Lecture: Localisation

Mathematical Basics, Dead-Reckoning and Localization

MOBILE ROBOT CONTROL 2024

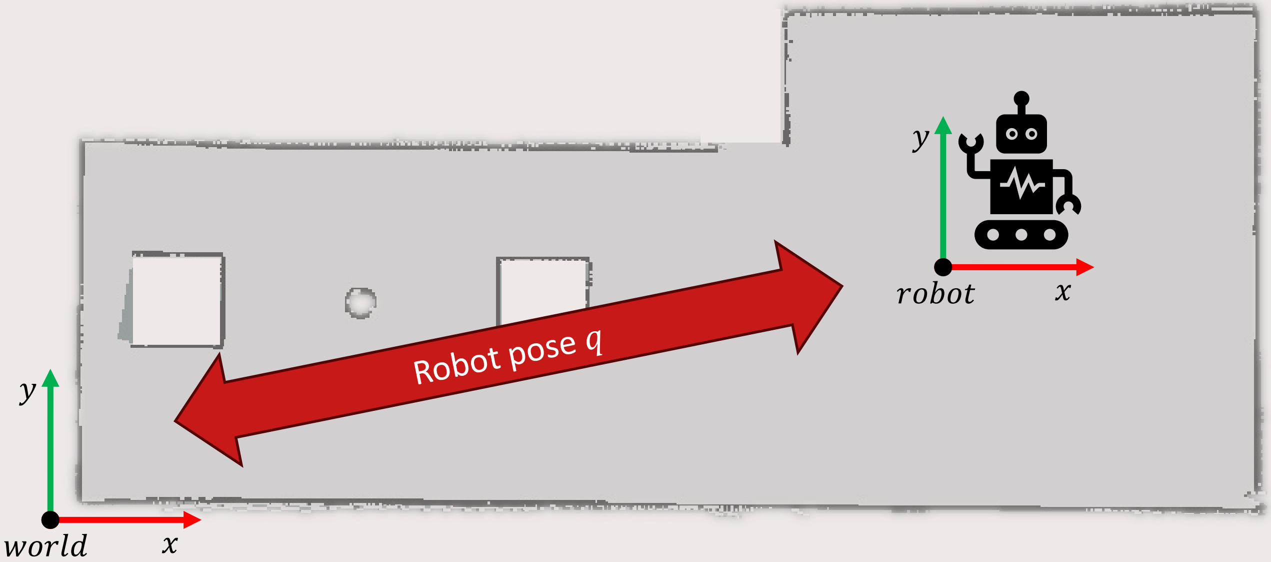
Gijs van Rhijn, Gijs van de Brandt, Koen de Vos, Jos Elfring

What is (robot) localization?

Compute the robot pose with respect to some frame of reference (e.g. a map)

Why do we need robot localization?

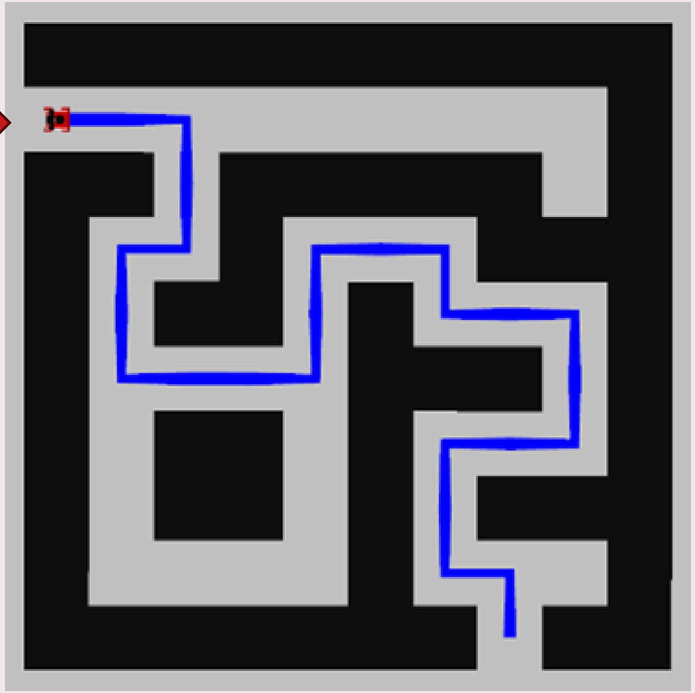
Being able to use a map, requires the pose of our robot with respect to the map



Why do we need robot localization?

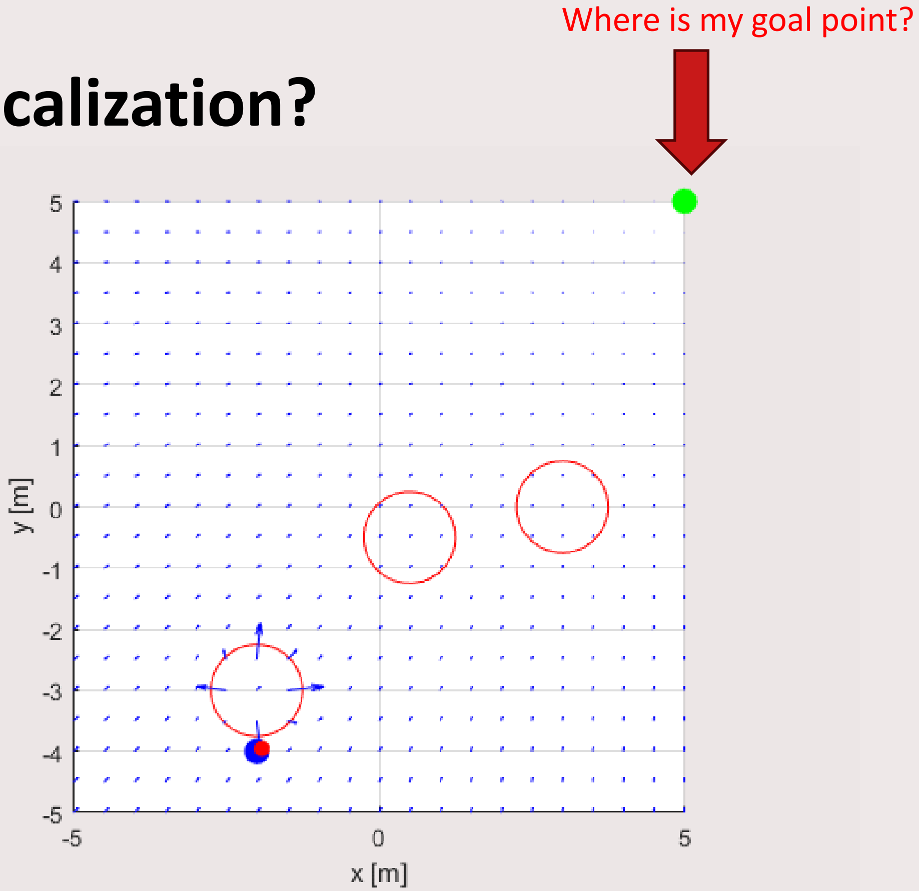
Global path planning:
we cannot plan a path if we do not know where we are!

We need to know where we start!



Why do we need robot localization?

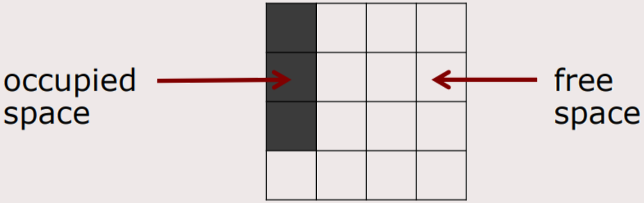
Local path planning:
we need to know the location of our waypoints.



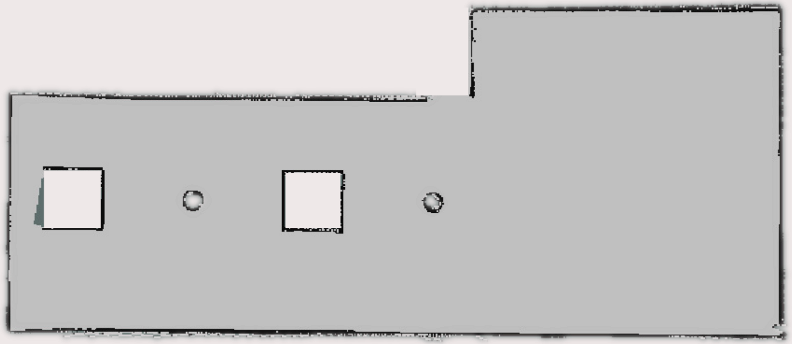
“What” do we localize on?

Today, maps used for localization will be represented by an **occupancy grid**

- Discretized world in which is cell is either occupied or empty



http://ais.informatik.uni-freiburg.de/teaching/ws22/mapping/



Intermezzo – brief recap on probability theory

Recap: random variables

Discrete random variable

- X can take on a countable number of values in the set $\{x_1, x_2, \ldots, x_\mathrm{n}\}$
- P(X=x_i), or P(x_i), is the probability that the random variable X takes on value x_i and P(·) is called **probability mass function**

Continuous random variable

- X takes values in the continuum
- p(X=x), or p(x), is a **probability density function** and

$$\Pr(x \in (a,b)) = \int_a^b p(x)dx$$

Recap: joint and conditional probabilities

Joint probability: $P(X{=}x \text{ and } Y{=}y) = P(x,y)$

- If *X* and *Y* are **independent**, then

$$P(x,y) = P(x)\,P(y)$$

Conditional probability: $P(x \mid y)$ is the probability of ***x* given *y***

$$P(x \mid y) = P(x,y) \,/\, P(y)$$

$$P(x,y) = P(x \mid y)\,P(y)$$

- If X and Y are **independent**, then

$$P(x \mid y) = P(x)$$

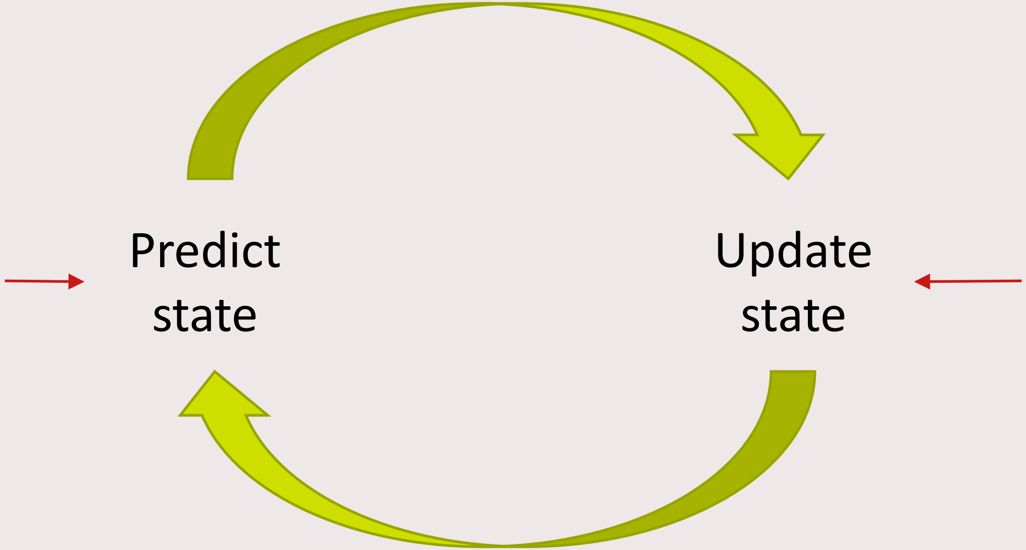
Recap: Bayes theorem

$$P(x|z) = \frac{P(z|x)P(x)}{P(z)} = \frac{\text{likelihood } \cdot \text{prior}}{\text{evidence}},$$

