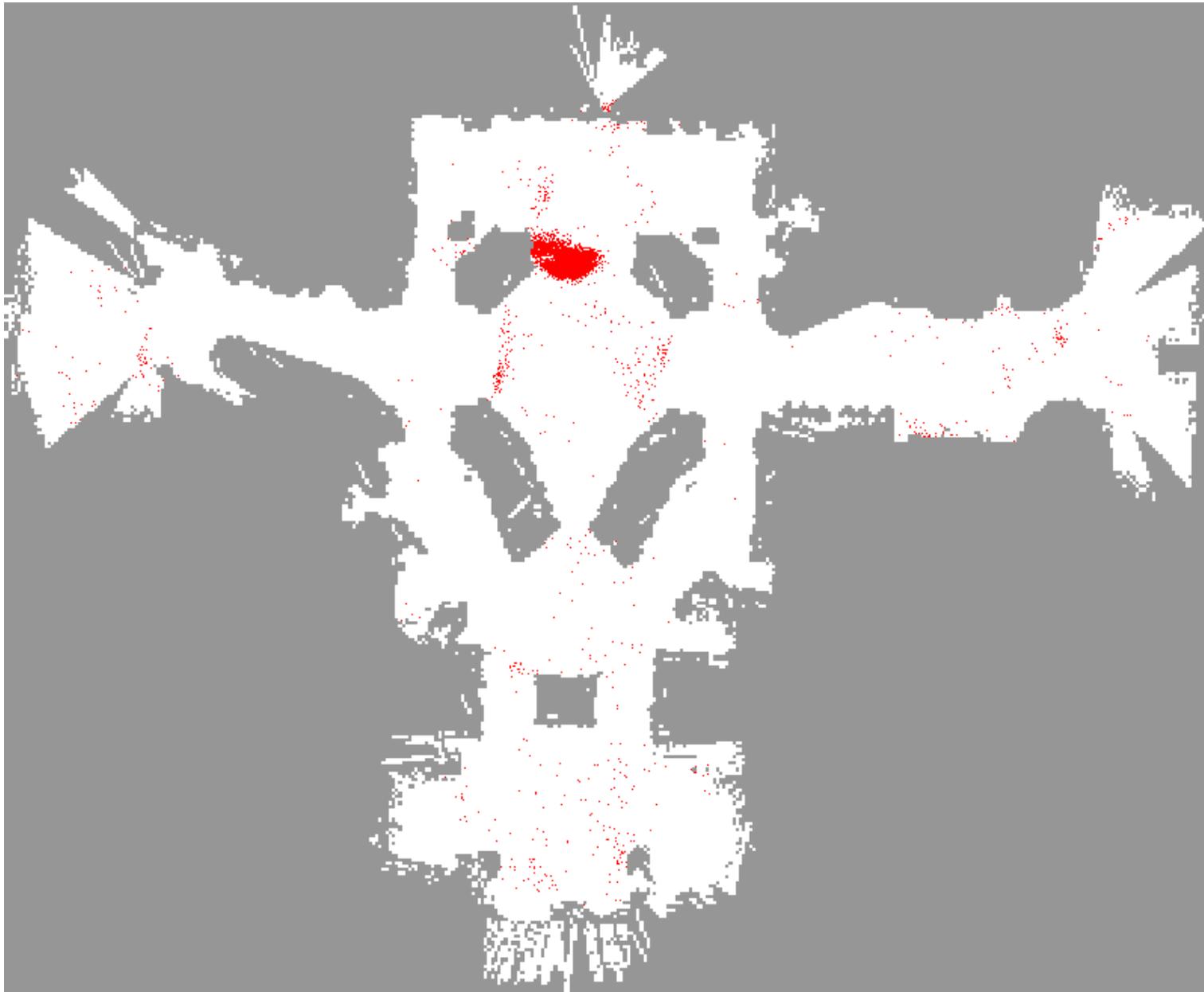
# Beispiel (5)

---



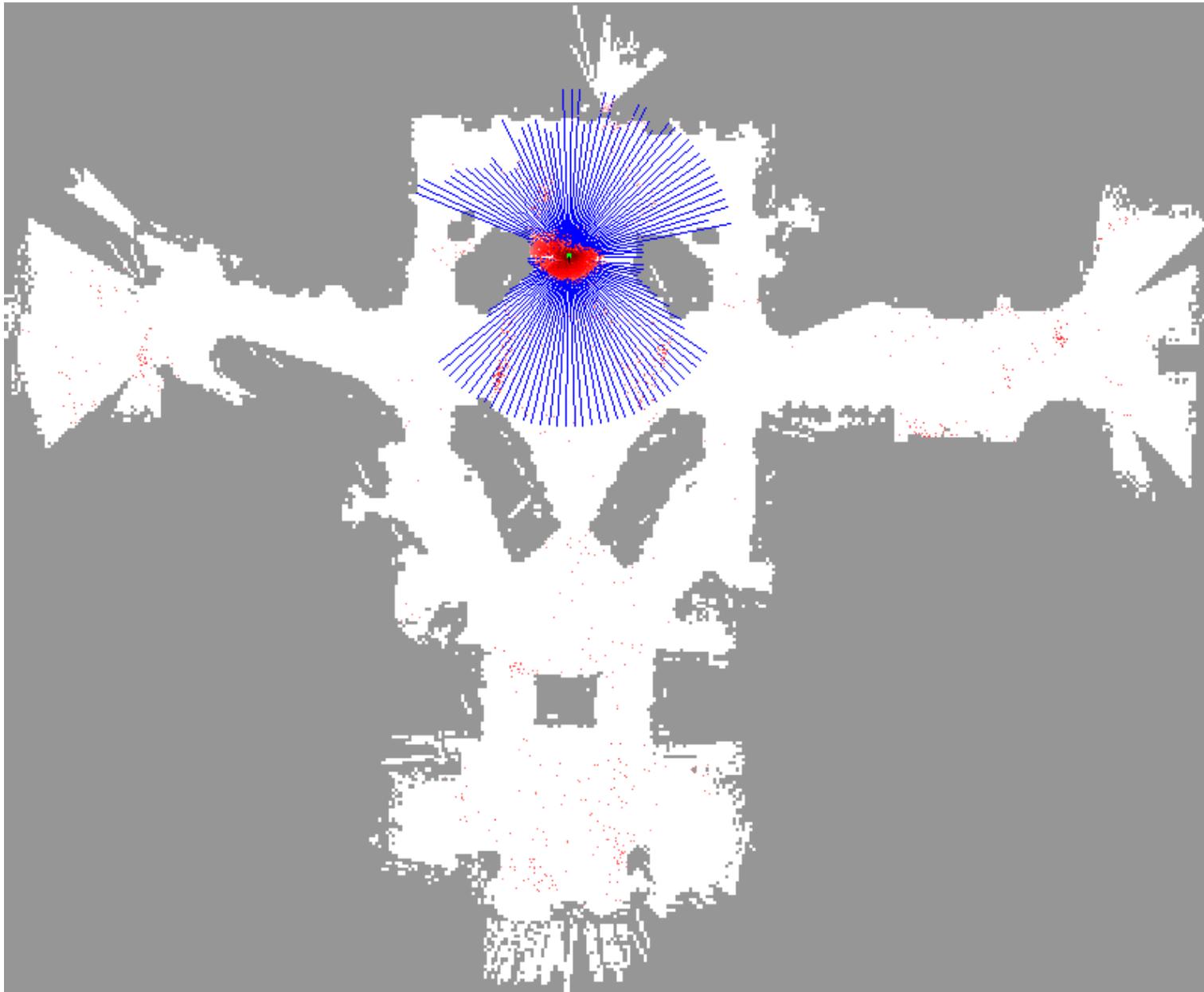
# Beispiel (6)

