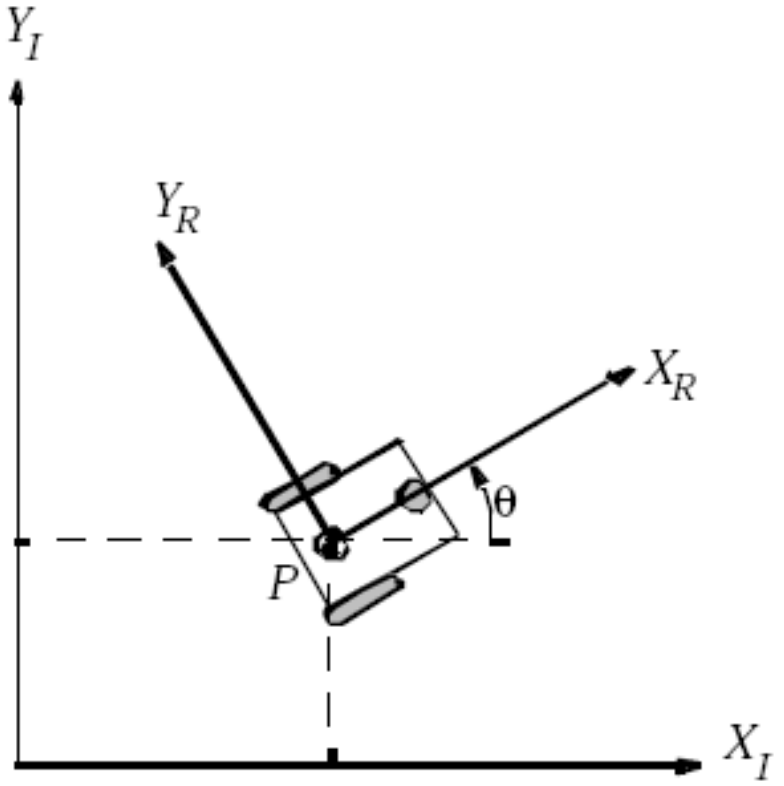
Robot pose in 2D

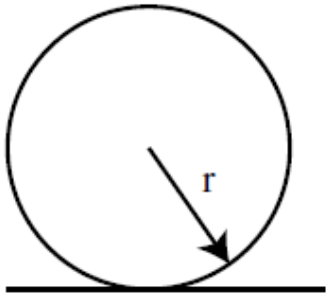
- We assume here that we deal with a rigid body robot moving on a horizontal plane.
- 3 Degrees of freedom (DOF)
  - X
  - Y
  - Θ
- Robot pose = position + orientation

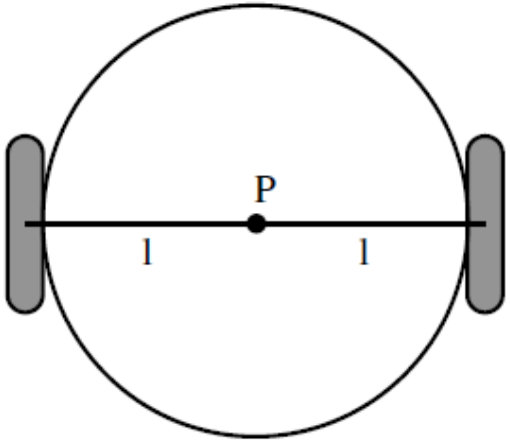


Differential drive kinematics

- Wheel radius r (assume same for both wheels)
- Distance between the wheel B = 2l
- Wheel rotation angles φ_L and φ_R

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \dot{\varphi}_R, \dot{\varphi}_L)$$





Differential drive kinematics cont'd

- Translation speed of each wheel given by r*(dφ/dt)
- Translation and rotation speed of the robot as a function of the wheel speeds are given by

$$v = \frac{r}{2}(\dot{\varphi}_R + \dot{\varphi}_L)$$

$$\omega = \frac{r}{2l}(\dot{\varphi}_R - \dot{\varphi}_L)$$

Odometry

- By attaching encoders on the wheels their rotation can be measured
- The so called **odometry** is given by integrating encoder measurements over time based on the kinematic model
- The better the kinematic model, the better the odometry

Kinematics

- From before

  $v = r/2(d\varphi_R/dt + d\varphi_L/dt)$

  $\omega = r/B(d\varphi_R/dt - d\varphi_L/dt)$

- Let ΔE_R and ΔE_L denote the difference in encoder reading over a period of time ΔT

- If K is the scaling factor to transform encoder difference to angle, $\Delta\varphi = K\Delta E$

- $v = r/2(K\Delta E_R + K\Delta E_L)/\Delta T \rightarrow D = v\Delta T = r/2(K\Delta E_R + K\Delta E_L)$
- $\omega = r/B(K\Delta E_R - K\Delta E_L)/\Delta T \quad \rightarrow \Delta\theta = \omega\Delta T = r/B(K\Delta E_R - K\Delta E_L)$

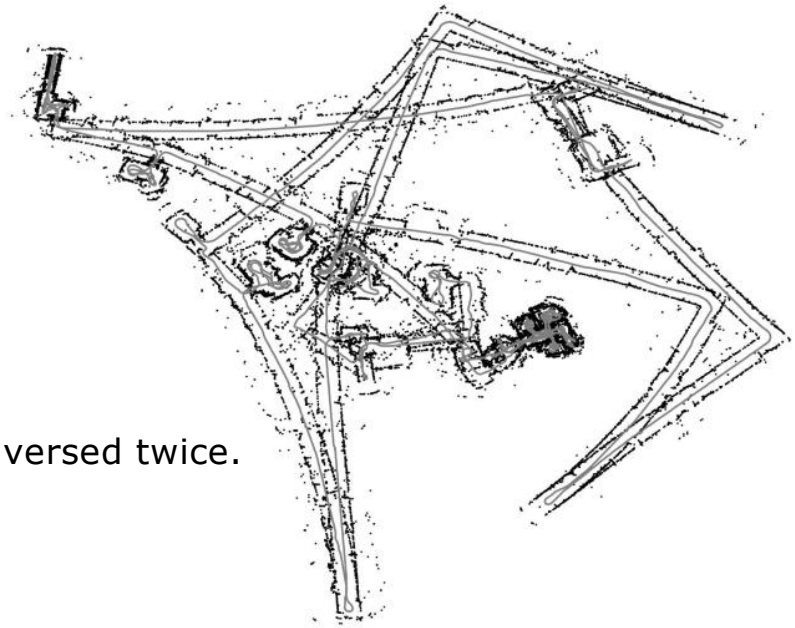
Odometry

- Odometry
  - $x(k+1) = x(k) + v*dt*\cos(\theta) = x(k) + D*\cos(\theta)$
  - $y(k+1) = y(k) + v*dt*\sin(\theta) = y(k) + D*\sin(\theta)$
  - $\theta(k+1) = \theta(k) + \omega*dt = \theta(k) + \Delta\theta$

  $pose(k+1) = f_1(pose(k),v,\omega,dt) = f_2(pose(k),D,\Delta\theta)$

Odometry

- Typically very accurate over short distances
- Will drift
- Error in dead-reckoning unbounded
- Angular error → large position errors



The path in the image should be a a U-shape with right angles, traversed twice.

Uncertainty in odometry

- Odometry with noise
  - $x(k+1) = x(k) + (v*dt + \vartheta_D)*\cos(\theta)$
  - $y(k+1) = y(k) + (v*dt + \vartheta_D) *\sin(\theta)$
  - $\theta(k+1) = \theta(k) + (\omega*dt + \vartheta_{\theta,\omega}) + \vartheta_{\theta,v}$

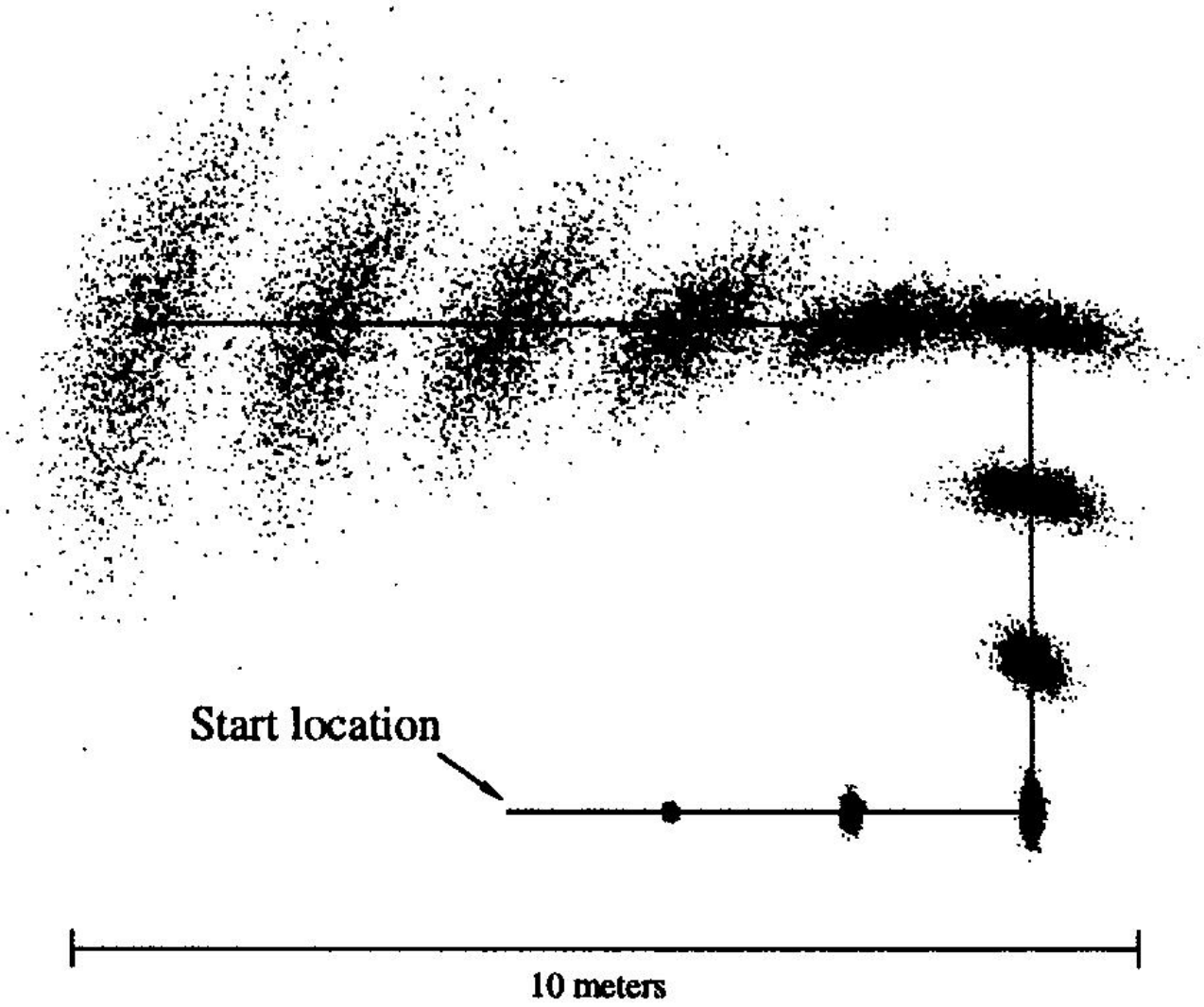
  - Where ϑ_D, $\vartheta_{\theta,v}$ and $\vartheta_{\theta,\omega}$ are typically assumed to be zero-mean Gaussian i.e. $N(0,\sigma^2)$.
  - σ^2 is typically modeled as depending on the (rotation)speed.
    - $\vartheta_D = N(0, (v*dt*k_D)^2)$
    - $\vartheta_{\theta,v} = N(0, (v*dt*k_v)^2)$
    - $\vartheta_{\theta,\omega} = N(0, (\omega*dt*k_\omega)^2)$

