# SLAM-Verfahren

## Synchrone Lokalisierung und Kartenerstellung

- EKF-SLAM:
  Landmarkenbasiertes SLAM-Verfahren mit einem erweiterten Kalmanfilter
- Fast-Slam: Gitterbasiertes SLAM-Verfahren mit einem Partikelfilter

# Problemstellung

- Roboter exploriert eine unbekannte, statische Umgebung mit Landmarken.
- Dabei wird Roboter mit Steuerdaten u bewegt und es werden Sensordaten z erfasst:

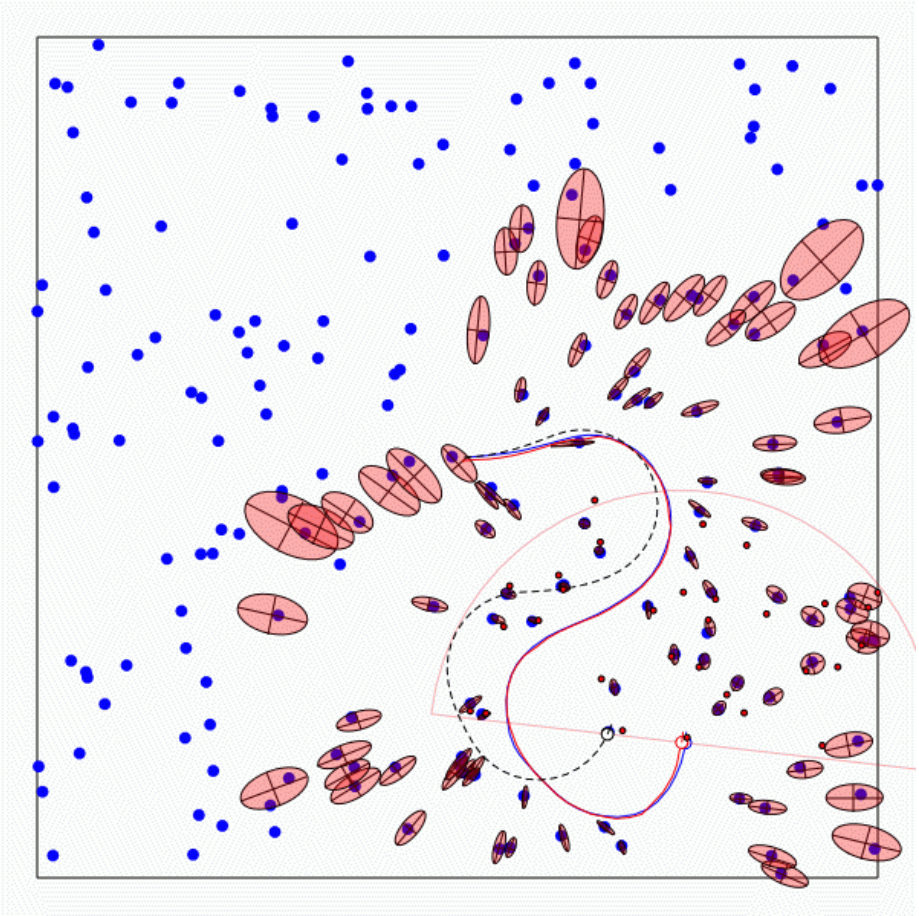
  $u_1, z_1, u_2, z_2, \ldots, u_t, z_t$