---



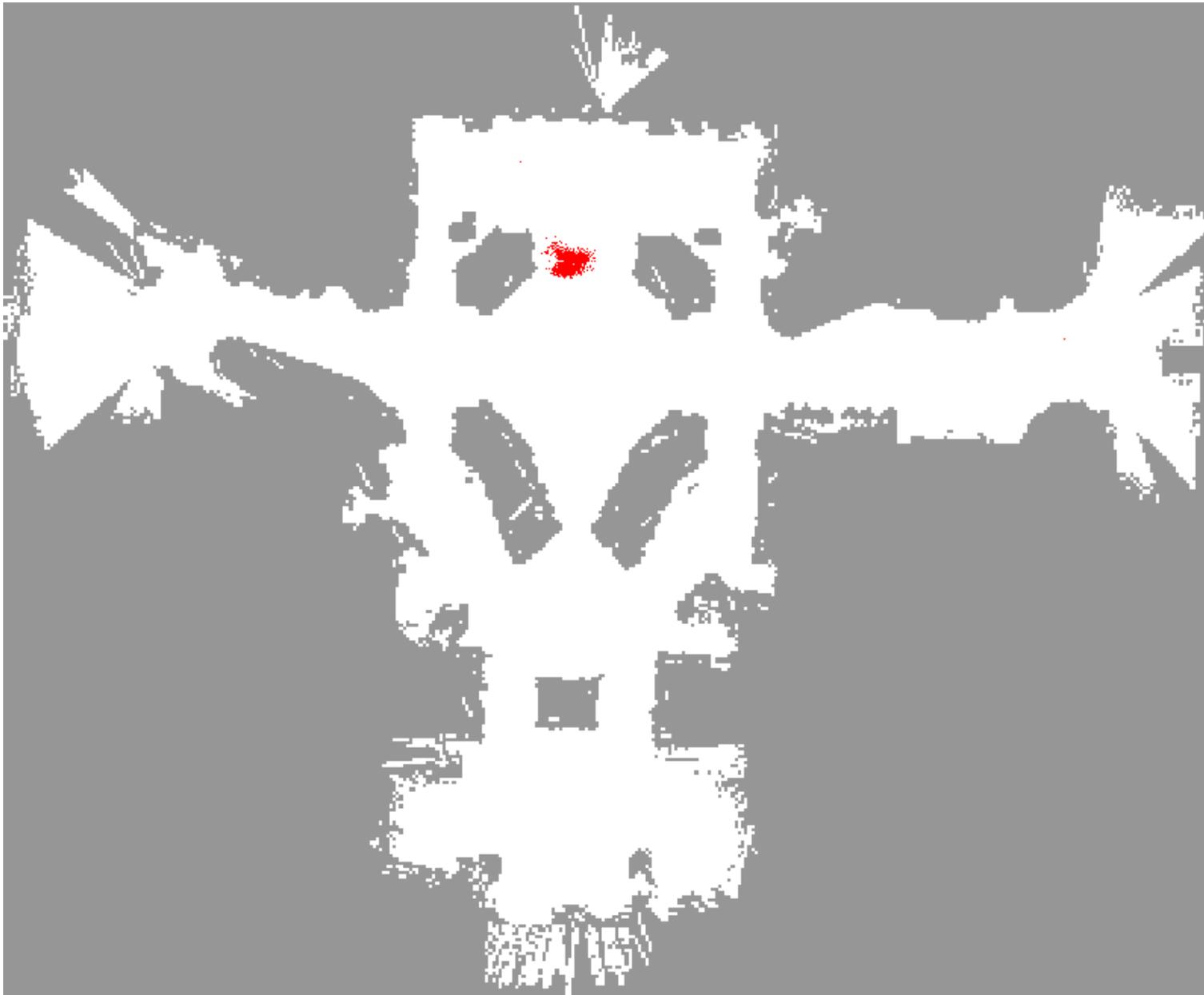
# Beispiel (7)

---



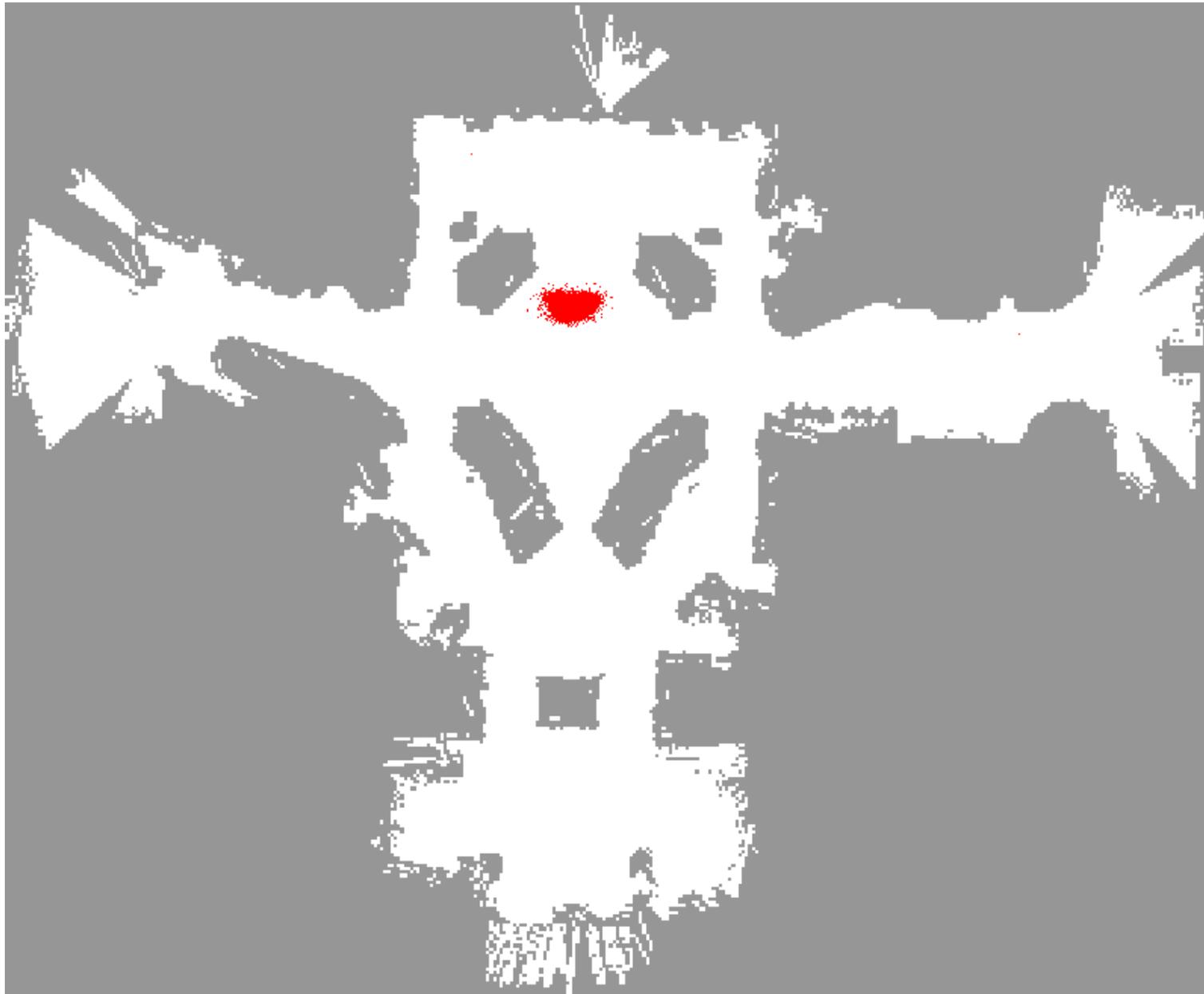
# Beispiel (8)