  - You could model the noise in many ways! For example as three independent noise sources, directly on x,y,θ

Monte Carlo simulation

- Simulate N paths where each path is effected by a different realization of the noise

Motion increases uncertainty



Sources of odometry errors

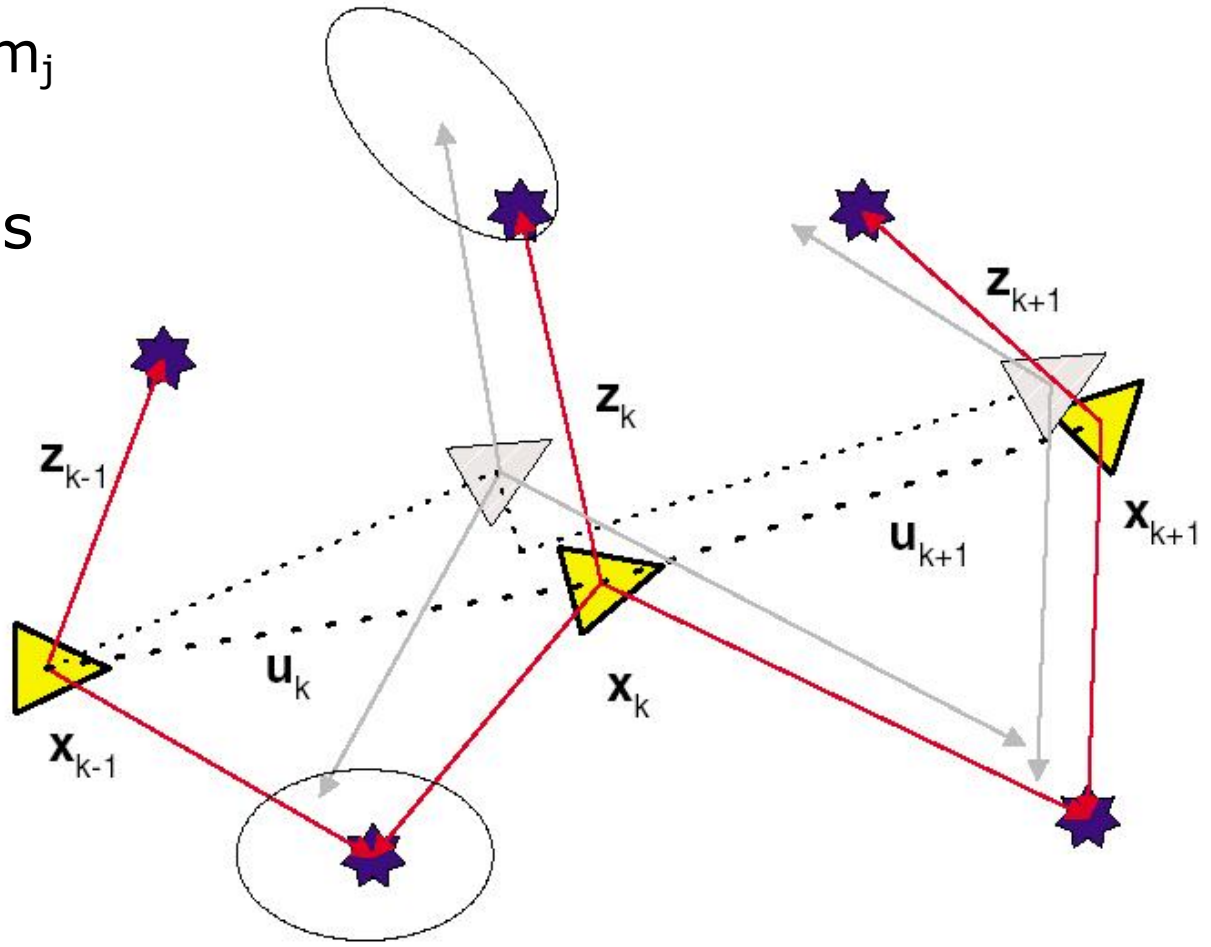
- Systematic errors (some of them)
  - Wrong size of wheels in model
  - Misalignment of the wheels
  - Limited encoder resolution and sampling rate
  - . . .
- Non-systematic errors
  - Uneven surface → wheels travel longer/shorter
  - Objects on the floor → large disturbance
  - Wheel slippage (slippery floors, over-acceleration, fast-turning,
  - external forces, non-point wheel contact with floor
  - . . .

Some other ways to get dead reckoning

- IMU
  - Integrate gyro and accelerometer data to get the pose
  - Typically only reliable over very short periods of time
- Visual odometry
  - Integrate images to estimate motion between camera positions
  - Amazing performance if done well
- Scan matching
  - Estimate motion by comparing laser scans acquired from different positions
  - Very reliable

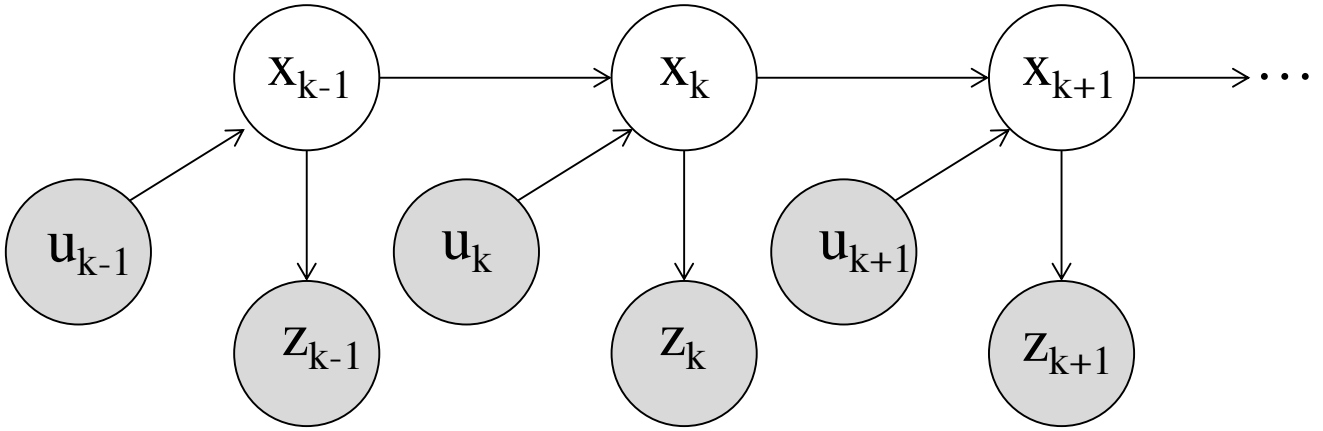
Localization problem

- Components
  - Robot position at time k, x_k
  - Measurements z_k
  - Map M, with landmarks m_j
- Map / landmark positions known
- Estimate robot pose



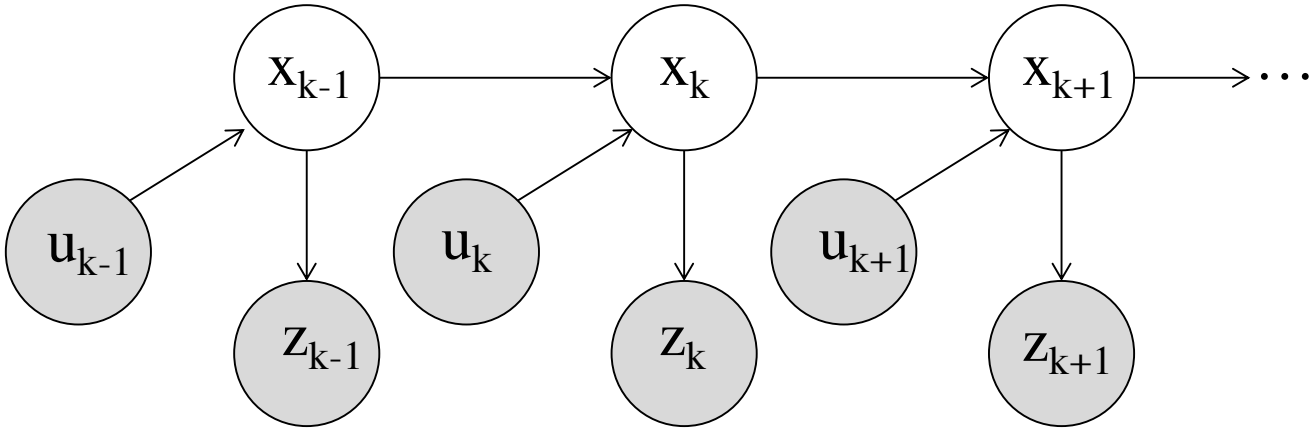
Formalizing the localization problem

- Variables in the Bayesian network below
  - x is the pose of the robot (the state)
  - z is an observation
  - u is a control input (odometry based on encoders or motion cmd)
- Grey variables are known by the robot, white unknown
- What assumptions does the Bayesian network below encode?