- Gesucht ist Karte m mit n Landmarken

  $m = \ell_{1,x}, \ell_{1,y}, \ldots, \ell_{n,x}, \ell_{n,y}$

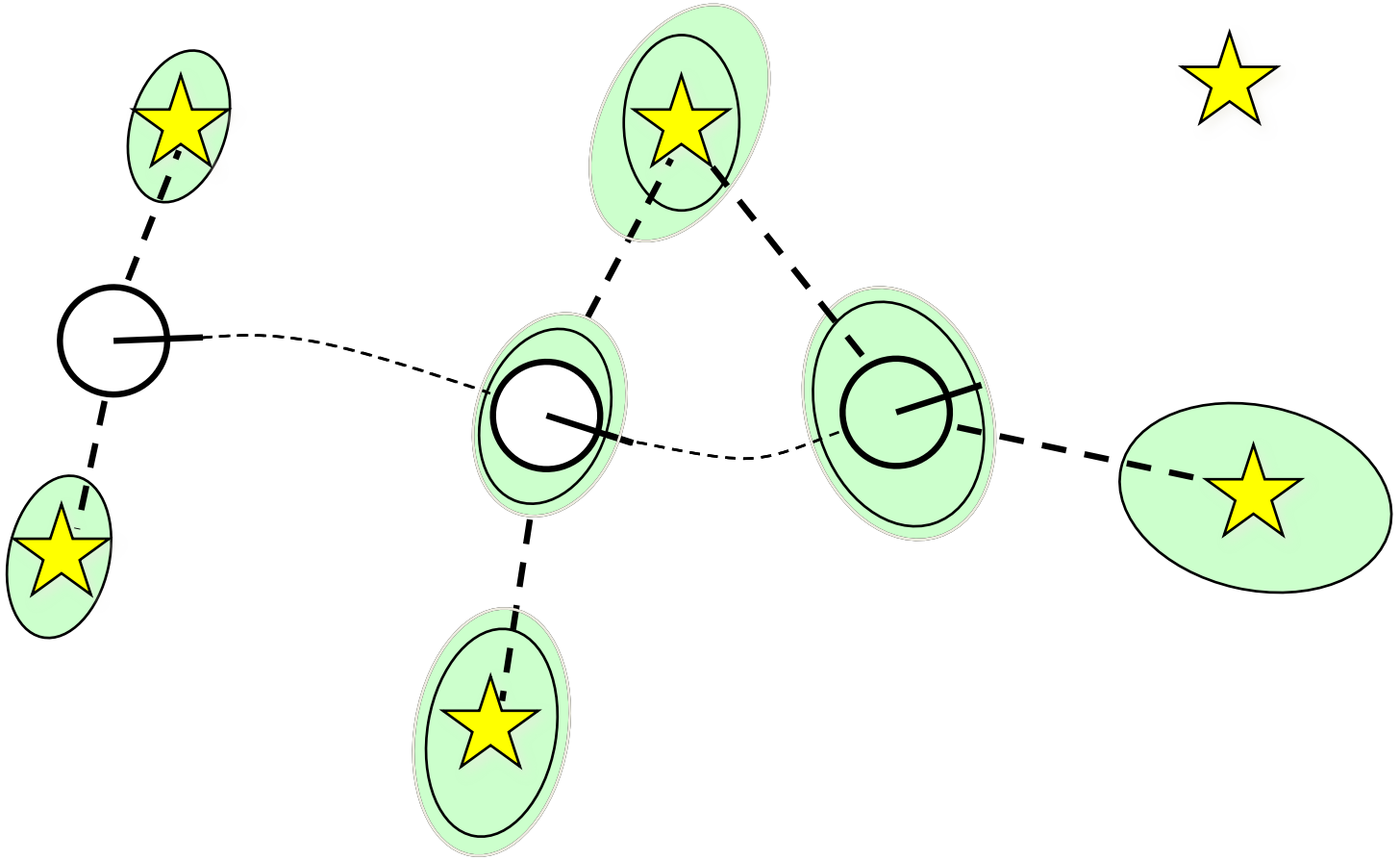
  und Weg des Roboters

  $x_1, x_2, \ldots, x_t$



# Warum ist SLAM ein schwieriges Problem?

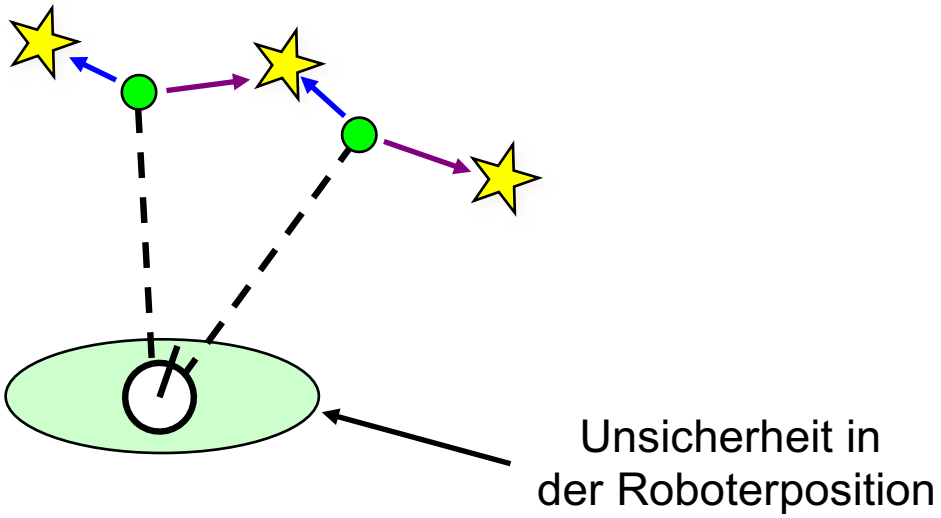
- Sowohl Positionen der Landmarken als auch Roboterweg sind unbekannt.



- Kartenfehler und Fehler im Roboterweg sind korreliert.

# Warum ist SLAM ein schwieriges Problem?

- Die Zuordnung von Messdaten zu Landmarken sind in der Regel unbekannt.
- Roboter muss entscheiden, ob Messdaten zu einer bereits beobachteten Landmarke zugeordnet werden können oder zu einer noch nicht gesehenen Landmarke.
- Zuordnungsproblematik wird durch Fehler im Roboterweg verstärkt.



# EKF-SLAM (1)

- Karte mit n Landmarken und Roboterposition wird durch 2n+3-stelligen Zustandsvektor und Kovarianzmatrix dargestellt.

$$\mathbf{x}_k = \begin{pmatrix} x \\ y \\ \theta \\ l_1 \\ l_2 \\ \vdots \\ l_n \end{pmatrix} \qquad \Sigma_k = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xl_1} & \sigma_{xl_2} & \cdots & \sigma_{xl_n} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\theta} & \sigma_{yl_1} & \sigma_{yl_2} & \cdots & \sigma_{yl_n} \\ \sigma_{x\theta} & \sigma_{y\theta} & \sigma_\theta^2 & \sigma_{\theta l_1} & \sigma_{\theta l_2} & \cdots & \sigma_{\theta l_n} \\ \sigma_{xl_1} & \sigma_{yl_1} & \sigma_{\theta l_1} & \sigma_{l_1}^2 & \sigma_{l_1 l_2} & \cdots & \sigma_{l_1 l_n} \\ \sigma_{xl_2} & \sigma_{yl_2} & \sigma_{\theta l_2} & \sigma_{l_1 l_2} & \sigma_{l_2}^2 & \cdots & \sigma_{l_2 l_n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{xl_n} & \sigma_{yl_n} & \sigma_{\theta l_n} & \sigma_{l_1 l_n} & \sigma_{l_2 l_n} & \cdots & \sigma_{l_n}^2 \end{pmatrix}$$

- Roboter-Position: $(x,y,\theta)$
- n Landmarken-Positionen: $l_i = (x_i, y_i)$ für $1 \leq i \leq n$
- EKF-SLAM kann einige hundert Landmarken behandeln.

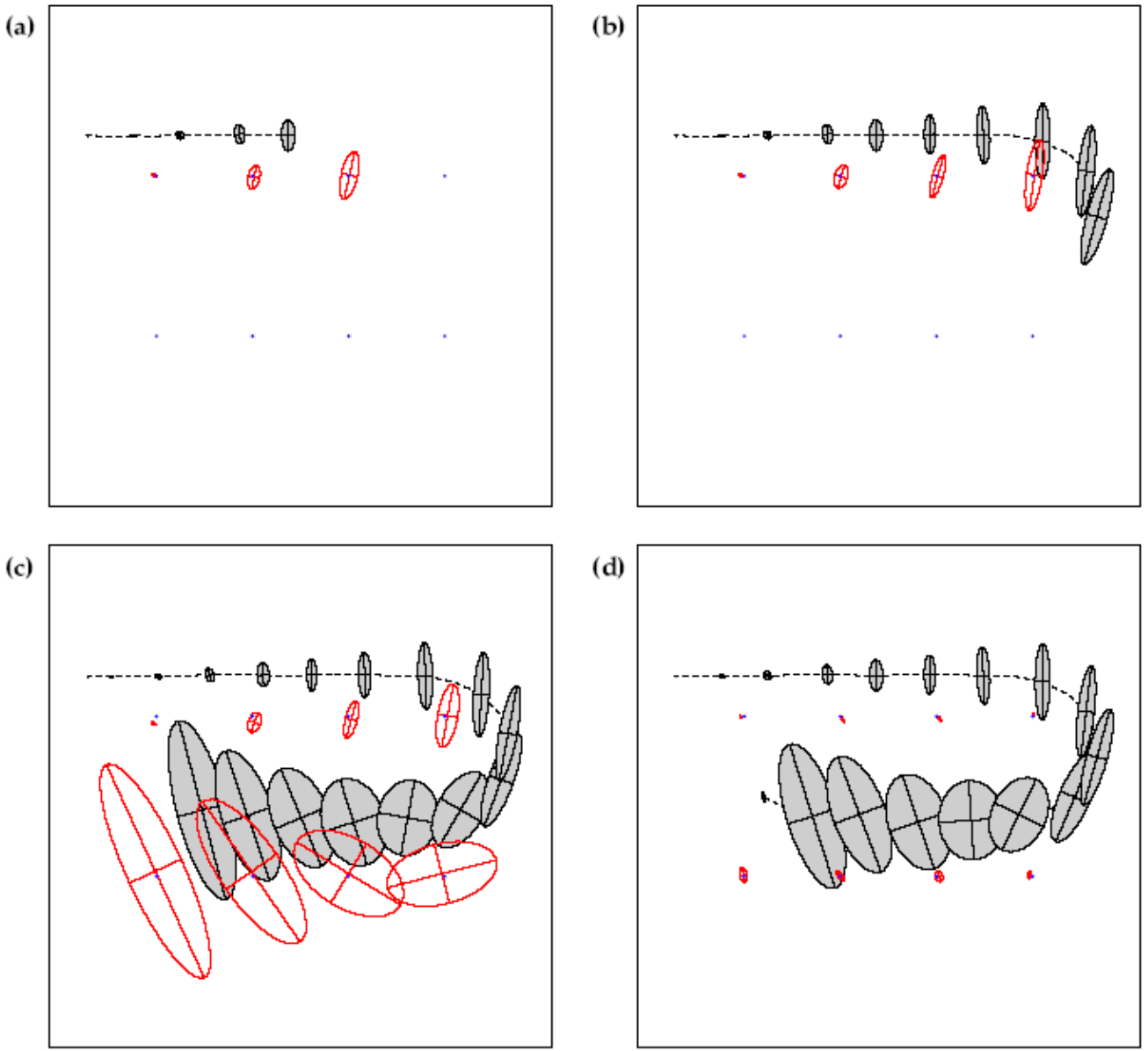
# EKF-SLAM (2)