---



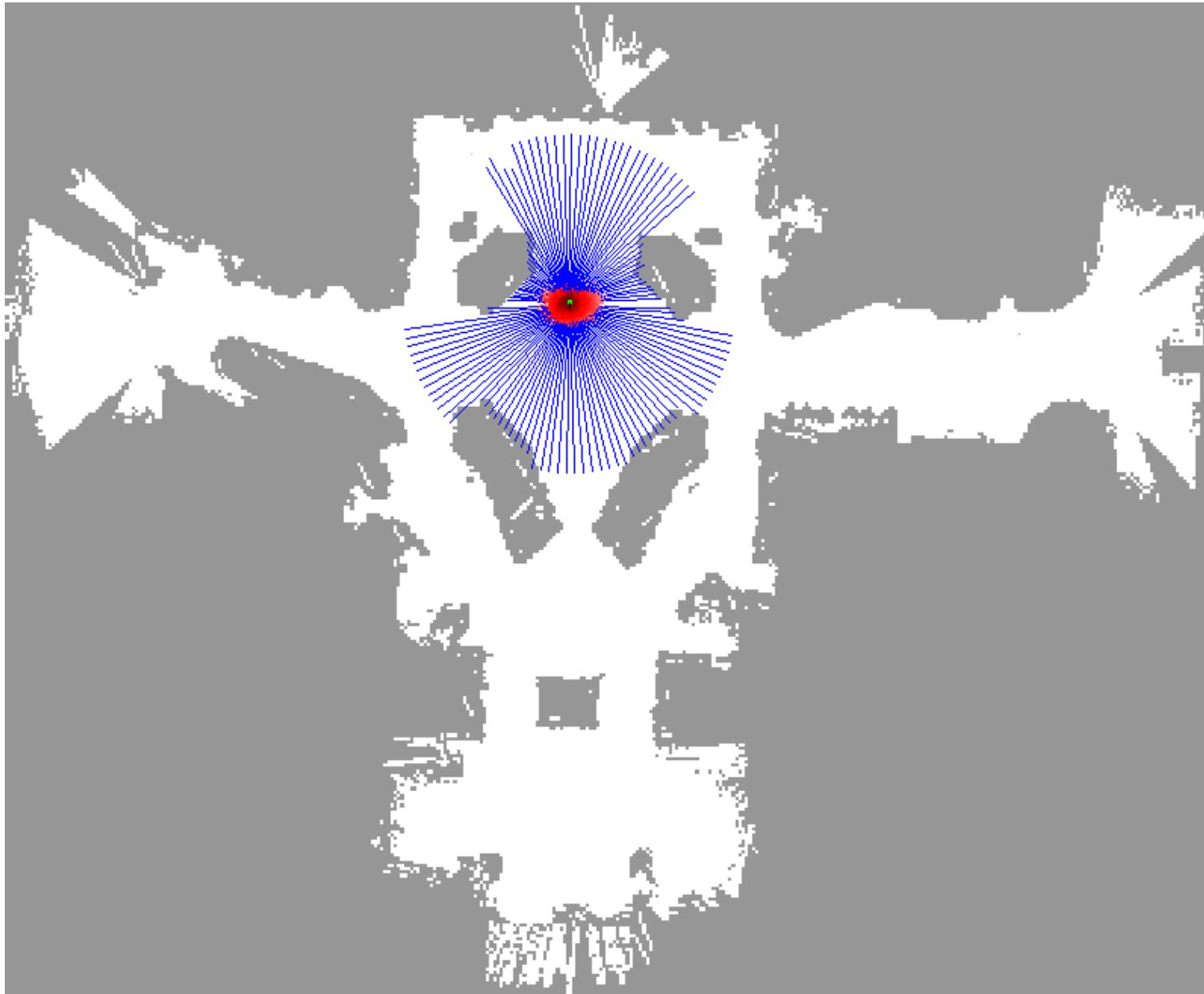
# Beispiel (9)

---



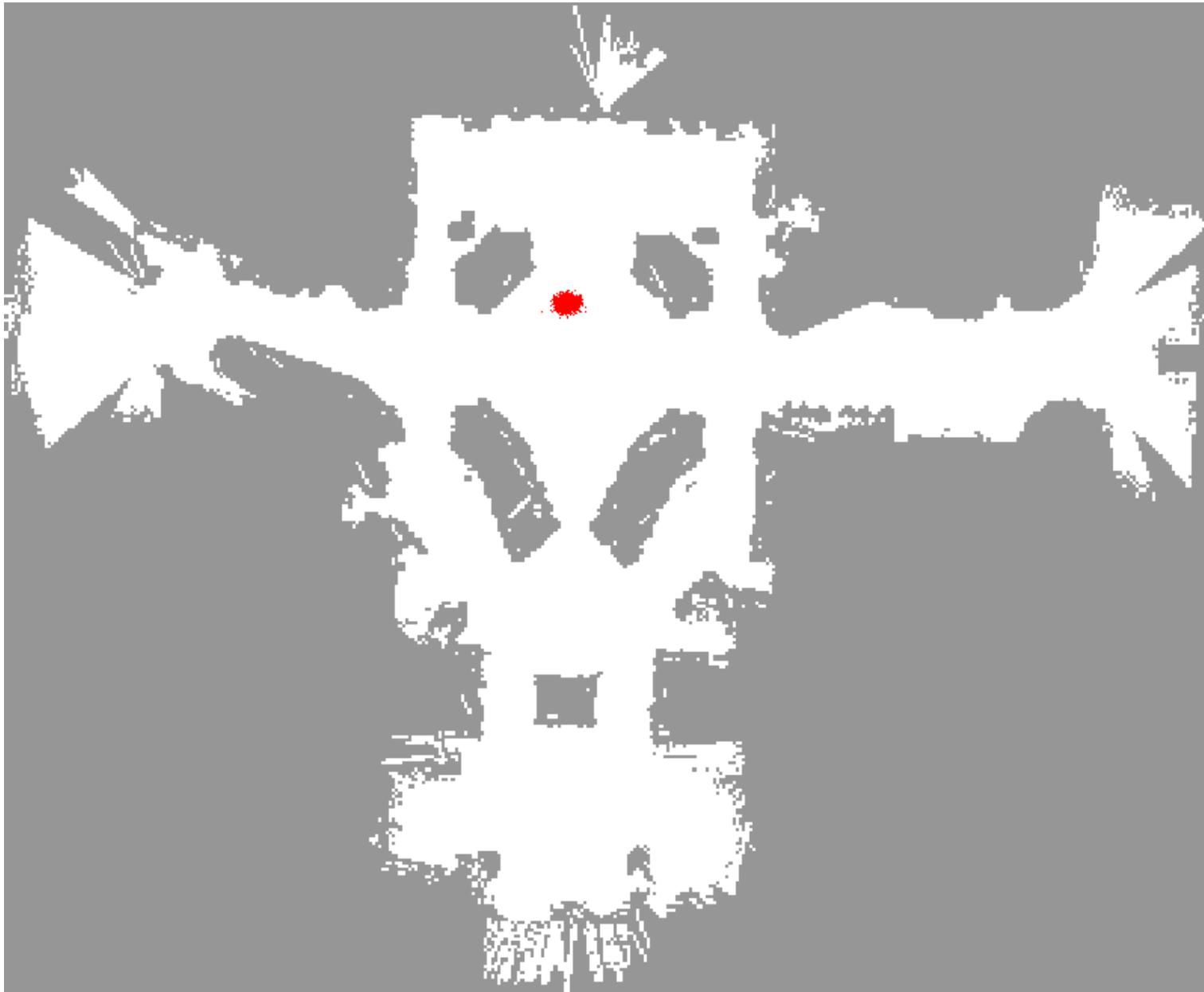
# Beispiel (10)