$$P(z) = \sum_{x} P(z|x)P(x)$$

Recap: Bayesian filter

State – vector with quantities that must be estimated



System model that describes how the state changes over time

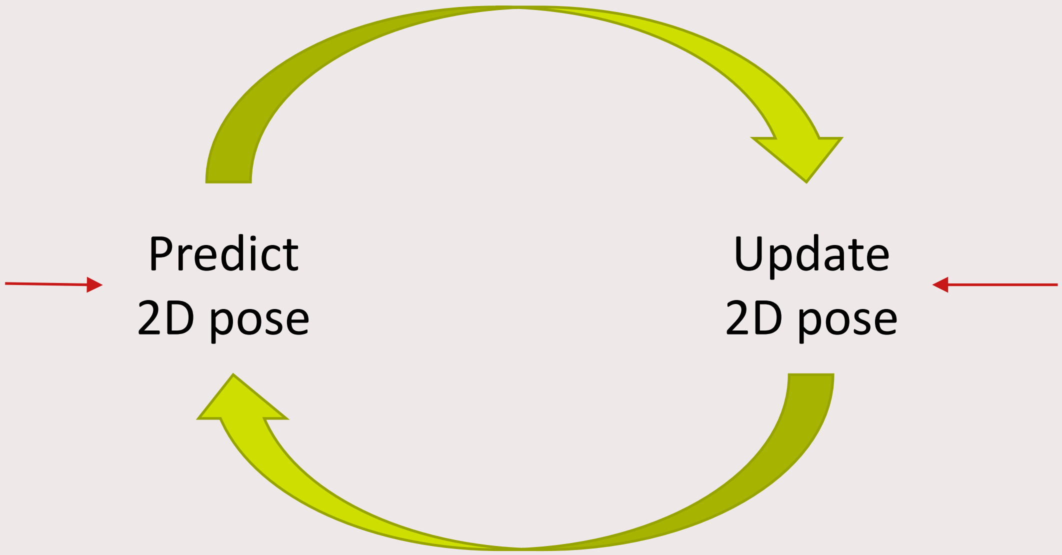
Predict state

Update state

Measurement model that relates measurements to state that must be estimated

Recap: Bayesian filter → localization

State – 2D robot pose: x-position, y-position, orientation



System model that describes how the robot moved over time (e.g. using wheel encoders)

Predict
2D pose

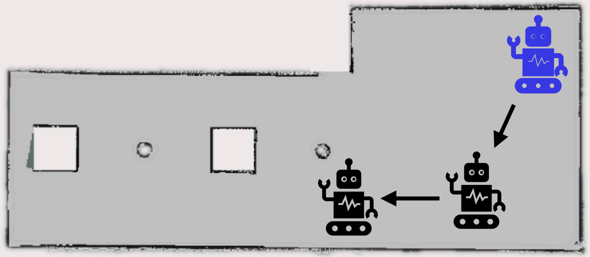
Update
2D pose

Measurement model that relates map-related measurements to robot pose

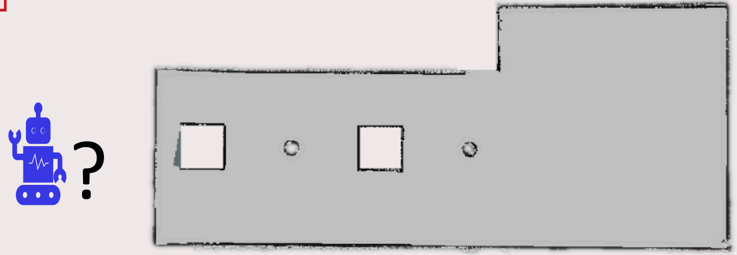
Types of localization problems

- “Tracking”
  - Initial position is known
  - Keep track of position while moving

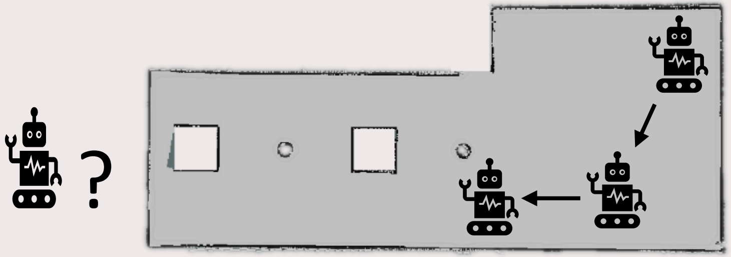
This will be the scenario in the final challenge



- “Global localization”
  - Initial position can be anywhere
  - Once position has been found start tracking

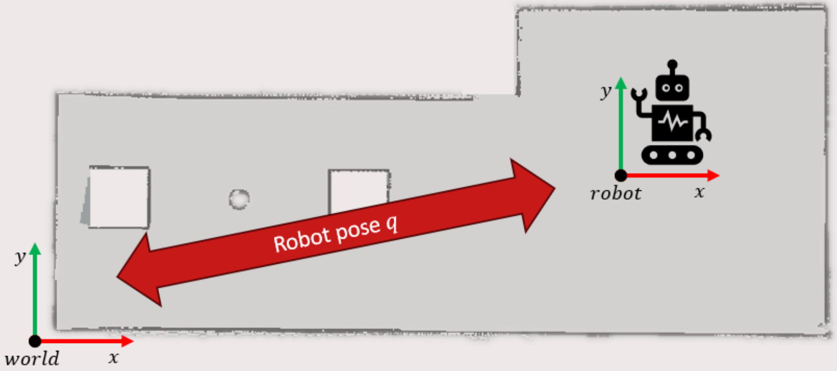


- “kidnapped robot”
  - Start by tracking
  - Trigger global localization when needed



Problem statement

Goal: estimate 2D robot pose $\mathrm{x} = \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}$

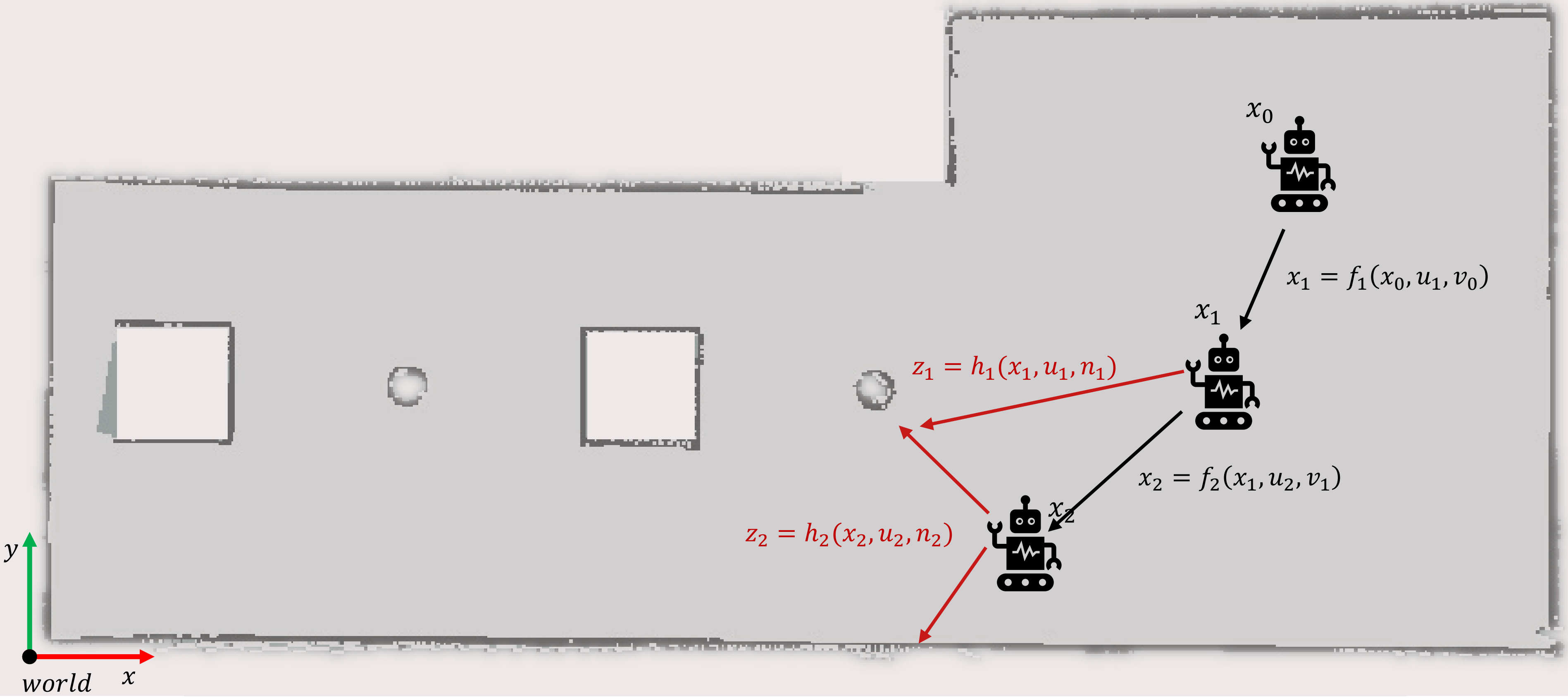


We have:

- Prediction model:
  - $x_t = f_k(x_{t-1}, u_t, v_{t-1})$
  - Knowledge on how state x evolves over time – noise represents confidence model

- Measurement model:
  - $z_t = h_t(x_t, u_t, n_t)$
  - Way to relate measurements to the state x – noise represents measurement noise