- System- und Messgleichungen sind i.a. nicht linear.

$$\mathbf{x}_{k+1} = g(\mathbf{x}_k, \mathbf{u}_k)$$
$$\mathbf{z}_k = h(\mathbf{x}_k)$$

- Bei der Systemgleichung wird angenommen, dass sich die Landmarken nicht bewegen!

# Beispiel mit 8 Landmarken

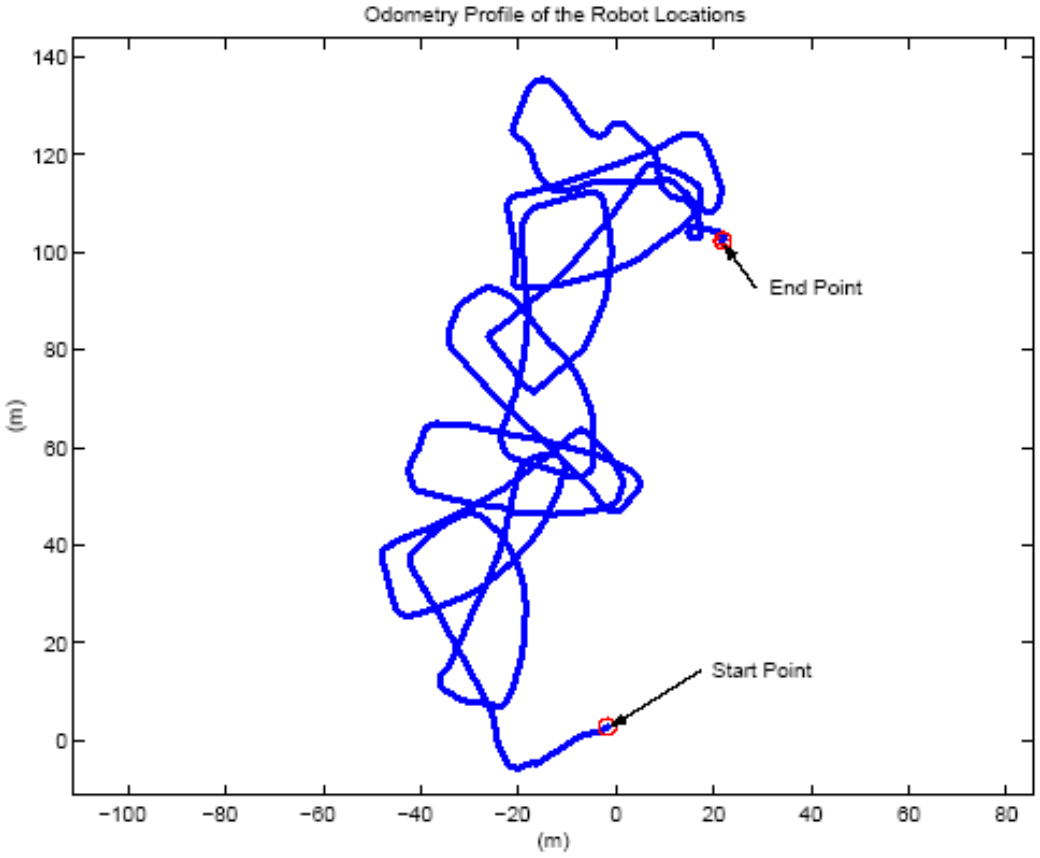



- Geschätzte Roboter-positionen sind grau dargestellt.
- Landmarken mit Positionsunsicher-heiten sind rot dargestellt.
- In (d) sieht der Roboter die erste Landmarke erneut. Sämtliche Unsicher-heiten werden erheblich kleiner.