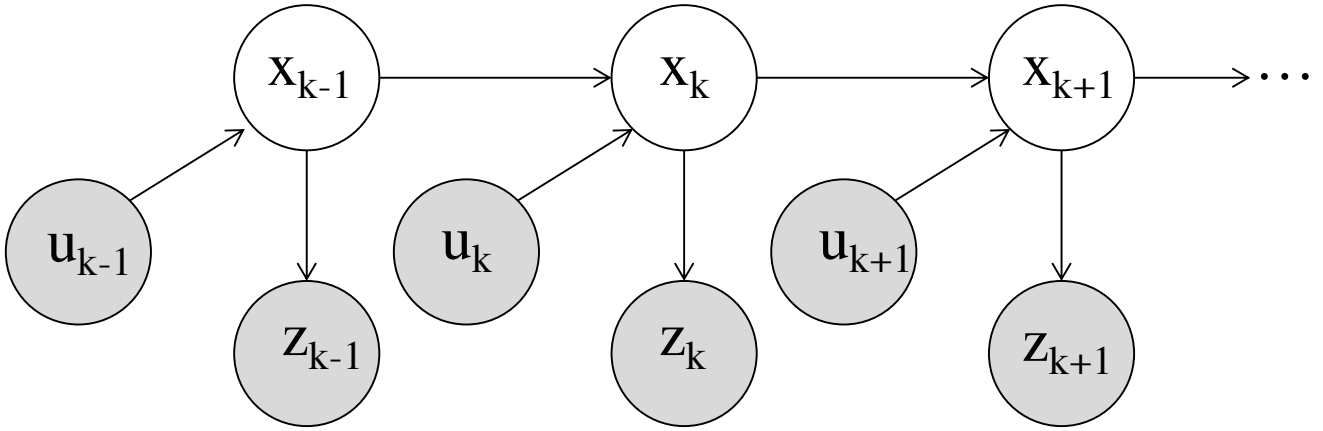
What can we read out from the network?

- Relation between variables
  - $p(z_k|x_k)$ {observation depends only on the state}
  - $p(x_{k+1}|x_k,u_{k+1})$ {Markov assumption, state at time k depends only on the previous state (and the control input), not states old than that}



What does this mean?

- To determine the pose at time k+1, the pose at k contains all information from past states
- Measurement at time k depends on the pose at that time and nothing else
- The pose at time k+1 depends on the pose at time k and the control input.



The localization problem

- Localization: Determine $p(x_k|Z_k,U_k)$
  where $Z_k=\{z_1,z_2,\ldots,z_k\}$ and $U_k=\{u_1,u_2,\ldots,u_k\}$

- Remember Bayes' rule

$$p(A|B,C) = p(B|A,C)\, p(A|C) / p(B|C)$$

introduce normalization factor η

$$p(A|B,C) = \eta\, p(B|A,C)\, p(A|C)$$

The localization problem

- Localization: Determine $p(x_k|Z_k,U_k)$
- Calculate it recursively

$p(x_{k+1}|Z_{k+1},U_{k+1})$

$Z_{k+1} = \{z_{k+1},U_k\}$

$$= p(x_{k+1}|z_{k+1},Z_k, U_{k+1})$$
$$= \{\text{Bayes with } A=x_{k+1},\ B=z_{k+1},\ C=Z_k,\ U_{k+1}\}$$
$$= \eta\ p(z_{k+1}|x_{k+1},Z_k, U_{k+1})p(x_{k+1}|Z_k,U_{k+1})$$
$$= \eta\ p(z_{k+1}|x_{k+1})p(x_{k+1}|Z_k,U_{k+1})$$

$p(x_{k+1}|Z_k,U_{k+1}) = \{\text{Use sum rule}\}$

$U_{k+1} = \{U_k, u_{k+1}\}$

$$= \int p(x_{k+1}|Z_k,U_{k+1},x_k)\ p(x_k|Z_k,U_{k+1})\ dx_k$$
$$= \{x_{k+1} \text{ indep of } Z_k \text{ and } U_k \text{ given } x_k,\ x_k \text{ indep of } u_{k+1}\}$$
$$= \int p(x_{k+1}|x_k,u_{k+1})\ p(x_k|Z_k,U_k)\ dx_k$$

The localization problem

Prediction based on control input u_k:

$$p(x_{k+1}|Z_k,U_{k+1}) = \int p(x_{k+1}|u_{k+1},x_k)\, p(x_k|Z_k,U_k)\, dx_k$$

where $p(x_{k+1}|u_k,x_k)$ is the motion model often given by odometry.

Update with new measurement z_{k+1}:

$$p(x_{k+1}|Z_{k+1},U_{k+1}) = \eta\, p(z_{k+1}|x_{k+1})p(x_{k+1}|Z_k,U_{k+1})$$

where $p(z_{k+1}|x_{k+1})$ is the measurement model

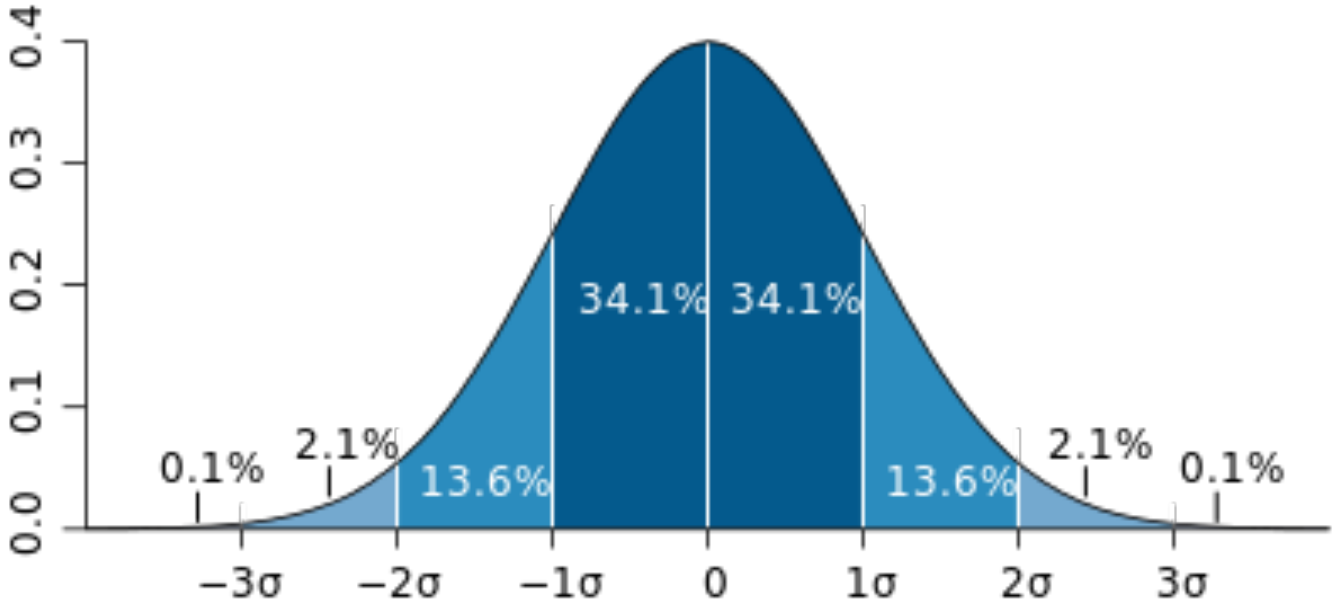
Representing probabilistic information

- What we described so far is general.
- To implement this we need to determine how to represent the distributions, such as $p(x_k|Z_k,U_k)$

- Two most common choices
  - Gaussian distribution
  - Set of particles (sampled based representation)

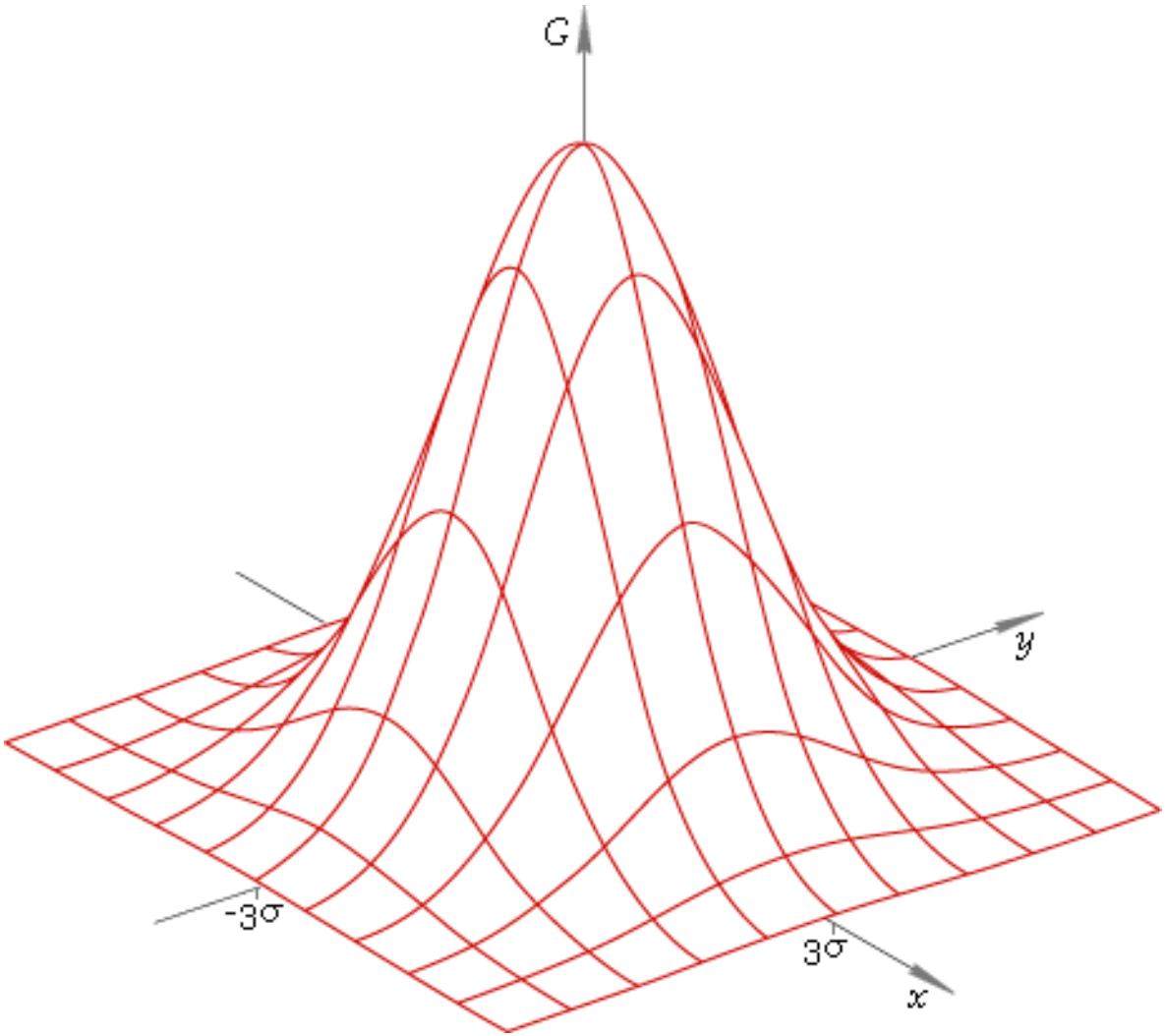
Gaussian/Normal Distribution

- Defined by
  - Mean (symmetric around the mean)
  - Variance (covariance for multidimensional data)



