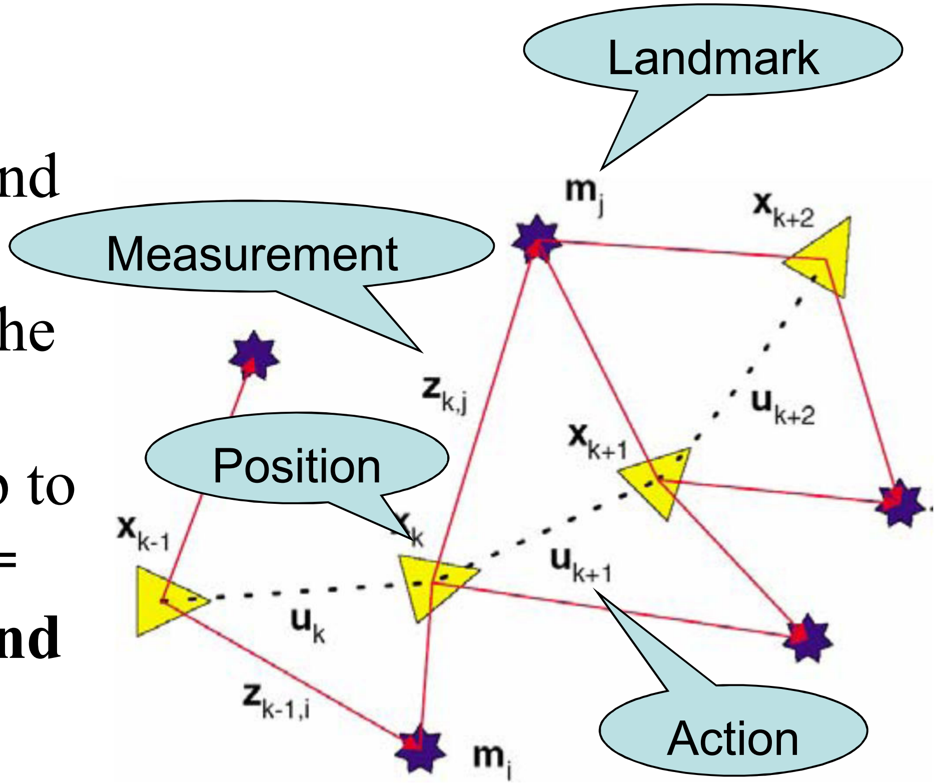
Simultaneous Localisation and Mapping

IAR Lecture 10

Barbara Webb

What is SLAM?

Start in an unknown location and unknown environment and incrementally build a **map** of the environment while **simultaneously** using this map to compute vehicle **location** = **Simultaneous Localisation And Mapping**

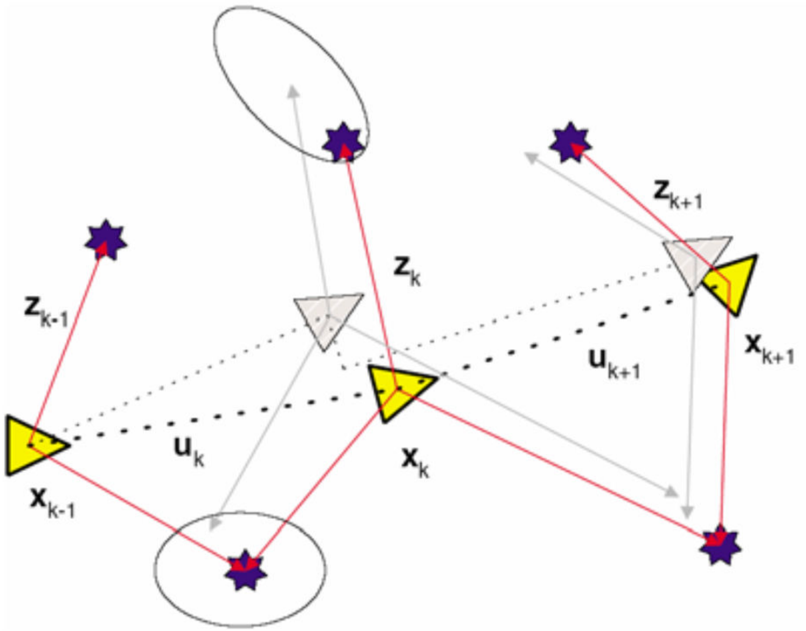


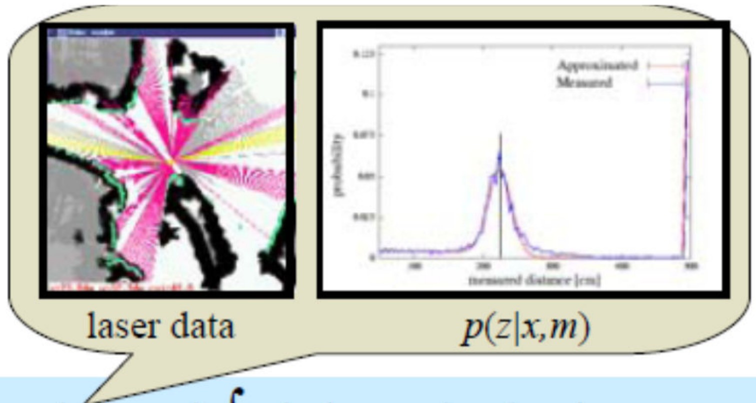
Estimate the pose and the map of a mobile robot at the same time

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

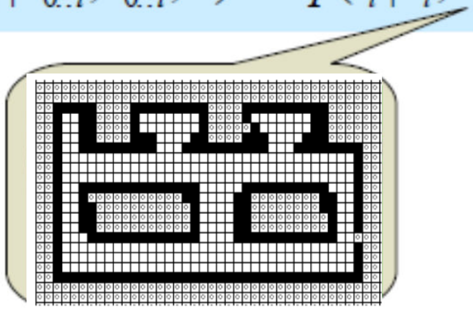
poses map measurements & actions

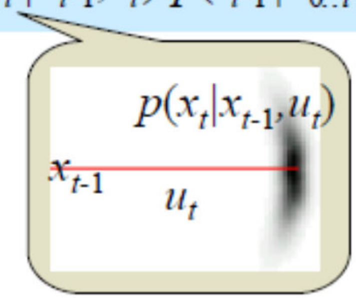
So far, we have been discussing the localisation problem, i.e., a map **m** is known *a priori*. From a sequence of control actions **U** and measurements **Z** we can infer the locations of the robot **X.**



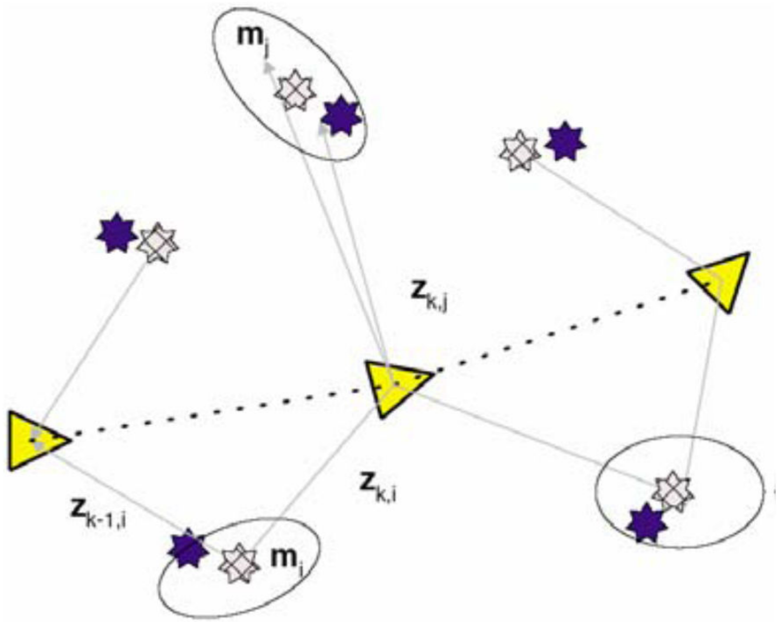


$$p(x_t \mid z_{0..t}, u_{0..t}, m) = p(z_t \mid x_t, m) \int p(x_t \mid x_{t-1}, u_t)\, p(x_{t-1} \mid z_{0..t-1}, u_{0..t-1}, m)\, dx_{t-1}$$