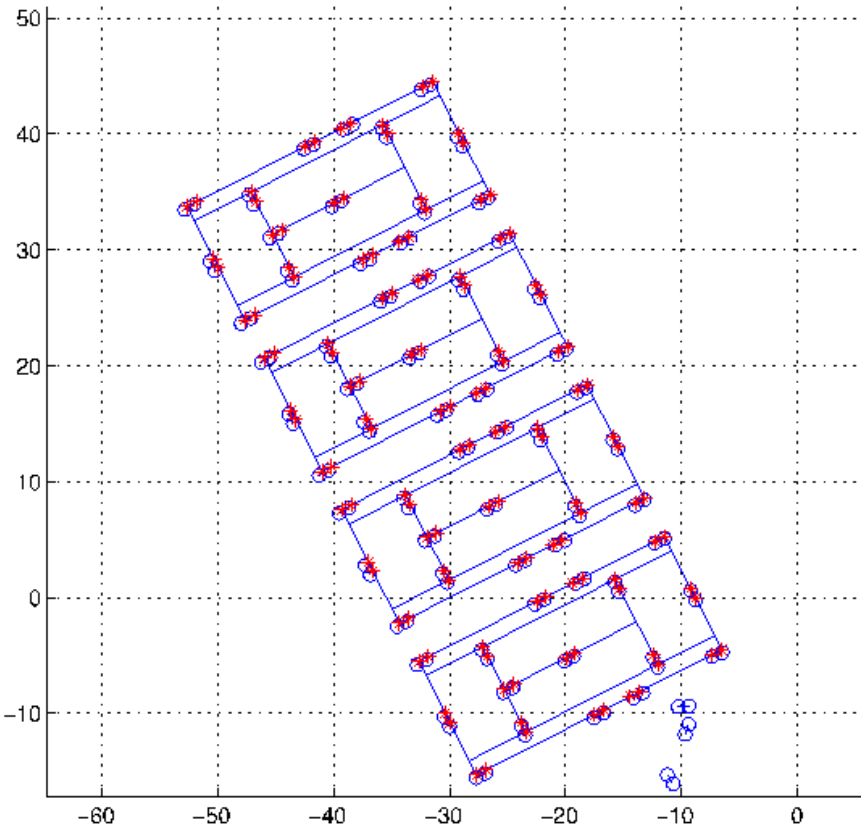
# SLAM-Kartierung eines Tennisplatzes (1)


[J. Leonhard, MIT]

# SLAM-Kartierung eines Tennisplatzes (2)




Mit Koppelnavigation ermittelter Weg



Mit EKF-SLAM ermittelte Karte

# SLAM-Verfahren
## Synchrone Lokalisierung und Kartenerstellung

- EKF-SLAM:
  Landmarkenbasiertes SLAM-Verfahren mit einem erweiterten Kalmanfilter
- Fast-Slam: Gitterbasiertes SLAM-Verfahren mit einem Partikelfilter

# Problemstellung

- Roboter exploriert eine unbekannte, statische Umgebung z.B. mit Laser-Scanner.
- Dabei wird Roboter mit Steuerdaten u bewegt und es werden Sensordaten z erfasst:

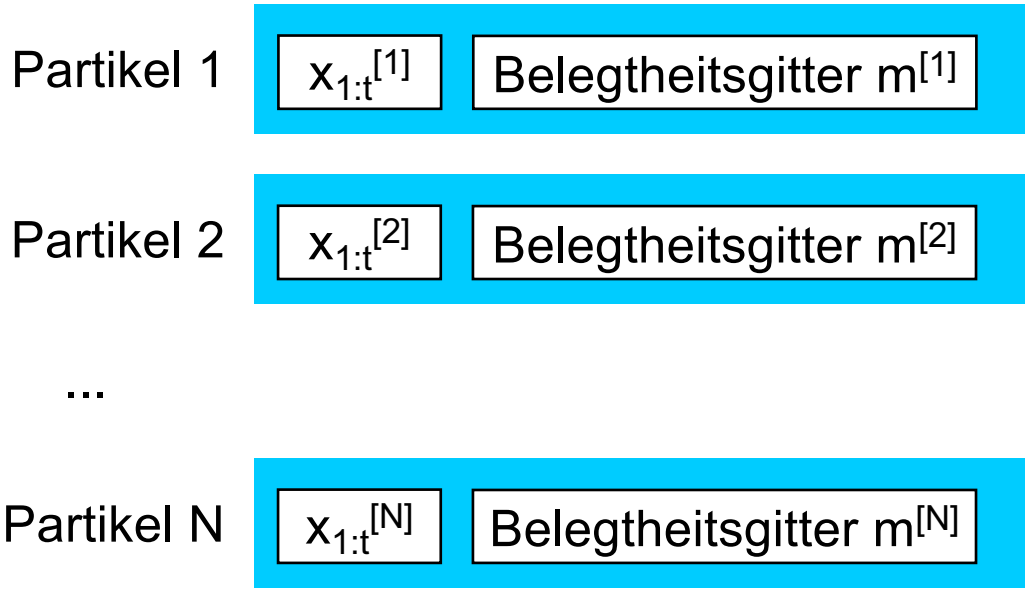
  $u_1, z_1, u_2, z_2, \ldots, u_t, z_t$

- Gesucht ist Belegtheitsgitter m und Weg des Roboters

  $x_1, x_2, \ldots, x_t$

# Gitterbasiertes SLAM-Verfahren als Partikel-Filter

- Verwalte zu jedem Zeitpunkt $t_k$ Partikelmenge $\Gamma_t$:




- $x_{1:t}^{[k]}$ ist der im Partikel k gespeicherte Roboterweg.

# Algorithmus

**Algorithmus FastSLAM_OccupancyGrids**( $\chi_{t-1}$, $u_t$, $z_t$):

for k = 1 to M do

Integration des Steuerbefehls:

$x_t^{[k]} = \text{sampleMotionModel}(u_t, x_{t-1}^{[k]})$;

Gewicht für jeden Partikel brechnen:

$w_t^{[k]} = \text{measurementModelMap}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$;

Integration der Sensordaten:

$m_t^{[k]} = \text{updateOccupancyGrid}(m_{t-1}^{[k]}, x_t^{[k]}, z_t)$;

endfor

Resampling:

$\chi_t = \emptyset$;

for i = 1 to M do

ziehe k zufällig mit Wahrscheinlichkeit $w_t^{[k]}$ ;