Problem statement: graphical representation


x_0
$x_1 = f_1(x_0, u_1, v_0)$
x_1
$z_1 = h_1(x_1, u_1, n_1)$
$x_2 = f_2(x_1, u_2, v_1)$
x_2
$z_2 = h_2(x_2, u_2, n_2)$
y
$world$
x

Remark on probability notations

In these slides:

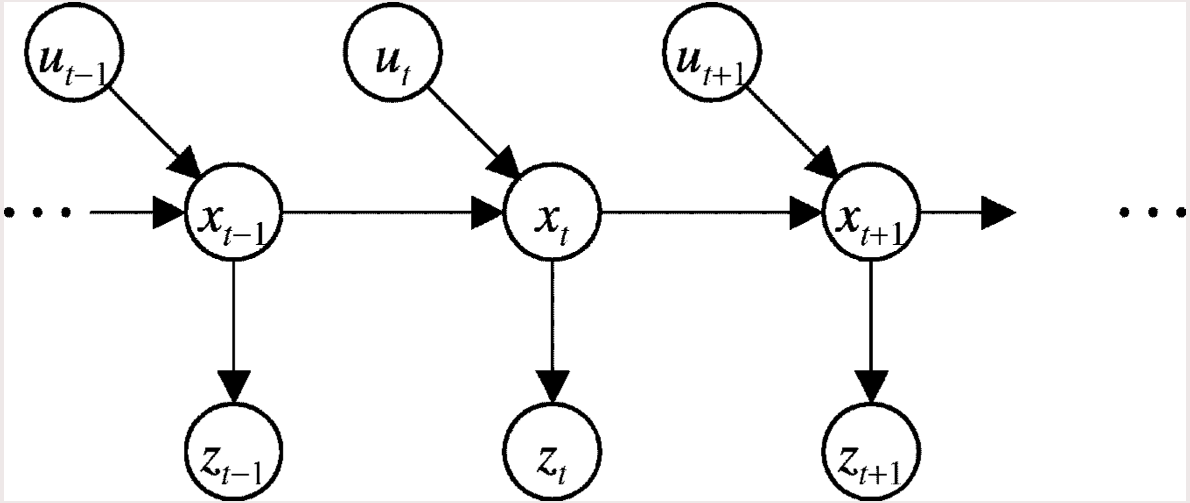
- PDF representing the 2D robot pose:

$$p(x_t)$$

- PDF representing a measurement:

$$p(z_t)$$

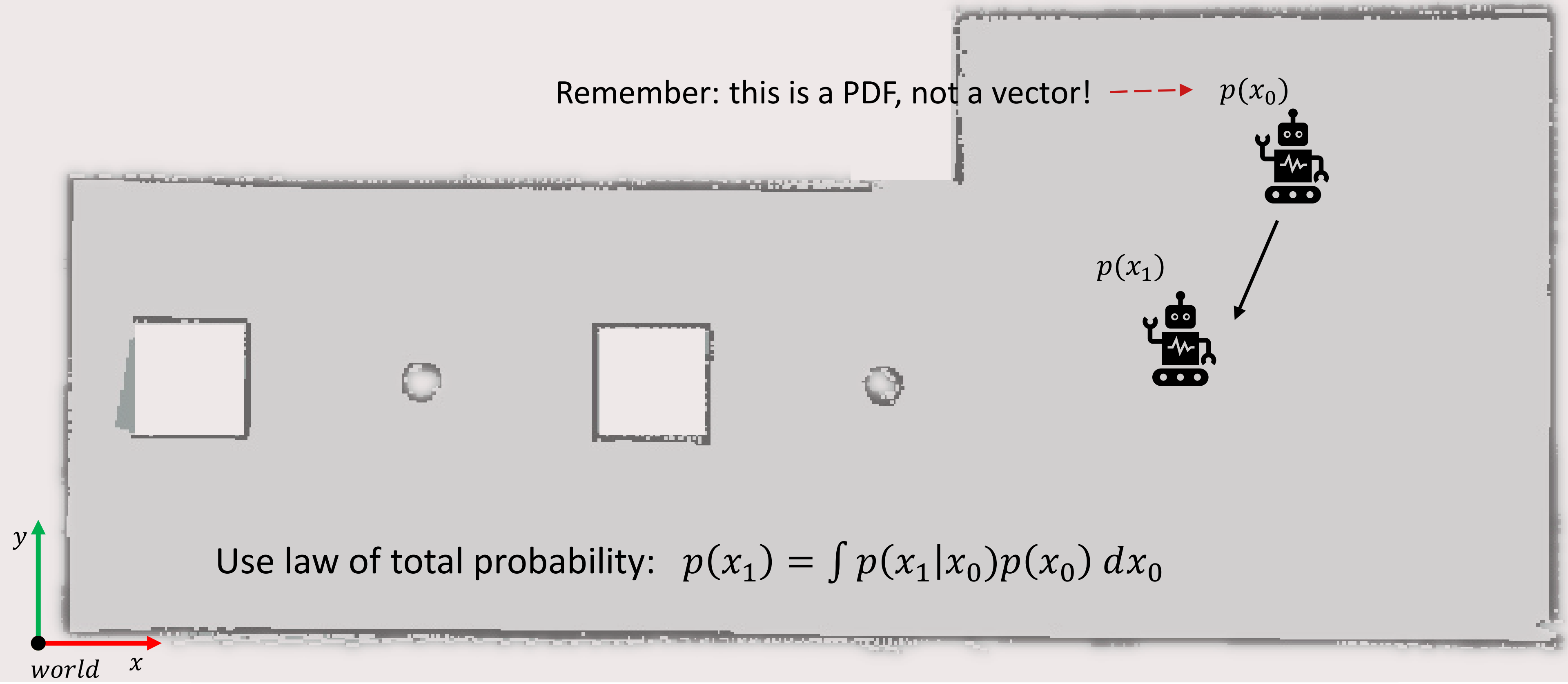
Markov assumption for sequence modeling



$$p(z_t \mid x_{0:t}, z_{1:t}, u_{1:t}) = p(z_t \mid x_t)$$

$$p(x_t \mid x_{1:t-1}, z_{1:t}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t)$$

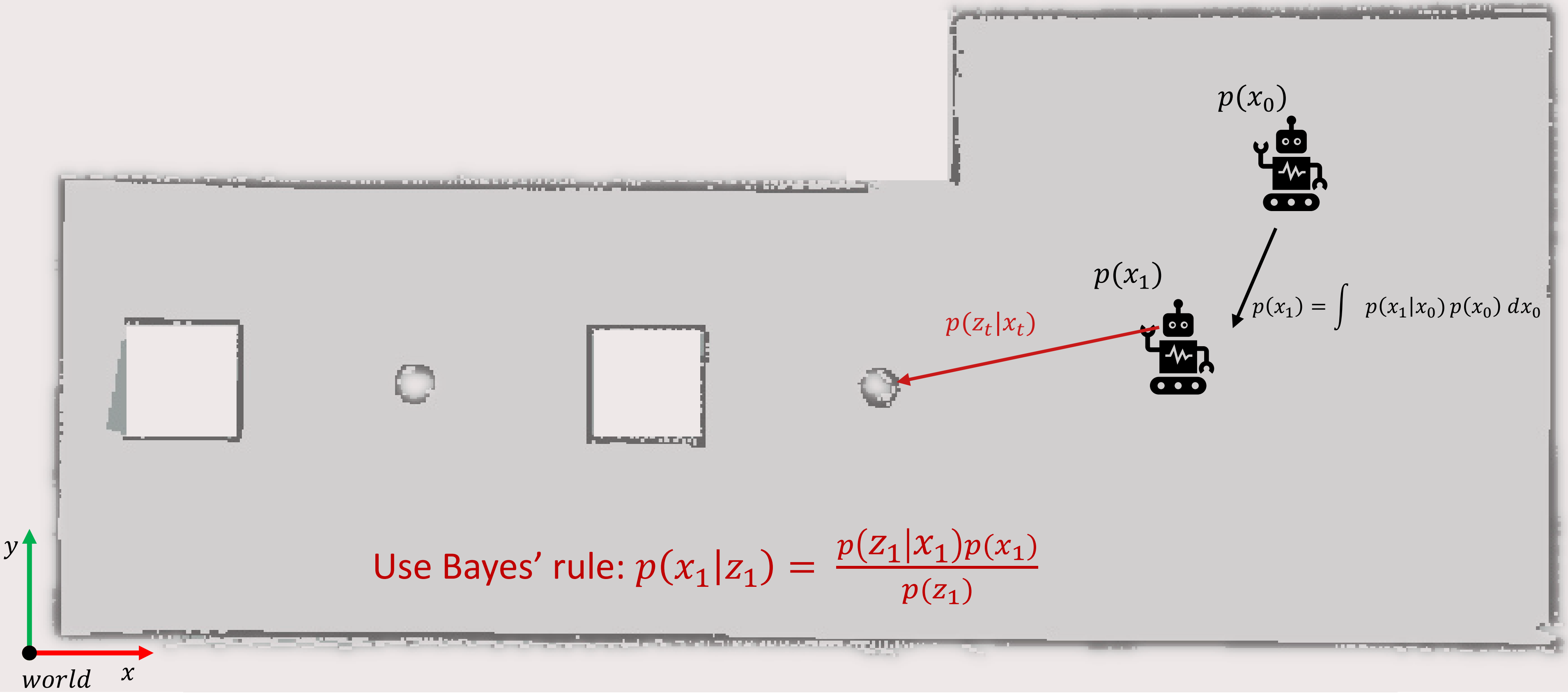
Probabilistic modelling: location prediction



Remember: this is a PDF, not a vector!

