Assignment 4.3

Monte-Carlo Localization Using A Particle Filter

Technische Universität Berlin

Particle Filter

- A popular instance of the Bayes Filter (besides Kalman Filters, Discrete Filters, Hidden Markov Models, etc.)

$$Bel(x_t) = \eta \ P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1}) \ Bel(x_{t-1}) \ dx_{t-1}$$

x: state
z: observation
u: action

- Basic Principle:
  - Set of state hypotheses („particles“)
  - Survival-of-the-fittest

- Efficiently represent non-Gaussian distributions

Mobile Robot Localization

- Each particle is a potential pose of the robot
- **Prediction step**: Proposal distribution is the motion model of the robot
- **Correction step**: The observation model is used to compute the importance weight
- **Resampling step**: A new set of particles is drawn according to their importance weights

Particle Filter Algorithm

1. Algorithm **particle_filter**(S_{t-1}, u_{t-1} z_t):
2. $S_t = \emptyset, \quad \eta = 0$
3. **For** $i = 1 \ldots n$ — *Generate new samples*
4. Sample index $j(i)$ from the discrete distribution given by w_{t-1} — **Resampling**
5. Sample x_t^i from $p(x_t \mid x_{t-1}, u_{t-1})$ using $x_{t-1}^{j(i)}$ and u_{t-1} — **Motion model**
6. $w_t^i = p(z_t \mid x_t^i)$ — *Compute importance weight* — **Sensor model**
7. $\eta = \eta + w_t^i$ — *Update normalization factor*
8. $S_t = S_t \cup \{< x_t^i, w_t^i >\}$ — *Insert*
9. **For** $i = 1 \ldots n$
10. $w_t^i = w_t^i / \eta$ — *Normalize weights*

Motion Model *p(x|x‘,u)*

- In practice, one often finds two types of motion models:
  - **Odometry-based** ***(what we‘ll implement)***
  - **Velocity-based (dead reckoning)**
- Odometry-based models are used when systems are equipped with wheel encoders
- Velocity-based models have to be applied when no wheel encoders are given
- They calculate the new pose based on the velocities and the time elapsed

Odometry Model "Probabilistic Robotics", p. 132

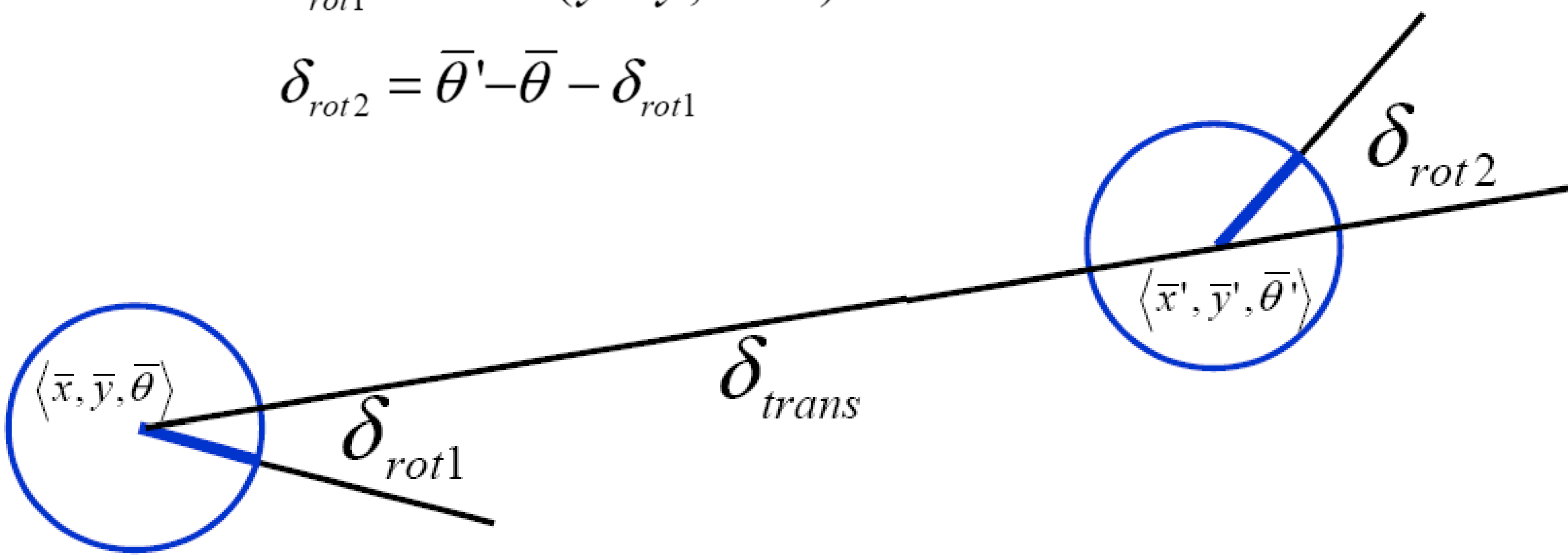
- Robot moves from $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ to $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$

- Odometry information $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle$

$$\delta_{trans} = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2}$$

$$\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$

$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$



Noise Model for Odometry

- The measured motion is given by the true motion corrupted with noise:

$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}|}$$

$$\hat{\delta}_{trans} = \delta_{trans} + \varepsilon_{\alpha_3 |\delta_{trans}| + \alpha_4 |\delta_{rot1} + \delta_{rot2}|}$$

$$\hat{\delta}_{rot2} = \delta_{rot2} + \varepsilon_{\alpha_1 |\delta_{rot2}| + \alpha_2 |\delta_{trans}|}$$