https://en.wikipedia.org/wiki/Standard_deviation

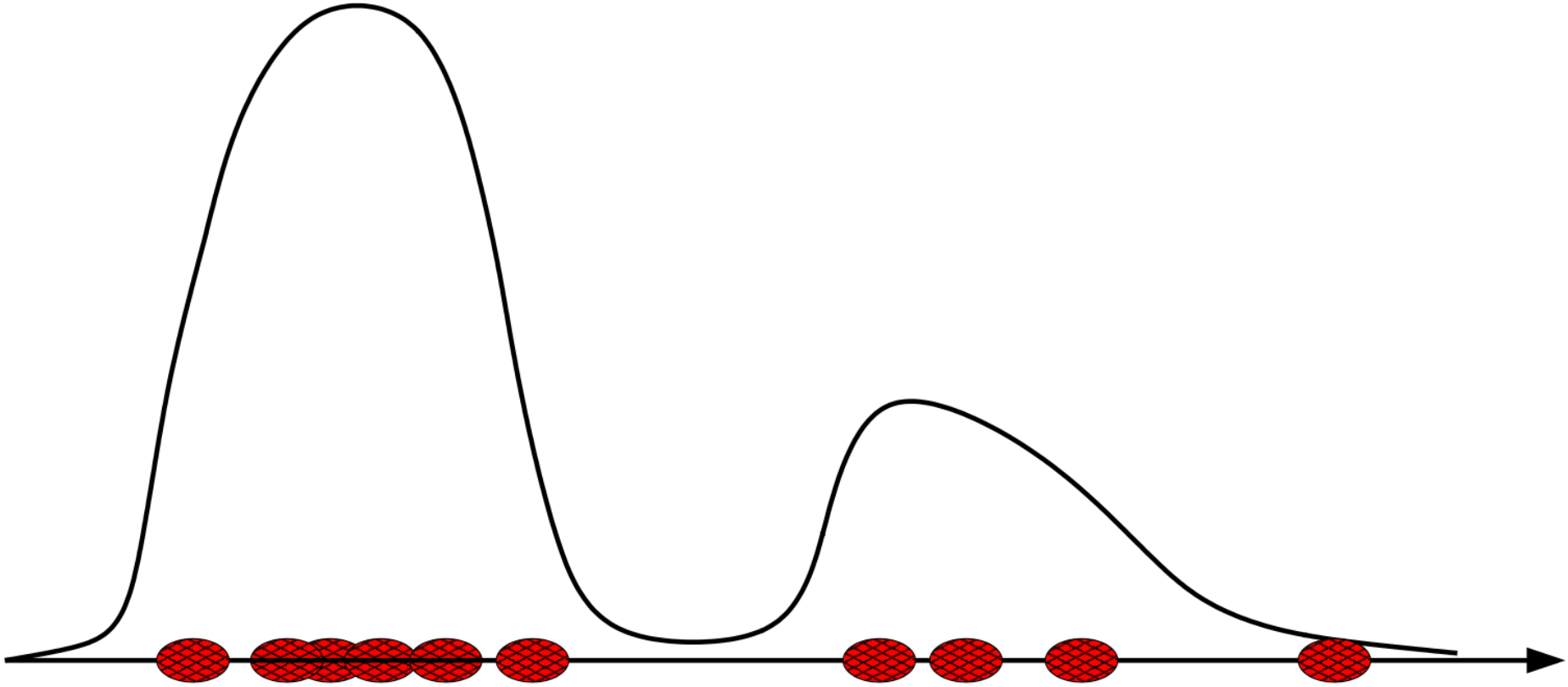
Multidimensional Gaussian



http://www.librow.com/content/common/images/articles/article-9/2d_distribution.gif

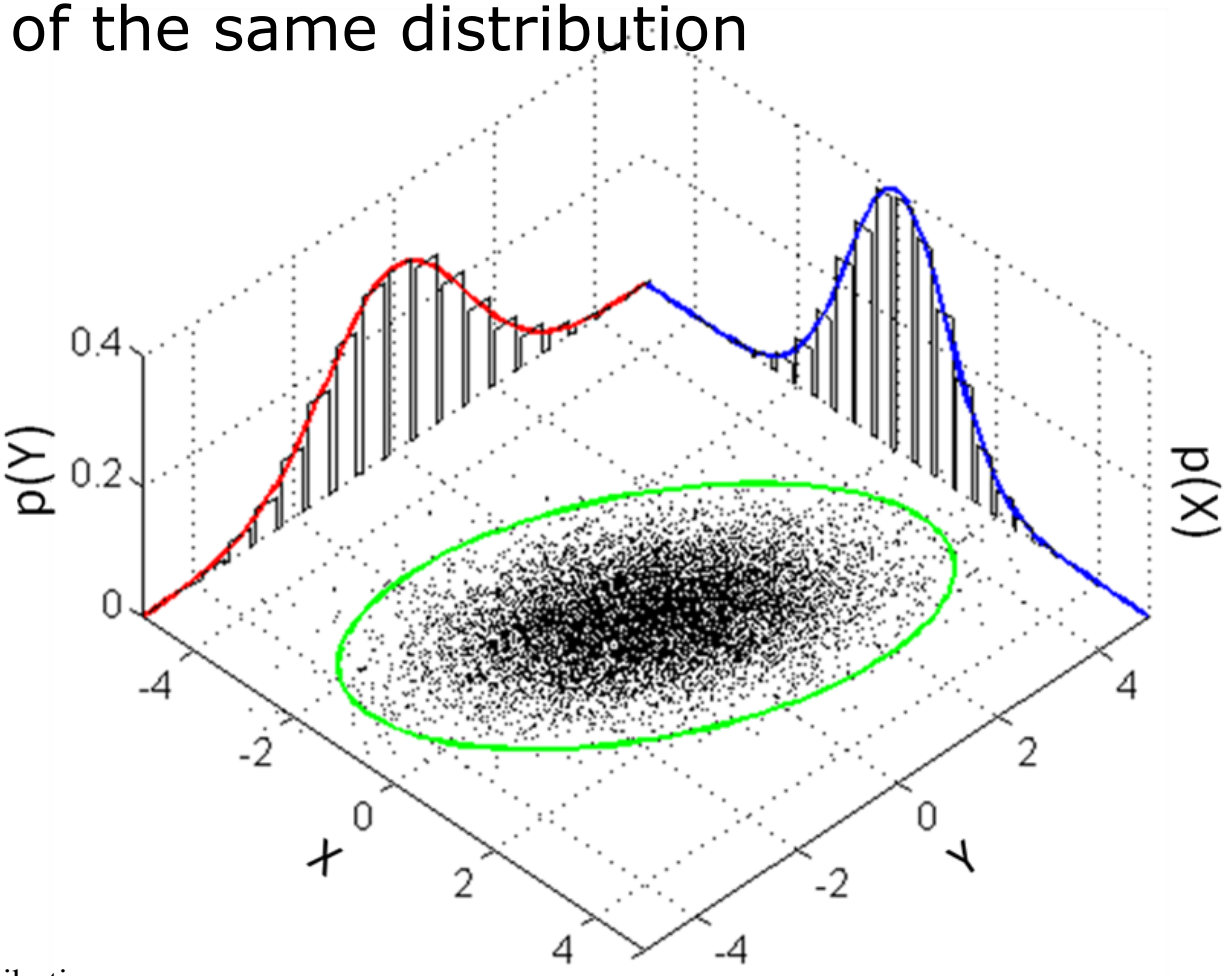
Sample based representations

- Can also sample the distribution
- A set of “particles” approximate the distribution
- Each particle represents a possible state the system can be in
- The more particles the better the approximation



Gauss vs particle set

- Green ellipse: 2D Gaussian
- Black dots: Samples of the same distribution



https://en.wikipedia.org/wiki/Multivariate_normal_distribution

Algorithms for localization

- The two most frequently used algorithms
  - Particle filter ←→ Sample based representation
  - Kalman filter ←→ Gaussian

Particle filter

The particle filter represents probability distributions using a set of particles, p_k, sampled from the distribution.

Each particle represents one "hypothesis" about the state.

Each particle also has a weight, initialized as $\pi=1/N$.