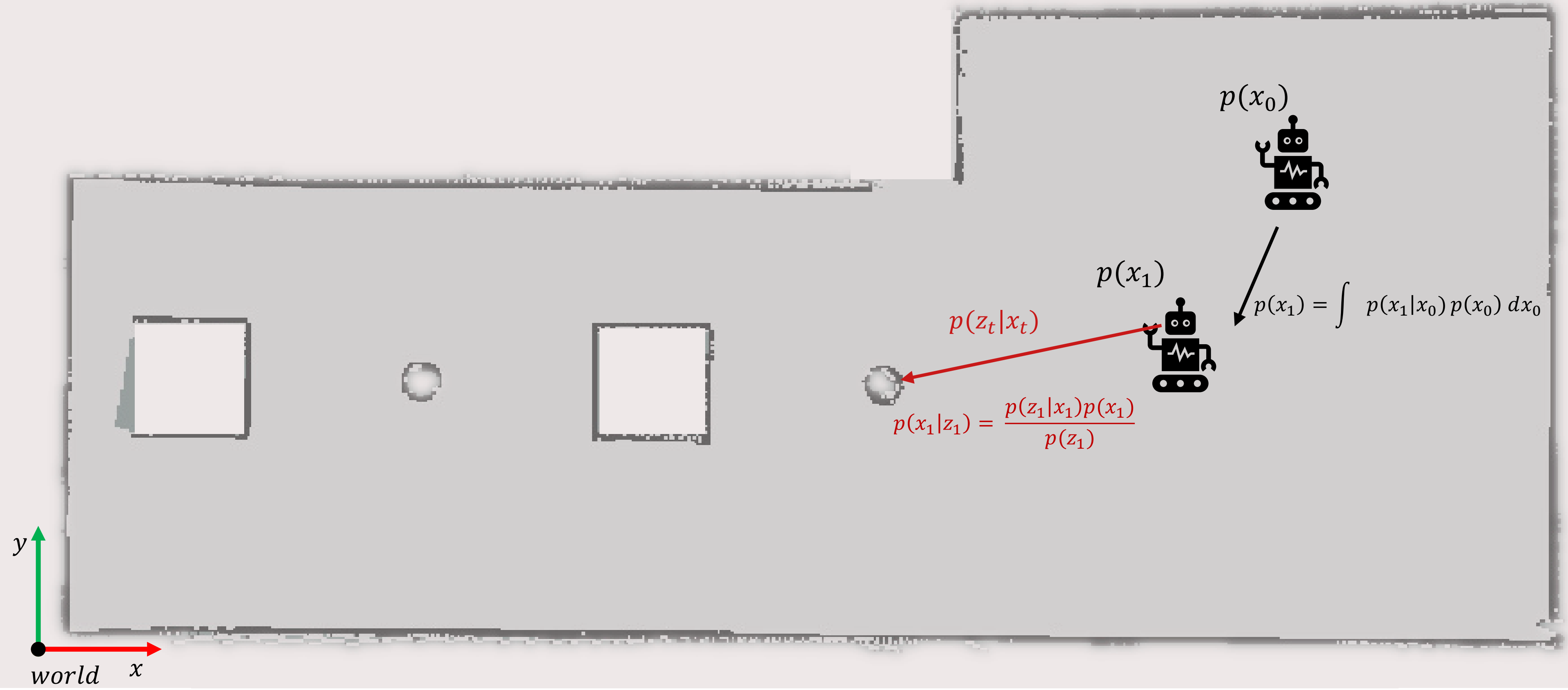
Use law of total probability: $p(x_1) = \int p(x_1|x_0)p(x_0)\, dx_0$

Probabilistic modelling: sensor update



Use Bayes' rule: $p(x_1|z_1) = \dfrac{p(z_1|x_1)p(x_1)}{p(z_1)}$

Probabilistic modelling: sensor update



$p(x_0)$

$p(x_1)$

$p(x_1) = \int p(x_1|x_0)\,p(x_0)\,dx_0$

$p(z_t|x_t)$

$p(x_1|z_1) = \frac{p(z_1|x_1)p(x_1)}{p(z_1)}$

y

x

$world$

Probabilistic modelling: overview of steps

Initialize

- $p(x_{t-1})$

- Predict robot pose in next timestep using prediction model:

$$p(x_t|\, z_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|z_{1:t-1})\, dx_{t-1}$$

The integrals can be hard to compute! How about an approximation?

- Update robot pose using measurement model

$$p(x_t|z_t) = \frac{p(z_t|x_t)p(x_t|z_{1:t-1})}{p(z_t|z_{1:t-1})},$$

where: $p(z_t|z_{1:t-1}) = \int p(z_t|x_t)p(x_t|z_{1:t-1})\, dx_t$ (normalization constant)

- Repeat

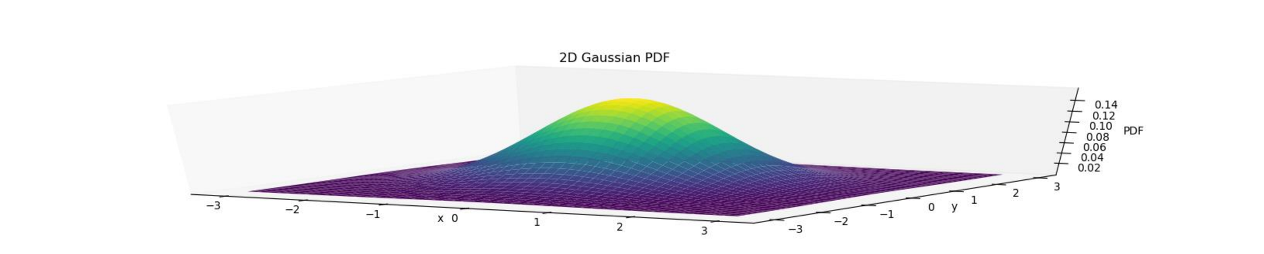
Particle filter idea

- Approximate a PDF representing a continuous random variable by a set of *N* ‘particles’

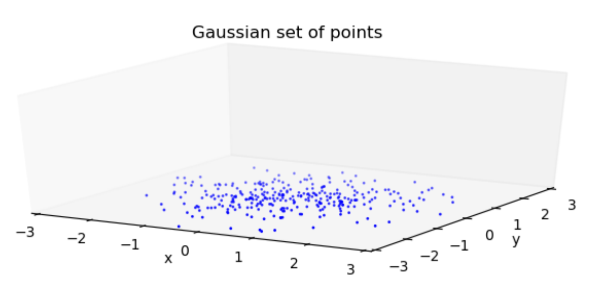
- Each particle represents a *possible* value of the state
(i.e. each particle is a 3D vector representing a 2D robot pose)

- Use prediction and update steps introduced on previous slide

Particle filter intuition



2D Gaussian PDF

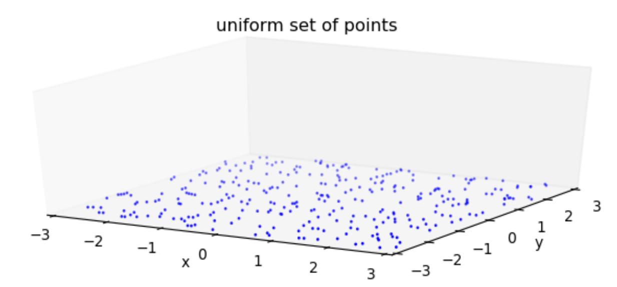


Gaussian set of points

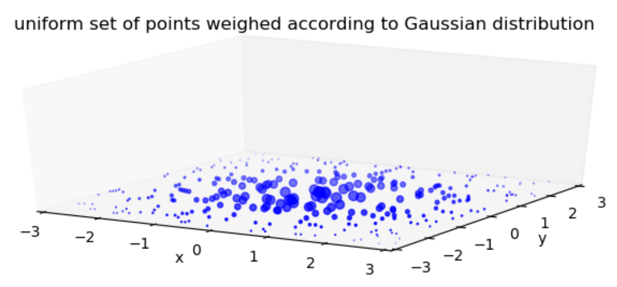
Problem: we cannot sample from the unknown distribution we would like to estimate

- Sample from "Proposal distribution“ (q) instead
- Use weights to compensate for sampling from q

Example “proposal distribution“ (q)



uniform set of points



uniform set of points weighed according to Gaussian distribution

The particle filter represents a distribution by a set of weighted samples!

Particle filter properties

Advantages:

- Hard-to-compute integrals turn into summations over *N* particles
- Particles can be distributed over map in any form → flexibility in ‘shape’ of PDF
- Prediction and measurement models have minimal restrictions (e.g. noise can be non-Gaussian, models van be non-linear)

Disadvantages:

- Computational load proportional to number of particles *N*
- Number of required particles scales poorly with dimension of state (which is 3 in our case)

Particle filtering: representing the PDF by a set of weighted points

- $p(x_{0:t}|z_{0:t}) \approx \sum_{i=1}^{N_s} w_t^i\, \delta\left(x_{0:t} - x_{0:t}^i\right) = \hat{p}\ (x_{0:t}|z_{0:t})$

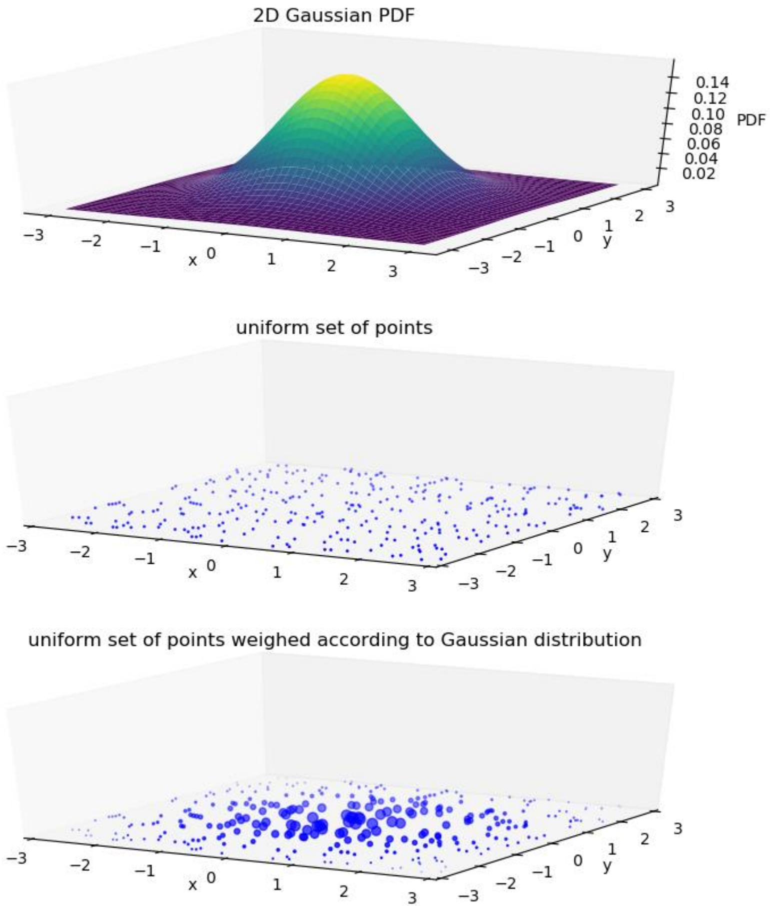
  weight

  coordinates

- $\delta(x) = \begin{cases} 0 & x \neq 0 \\ \infty & x = 0 \end{cases}$
  - $\int \delta(x)\ dx = 1$

- $w_t^i \propto \frac{p(x_t^i)}{q(x_t^i)}$ ← the weight compensates for the proposal density