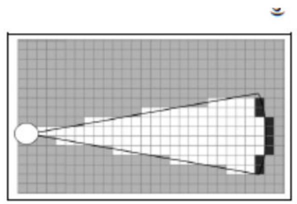
Complementary to localisation is the mapping problem: If we knew the location **X** of the robot (e.g. precise GPS) then from the measurements **Z** we could infer the map **M.**



- E.g. represent environment by a grid and estimate the (assumed independent) probability that each location is occupied by an obstacle.

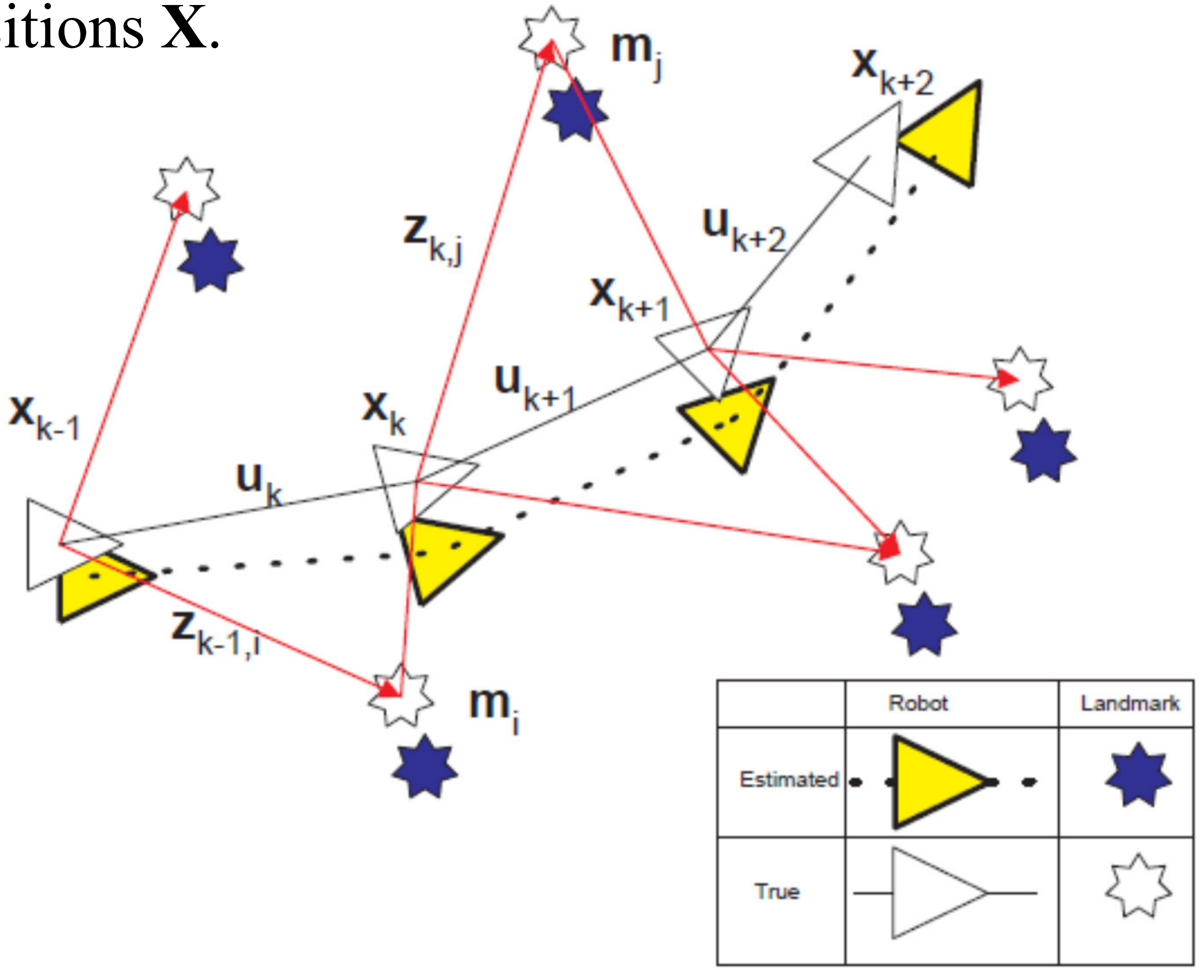
$$p(m \mid z_{1:t}, x_{1:t}) = \prod_i p(m_i \mid z_{1:t}, x_{1:t})$$