$p_k=\{x_k, \pi_k\}$

particle state weight

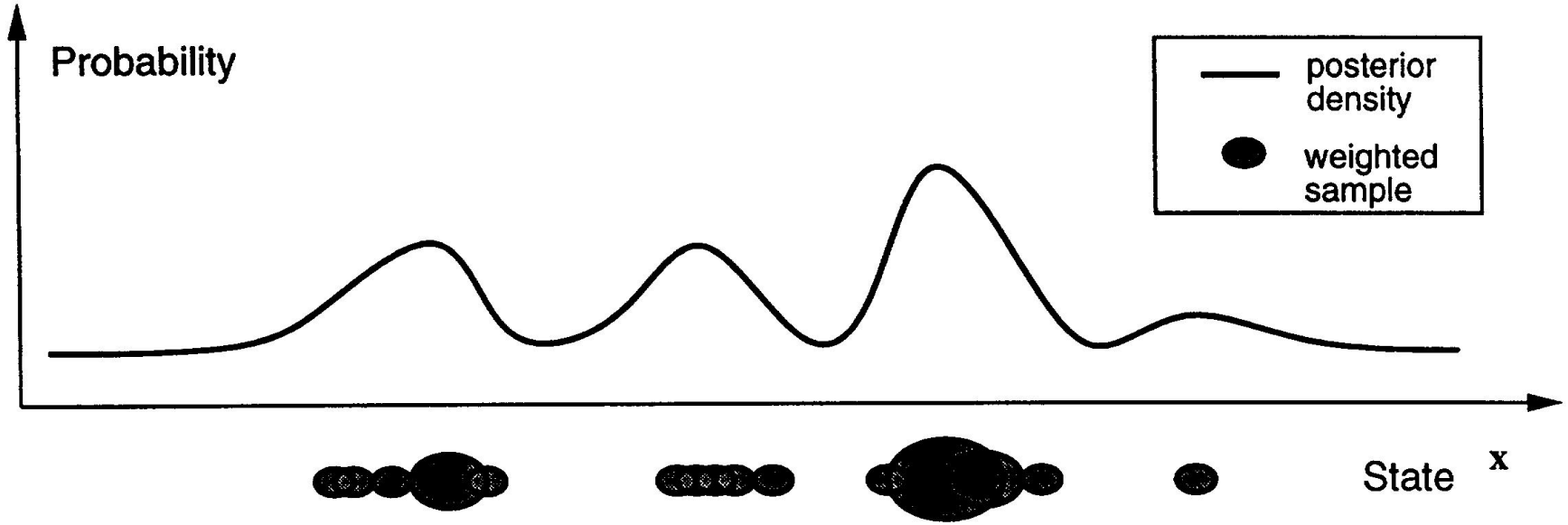
Particle filter

$p_k = \{x_k, \pi_k\}$

particle state weight

The weight allows us to use fewer particles.

For example, can replace 5 particles by one with 5 times the weight



Particle filter motion prediction

Let us consider the case where we have no measurements, that is, pure prediction. The filter equation is given by

$$p(x_{k+1}|Z_k,U_{k+1}) = \eta \int p(x_{k+1}|x_k,u_{k+1})\ p(x_k|Z_k,U_k)\ dx_k$$

The particle filter approximates this by, for each particle, predict the new state using the motion model $p(x_{k+1}|x_k,u_{k+1})$.

Localization using particle filter is also called Monte Carlo Localization (MCL)

Particle filter motion prediction

- Each particle represents a pose hypothesis $x_k=x(k)=(x,y,\theta)_k$
- $p(x_{k+1}|x_k,u_{k+1})$ is given by our motion model
  - $x(k+1) = x(k) + (v*dt + \vartheta_D)*\cos(\theta(k))$
  - $y(k+1) = y(k) + (v*dt + \vartheta_D) *\sin(\theta(k))$
  - $\theta(k+1) = \theta(k) + (\omega*dt + \vartheta_{\theta,\omega}) + \vartheta_{\theta,v}$

Use measurements

- Predicting the pose will make the particles spread, i.e., the uncertainty increase.

- How to decrease the uncertainty?
  - Use IMU, calibrate odometry, etc → reduce rate of increase, but still unbounded error
  - Use exteroceptive sensors to bound the error

Measurements in particle filter

Measurement update

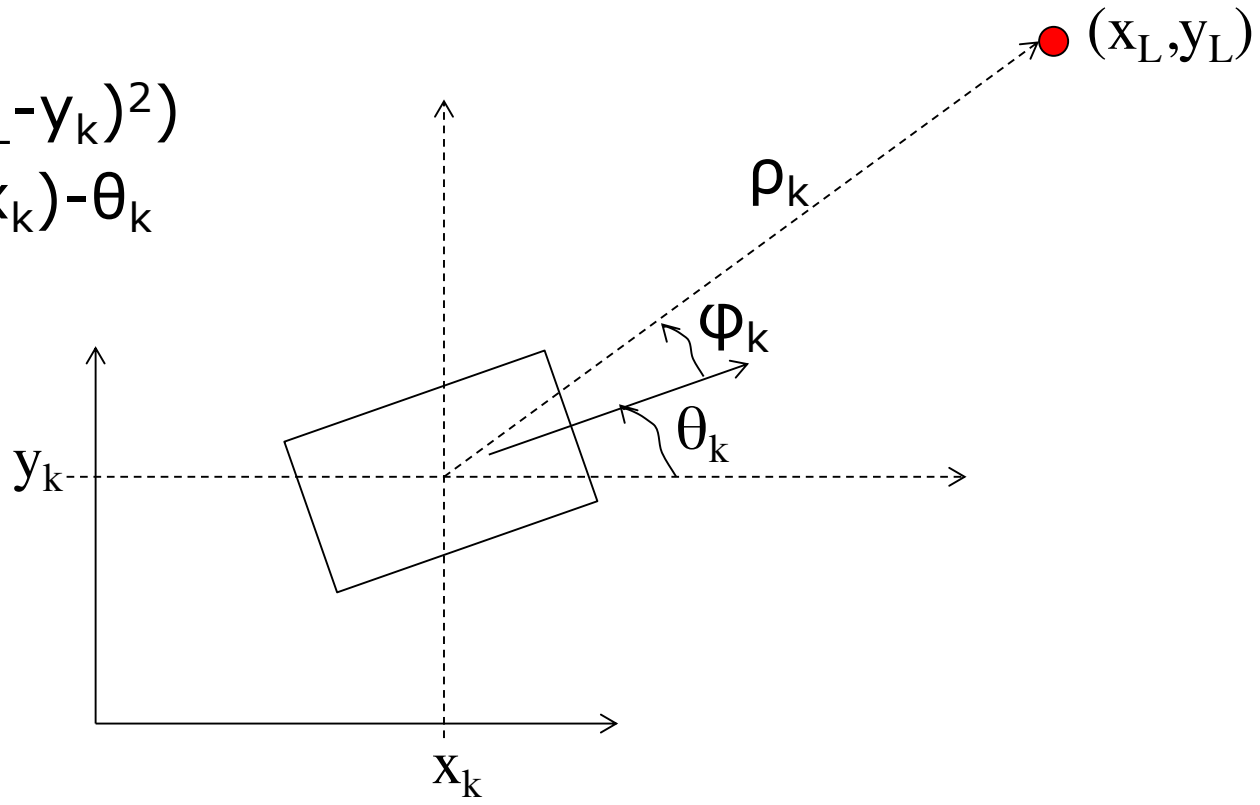
$p(x_{k+1}|Z_{k+1},U_{k+1}) = \eta\ p(z_{k+1}|x_{k+1})p(x_{k+1}|Z_k,U_{k+1})$

The particle filter approximates this by, for each particle, multiply the weight by the measurement likelihood given by the sensor model, $p(z_{k+1}|x_{k+1})$.

Example of measurement model

- Assume that we have some landmark in the environment
- We know their locations
- We can measure range and bearing to them

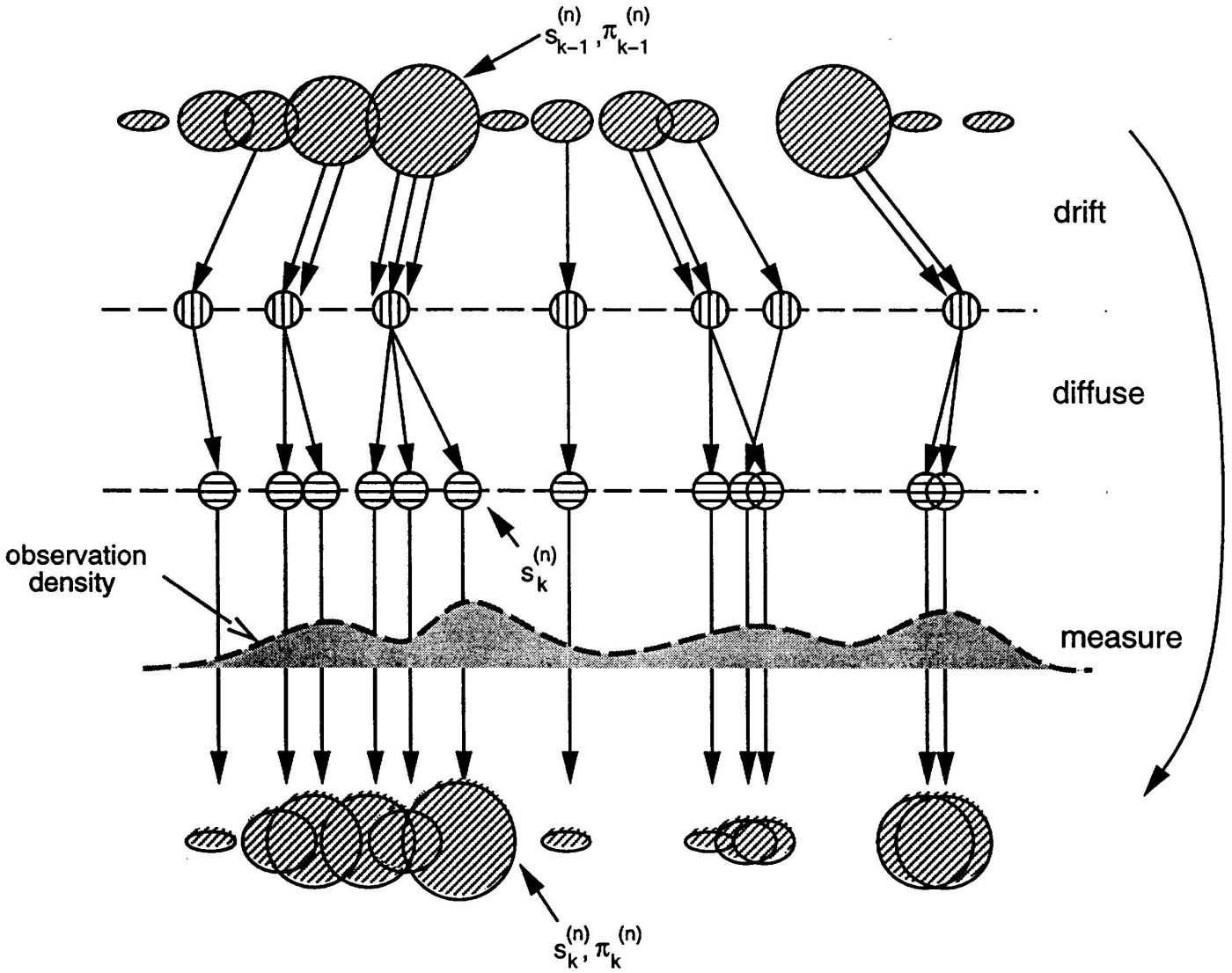
- $\rho_k = \text{sqrt}((x_L - x_k)^2 + (y_L - y_k)^2)$
- $\varphi_k = \text{atan2}(y_L - y_k, x_L - x_k) - \theta_k$



Resampling

- As the particles spread, fewer and fewer of the particles are in regions where $p(x_k|Z_k,U_k)$ is high.
- The approximation of the true distribution becomes bad!
- Solution? Importance resampling!
- How?
  - Create a new particle set.
  - Probability to copy a particle from the old set is proportional to the weight. Can have multiple copies.
  - Set weight to 1/N again
  - High weights results in many copies
  - Resources better spent

The particle filter



Algorithm

- 0. Initialize the particles given what you know to start with (nothing→uniform, a lot→ very small spread) and with weight 1/N.
- 1. Use odometry to update all poses of particles and perturb each particle according to odomety noise (different for all).
- 2. Use measurements and multiply the weight of each particle, i, with $p(z_k|x^i_k)$
- 3. Re-sample "if needed" and then return to 1.