---



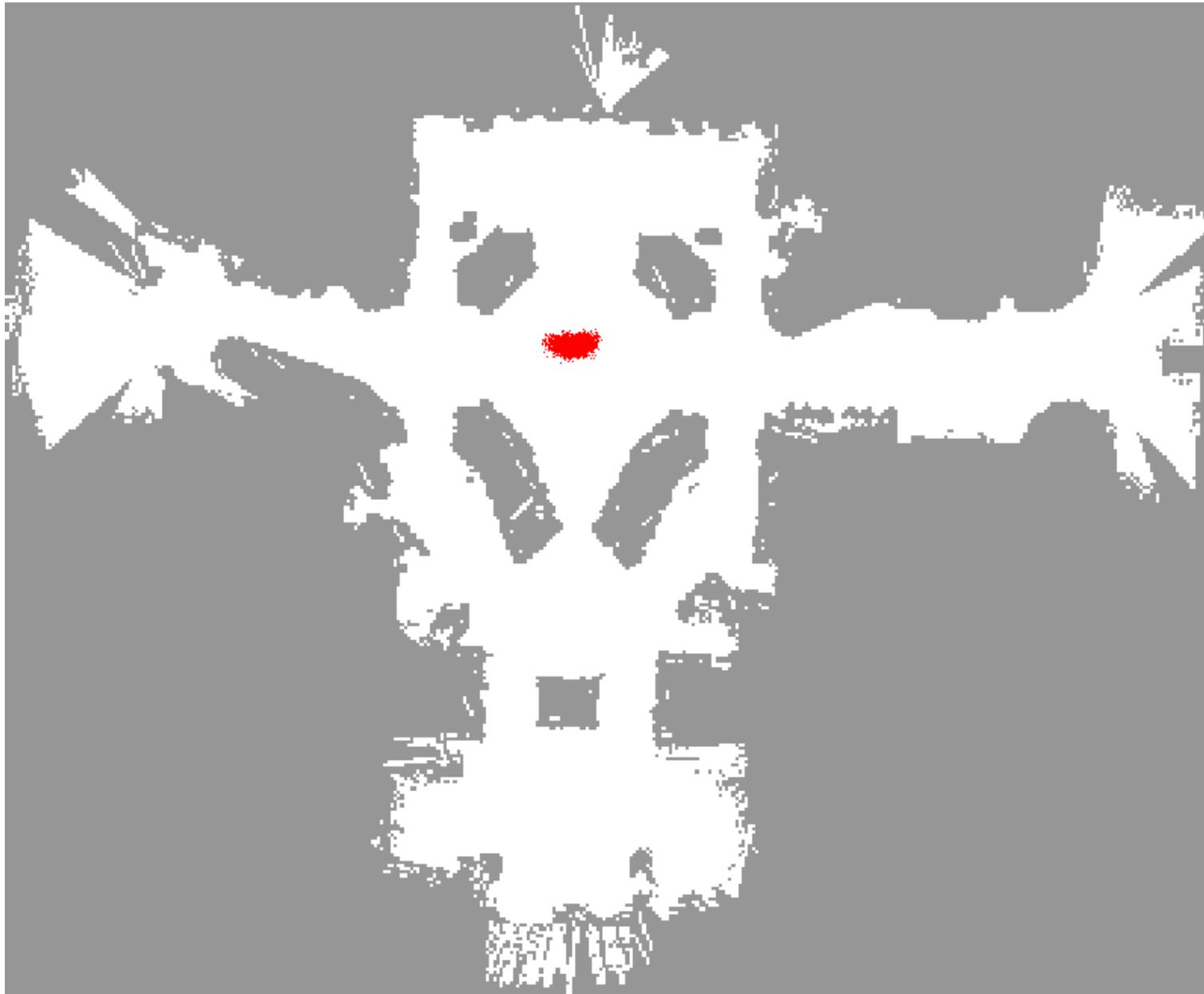
# Beispiel (11)

---



# Beispiel (12)

---

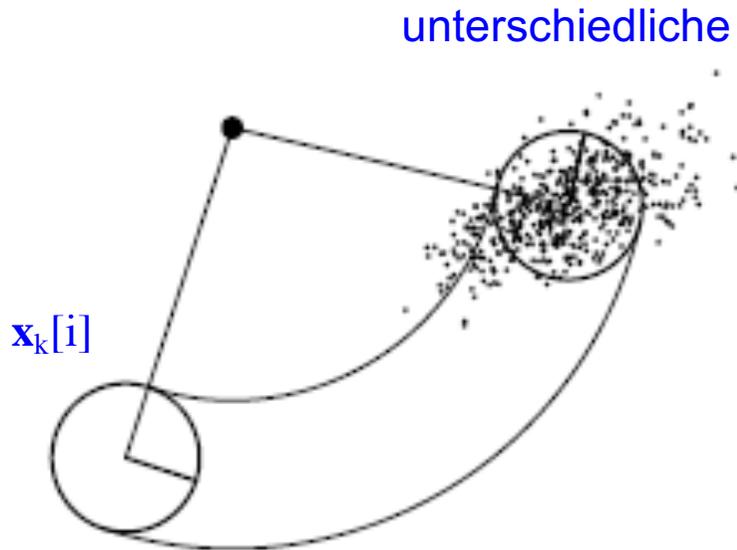


# Bewegungsmodell

- Die Funktion `sampleMotionModell` addiert zunächst zum Steuerbefehl  $\mathbf{u}_k$  einen zufällig generierten Rauschterm mit der Varianz  $\Sigma_u$
- Auf den verrauschten Steuerbefehl  $\tilde{\mathbf{u}}_k$  und einem Partikel  $\mathbf{x}_k[i]$  wird dann die Funktion  $f$  des Bewegungsmodells angewendet:

$$\mathbf{x}_{k+1}[i] = f(\mathbf{x}_k[i], \tilde{\mathbf{u}}_k)$$

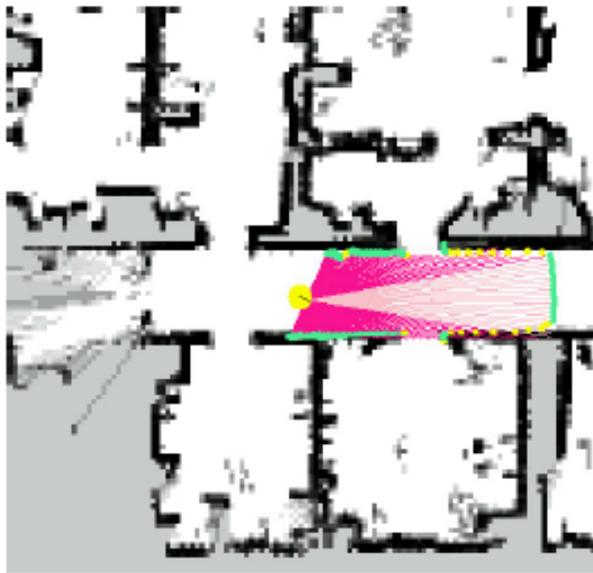
- Das Bewegungsmodell  $f$  ist üblicherweise eines der Koppelnavigationsmodelle.



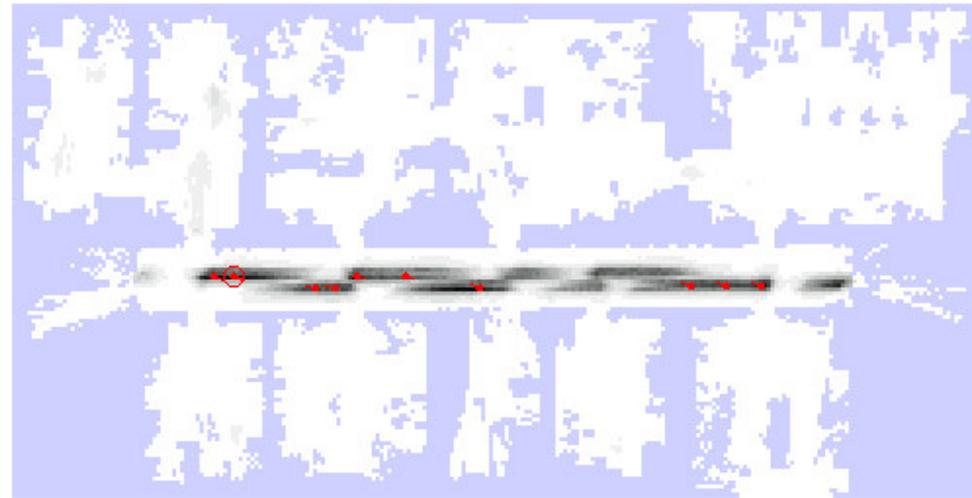
Das Bewegungsmodell  $f$  wurde auf ein Partikel  $\mathbf{x}_k[i]$  mit unterschiedlichen zufällig generierten Steuerbefehlen  $\tilde{\mathbf{u}}_k$  angewendet.

# Messmodell (1)

- Es muss geprüft werden, wie gut die Sensorwerte  $z_{k+1}$  zur Partikel-Position  $x_{k+1}[i]$  in einer Umgebungskarte  $m$  passen.



Sensorwerte  $z_{k+1}$   
(z.B Laser-Scan)



Wahrscheinlichkeit für jede Position  $x_{k+1}[i]$ , dass dort Laser-Scan  $z_{k+1}$  gemessen wurde. Je dunkler, desto wahrscheinlicher.

# Messmodell (2)

---

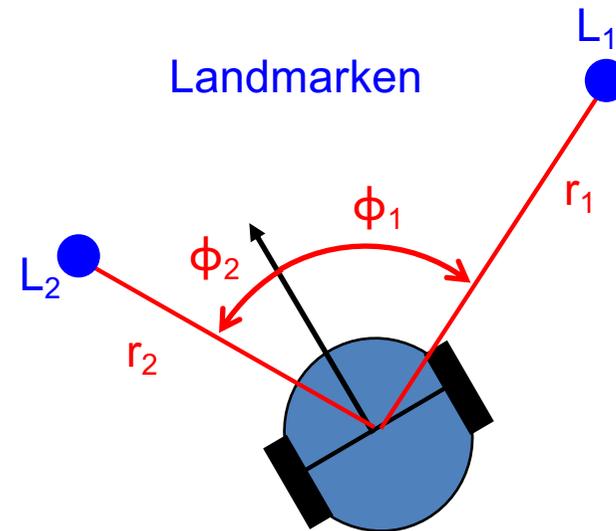
- Es gibt in der Literatur zahlreiche Verfahren, wovon einige hier kurz behandelt werden.
- Wir wollen davon ausgehen, dass außer beim Landmarkenbasiertem Verfahren die Sensordaten aus einem Laser-Scan bestehen.

Karte	Verfahren
Menge von Landmarken	Landmarkenbasiertes Verfahren
Metrische Karte	Strahlenmodell
Metrische Karte	Likelihood-Field

# Landmarkenbasiertes Verfahren (1)

---

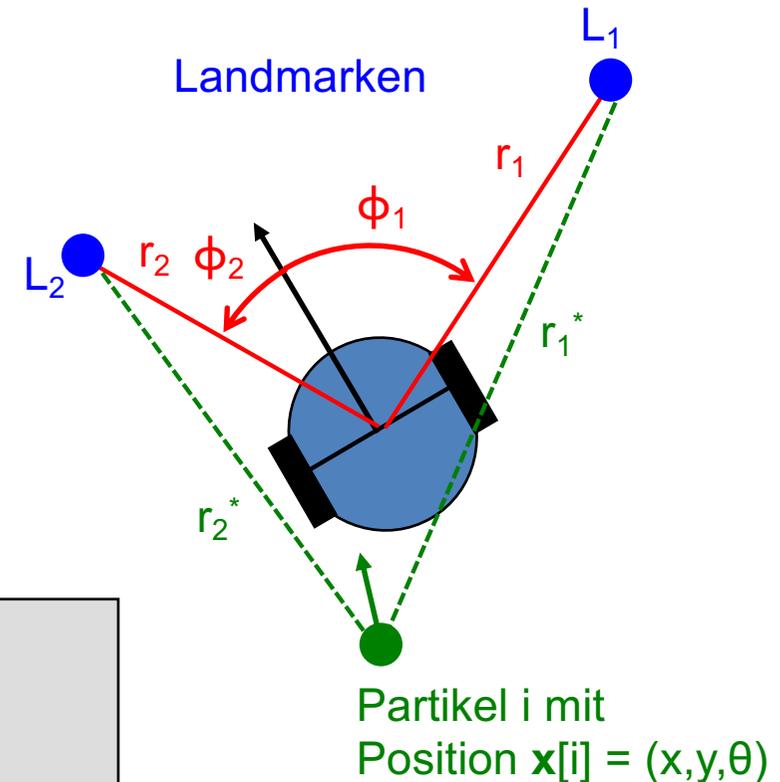
- Sensor misst zur  $j$ -ten Landmarke Entfernung  $r_j$  und relative Orientierung  $\phi_j$ .
- Dabei wird ein normalverteilter Fehler angenommen:
  - $\text{err}_r \sim N(0, \sigma_r^2)$
  - $\text{err}_\phi \sim N(0, \sigma_\phi^2)$
- Es kann auch nur Entfernung oder Orientierung gemessen werden.