Inverse sensor model

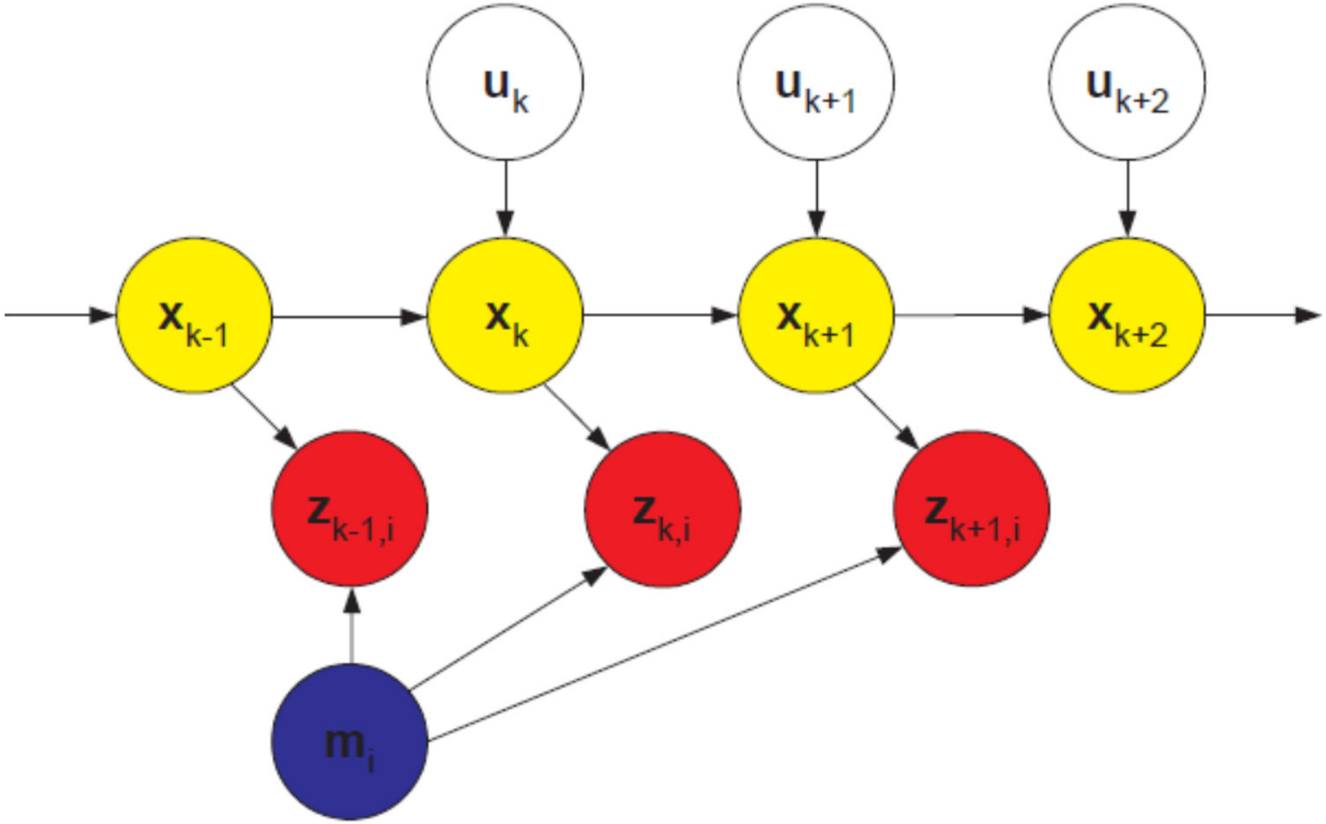


But can we solve the 'chicken and egg' problem?

If we only know the robot's position at $\mathbf{x_0}$, use the sequence of actions **U** and measurements **Z** to infer both the map **M** and the robot positions **X**.



Bayesian SLAM



Bayesian SLAM

- Recursive filter for estimating robot positions and map
- Prediction (time update)

Motion model

$$\underbrace{P(x_t, m \mid z_{0:t-1}, u_{0:t}, x_0)}_{\overline{\mathrm{Bel}}(x_t, m)} = \int P(x_t \mid u_t, x_{t-1}) \times \underbrace{P(x_{t-1}, m \mid z_{0:t-1}, u_{0:t-1}, x_0)}_{\text{Estimate at previous time step, } \mathrm{Bel}(x_{t-1}, m)} \, dx_{t-1}$$

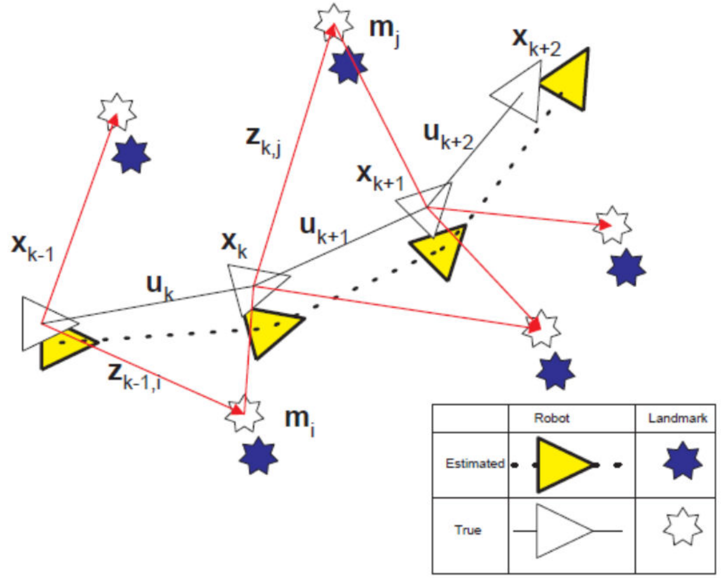
- Correction (measurement update)

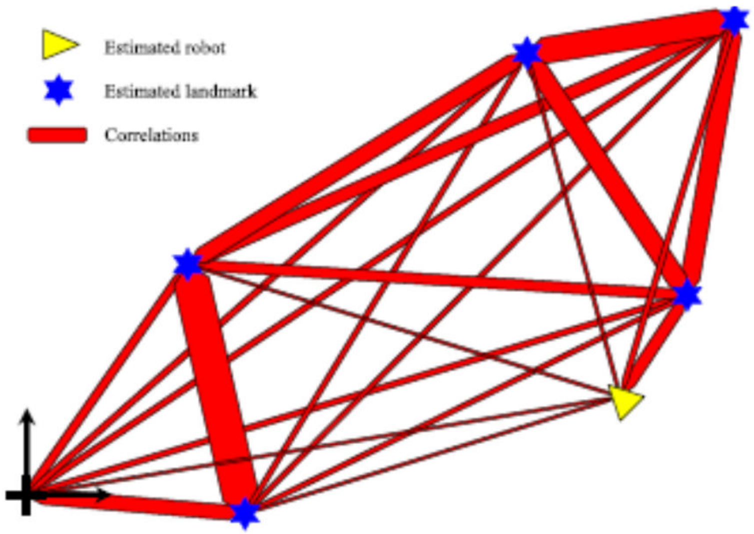
Sensor model

$$\underbrace{P(x_t, m \mid z_{0:t}, u_{0:t}, x_0)}_{\mathrm{Bel}(x_t, m)} = \eta P(z_t \mid x_t, m) \times \underbrace{P(x_t, m \mid z_{0:t-1}, u_{0:t}, x_0)}_{\overline{\mathrm{Bel}}(x_t, m)}$$

Bayesian SLAM

- Bayesian SLAM works because the error between estimated and true landmark location depends mostly on the error in the position estimate, which implies error is *correlated* between different landmarks.
- This means knowledge of the *relative* location of landmarks can only improve as more observations are made.
- As a consequence, accuracy of map and location estimates will converge, bounded only by the quality of the possible map.