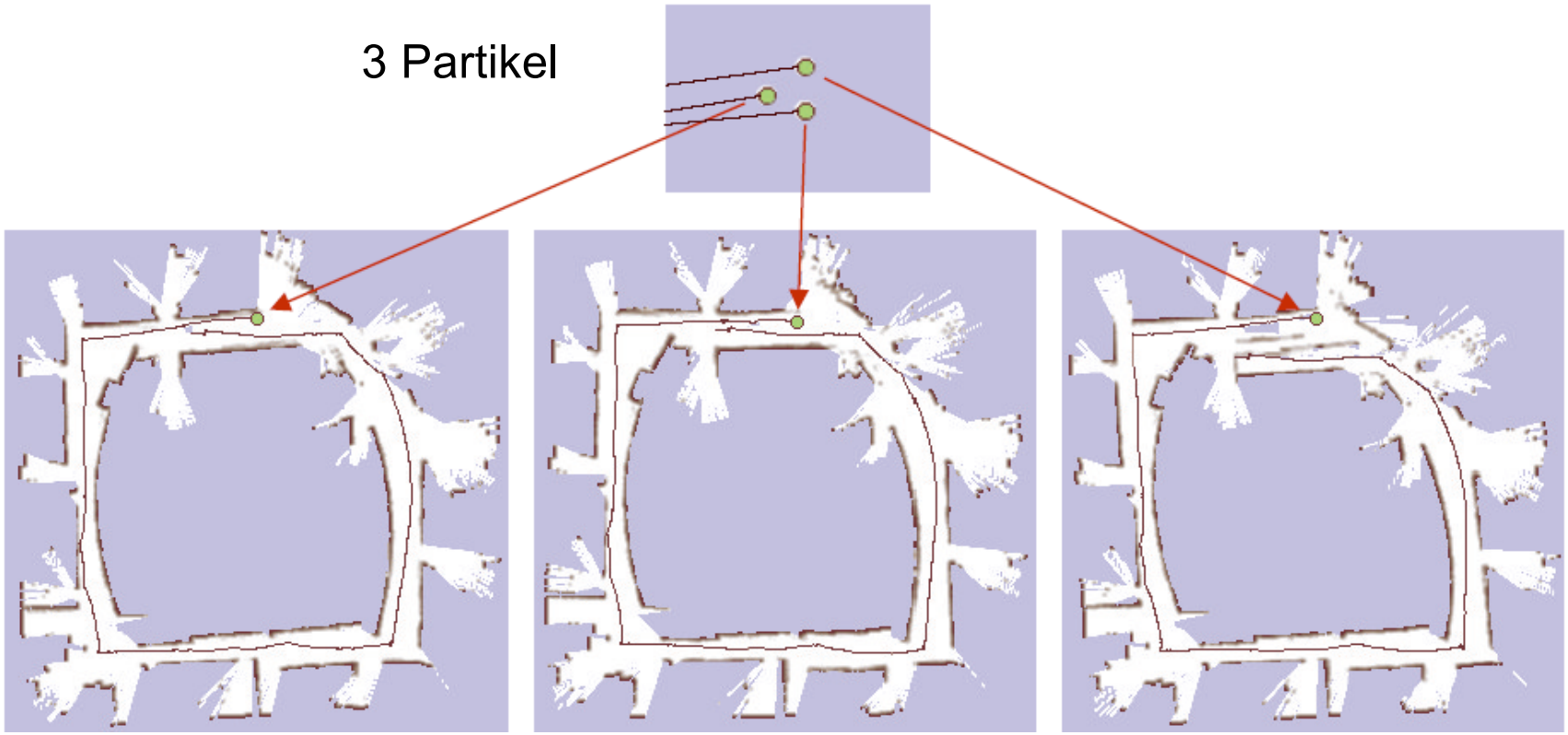
$\chi_t = \chi_t \cup \{ <x_{1:t}^{[k]}, m_t^{[k]} > \}$;

endfor

return $\chi_t$;

wie beim MCL-Algorithmus aus Kap. 4

# Beispiel mit 3 Partikel

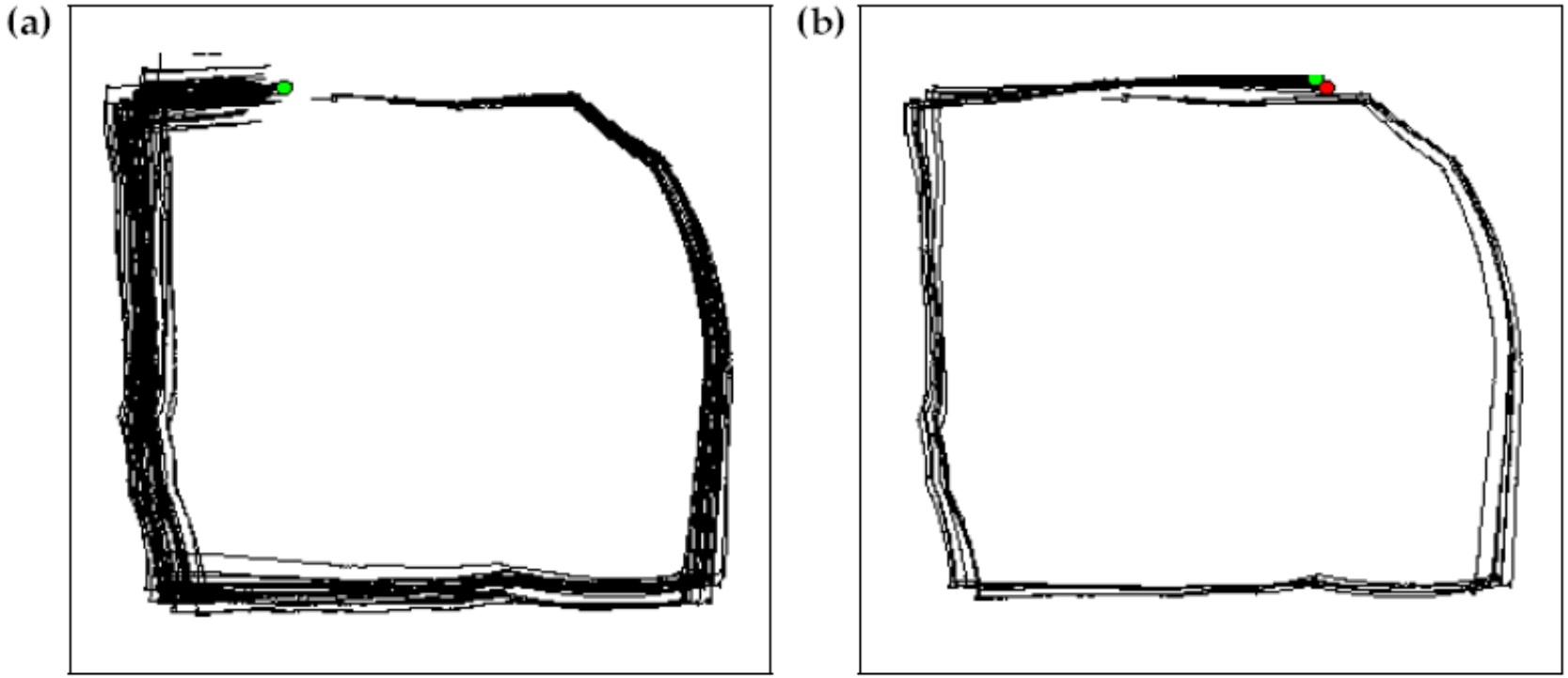



Roboterweg und Karte von Partikel 1

Roboterweg und Karte von Partikel 2

Roboterweg und Karte von Partikel 3

# Beispiel mit mehreren Partikel (nur Roboterwege)




(a) Aufgrund der langen Strecke steigt die Unsicherheit in der Position.
Die Partikel spreizen stark.

(b) Die Schleife wird geschlossen.
Der Roboter gelangt in einem Kartenbereich, der mit größerer Genauigkeit zu Beginn aufgenommen wurde.
Viele unwahrscheinliche Partikel fallen beim Resampling weg.
Die Partikel spreizen jetzt weniger stark.