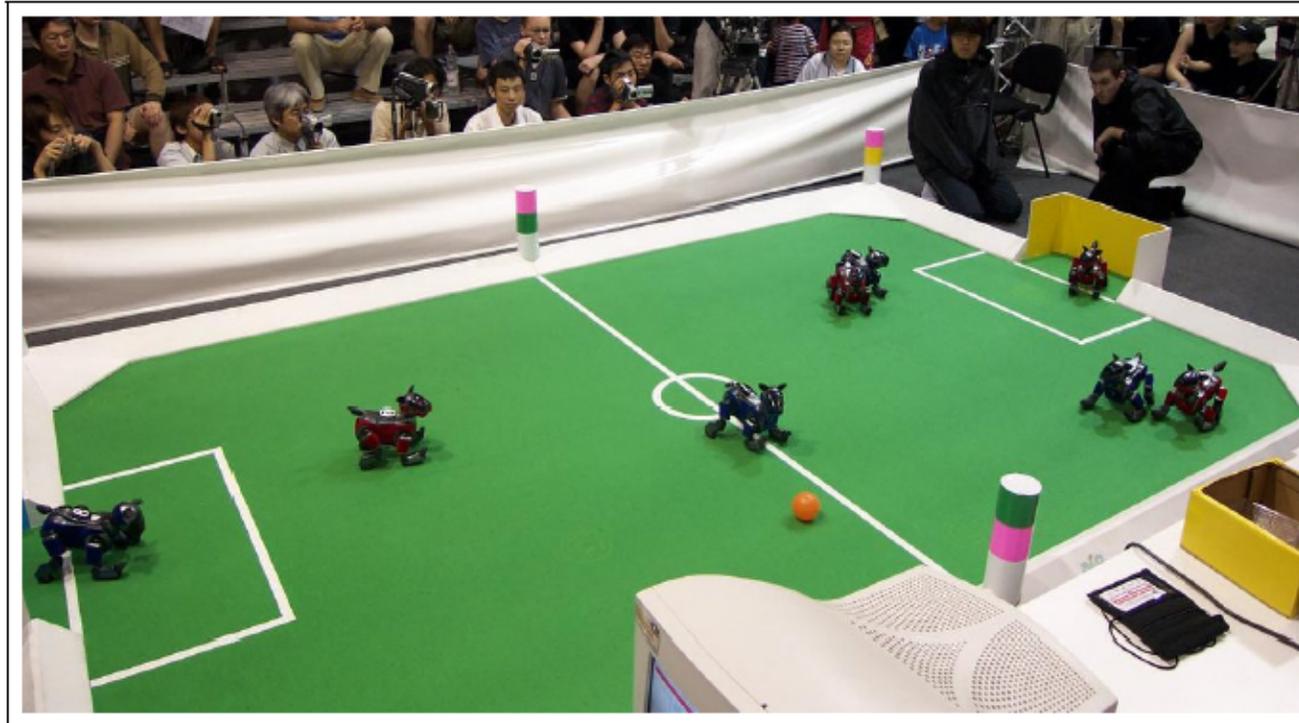
# Landmarkbasiertes Verfahren (2)

- Um das Gewicht  $w_i$  des Partikels  $i$  zu bestimmen, wird ermittelt wie wahrscheinlich die Sensorwerte an der Partikelposition  $\mathbf{x}[i] = (x, y, \theta)$  sind:

```
p = 1;  
for all Sensorwerte  $z_j = (r_j, \phi_j)$  do  
  rechne für Partikelposition  $\mathbf{x}[i] = (x, y, \theta)$   
  erwartete Entfernung  $r_j^*$  und relative  
  Orientierung  $\phi_j^*$  zur  $j$ -ten Landmarke aus;  
   $p = p * N(0, \sigma_r^2)(r_j - r_j^*) * N(0, \sigma_\phi^2)(\phi_j - \phi_j^*)$  ;  
return p;
```



# Beispiel (1)



- Aibo und Robo-Soccer
- 6 Landmarken am Spielfeldrand; farblich gekennzeichnet

- **Bewegungsmodell** auf 5-Bein-Antrieb angepasst.
- **Sensormodell** auf Landmarkenerkennung angepasst. Sensorwert  $z_t$  besteht aus Richtung, Abstand und Landmarkennummer. Sensordaten durch Kamera.

# Beispiel (2)

(c) **Partikelwolken**  
zu unterschiedlichen Zeitpunkten.  
Jeweils vor und nach Resampling.

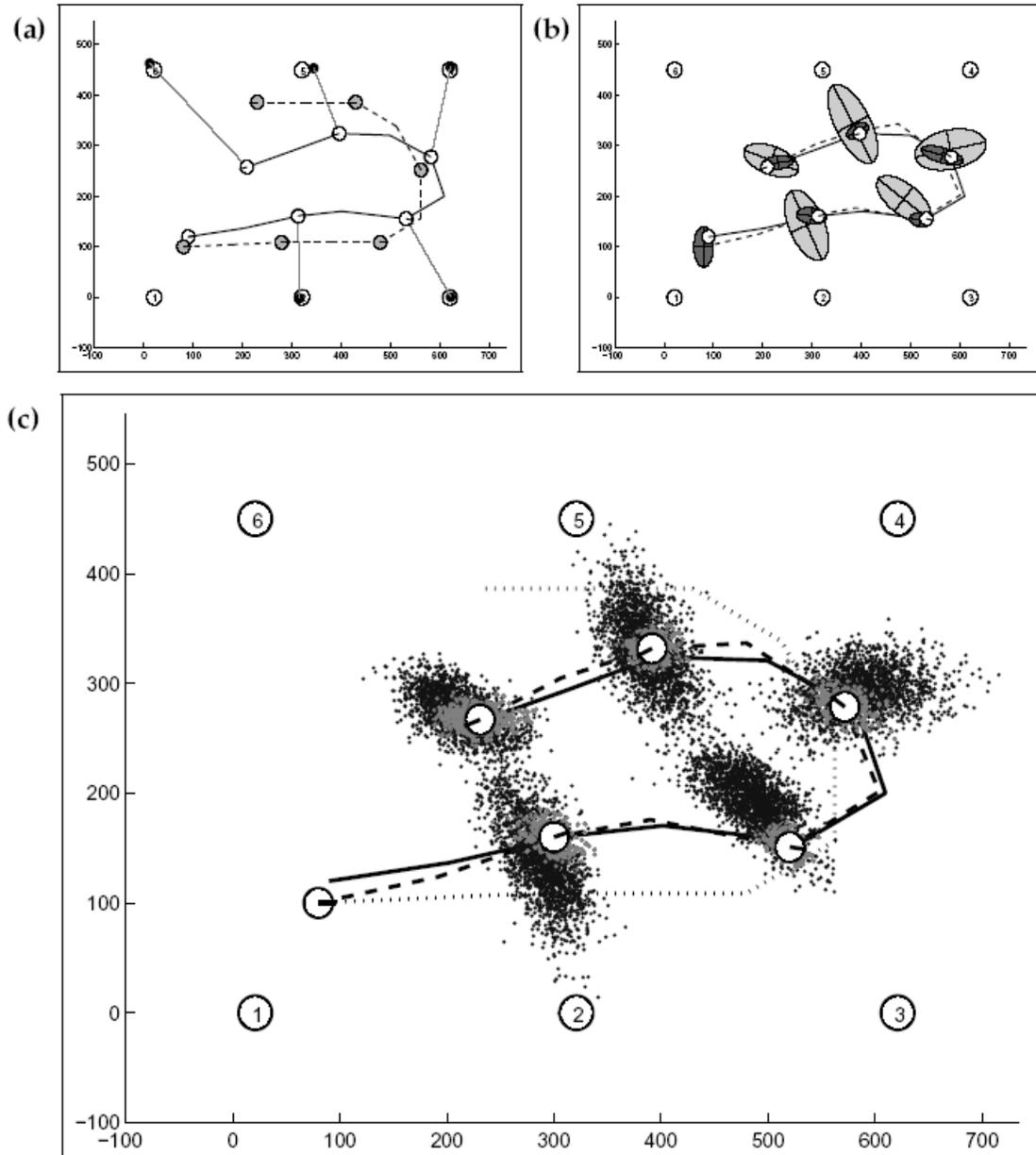
**Durchgezogene Linie:**  
tatsächlicher Weg