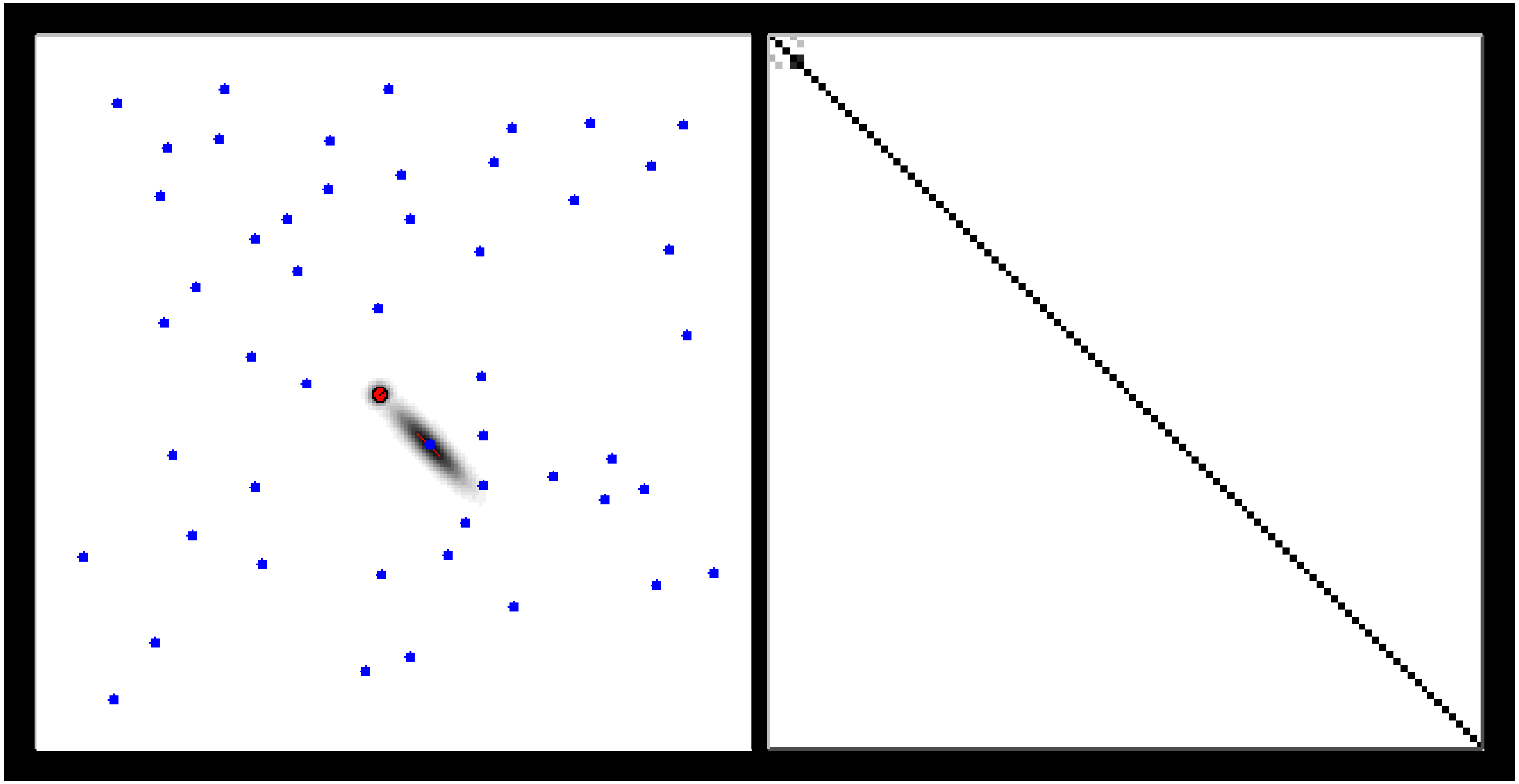
(Extended) Kalman Filter SLAM

- Basic idea is 'simply' to include the map as part of the state to be estimated, then apply methods as before
- Map with N landmarks:(3+2N)-dimensional Gaussian

$$Bel(x_t, m_t) = \left\langle \begin{pmatrix} x \\ y \\ \theta \\ l_1 \\ l_2 \\ \vdots \\ l_N \end{pmatrix}, \begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xl_1} & \sigma_{xl_2} & \cdots & \sigma_{xl_N} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\theta} & \sigma_{yl_1} & \sigma_{yl_2} & \cdots & \sigma_{yl_N} \\ \sigma_{x\theta} & \sigma_{y\theta} & \sigma_\theta^2 & \sigma_{\theta l_1} & \sigma_{\theta l_2} & \cdots & \sigma_{\theta l_N} \\ \sigma_{xl_1} & \sigma_{yl_1} & \sigma_{\theta l_1} & \sigma_{l_1}^2 & \sigma_{l_1 l_2} & \cdots & \sigma_{l_1 l_N} \\ \sigma_{xl_2} & \sigma_{yl_2} & \sigma_{\theta l_2} & \sigma_{l_1 l_2} & \sigma_{l_2}^2 & \cdots & \sigma_{l_2 l_N} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{xl_N} & \sigma_{yl_N} & \sigma_{\theta l_N} & \sigma_{l_1 l_N} & \sigma_{l_2 l_N} & \cdots & \sigma_{l_N}^2 \end{pmatrix} \right\rangle$$