# Wichtig: Anzahl der Partikel klein halten

**Problem:**

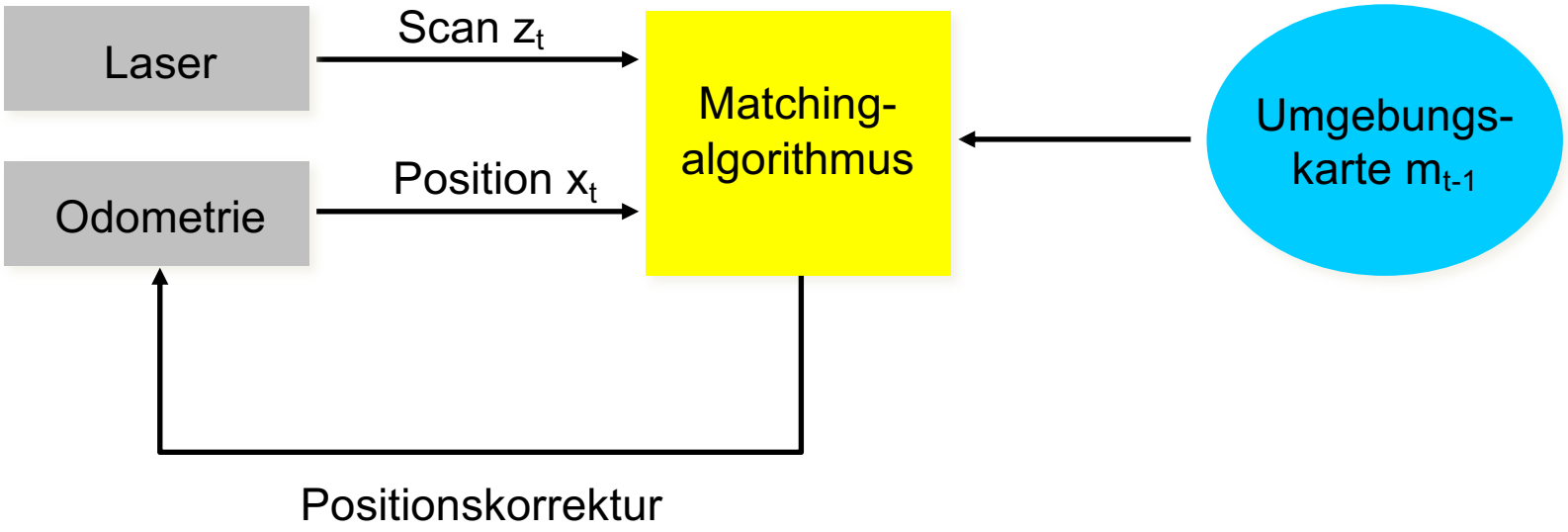
- bei der gitterbasierten Kartenerstellung kann ein Gitter sehr groß werden.
- Jeder Partikel enthält ein eigenes Belegtheitsgitter.
- Daher müssen die Anzahl der Partikel klein gehalten werden!

**Lösungsansätze:**

- Verbesserung der Odometrie mit Scan-Matching
- Verbesserung von sampleMotionModel: auf Sensordaten ausgerichtete Generierung von Positionen

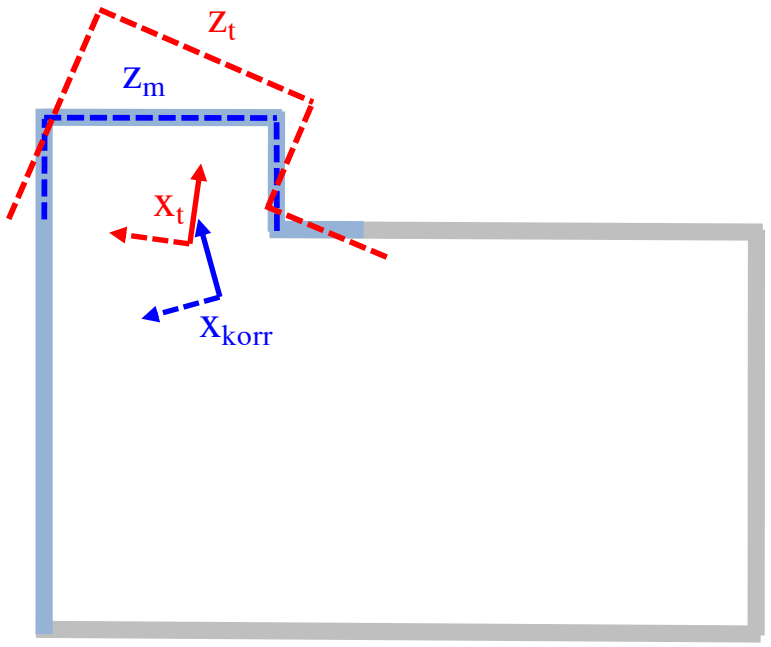
# Verbesserung der Odometrie durch Scanmatching

- verbessere die durch Odometrie ermittelte Position $x_t$, indem der aktuelle Laserscan $z_t$ mit der bisherigen Karte $m_{t-1}$ abgeglichen wird.



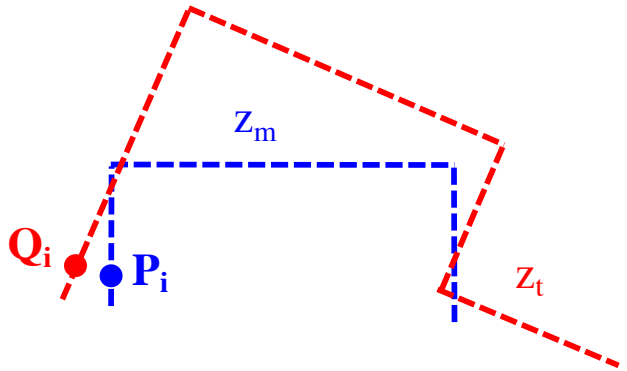
# Scanmatching (1)

- Der Roboter bewegt sich in der grau dargestellten Umgebung.
- Er hat zum Zeitpunkt t bereits den hellblau dargestellten Bereich $m_{t-1}$ kartiert.
- An der geschätzten (aber fehlerhaften) Position $x_t$ wird der Scan $z_t$ aufgenommen.
- Aufgrund der Umgebungskarte $m_t$ und der Position $x_t$ müsste sich aber der Scan $z_m$ ergeben.
- Durch ein Scanmatching-Verfahren wird $z_t$ möglichst gut auf $z_m$ transformiert (verschoben und gedreht).
- Mit der berechneten Transformation lässt sich die Position zu $x_{korr}$ korrigieren.