**Gepunktete Linie:**  
über Odometrie gemessener Weg

**Gestrichelte Linie:**  
über MCL ermittelter Weg

(a) gibt an welche Landmarke  
zu den einzelnen  
Zeitpunkten erkannt wird.

(b) Kovarianzen zu den  
Partikelwolken aus (a)



# Strahlenmodell

- Laser misst strahlenförmig Abstände  $z = d_1, d_2, \dots, d_n$  zu den nächsten Hindernissen.
- Es wird die Wahrscheinlichkeit  $p$  geschätzt, dass an der Partikelposition  $(x[i], y[i], \theta[i])$  diese Abstandsmessungen beobachtet werden.

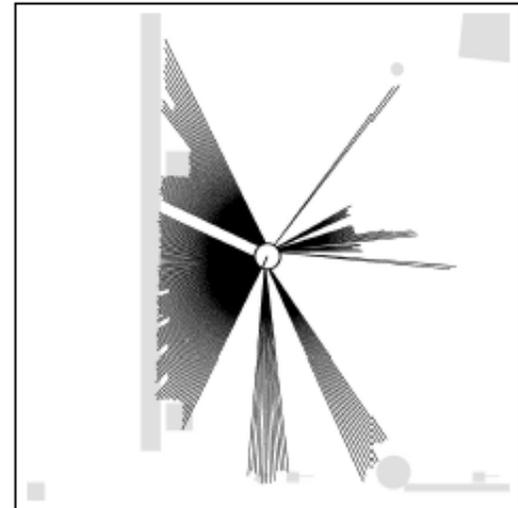
```
p = 1;
```

```
for k = 1 to n do
```

```
    berechne in Richtung der Messung  $d_k$  durch  
    Strahlverfolgung von Position  $(x[i], y[i], \theta[i])$  aus  
    in der Karte  $m$  den Abstand  $d_k^*$   
    zum ersten Hindernispunkt;
```

```
     $p = p * N(0, \sigma^2)(d_k - d_k^*);$ 
```

```
return p;
```



Beobachtete Abstandsmessungen  $z_t = z_1, z_2, \dots, z_n$

# Likelihoodfield (1)

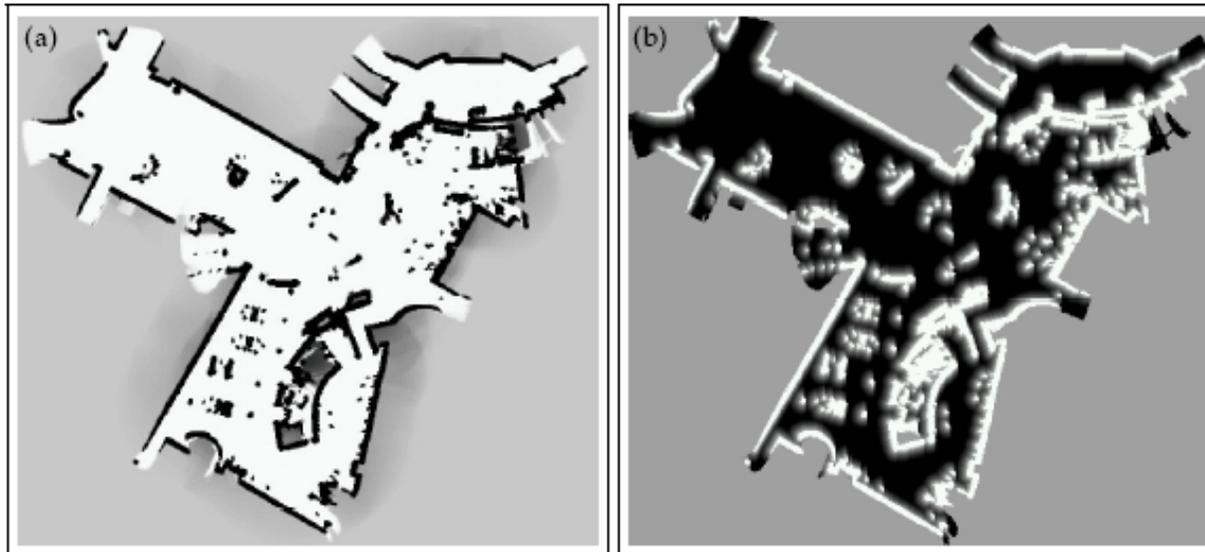
---

- Für eine Lasermessung  $z = d_1, d_2, \dots, d_n$  und eine Partikelposition  $(x[i], y[i], \theta[i])$  werden Hindernispunkte  $(x_{z1}, y_{z1}), \dots, (x_{zn}, y_{zn})$  im globalen Koordinatensystem berechnet.
- Es wird die Wahrscheinlichkeit  $p$  geschätzt, dass an der Partikelposition  $(x[i], y[i], \theta[i])$  die Lasermessung beobachtet wird, indem berechnet wird, wie nah die Hindernispunkte von tatsächlichen Hindernissen in der Umgebungskarte entfernt sind.

```
p = 1;
for k = 1 to n do
    berechne aus lokalen Polarkoordinaten  $(d_k, \phi_k)$  globale Koordinaten  $(x_{zk}, y_{zk})$ ;
    dist = Abstand von  $(x_{zk}, y_{zk})$  zum nächstgelegenen Hindernispunktes in m;
    p = p *  $N(0, \sigma^2)(dist)$ ;
end
return p;
```

# Likelihoodfield (2)

- Der Abstand  $\text{dist}$  von  $(x_{zk}, y_{zk})$  zum nächstgelegenen Hindernispunkt läßt sich vorab berechnen.
- Damit wird der Algorithmus (auf Kosten von Speicherplatz!) wesentlich beschleunigt.



Links:

Umgebungskarte m.