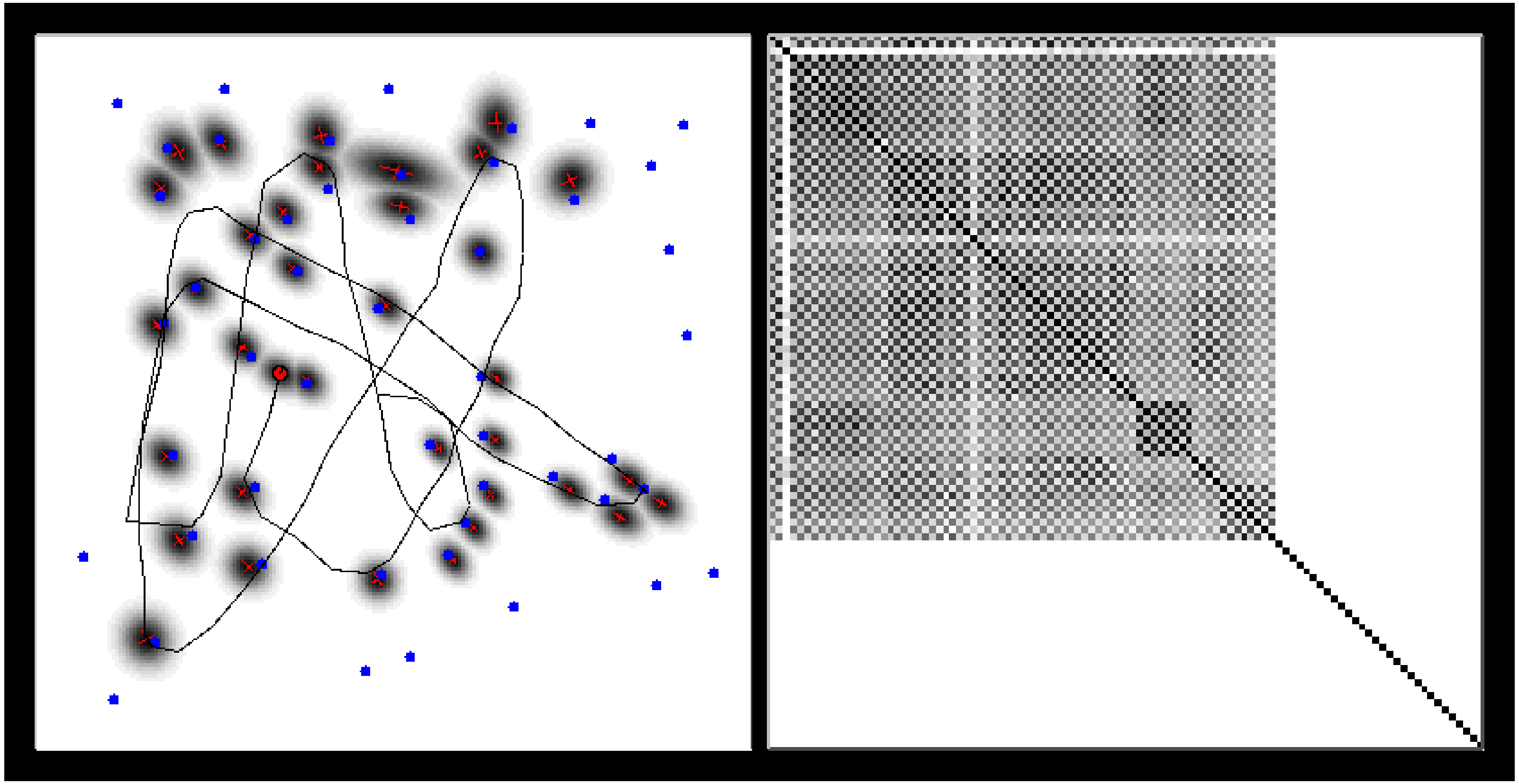
- Can handle hundreds of dimensions

EKF-SLAM



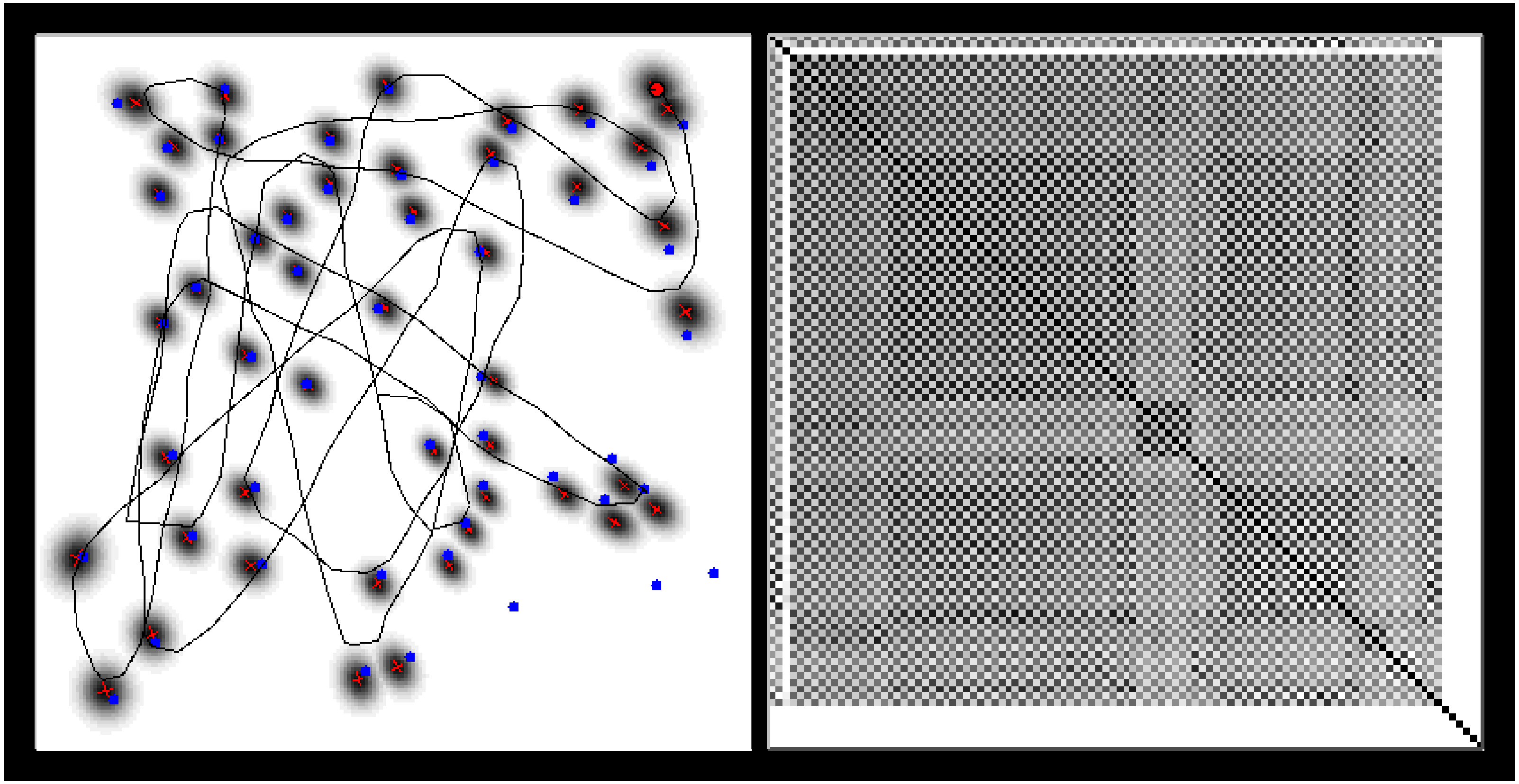
Map   Correlation matrix

EKF-SLAM

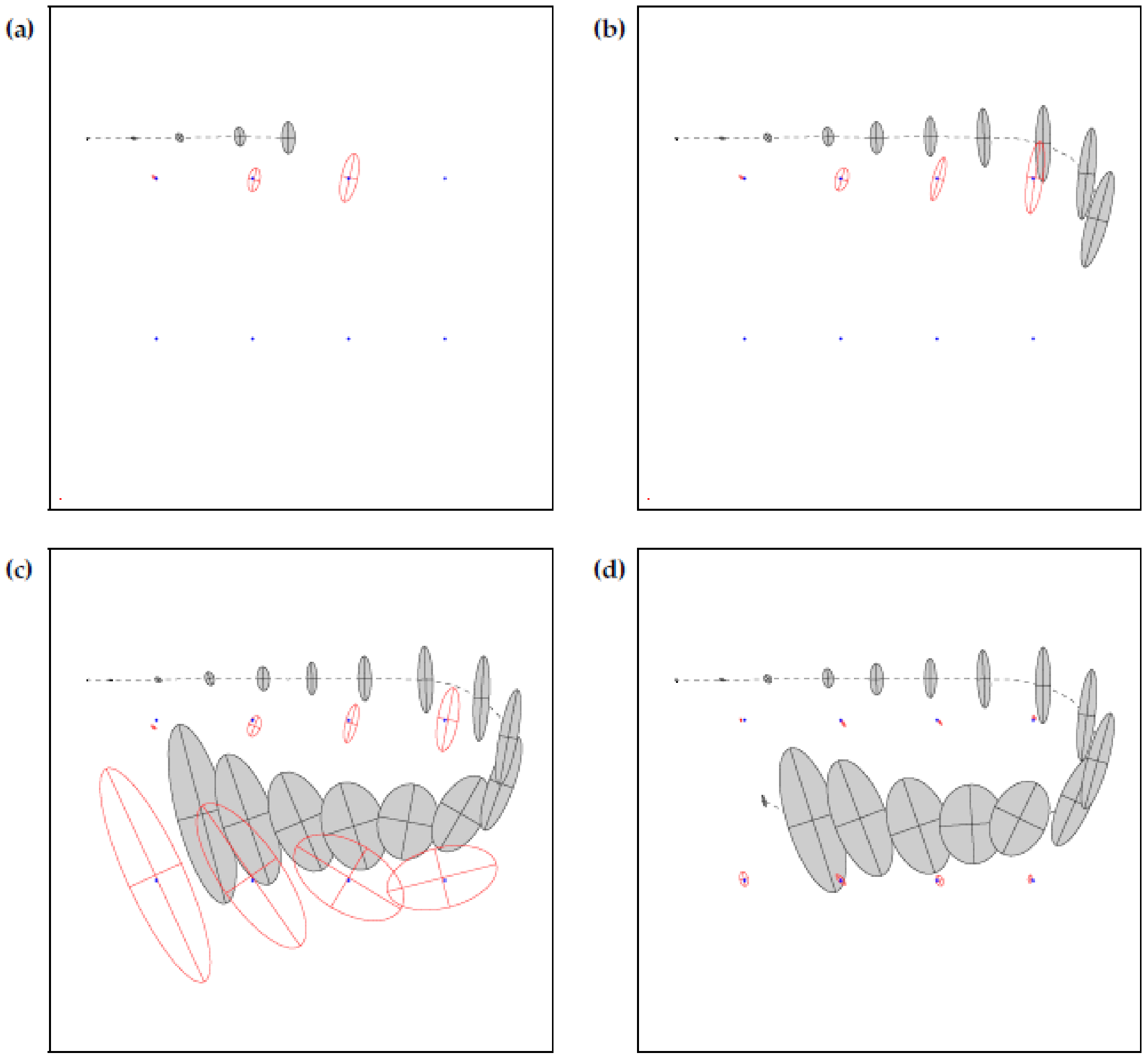


Map Correlation matrix

EKF-SLAM



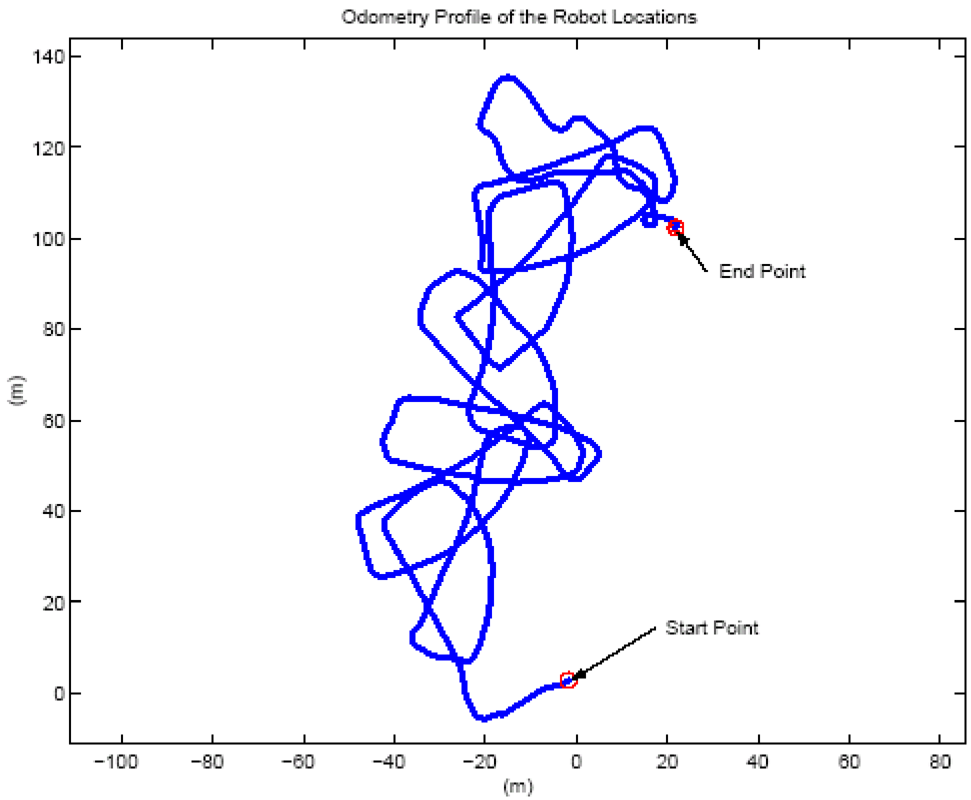
Map Correlation matrix



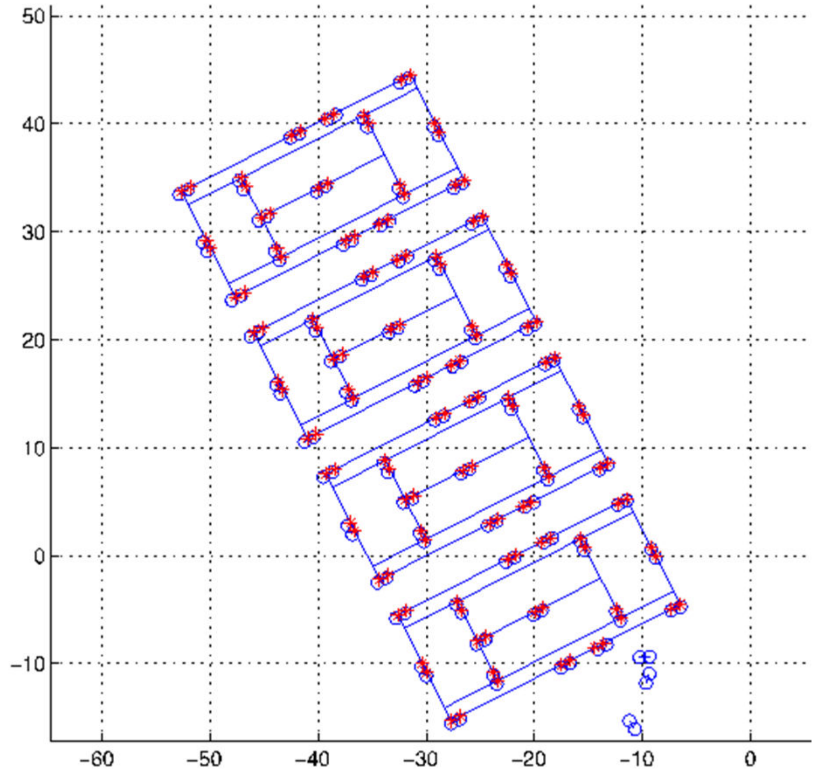
- a)-c) Pose uncertainty increases as robot moves
- Thus each successive landmark location estimate is also less certain
- But in (d) see first landmark again
- Uncertainty of all landmark locations decreases
- Pose uncertainty also decreases

EKF SLAM Application

EKF SLAM Application