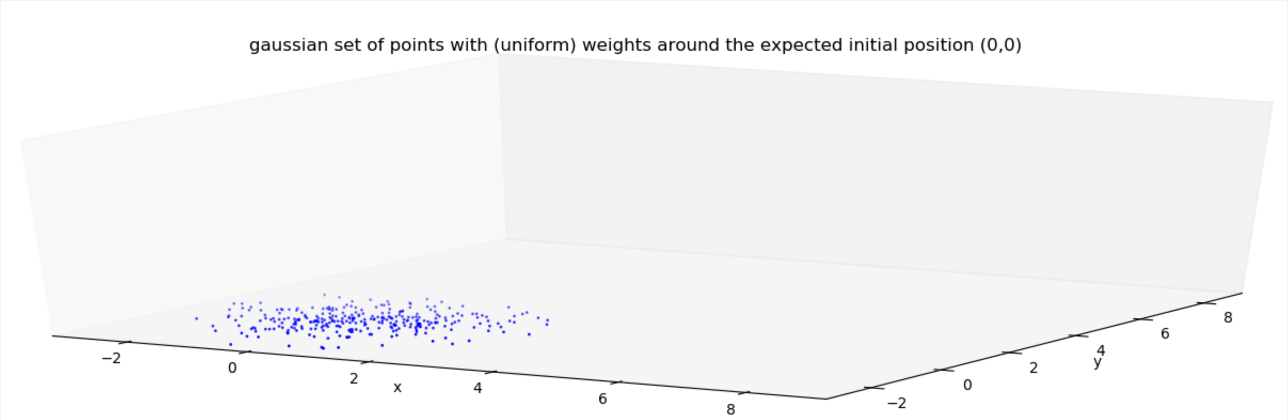
- $\sum_{i=1}^{N_s} w_t^i = 1$



Initializing a particle filter for robot localization

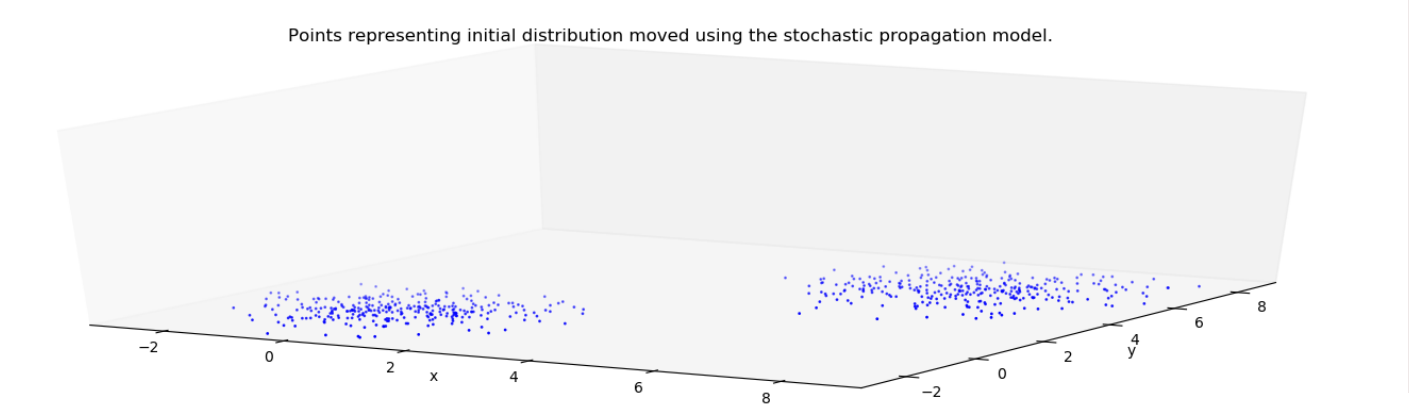
We have assumed an initial guess is available

- Sample N particles from the initial guess (e.g., a uniform distribution over a part of the map)
- Set all particle weights w_i to $1/N$



Prediction step

- Move each of the particles according to our model: $f(x_i, u, v)$
  - x_i: each of the particle states
  - u: control input that might be available (**same** for all particles)
  - v: independent noise sample (**different** for each particle) → 'diversifies' particles
- Values weights do not change

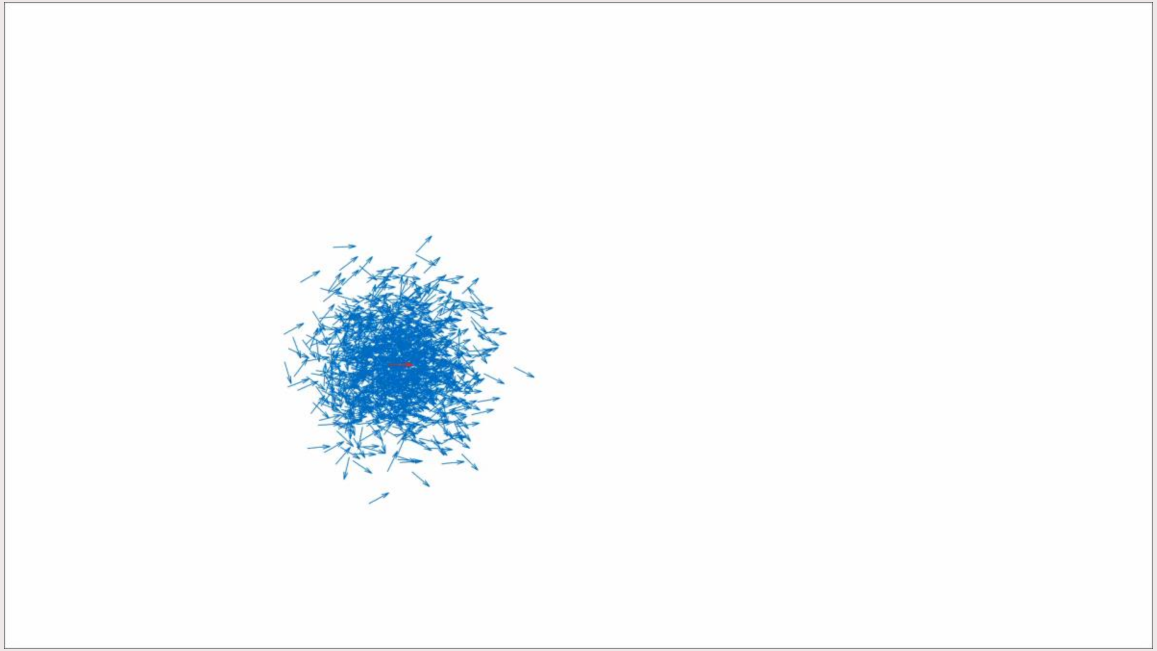


Predictions only – what about measurements?

- We can keep repeating this prediction step

- Our estimate will diverge

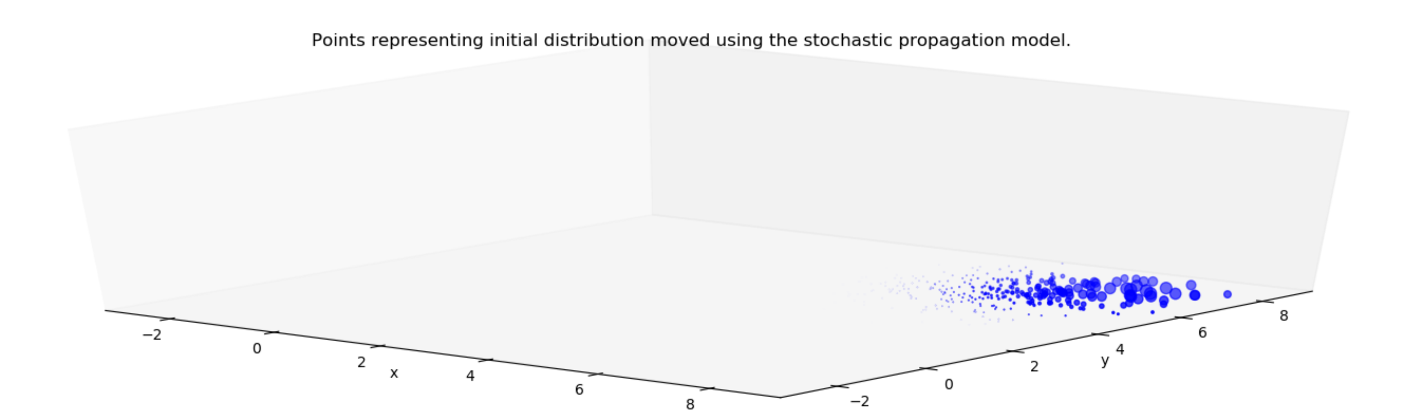
- How do we incorporate measurements?



Incorporate sensor data by weighing with Bayes’ rule

- Use predicted density as a proposal density
- Update particle weights using Bayes’ rule (do not change particle locations):

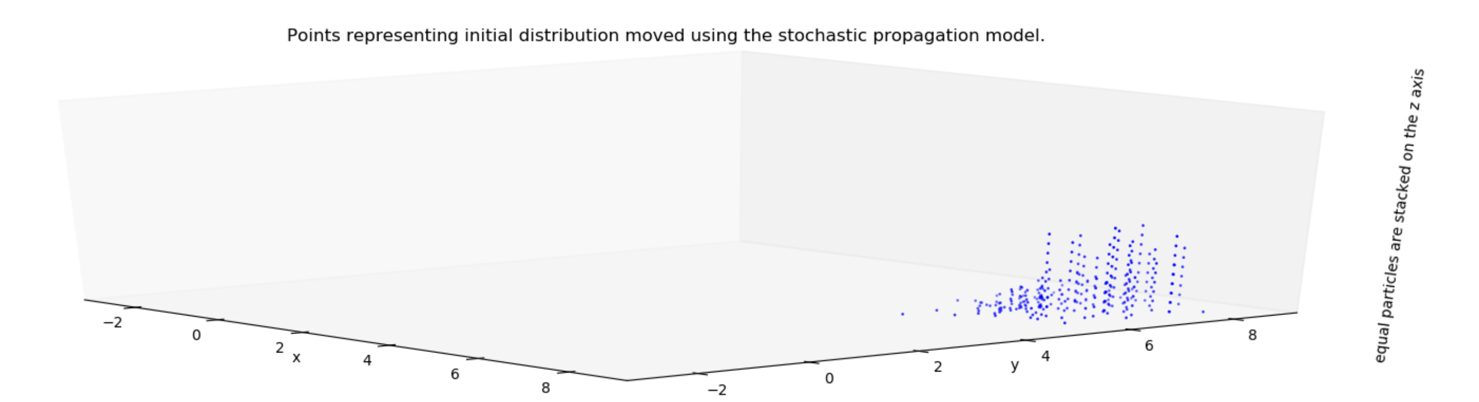
$$w_t = \frac{1}{c} p\left(z_t | x_t^i\right) w_{t-1}$$, where c is a normalizing constant