- In practice, the parameters have to be learned

Sample Odometry Motion Model

1. Algorithm **sample_motion_model**(u, x):

   $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle, x = \langle x, y, \theta \rangle$

1. $\hat{\delta}_{rot1} = \delta_{rot1} + \text{sample}(\alpha_1 \mid \delta_{rot1} \mid + \alpha_2\ \delta_{trans})$
2. $\hat{\delta}_{trans} = \delta_{trans} + \text{sample}(\alpha_3\ \delta_{trans} + \alpha_4\ (\mid \delta_{rot1} \mid + \mid \delta_{rot2} \mid))$
3. $\hat{\delta}_{rot2} = \delta_{rot2} + \text{sample}(\alpha_1 \mid \delta_{rot2} \mid + \alpha_2\ \delta_{trans})$

4. $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
5. $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
6. $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

7. Return $\langle x', y', \theta' \rangle$

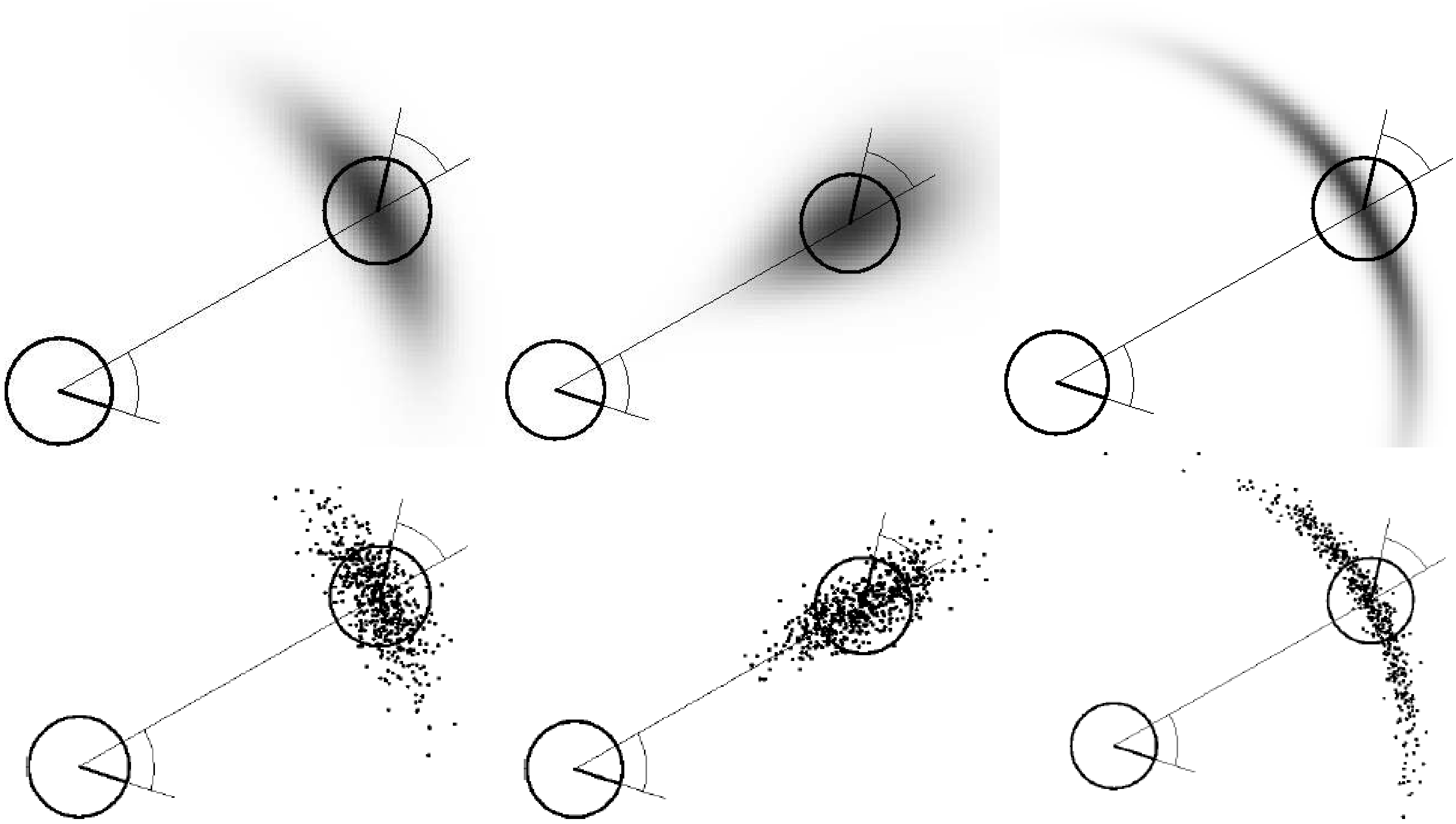
sample_normal_distribution

Examples (Odometry-based)