Some practical considerations

- Resampling too often can result in particle depletion as a result of overfitting to the latest measurements.
  - Particles gather in one or a few places which may not even be the true position if there is noise, outliers, etc
- One strategy is to wait to resample until the effective number of particles, n_{eff}, is lower than some threshold, typically N/2

$$n_{eff} = \frac{1}{\sum_i \left(w_t^{(i)} \right)^2}$$

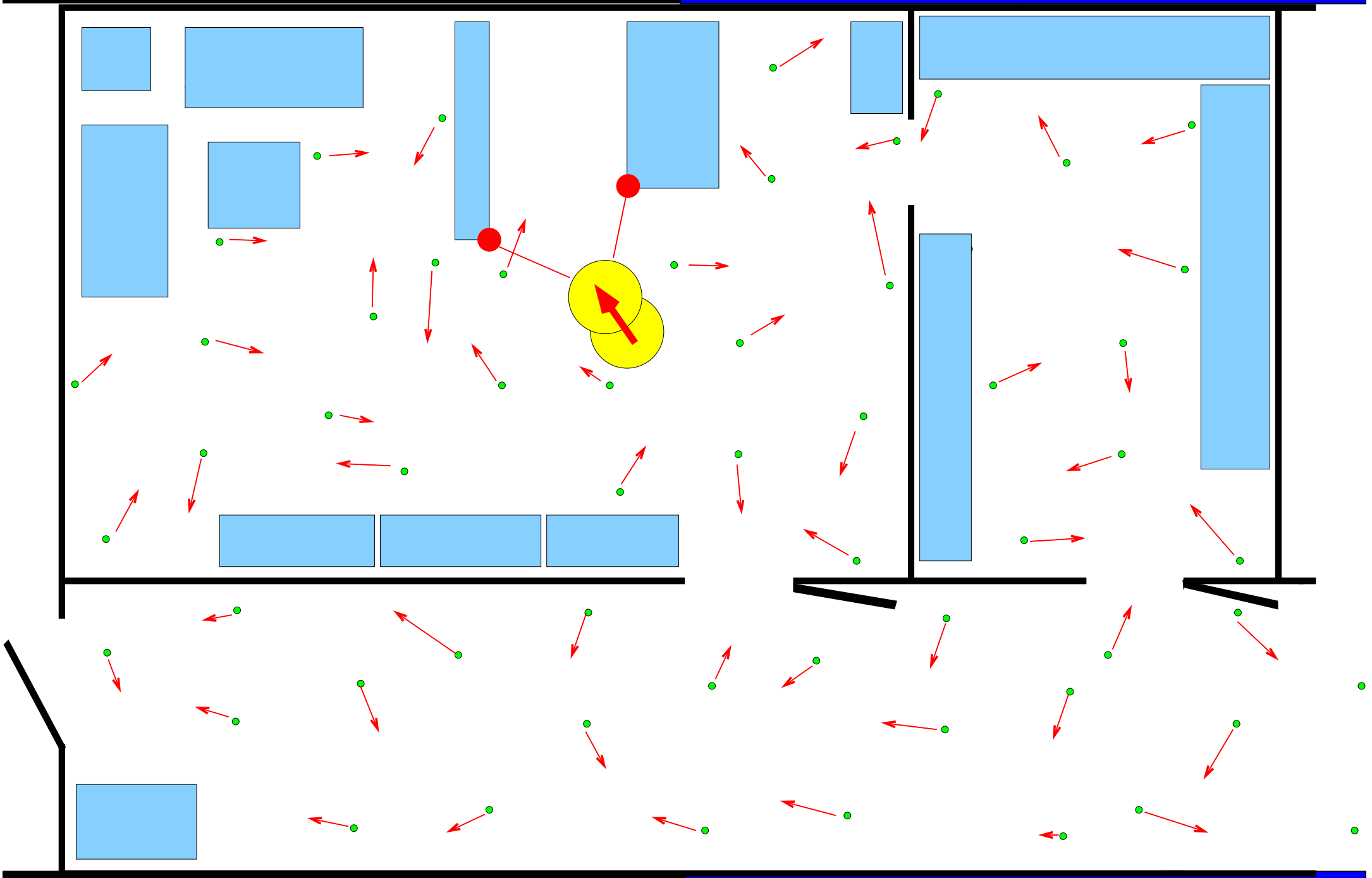
Some practical considerations

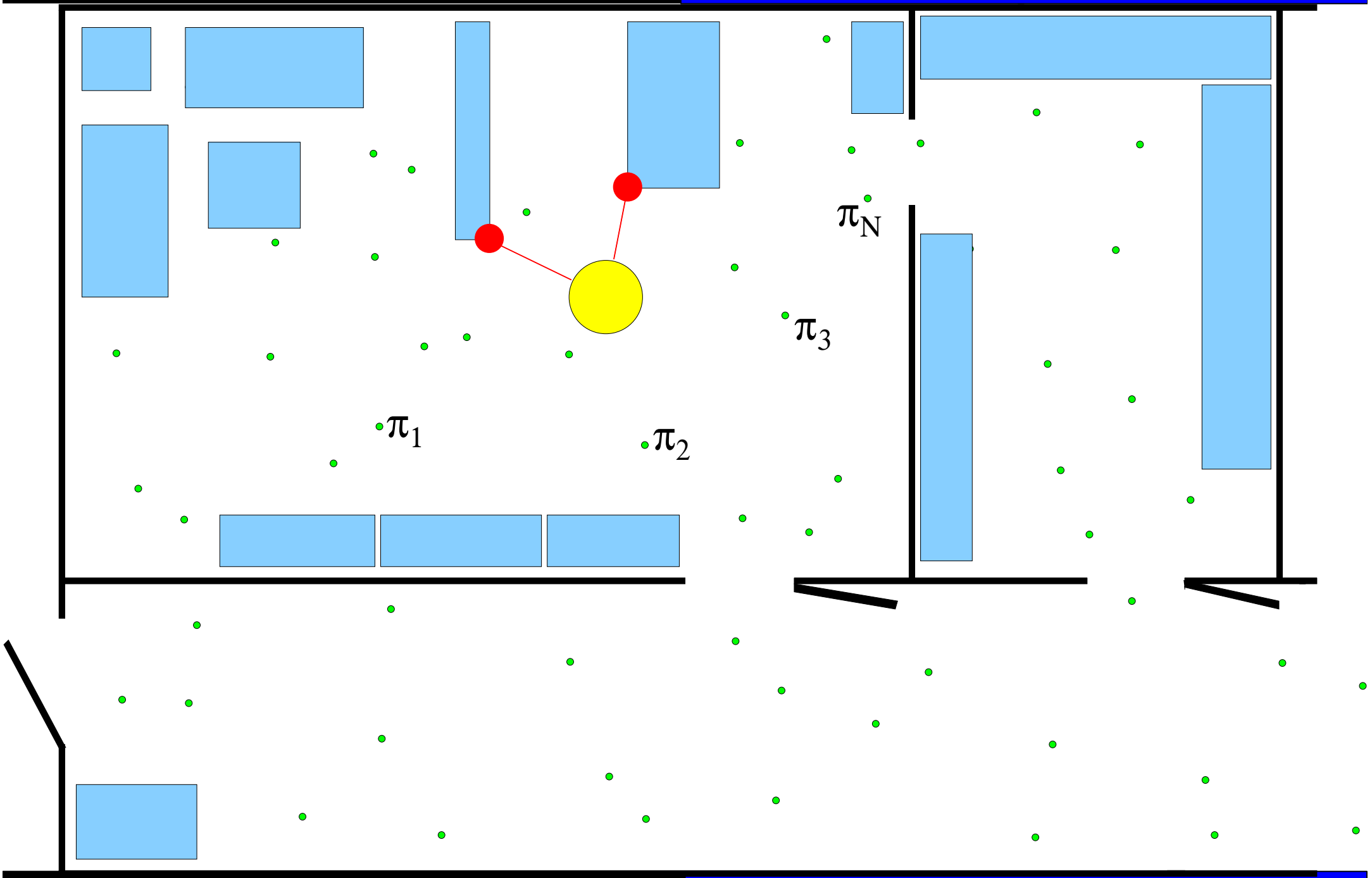
- We have a set of particles, each giving one hypothesis of the position of the robot. So, where is the robot?
- Need to calculate that from the particles.
  - Can calculate the weighted average of the particle positions
    - Remember that you need to be careful with the orientation, the average of 1° and 359° is 0° not 180°
    - Pay attention to multi-modal cases (several peaks)
- Remember that your estimate of the position will be old once you have it. The slower your filter runs, the older it will be. Might need to take that into account when you do control.