Resampling

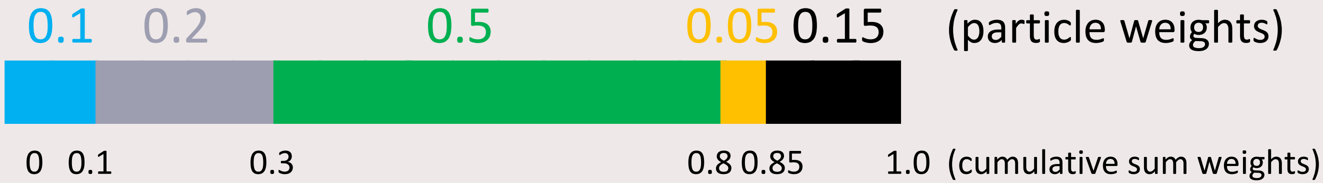
After a few time steps, all but one particle will have a weight of 0

- Resample (with replacement) each particle using its weight as a probability of being chosen
  - low-weight particles disappear, high-weight particles are duplicated
  - Reset the weight to 1/*N*



Multinomial resampling – implementation

Example with 5 particles



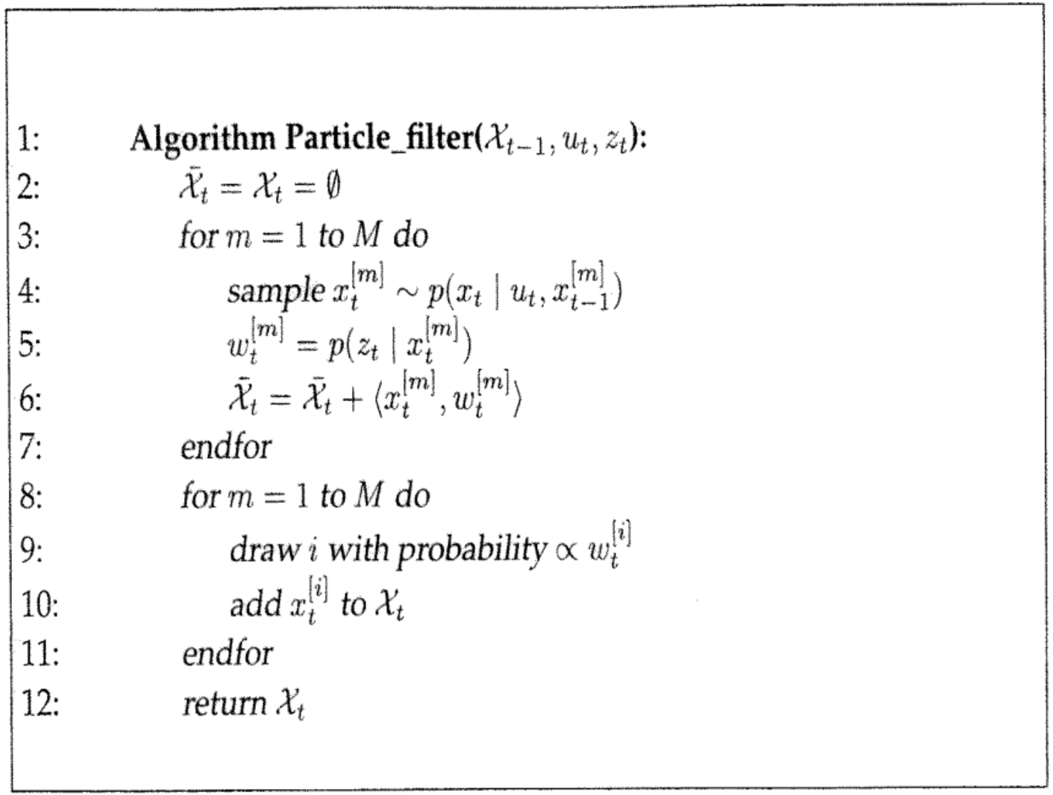
0.1 0.2 0.5 0.05 0.15 (particle weights)

0 0.1 0.3 0.8 0.85 1.0 (cumulative sum weights)

Sample from a uniform distribution U(0,1) → N_s = five times

- Sample between 0 and 0.1 → duplicate particle one
- Sample between 0.1 and 0.3 → duplicate particle two
- Sample between 0.3 and 0.8 → duplicate particle three
- …

Pseudocode



```
Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):
    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
    for $m = 1$ to $M$ do
        sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
        $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
        $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
    endfor
    for $m = 1$ to $M$ do
        draw $i$ with probability $\propto w_t^{[i]}$
        add $x_t^{[i]}$ to $\mathcal{X}_t$
    endfor
    return $\mathcal{X}_t$
```

For more detailed information see:

Elfring, J.; Torta, E.; van de Molengraft, R. Particle Filters: A Hands-On Tutorial. *Sensors* **2021**, *21*, 438. https://doi.org/10.3390/s21020438

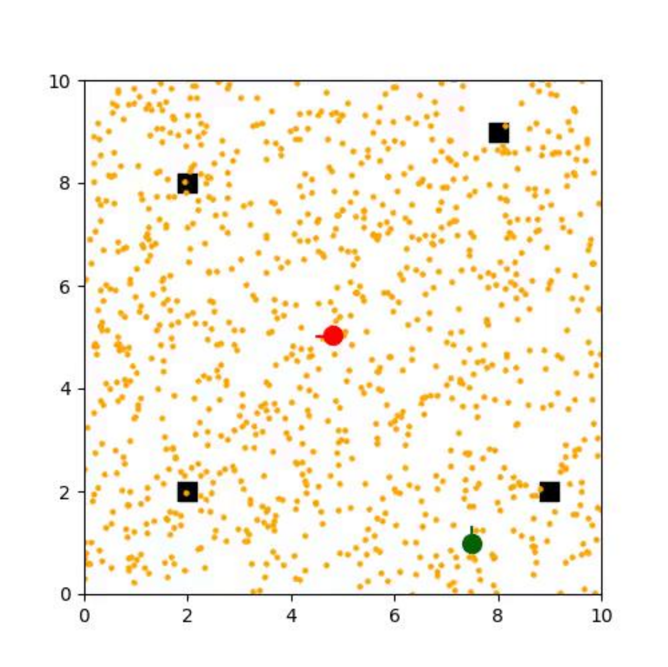
F. Gustafsson, "Particle filter theory and practice with positioning applications," in *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53-82, July 2010, doi: 10.1109/MAES.2010.5546308

Particle filter: example animation

- Note how the weighing and resampling steps aren't explicitly visualized here.
  - Only the result the prediction is shown
    - Uniform weights

- Rotation is also part of each sample
  - (samples are a random state of x, y, theta)

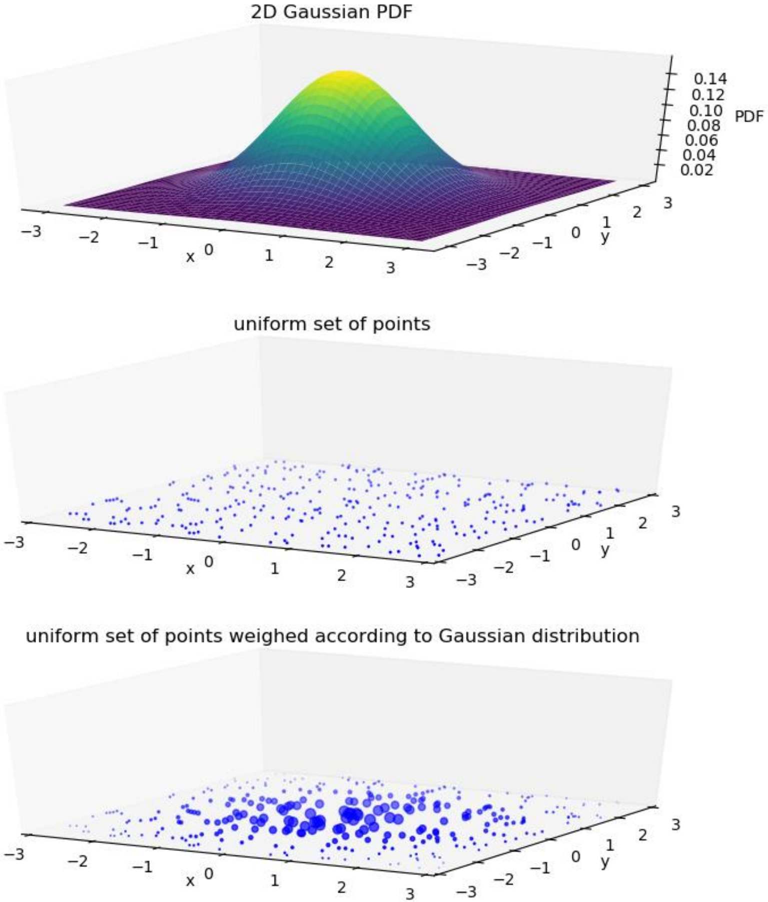
- Next: how to compute $p(z_t|x_t)$



Time for a break!

After the break:

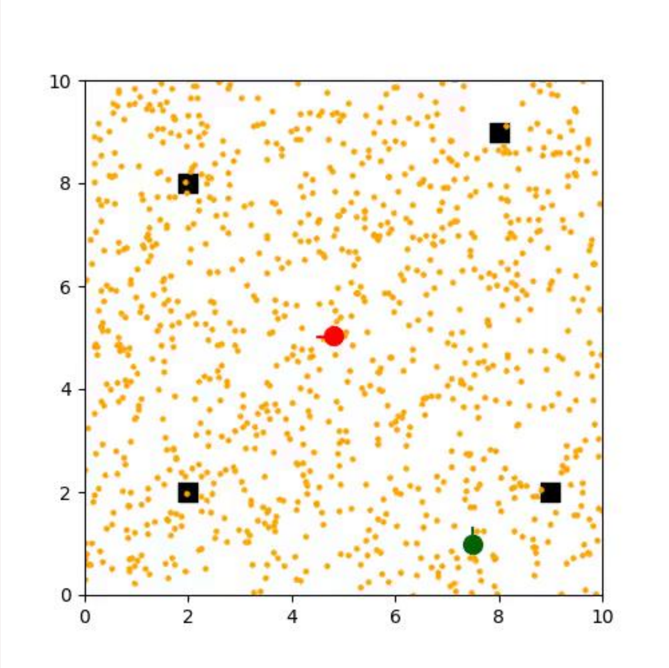
- How to calculate $p(z_t|x_t)$
- How to initialize a particle filter
- Obtaining a pose from the particle filter



Welcome back!