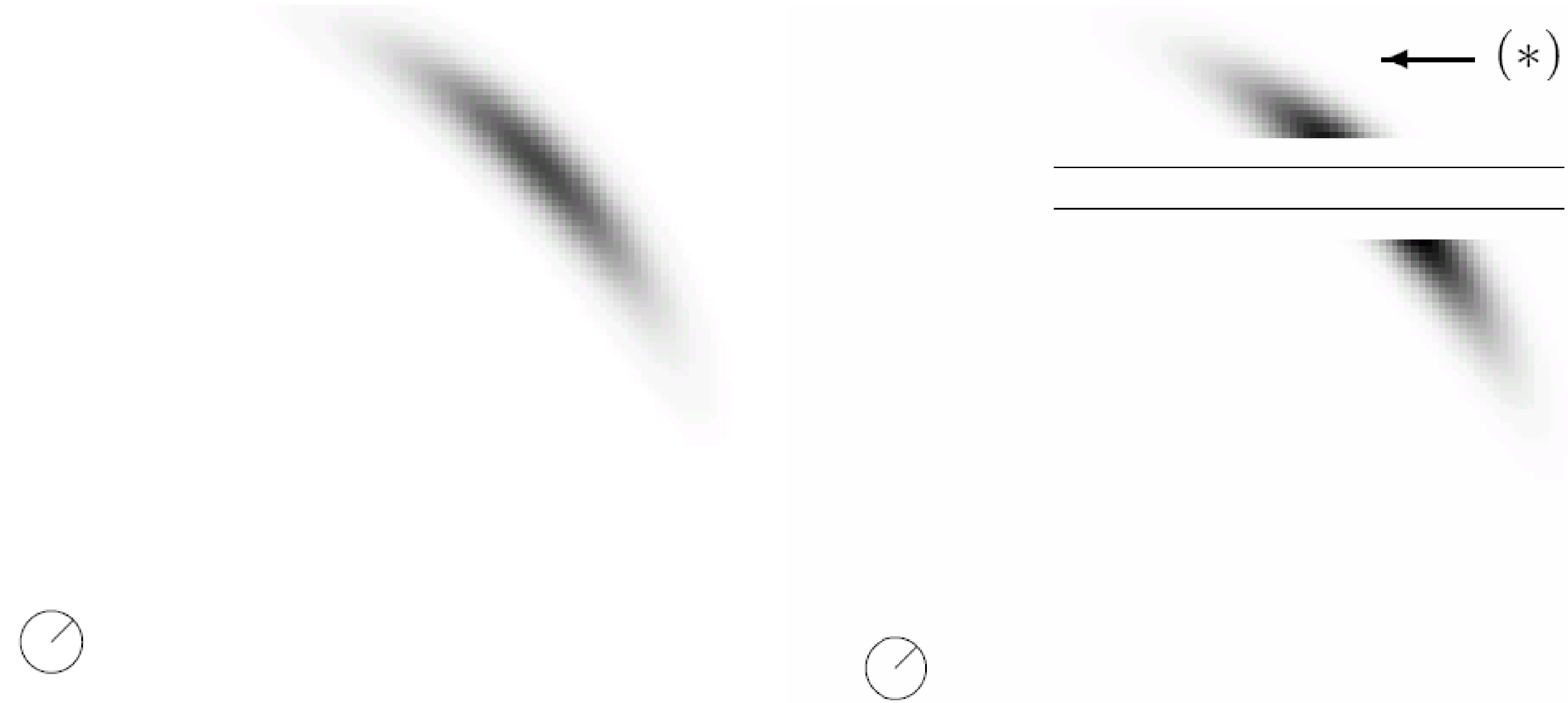
$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}|}$$

$$\hat{\delta}_{trans} = \delta_{trans} + \varepsilon_{\alpha_3 |\delta_{trans}| + \alpha_4 |\delta_{rot1} + \delta_{rot2}|}$$

$$\hat{\delta}_{rot2} = \delta_{rot2} + \varepsilon_{\alpha_1 |\delta_{rot2}| + \alpha_2 |\delta_{trans}|}$$



Map-consistent Motion Model



$$p(x \mid u, x') \neq p(x \mid u, x', m)$$

- Approximation (takes only final pose into account):

$$p(x \mid u, x', m) = \eta \; p(x \mid m) \; p(x \mid u, x')$$

Sensor Model *p(z|x,m)*

- In practice, one often finds two types of sensor models:
  - **Beam-based Proximity Model**
  - **Likelihood Field / Endpoint Model / Scan-based Model** ***(what we'll implement)***

- Scan z consists of K measurements

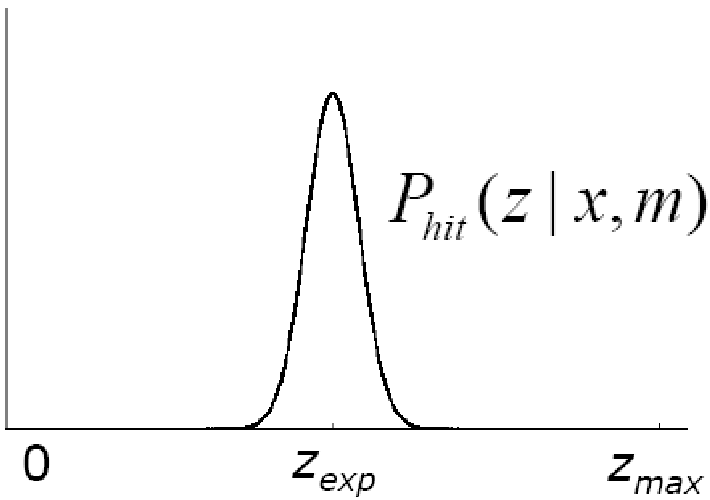
$$z = \{z_1, \ldots, z_K\}$$

- Independence assumption:

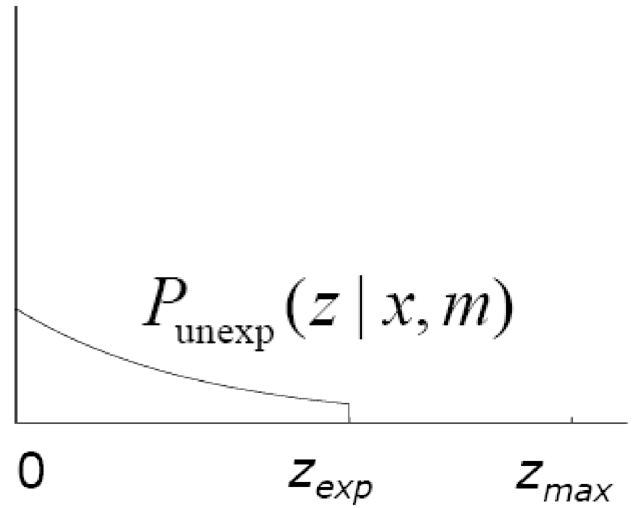
$$p(z \mid x, m) = \prod_{k=1}^{K} p(z_K \mid x, m)$$

Beam-based Proximity Model *(just for completeness)*

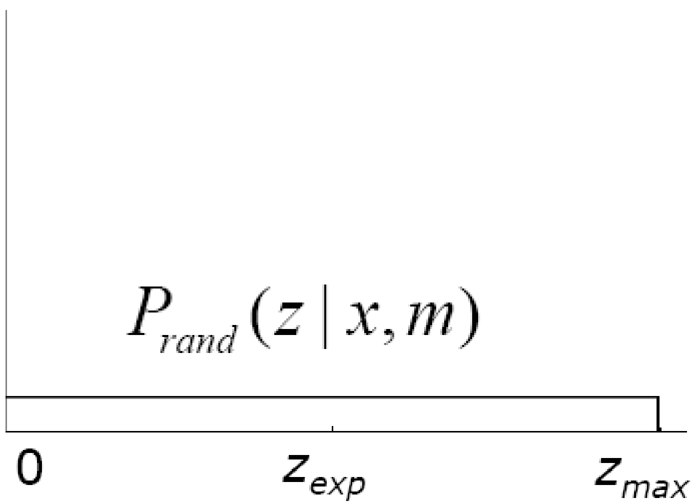
Measurement noise



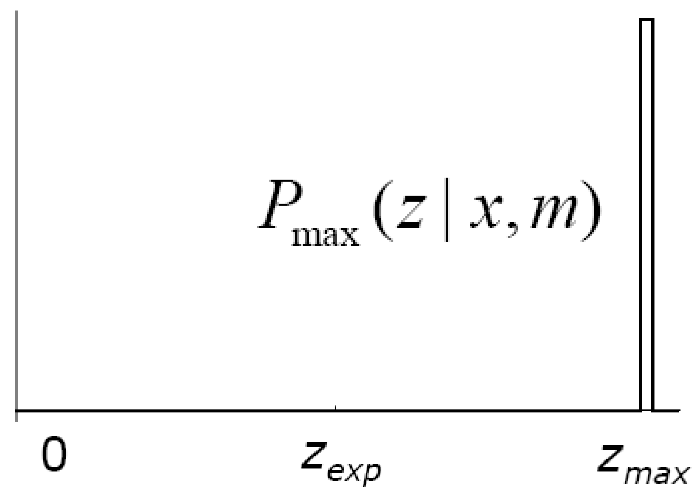
Unexpected obstacles



Random measurement

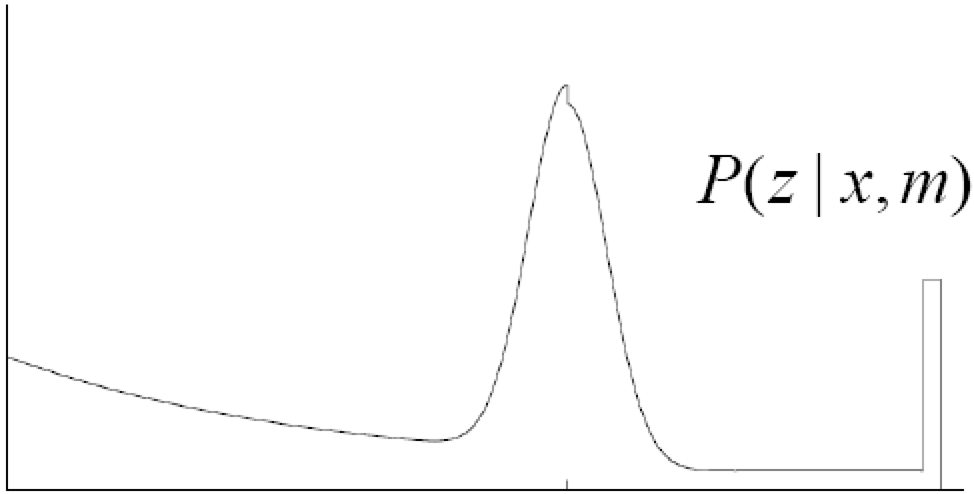


Max range



Beam-based Proximity Model *(just for completeness)*

- Resulting Mixture Density:



$$P(z \mid x, m) = \begin{pmatrix} \alpha_{\mathrm{hit}} \\ \alpha_{\mathrm{unexp}} \\ \alpha_{\mathrm{max}} \\ \alpha_{\mathrm{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} P_{\mathrm{hit}}(z \mid x, m) \\ P_{\mathrm{unexp}}(z \mid x, m) \\ P_{\mathrm{max}}(z \mid x, m) \\ P_{\mathrm{rand}}(z \mid x, m) \end{pmatrix}$$

- Not smooth for small obstacles and at edges
- Not very efficient

Scan-based Model *(what we'll implement)*

"Probabilistic Robotics", p. 169

"Instead of following along the beam, just check the end point."