Some practical considerations

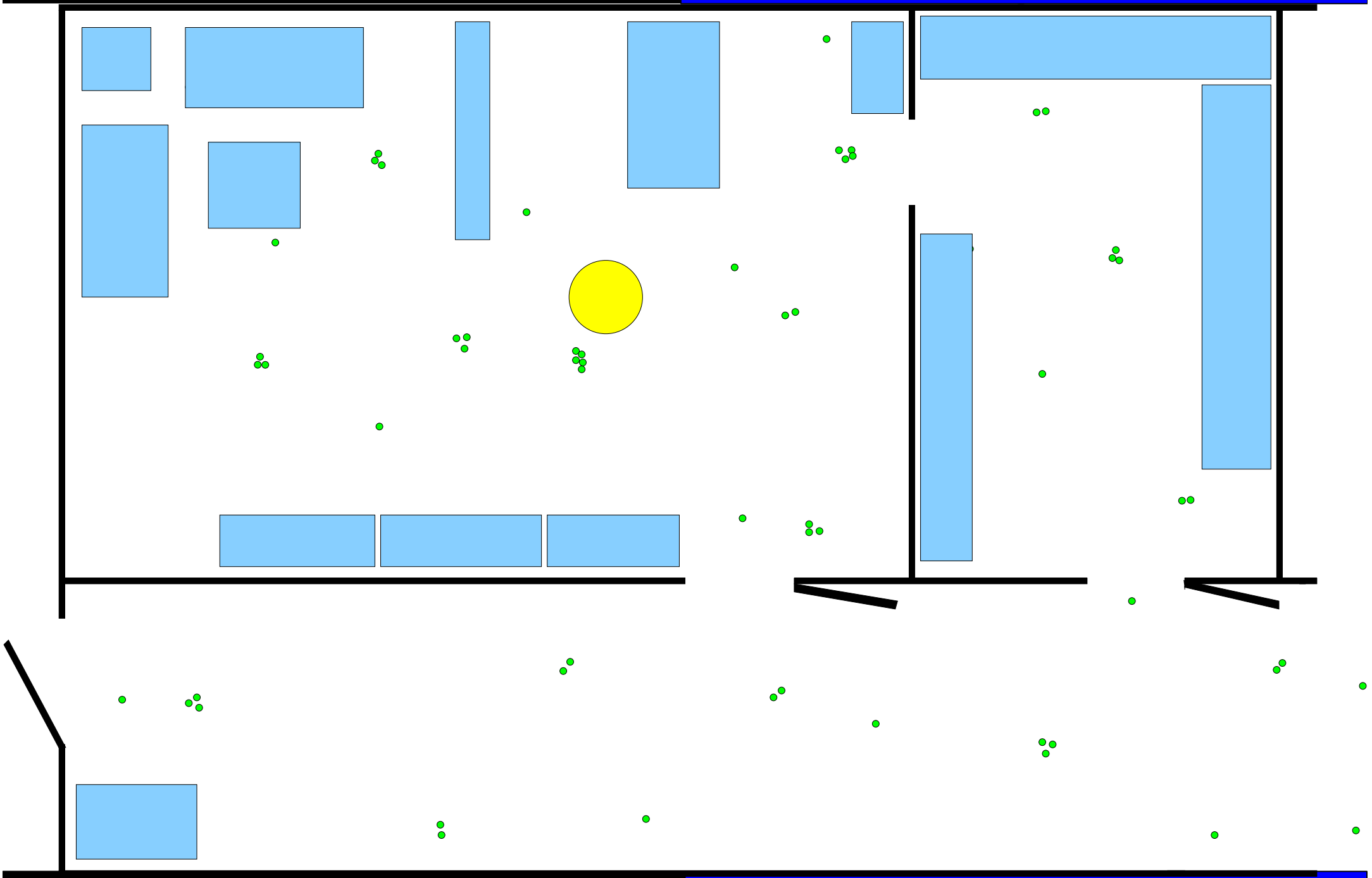
- If you use a sensor like a laser, you may not need to use all readings from the laser to get a good estimate of p(z|x)

Localization




π_N
π_3
π_1
π_2

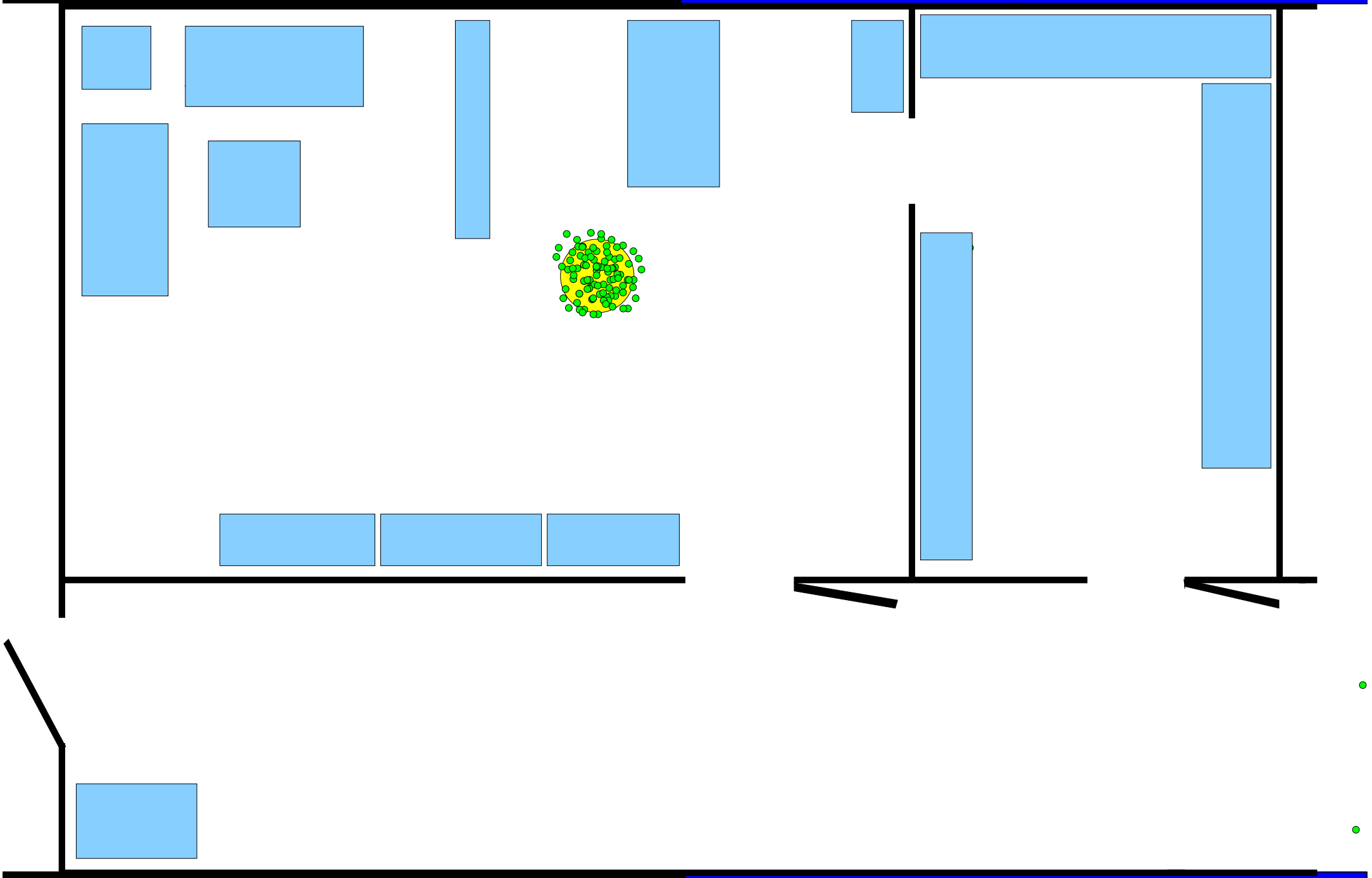
Localization



Patric Jensfelt

DD2410

Localization



Considerations

- Do not know anything about your position?
  - Need more particles!
  - Can reduce as you become more certain

- Noise levels
  - Small is not always better
    - Low noise on the motion model means particles do not spread much and thus the system is more sensitive to errors in odometry, like slippage (“the actual position will move away where there are no particles”).
    - Low sensor model noise will make the system more sensitive to outliers (“will be more likely to kill off particles that do not match the (outlier) measurement”

Particle filter summary

- Very powerful framework
  - Can be used for much more than localization
- Can represent any distribution (given enough particles)
  - Multi-modal no problem!
  - Can solve global localization, i.e. starting from uniform distribution
- Can handle non-linearities
- High dimensionality in the state a challenge
  - To sample the state space, the number of particles required grows exponentially with the number of dimensions.
    "the curse of dimensionality"
    - Too computationally expensive for some applications
- Accuracy depending on number of particles
- Requires some processing to get the "answer", i.e. what is the state? (often fits a Gaussian to distribution)

More reading

- "Monte carlo localization: Efficient position estimation for mobile robots", D. Fox, W. Burgard, F. Dellaert and S. Thrun, AAAI/IAAI, 1999
- "Adapting the Sample Size in Particle Filters Through KLD-Sampling", Dieter Fox, IJRR, 2003

Particle filter localization videos

Gaussian approximation

- The Gaussian approximation to the probability distribution is quite good when the uncertainty is kept low
- Much more efficient representation than particle set
  - N(x,P) where x is the state estimate (the pose) and P is the estimate error covariance of that estimate.
    - 9 parameters for 3D statespace (x:3 + P:6) compared to 3N, where N>>1 for particle filter
  - BUT limited to uni-modal distributions

- Updated by the Kalman Filter
  - When there are non-linearities use
    - Extended Kalman Filter (EKF) or
    - Unscented Kalman Filter (UKF)

System model

- System dynamics ("motion model")

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}),$$

- Measurement model

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

- Assumption on process and measurement noise (zero-mean and Gaussian)

$$p(w) \sim N(0, Q),$$

$$p(v) \sim N(0, R).$$

Approximations

- Cannot guess value of noise. Assume it is 0 for our predictions!