To discuss:

- How to calculate $p(z_t|x_t)$
- How to initialize a particle filter
- Obtaining a pose from the particle filter



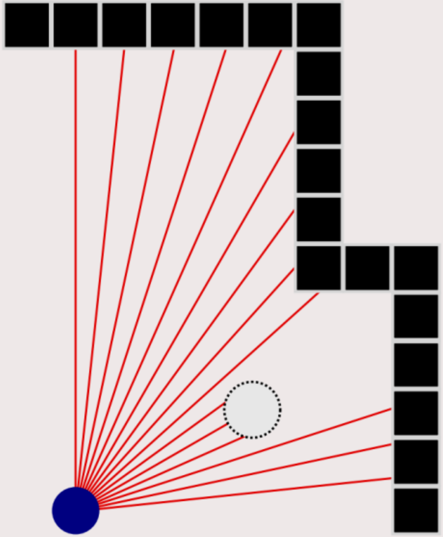
Recursive State Estimation
Beam-based model

For now, let’s define a measurement as a vector of ranges:

- $z_k = \begin{bmatrix} (r_0, \theta_0) \\ (r_1, \theta_1) \\ \vdots \\ (r_2, \theta_2) \end{bmatrix},$

- Given a map, a robot pose, and *appropriate algorithms* we can generate a prediction of this measurement should be

- $z_k^* = \begin{bmatrix} (r_0^*, \theta_0) \\ (r_1^*, \theta_1) \\ \vdots \\ (r_2^*, \theta_2) \end{bmatrix}$

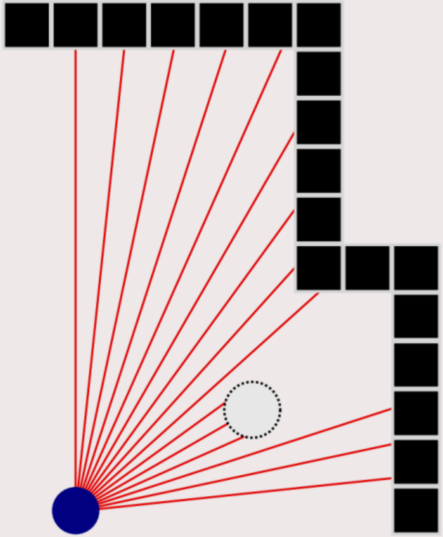


Recursive State Estimation
Beam-based model

Appropriate algorithms?

A family of algorithms called Ray casters.

Don’t worry about them for now,
we have provided you with one for the assignment :)

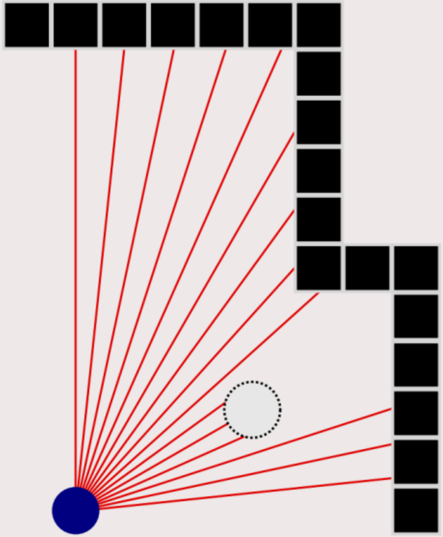


Recursive State Estimation
Beam-based model

Observe that we now have a measurement and a measurement prediction in the ideal (modeled) case.

Core Idea:

The mismatch between the two tell us something about whether the robot pose is correct.

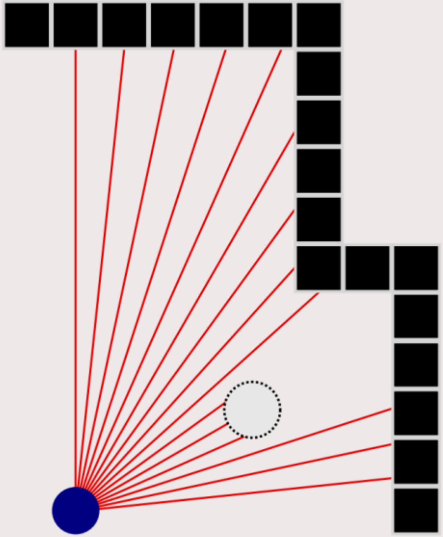


Recursive State Estimation
Beam-based model

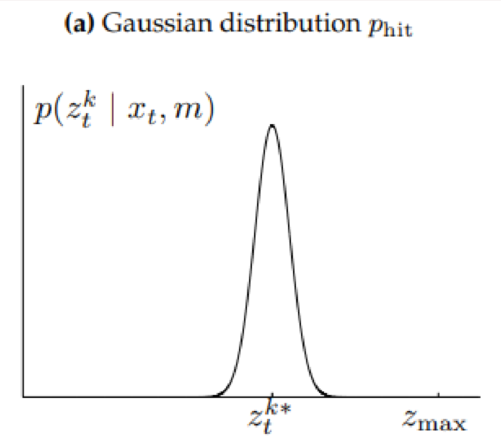
How to quantify this
mismatch as a probability $p(z_t|x_t)$?

- For a **single ray**, we identify four sources of “disturbances”
  1. Local measurement noise
  2. Unexpected obstacles (object not present in the map)
  3. Failures (Glass, Black obstacles)
  4. Random measurements

- We assign each source a distribution and probability of occurring



Recursive State Estimation – beam-based model



Local Measurement Noise

$$p_{short}(z_t^k | x_t, m) = \begin{cases} \eta\, \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) & if\ 0 \le z_t^k \le z_{max} \\ 0 & otherwise \end{cases}$$

Evaluating a Gaussian does not guarantee p_{short} is between 0 and 1, which is why a normalizer is needed:

$$\eta = \left(\int_0^{z_{max}} \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) dz_t^k \right)^{-1}$$

z_t^k : measured range
z_t^{k*}: true range
σ_{hit}: std. dev. measurement noise
$\mathcal{N}(x; \mu, \sigma_{hit}^2)$: evaluate Gaussian with mean μ and standard deviation σ at x

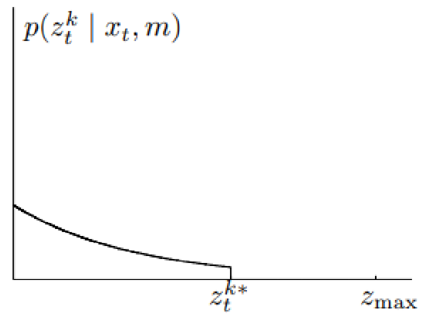
Recursive State Estimation – beam-based model

Could be that the robot measures unexpected obstacles (measure nearby objects not in the map). Modeled via exponential distribution.

$$p_{short}(z_t^k | x_t, m) = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short}\, z_t^k} & if\ 0 \leq z_t^k \leq z_t^{k*} \\ 0 & otherwise \end{cases}$$

$$\eta = \frac{1}{1 - e^{-\lambda_{short}\, z_t^{k*}}}$$

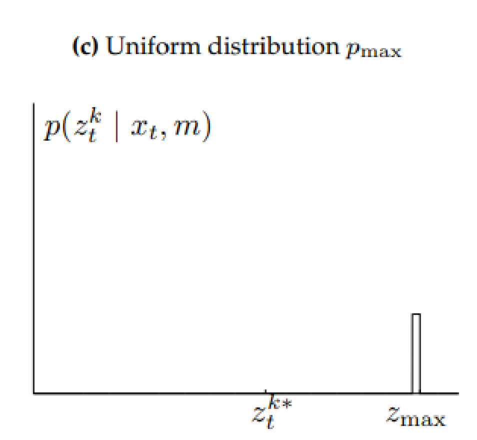
(b) Exponential distribution p_{short}



Recursive State Estimation – beam-based model

Well-known measurement failures happen on black or non-reflective objects or glass. In that case typically, a max range measurement is returned.

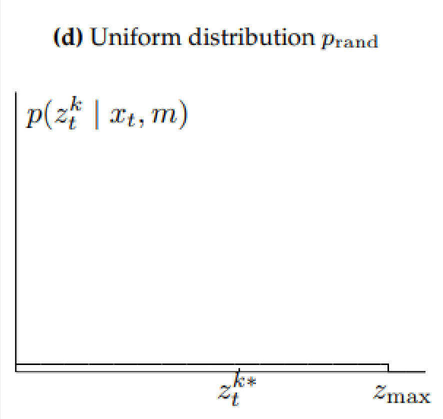
$$p_{max}\left(z_t^k \middle| x_t, m\right) = \begin{cases} 1 & z_k^t = z_{max} \\ 0 & otherwise \end{cases}$$



Recursive State Estimation – beam-based model

Random measurements that are entirely unexplained may occur (although not frequently):

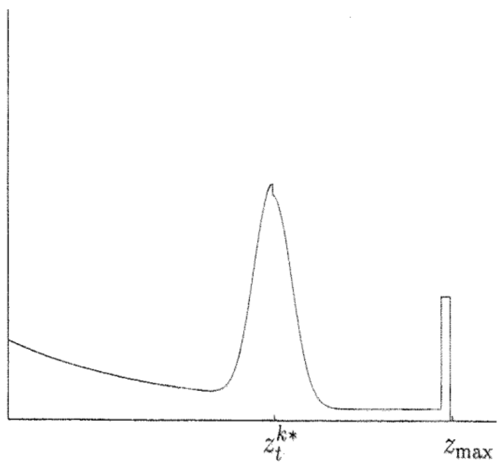
$$p_{rand}\left(z_t^k \middle| x_t, m\right) = \begin{cases} \dfrac{1}{z_{max}} & if\ 0 \leq z_k^t < z_{max} \\ 0 & otherwise \end{cases}$$



Recursive State Estimation – beam-based model

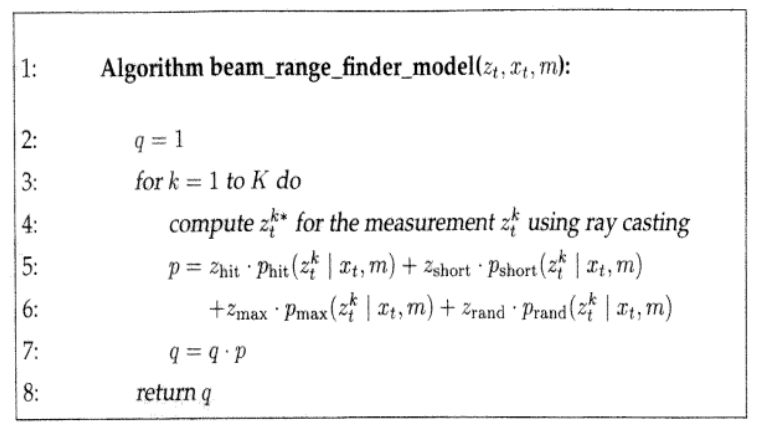
Taking the weighted average of these distributions yields the overall model:

$$p\left(z_t^k \middle| x_t, m\right) = z_{hit}\, p_{hit}\left(z_t^k \middle| x_t, m\right) + z_{short} p_{short}\left(z_t^k \middle| x_t, m\right) + z_{max} p_{max}\left(z_t^k \middle| x_t, m\right) + z_{rand} p_{rand}\left(z_t^k \middle| x_t, m\right)$$



Recursive State Estimation – beam-based model

Probability of entire measurement vector by assuming **independence** of rays.