- Probability is a mixture of …
  - a Gaussian distribution with mean at distance to closest obstacle,
  - a uniform distribution for random measurements, and
  - a small uniform distribution for max range measurements.

inferred from map

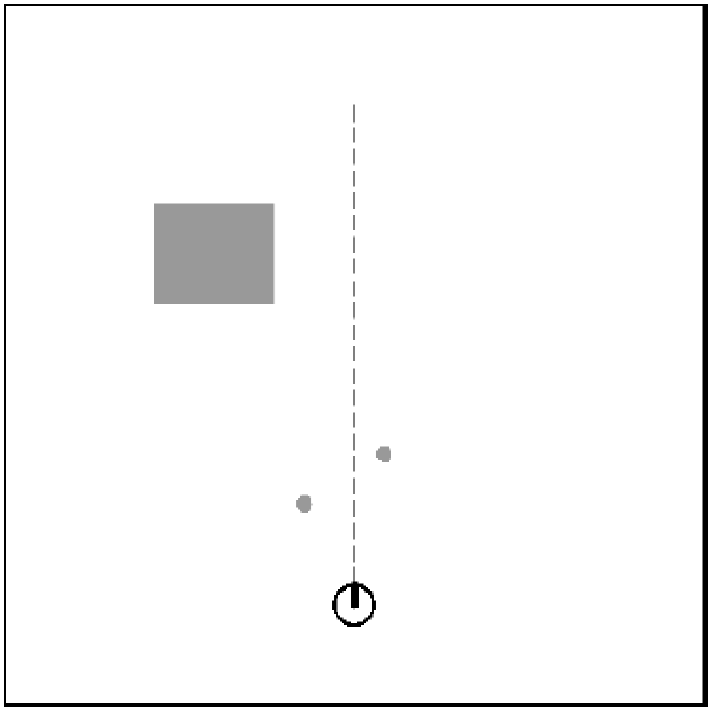
not used in the assignment!

$$p(z_k \mid x, m) = \underbrace{z_{hit} \cdot p_{hit} + z_{rand}}_{\text{weighting } z_{hit}+z_{rand}=1,\ (z_{max}=0)} \cdot p_{rand} + z_{\max} \cdot p_{\max}$$

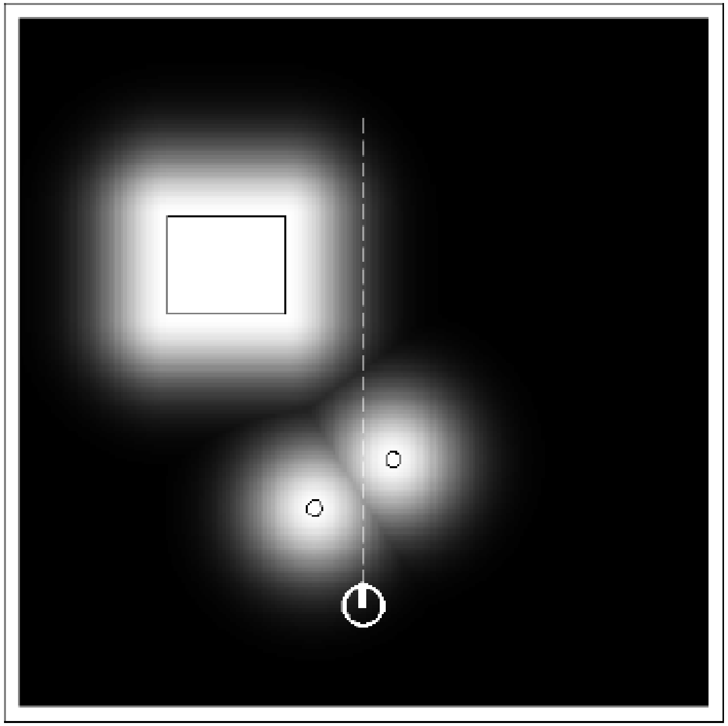
we assume $p_{rand}=1$

- This probability is pre-calculated (for any possible measurement) and stored in the likelihood field