odometry



estimated trajectory

Particle Filter SLAM

- SLAM: state space $< x, y, \theta, map>$
  - for landmark maps = $< l_1, l_2, \ldots, l_m>$
  - for grid maps = $< c_{11}, c_{12}, \ldots, c_{1n}, c_{21}, \ldots, c_{nm}>$
- **Problem:** The number of particles needed to represent the estimate grows exponentially with the dimension of the state space!

Solution: Factored Posterior (Landmarks)

poses map observations & movements

$$p(x_{1:t}, l_{1:m} \mid z_{1:t}, u_{0:t-1}) =$$

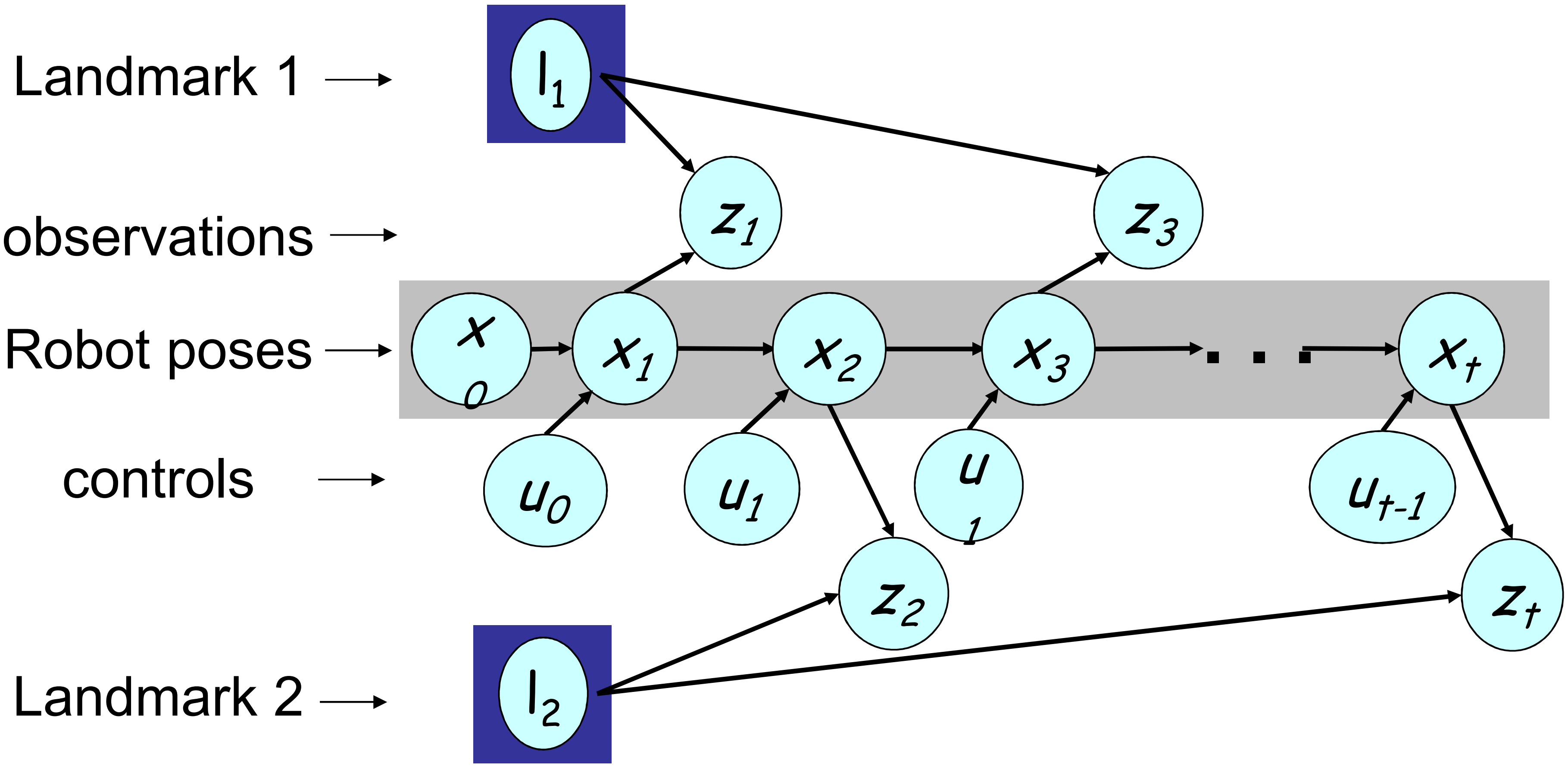
$$p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(l_{1:m} \mid x_{1:t}, z_{1:t})$$