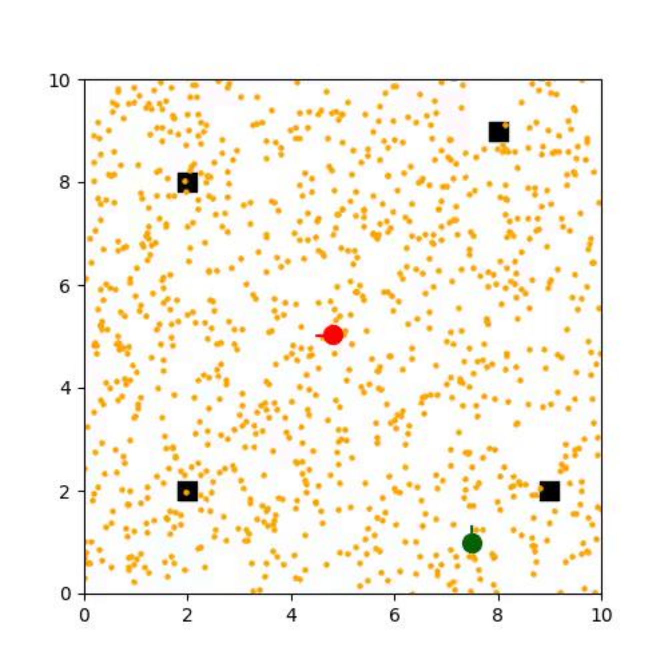
```
Algorithm beam_range_finder_model($z_t, x_t, m$):

    $q = 1$
    for $k = 1$ to $K$ do
        compute $z_t^{k*}$ for the measurement $z_t^k$ using ray casting
        $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$
            $+ z_{\max} \cdot p_{\max}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$
        $q = q \cdot p$
    return $q$
```

- Does this assumption really hold true?

Particle filter: example animation

- Sensor data not explicitly shown in this animation
- Particles are resampled based on the sensor model



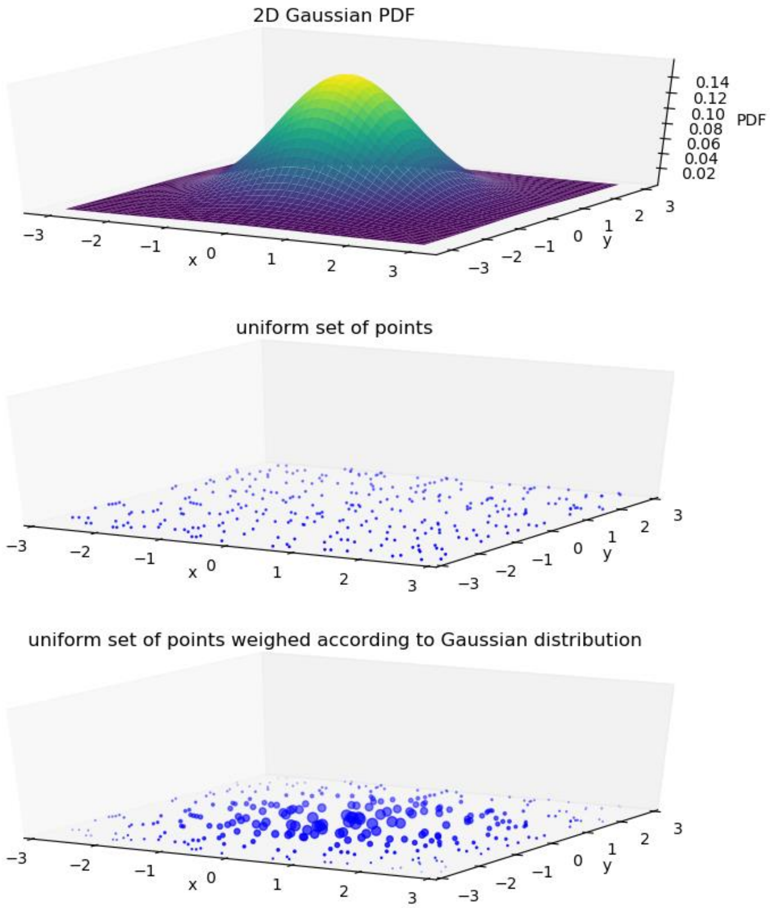
Estimating the pose from a particle filter

Remember:

  - $p(x_{0:t}|z_{0:t}) \approx \sum_{i=1}^{N_s} w_t^i \; \delta(x_{0:k} - x_{0:k}^i)$

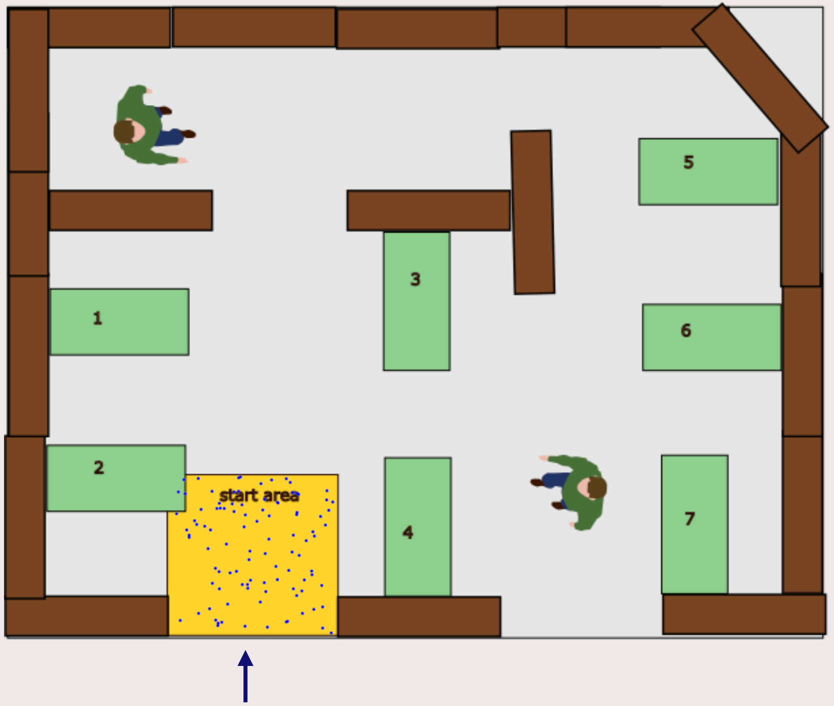
- $E(x_t) \approx \sum_{i=1}^{N} w_t^i \; x_{0:t}^i$

- Is this a good pose estimate to use?
  - When is it, when is it not?



Initializing a particle filter

- Decide on the number of particles N
- Draw N particles from the initial distribution
- Run consecutive update/prediction steps!



Initial distribution

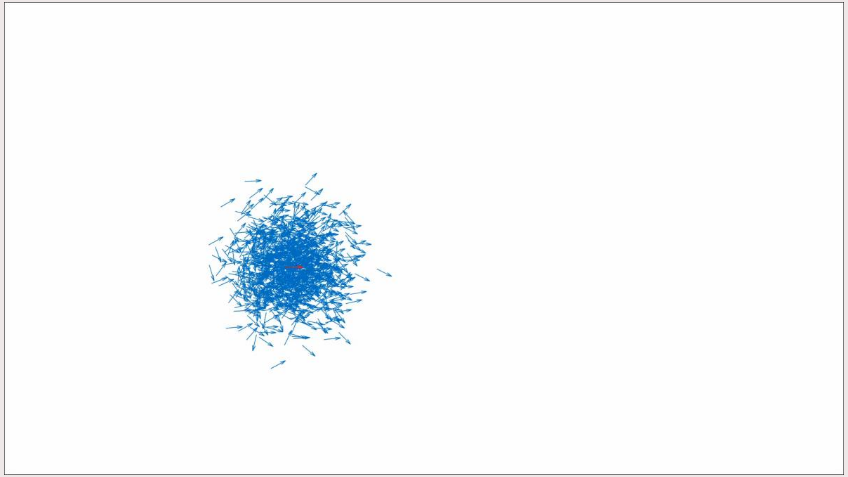
This week's exercise

- Particle filter predictions

- Generate new samples from the proposal distribution $p(x_{t+1}|x_t)$
  - Skip Bayes' update

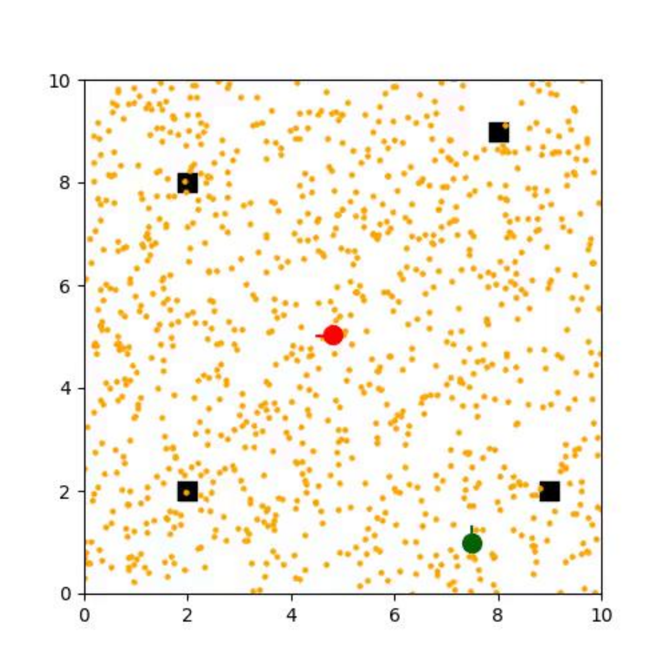
- What happens to the prediction over time?

- What would be the benefit of adding measurement updates?



Next week’s exercise

- Measurement data updates!
- Update the weight of our particles using Bayes’ rule.
- Close the loop by resampling the particles.
  - Fully functioning particle filter!



0
2
4
6
8
10
0
2
4
6
8
10