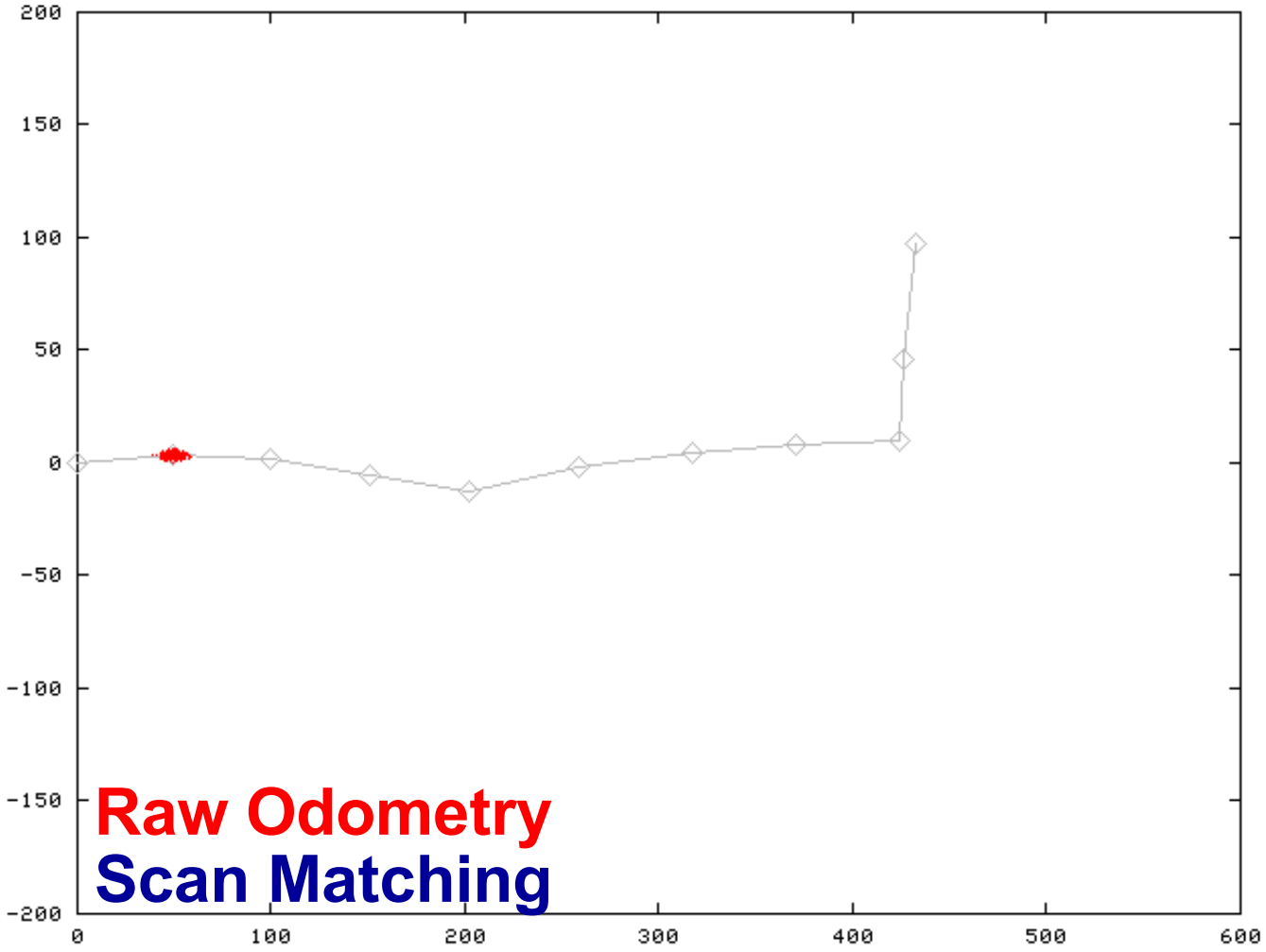


# Scanmatching (2)

- IDC (Iterative Dual Correspondance)
  ist ein bekanntes Scanmatching-Verfahren, das iterativ solange arbeitet, bis entweder eine Grenze für die Anzahl der Iterationen oder aber eine bestimmte Überdeckungsgüte erreicht wird.
- Bei jeder Iteration werden aus den beiden Scans zunächst eine Menge von korrespondierenden Punktepaare ($P_i$, $Q_i$) bestimmt.
- Durch ein LMS-Ansatz (Least Mean Square) wird eine Transformation T so bestimmt, dass die transformierten Punkte $T(Q_i)$ von $P_i$ möglich gering entfernt sind.
- Bricht die Iteration ab und wurde die gewünschte Überdeckungsgüte nicht erreicht, dann werden die Scans als nicht matchbar eingestuft.
- Wurde die Überdeckungsgüte erreicht, wird die Transformation mit einer Kovarianz zurückgeliefert, die die Überdeckunsgüte angibt.