SLAM posterior

Robot path posterior

landmark positions

Mapping using Landmarks



Knowledge of the robot's true path renders landmark positions conditionally independent

Factored Posterior

$$
\begin{aligned}
& p(x_{1:t}, l_{1:m} \mid z_{1:t}, u_{0:t-1}) \\
& \quad = \; p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(l_{1:m} \mid x_{1:t}, z_{1:t}) \\
& \quad = \; p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot \prod_{i=1}^{M} p(l_i \mid x_{1:t}, z_{1:t})
\end{aligned}
$$

Robot path posterior (localization problem)

Conditionally independent landmark positions

Rao-Blackwellization

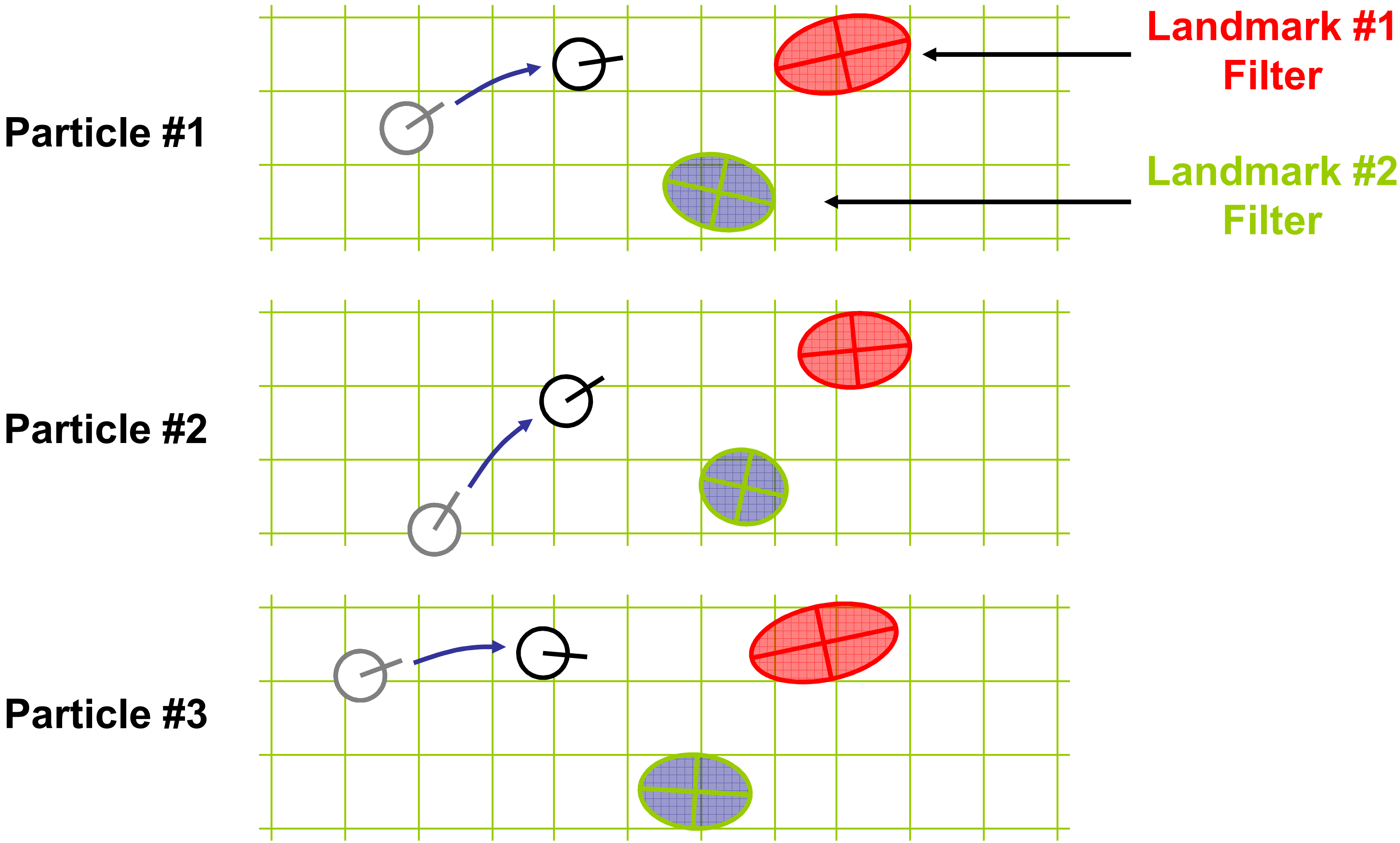
$$p(x_{1:t}, l_{1:m} \mid z_{1:t}, u_{0:t-1}) =$$
$$p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot \prod_{i=1}^{M} p(l_i \mid x_{1:t}, z_{1:t})$$

- This factorization is also called Rao-Blackwellization
- Given that the second term can be computed efficiently, particle filtering becomes possible.
- Particles represent the distribution of possible robot trajectories (the first term).