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0)$$
$$\tilde{z}_k = h(\tilde{x}_k, 0)$$

Approximations

- We can linearize system equations

$$x_k \approx \tilde{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + Ww_{k-1},$$

$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + Vv_k.$$

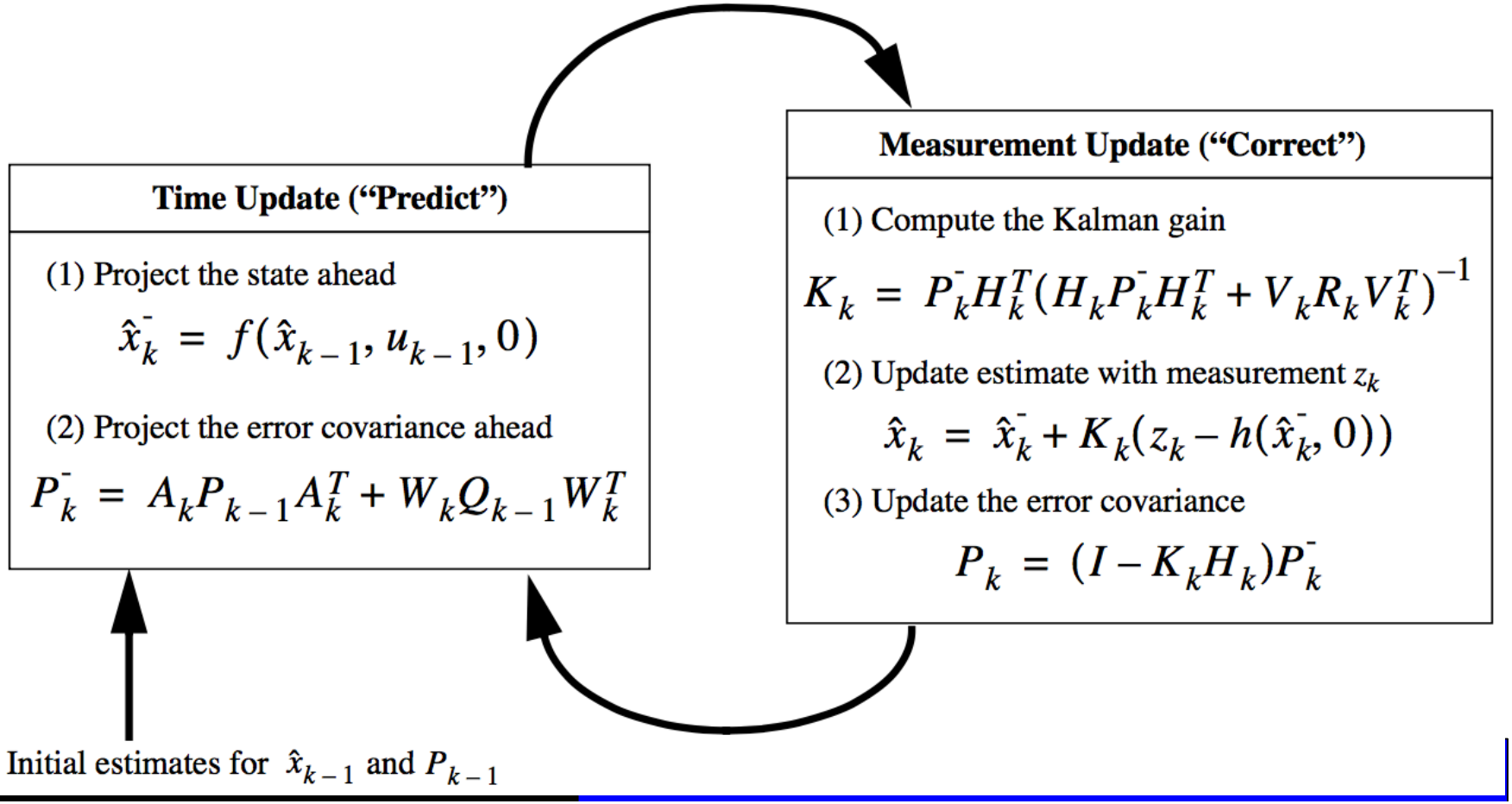
where A,W,H and V are Jacobians

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0) \qquad W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0)$$

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\tilde{x}_k, 0) \qquad V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\tilde{x}_k, 0)$$

Extended Kalman Filter (EKF)

- A prediction step and an update step like for the particle filter
- K is the Kalman gain, weights process vs measurement noise



EKF Localization

- Motion model is the same as for particle filter
- Measurement model the same
- Need to calculate Jacobians
- Need to set noise covariance matrices Q and R
  - You can think of these as tuning values (just like in the case for the particle filter)
  - Start at values derived from “reasoning” about errors and then adjust for good performance.

Noise levels

- Small is not always better
  - Small process noise means that the system will be sensitive to disturbances in odometry
  - Small measurement noise means that outliers will influence the estimate much and that the output will be less smooth (“jumps” more)

Properties of EKF Localization

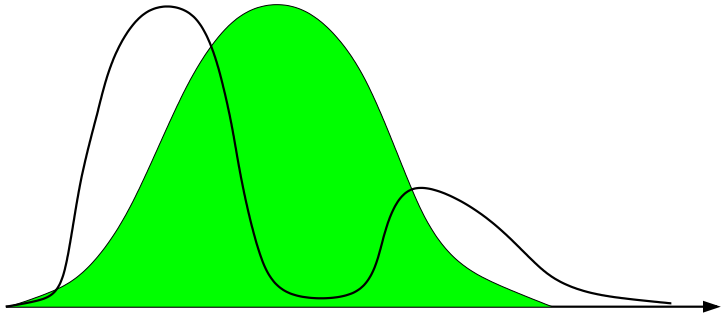
- Computationally very efficient
- Has been used extensively "everywhere" for a long time
  - The workhorse in target tracking for example
- Scales well to high dimensions
- Cannot handle multi-modal distributions
- Linearization problematic in cases of large uncertainty w.r.t to the non-linearities.
  - Dynamics very different between estimated (around which we linearize) and true state (where the system actually is) → bad estimates
- Need "good enough" initial guess to converge which depends on the type of use case.

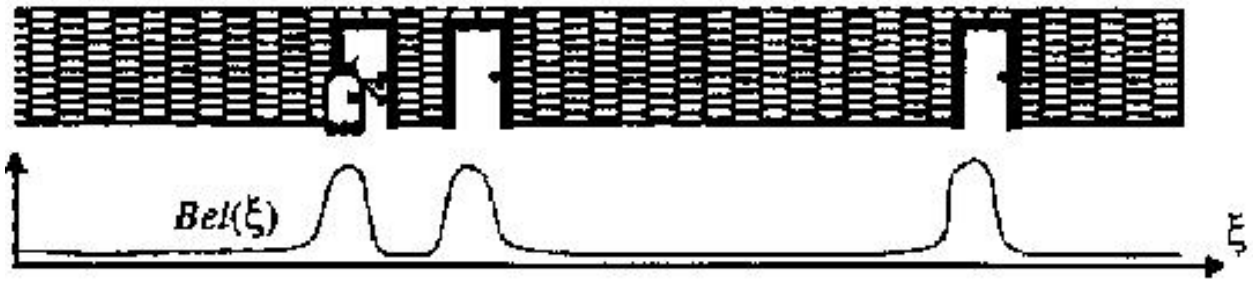
More reading

- "An Introduction to the Kalman Filter",
  G. Welch and G. Bishop,
  https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

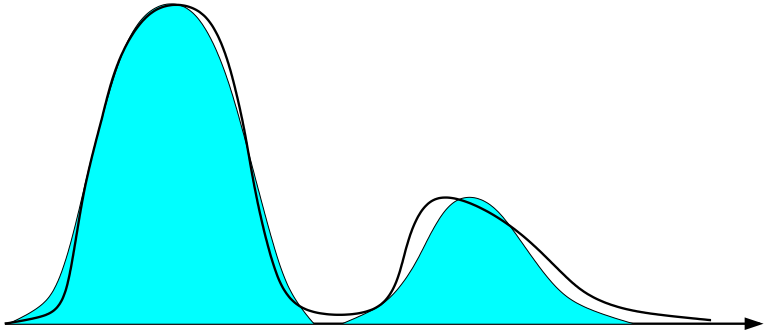
Limitations of a Gaussian

- A Gaussian cannot represent multi modal distributions!





- "Active global localization for a mobile robot using multiple hypothesis tracking", P. Jensfelt and S. Kristensen, IEEE Transactions on Robotics and Automation, 2001.
  - Shows that you can handle multimodal distributions using a mixture of Gaussians



What about those measurements?

- What can we measure in practice
  - Range and bearing to a landmark (e.g. a corner)
  - Lines/planes → distances and angles
  - Transformation between images/laser scans
  - Ray trace in a map to get likelihood of measurement
  - …

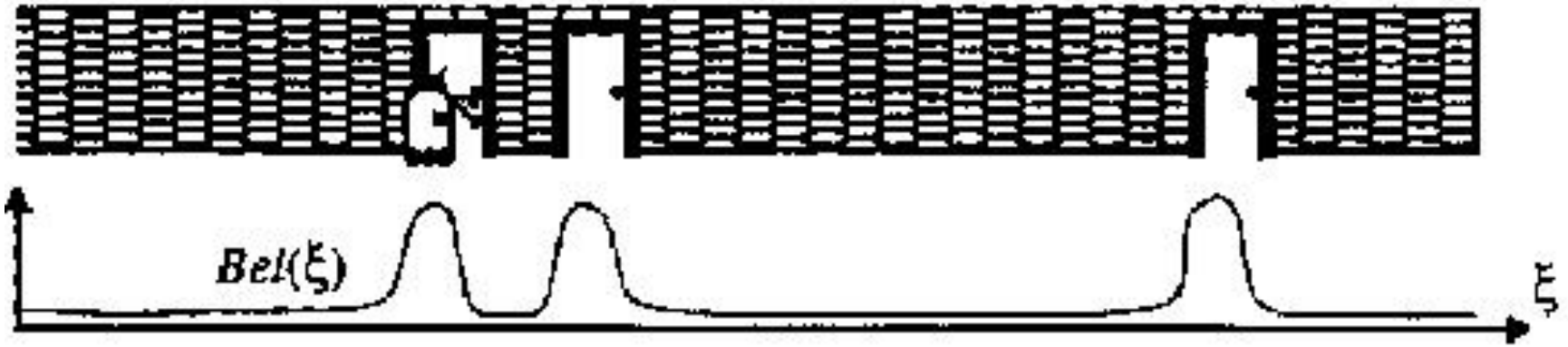
- What we need:
  - Provide likelihood of the measurement given state p(z|x)

Additional real world challenges

- Data association is super hard!
  - Need to associate a measurement with a landmark / part of a map
  - Ex: What parts of the environment did the M measurements from the laser come from??

Topological localization

- Discrete positions (“places”).
  - Ex: A room, a subway station, near landmark X
- Pure topological localization depends on place recognition
  - Need to recognize the place and at least be able to tell all places connected to current place apart so that we know how we move
- Often combined with coarse metric information
- Ex: Discretize corridor into segments



So much more to say

- Localisation is an extremely heavily researched area
- Needed by all (more or less) systems that move
- Localiztion using
  - Signal strength of WiFi signals
  - BLE
  - UWB
  - …