Example

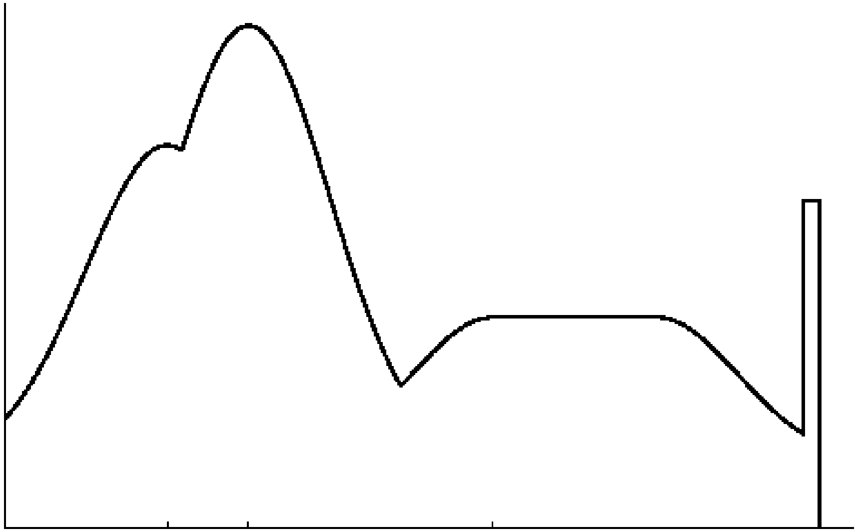


Map m



Likelihood field

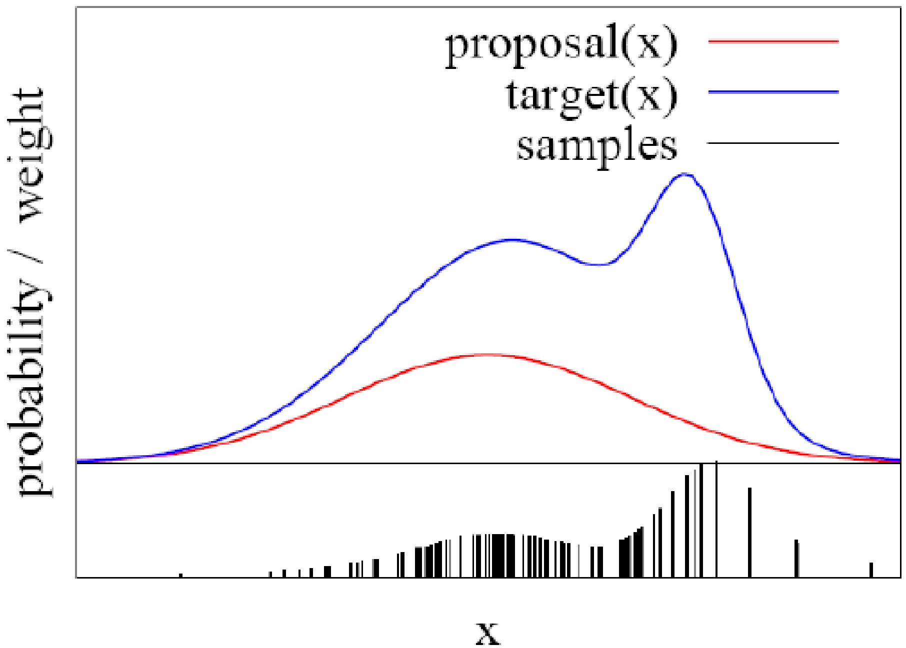
$P(z|x,m)$



Properties of Scan-based Model

- Ignores physical properties of beams! (explains measurement with distance to the closest obstacle)
- Highly efficient, uses 2D tables only
- Smooth w.r.t. to small changes in robot position
- Allows gradient descent, scan matching

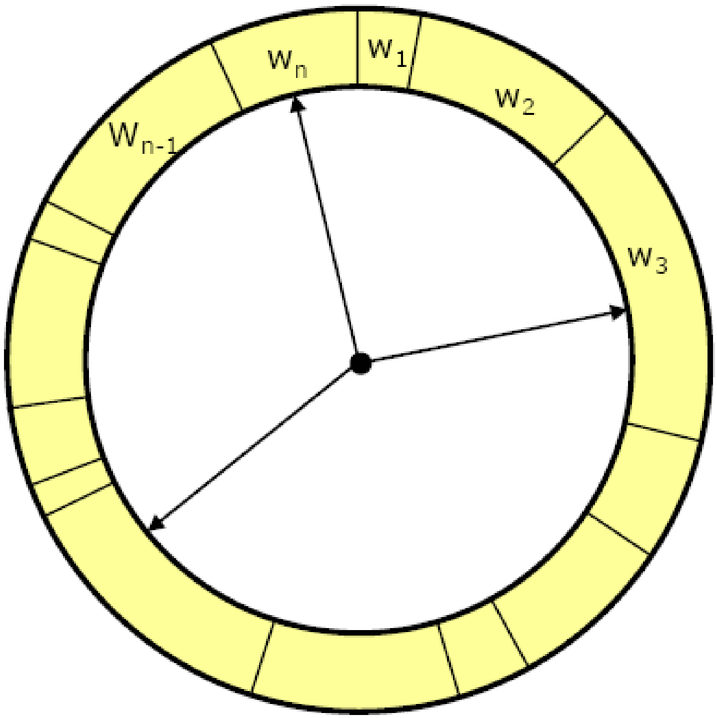
Importance Sampling Principle

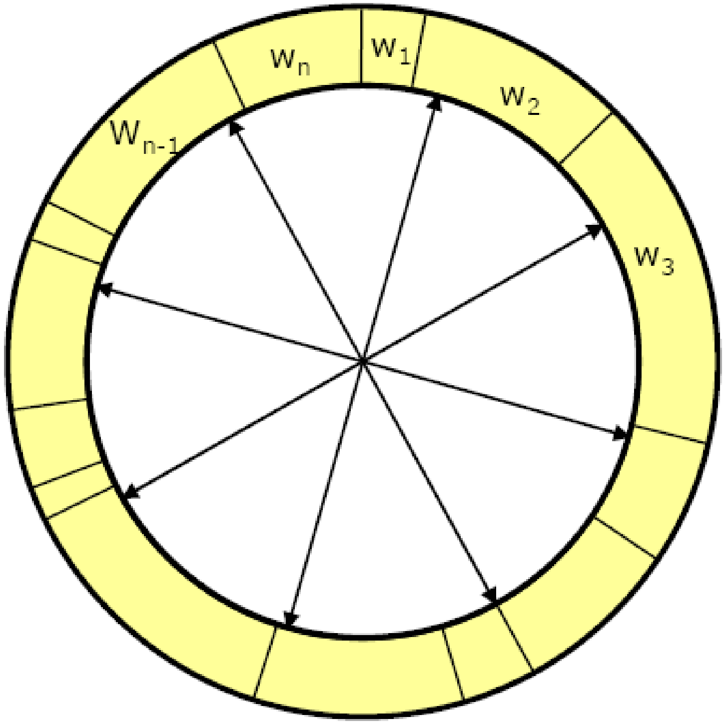


- We can even use a different distribution *g* (*proposal*) *to* generate samples from *f* (*target*)
- By introducing an importance weight w = *f / g*, we can account for the “differences between *g and f* ”