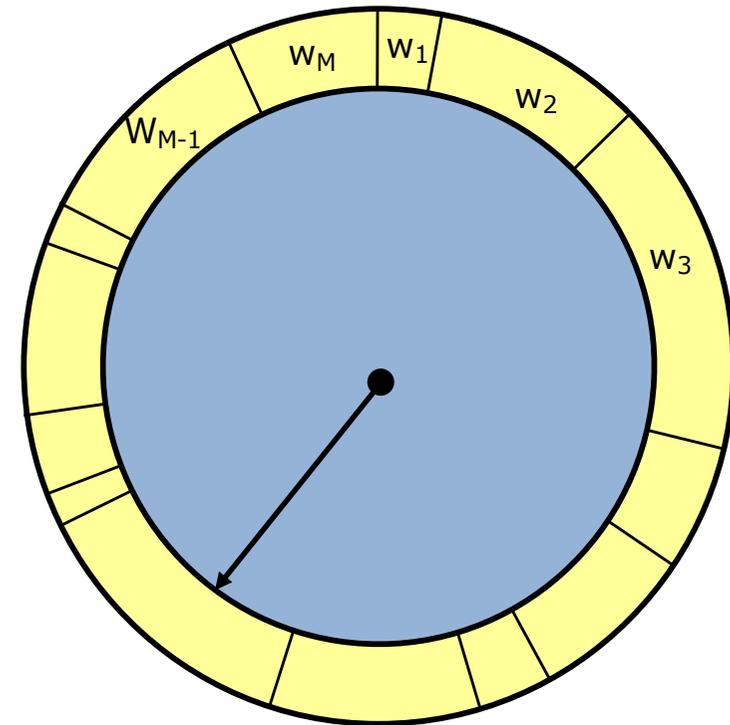
Rechts:

im Voraus berechnetes  
Likelihood-Field.

Je dunkler desdo  
unwahrscheinlicher ist es,  
dort ein Hindernispunkt zu  
beobachten.

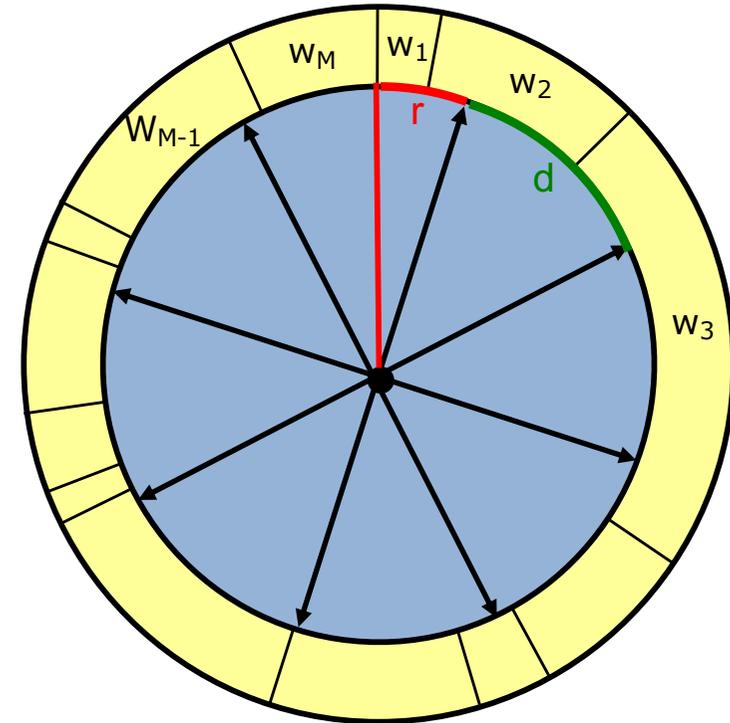
# Resampling mit Roulette-Rad-Verfahren

- Aus der Partikelmenge  $\{x[1], x[2], \dots, x[M]\}$  mit Gewichten  $w_1, \dots, w_M$  sollen  $M$  Partikeln zufällig entsprechend ihrem Gewicht gezogen werden.
- Bilde dazu  $M$  Intervalle (gelbe Streifen im Roulette-Rad):
  - $I_1 = [0, w_1]$
  - $I_2 = [w_1, w_1+w_2]$
  - ...
  - $I_M = [w_1+\dots+w_{M-1}, w_1+\dots+w_M]$
- Führe folgende Schritte  $M$ -mal aus:
  - Generiere gleichverteilte Zufallszahl  $z$  aus  $[0, w_1+\dots+w_M]$  (drehe Roulette-Rad);
  - Ermittle Intervall  $I_n$ , in dem  $z$  liegt mit binärer Suche;
  - Wähle Partikel  $x[n]$ ;
- Beachte: ein Partikel kann mehrfach ausgewählt werden.
- Laufzeit:  $O(M \log(M))$ .



# Resampling mit Speichenrad-Verfahren

- Aus der Partikelmenge  $\{x[1], x[2], \dots, x[M]\}$  mit Gewichten  $w_1, \dots, w_M$  sollen  $M$  Partikeln zufällig entsprechend ihrem Gewicht gezogen werden.
- Ordne Gewichte  $w_1, \dots, w_M$  auf einer kreisförmigen Skala an. Es sei  $W = w_1 + \dots + w_M$  die Gewichtssumme.
- Bilde ein Speichenrad mit  $M$  Speichen und einem Speichenabstand von  $d = W/M$ .
- Drehe 1. Speiche auf einen zufälligen Wert  $r \in [0, d]$ .
- Wähle jeden Partikel  $x[i]$  aus, auf dessen Gewicht  $w[i]$  eine Speiche zeigt. Zeigen mehrere Speichen auf einen Partikel, dann wird der Partikel mehrfach ausgewählt.
- Laufzeit:  $O(M)$ .



# Diversität der Partikelmenge

---

- Wünschenswert ist eine gewisse Streuung der Partikelmenge (Diversität). Im Idealfall ist die Partikelmenge normalverteilt um die tatsächliche Pose.
- Falls sich der Roboter im Extremfall über viele Schritte nicht bewegt (d.h. keine Integration eines Steuerbefehls) und darüberhinaus die Gewichte je Partikel immer ähnlich bleiben, dann tendiert das Roulette-Verfahren dazu, mit  $M$  Kopien des gleichen Partikels zu enden.

Die Posenvarianz der Partikel wird 0. D.h. die Diversität verschwindet komplett. Die Wahrscheinlichkeit eines Lokalisierungsfehlers ist jedoch sehr groß.

- Das Speichenrad-Verfahren behält dagegen eher die Diversität bei. Haben alle Partikel das gleiche Gewicht, dann bleibt die Partikelmenge unverändert.
- Die Diversität lässt sich auch durch eine Verringerung der Resampling-Frequenz erhalten.

Es wird nur in jedem  $N$ -ten Schritt ein Resampling durchgeführt.

In den Zwischenschritten werden die Partikeln zwar gewichtet, jedoch werden die Gewichte je Partikel nur aufmultipliziert. Das Resampling wird über das Gewichtsprodukt durchgeführt. Die Steuerbefehle werden dagegen in jedem Schritt integriert.

Ein guter Indikator, um festzustellen, ob ein Resampling durchgeführt werden soll, ist die Gewichtsvarianz. Nur bei hoher Gewichtsvarianz sollte ein Resampling durchgeführt werden.