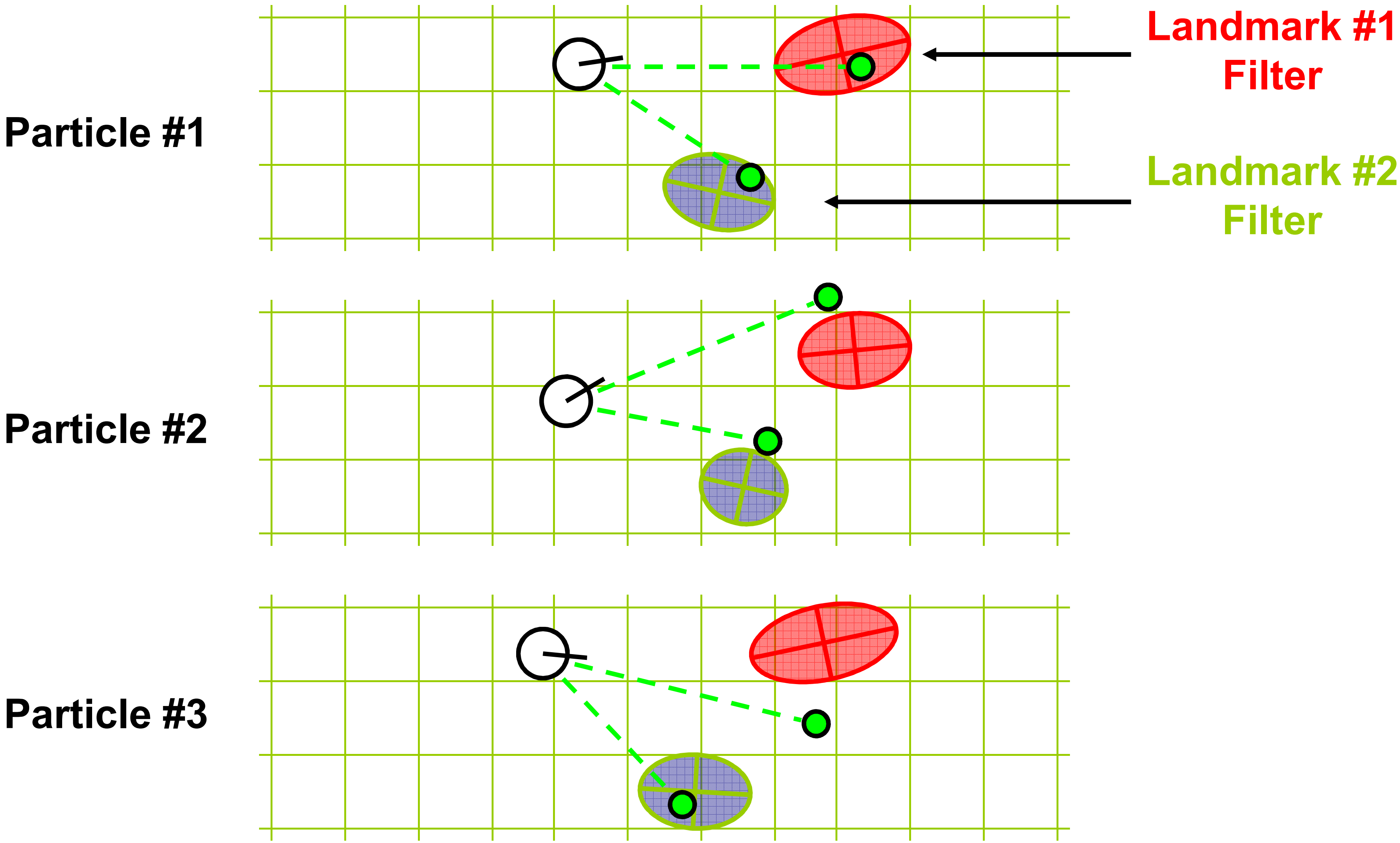
FastSLAM

- Rao-Blackwellized particle filtering based on landmarks
- Each landmark is represented by a Extended Kalman Filter (EKF)
- Each particle therefore has to maintain *M* EKFs

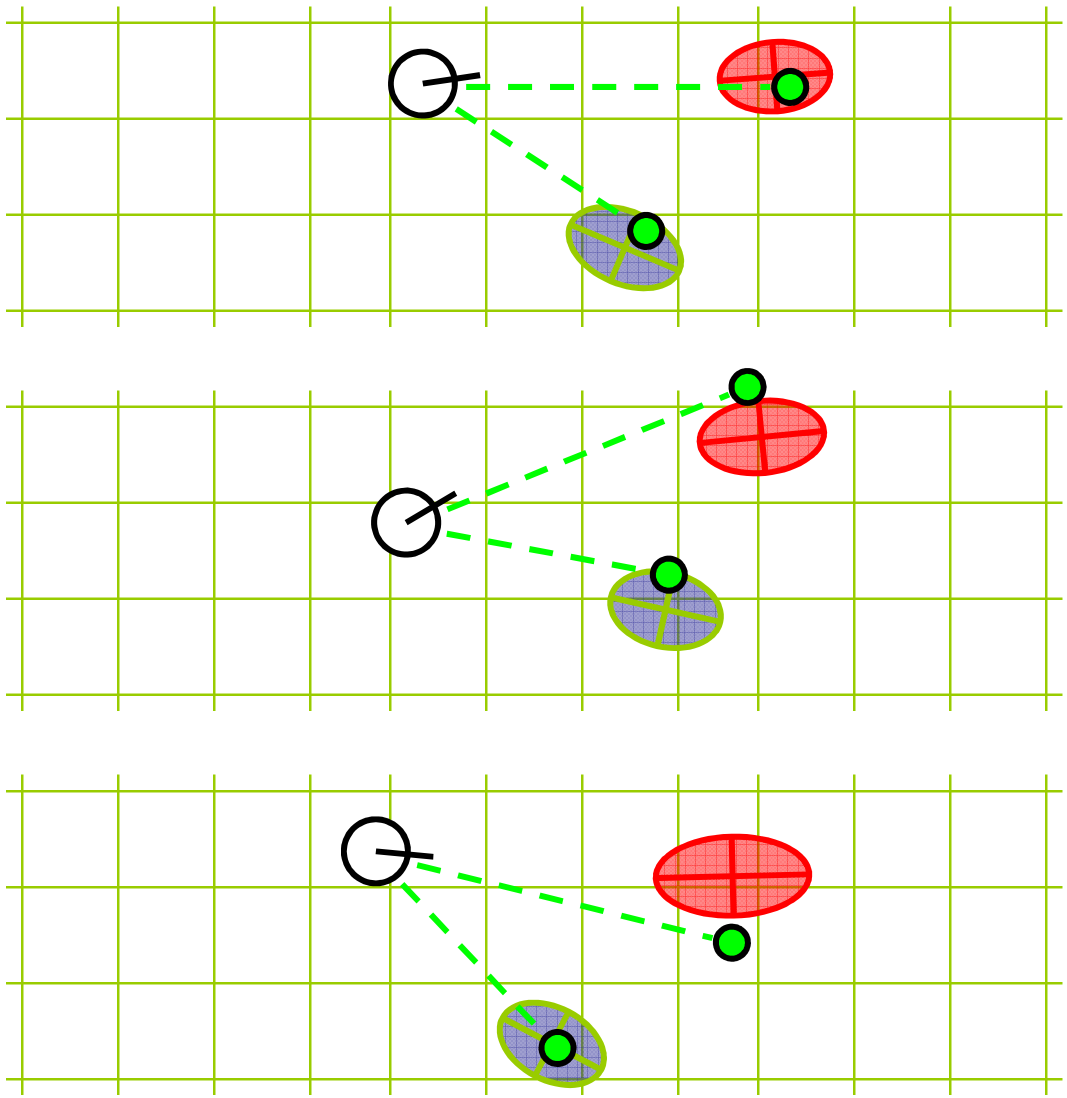
FastSLAM – Action Update



FastSLAM – Sensor Update



FastSLAM – Sensor Update



Particle #1 — **Weight = 0.8**

Particle #2 — **Weight = 0.4**

Particle #3 — **Weight = 0.1**