Resampling "Probabilistic Robotics", p. 108

"Replace unlikely samples by more likely ones"





- Roulette wheel
- Binary search, n log n

- Stochastic universal sampling
- Systematic resampling
- Linear time complexity
- Easy to implement, low variance

RBO

Stochastic Universal Sampling

(aka "Low variance sampling" in *Probabilistic Robotics* chapter 4.3)

Algorithm **systematic_resampling**(S,n):

$S' = \emptyset, c_1 = w^1$

For $i = 2 \ldots n$ — *Generate cdf*

$\quad c_i = c_{i-1} + w^i$

$u_1 \sim U]0, n^{-1}], i = 1$ — *Initialize threshold*

For $j = 1 \ldots n$ — *Draw samples …*

$\quad$ **While** $(u_j > c_i)$ — *Skip until next threshold reached*

$\quad\quad i = i + 1$

$\quad S' = S' \cup \{ < x^i, n^{-1} > \}$ — *Insert*

$\quad u_{j+1} = u_j + n^{-1}$ — *Increment threshold*

Return S'

ASSIGNMENT 4.3

Preliminaries

- Download the zip files *create_gui.zip* and *localization.zip* from ISIS and unzip to your ROS workspace
  - Dell laptops: /home/create/ros/
- Build the visualization tool

```
$ cd /home/create/ros/a4/create_gui
$ rosmake
```

- Build the localization node

```
$ cd /home/create/ros/a4/localization
$ rosmake
```

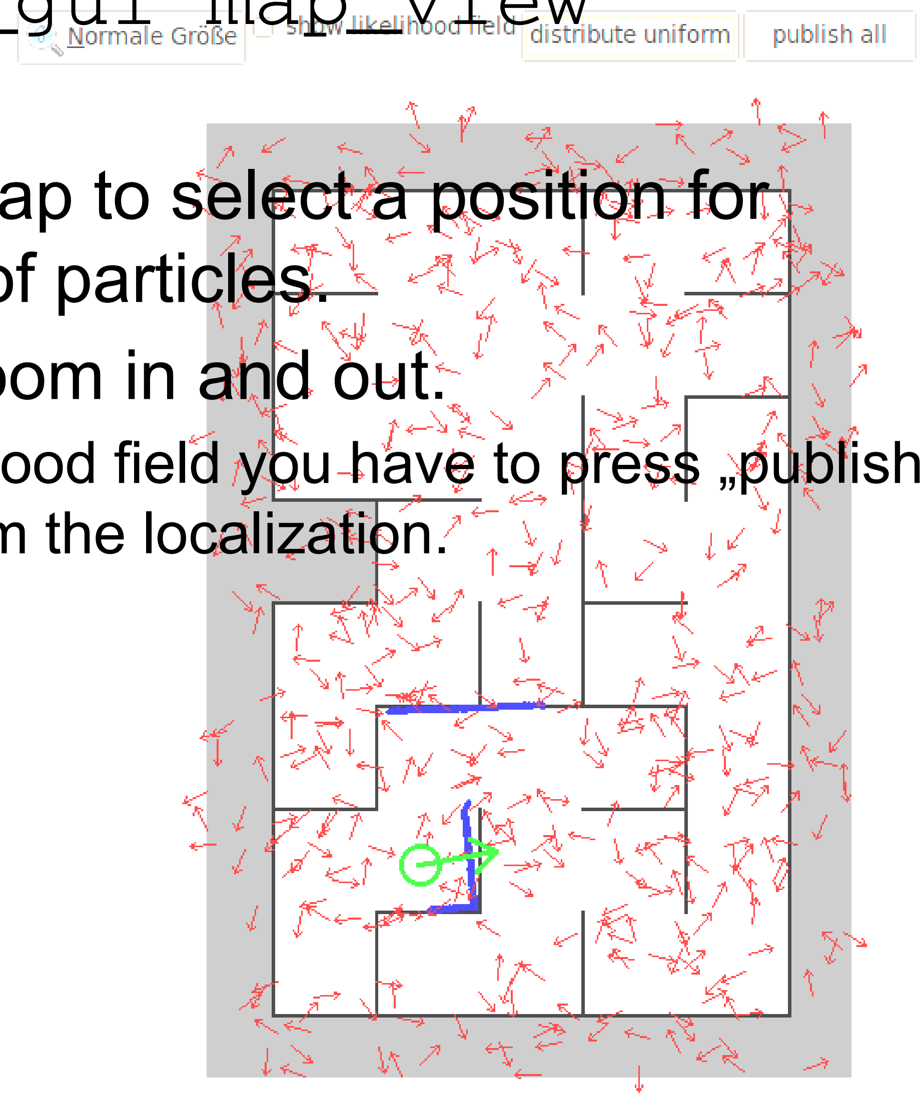
ROS Launch File

```
roslaunch localization mcl.launch
```