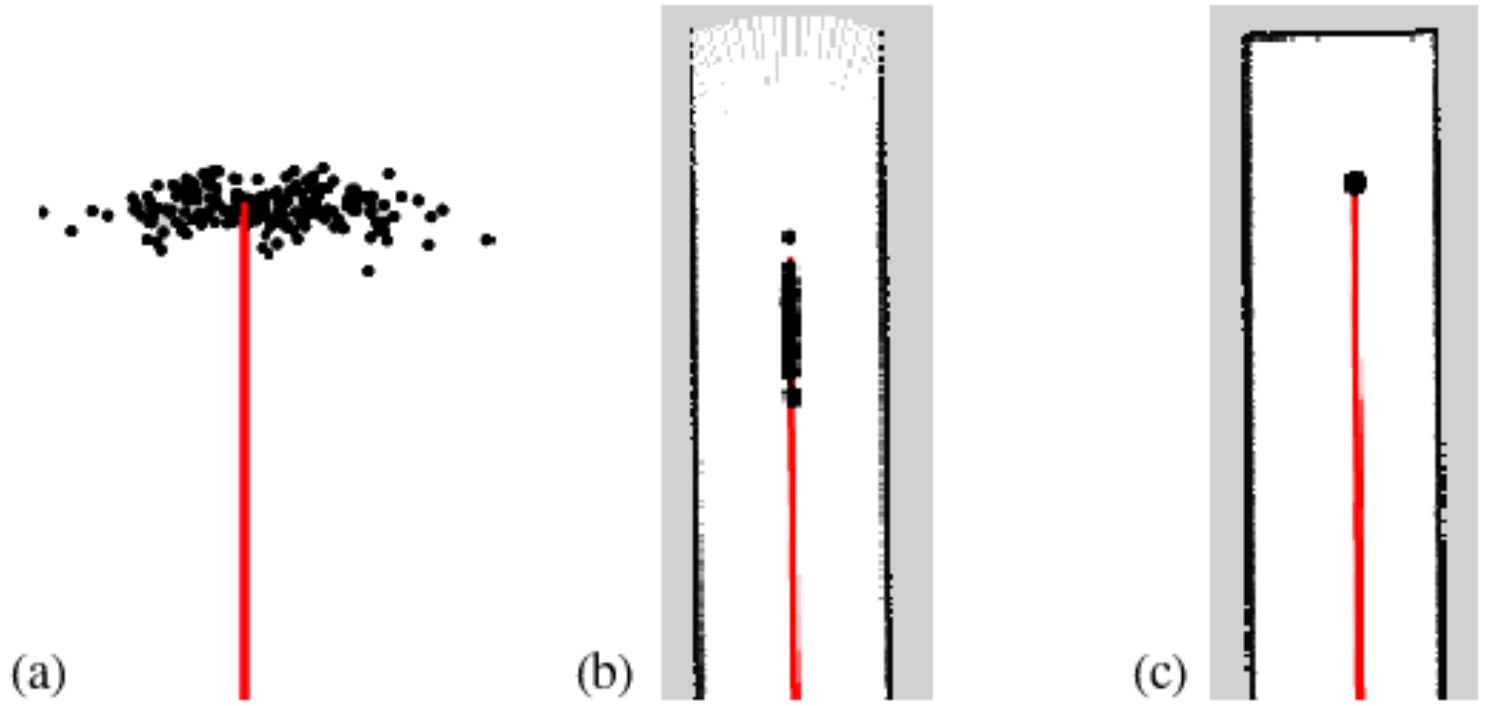


# Illustrierung der Positionsverbesserung beim Scanmatching



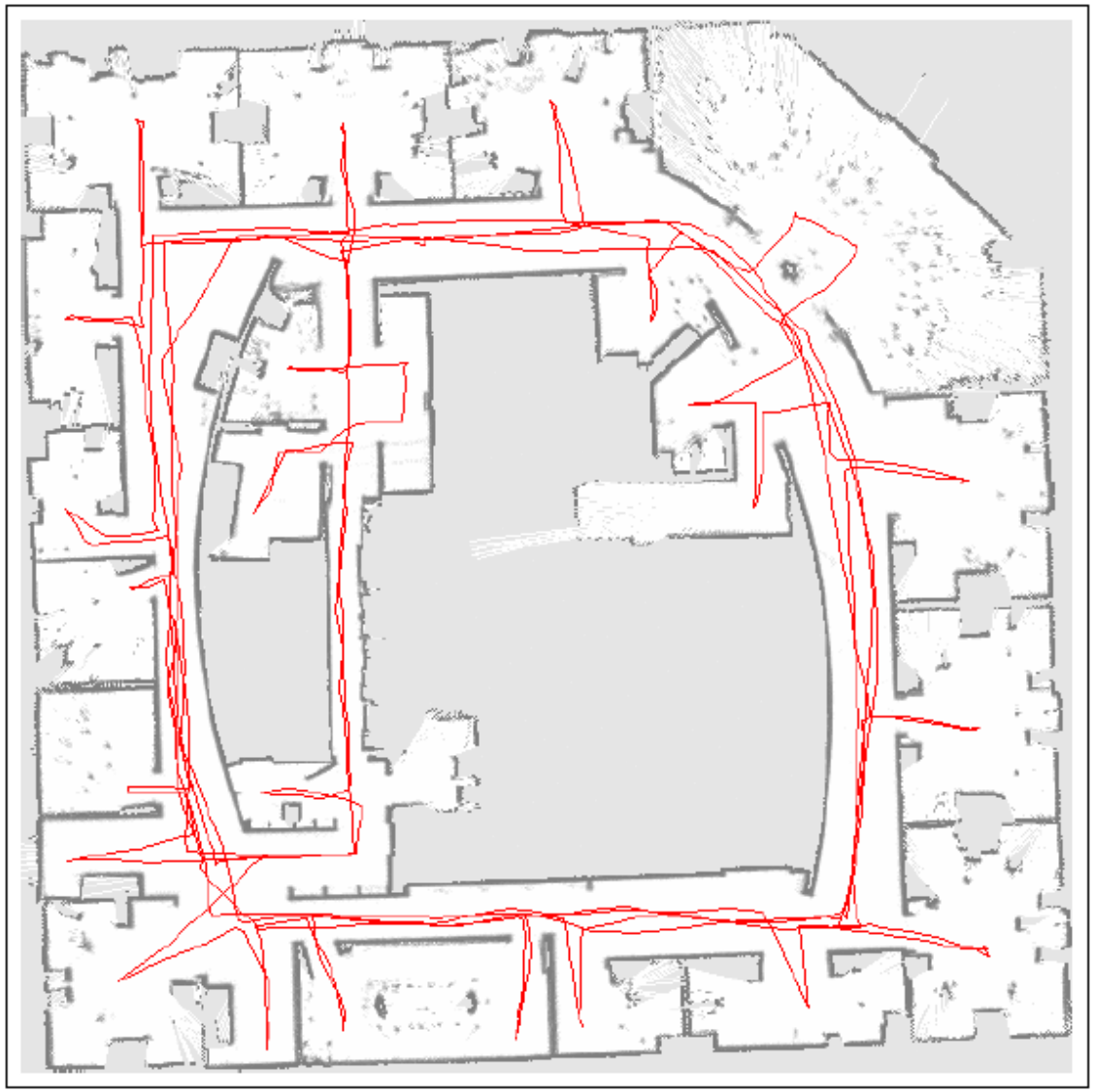
# Verbesserung von sampleMotionModel

- von Sensordaten (d.h. Umgebung) abhängige Generierung von Positionen



# Intel-Lab mit Standard-FastSlam (1)

- Belegheitsgitter für den Partikel mit höchsten (akkumuliertem) Gewicht
- 500 Partikel
- Roboterpfad: 491m
- Umgebung: 28m * 28m
- Durchschnitts-geschwindigkeit: ca. 0.2 m/sec



# Intel-Lab mit Standard-FastSlam (2)

- Darstellung der reinen Odometriedaten und des damit erzeugten Belegtheitsgitters

# Intel-Lab mit FastSlam und Scanmatching

- 15 Partikel
- 5 cm Gitter-Auflösung während des Scanmatchings
- 1 cm Auflösung in der endgültigen Karte

# Outdoor Campus Map

- 30 Partikel
- 250m x 250m
- ca. 1,7 km (Odometrie)
- 20 cm Auflösung während des Scanmatchings
- 30 cm Auflösung in der endgültigen Karte