- Launches:
  - **particle_filter**: Node that you have to implement
  - **map_view**: visulization tool
  - **map_server:** Publishing the map your robot should localize itself in
  - **map_transform:** Static transformation between map and world frame
  - **rosbag**: Recorded test data

Visualization / Debugging Tool

```
rosrun create_gui map_view
```

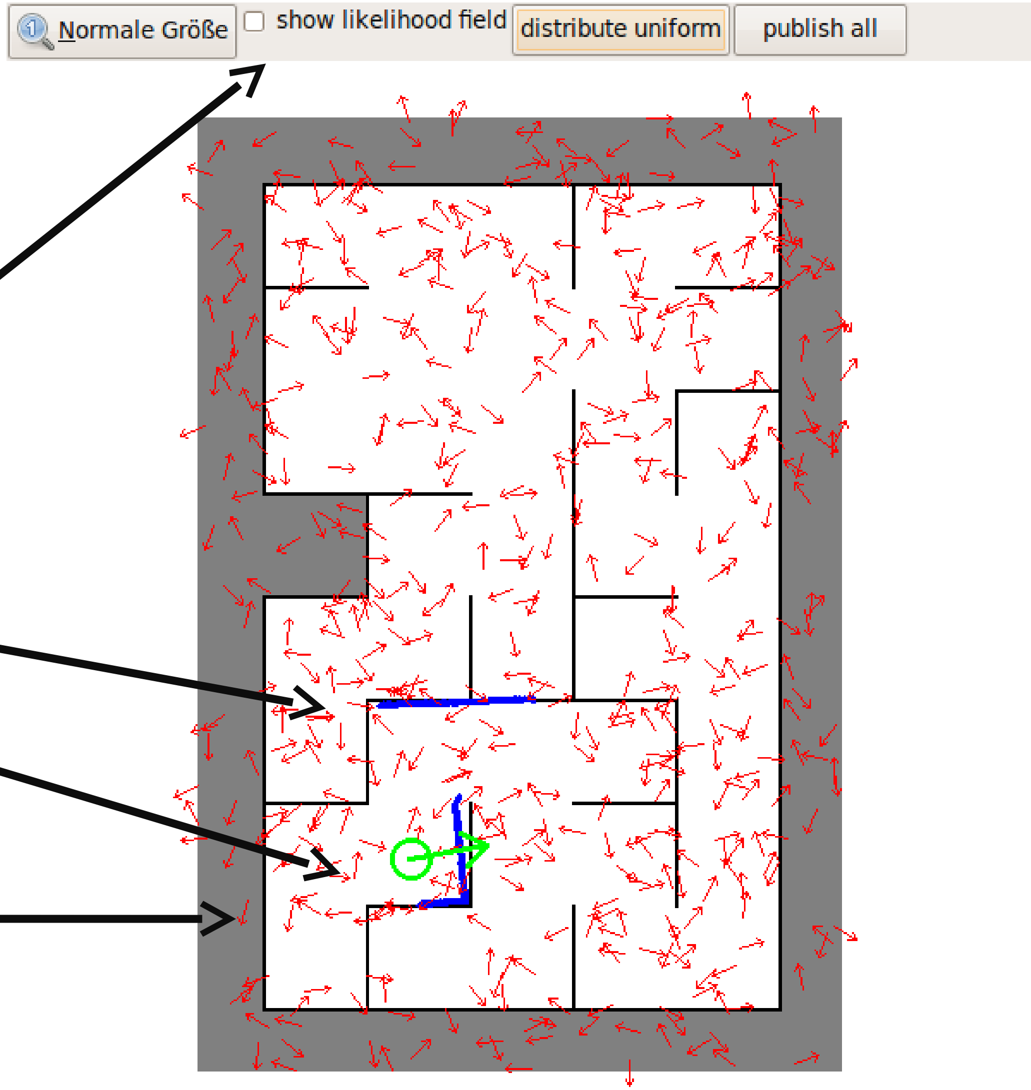


- Right click on the map to select a position for normal distribution of particles.
- Mouse wheel will zoom in and out.
  - If you select the likelihood field you have to press „publish all“ to get the data from the localization.

Visualization / Debugging Tool

Displays:

- Map
- likelihood field
- laser scan
- robot position
- robot path
- particles



Using the iRobot creates

Using the iRobot creates

- If you *want* you can record your own bag file (not mandatory for the assignment)

- Do not break them!
- Connect the robot and the Netbook to the charger when you put them into the locker.
- You should be able to connect via SSH to your netbook
  `ssh -X <color>.elektro.robotics.tu-berlin.de`
  - Experimental!
  - Please do not blame us for eduroam / network problems

Hokuyo Laser Range Finder

- URG-04LX-UG01
  - Range: 5,6m 240°
  - Scan rate: 10Hz
- http://www.ros.org/wiki/hokuyo_node

Steering the robot

You can drive them around using a GUI:

- Switch on the create robot (make sure the battery is charged)
- Start **`roscore`**
- Start the irobot driver:
  `rosrun irobot_create_2_1 driver.py`
- Start the GUI
  `rosrun create_gui button_move`.

Recording your own bag file

- Start create GUI as explained on the previous slide

- Additionally, Start the hokuyo driver
  `rosrun hokuyo_node hokuyo_node`
  - Publishes topic: /scan

- Use rosbag to record
  `rosbag record -a`