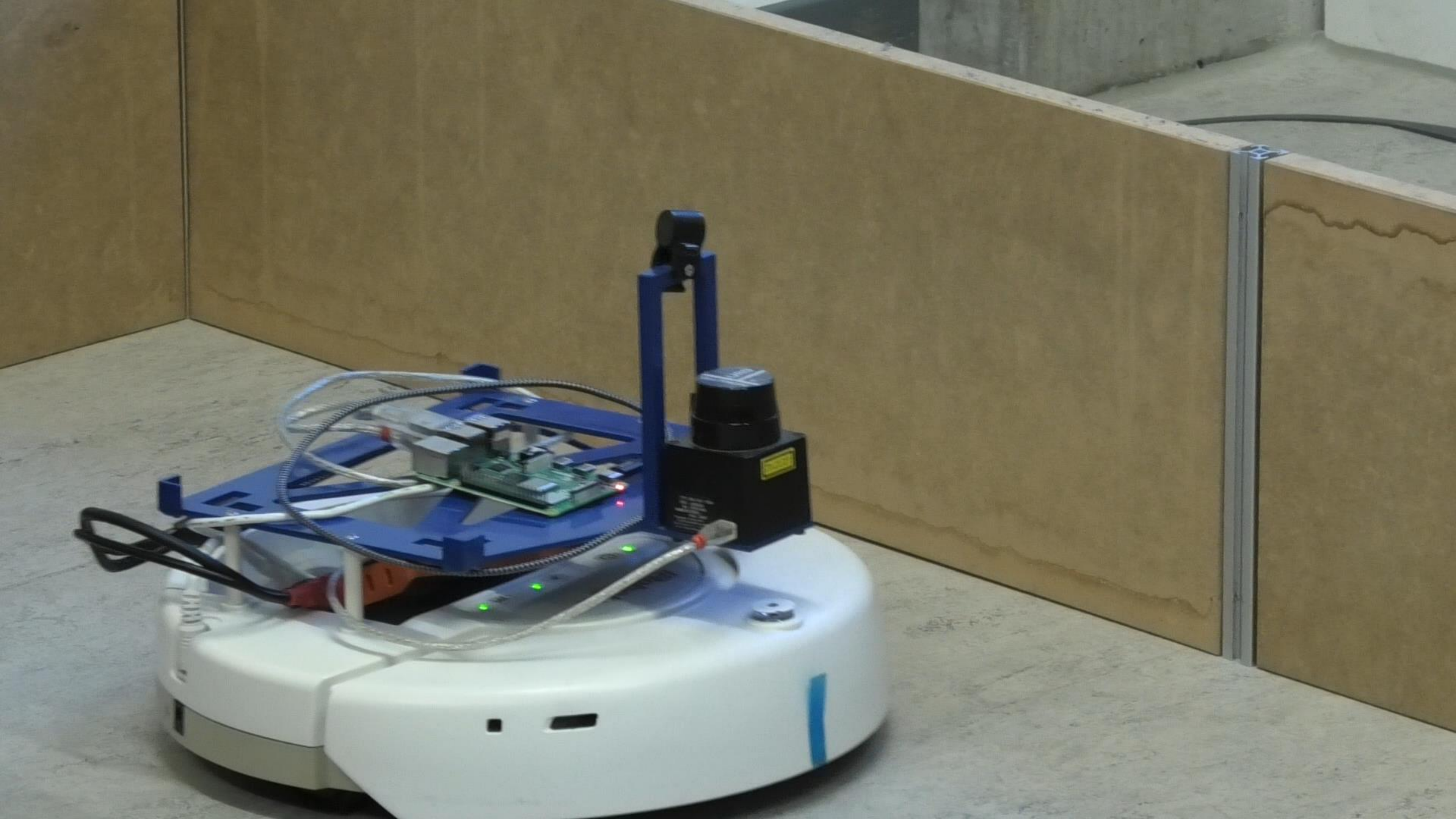
Localization

Aravind Battaje

Technische Universität Berlin



Particle Filter

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

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

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

x: state
z: observation
u: action

- 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 $w_{t-1}$
5.      Sample $x_t^i$ from $p(x_t \mid x_{t-1}, u_{t-1})$ using $x_{t-1}^{j(i)}$ and $u_{t-1}$
6.      $w_t^i = p(z_t \mid x_t^i)$                            Compute importance weight
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
```

Resampling

Motion model

Sensor model

Motion Model → $p(x_t|x_{t-1}, u_t)$

► In practice, one often finds two types of motion models:

- **Odometry-based** ***(what we'll implement)***
  - Used when systems are equipped with wheel encoders
- **Velocity-based (☠ dead reckoning ☠)**
  - Must 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

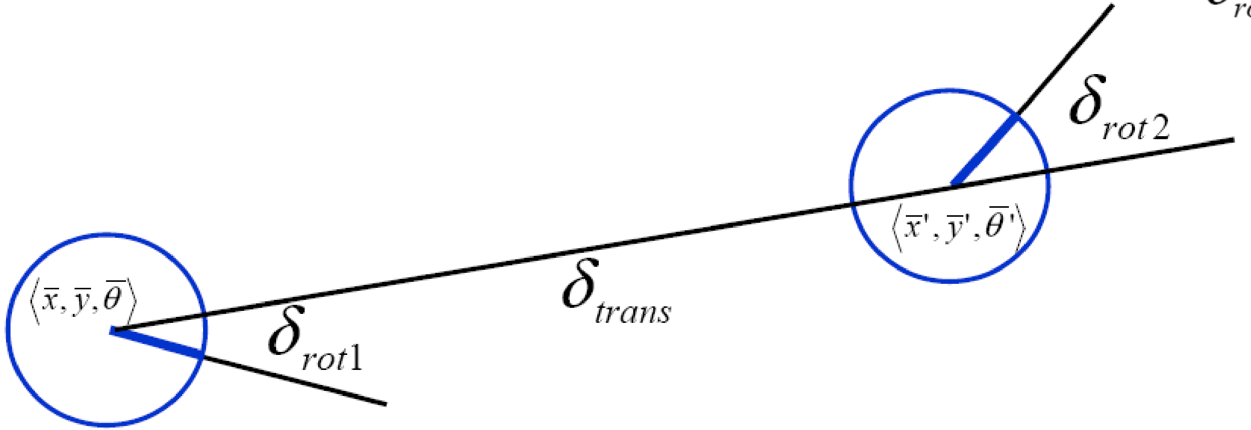
- Robot moves from $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ to $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$
- Odometry information

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

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

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

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


$\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$
δ_{rot1}
δ_{trans}
$\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$
δ_{rot2}

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 (alphas) must often be estimated using domain knowledge

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

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 $\langle x', y', \theta' \rangle$
```

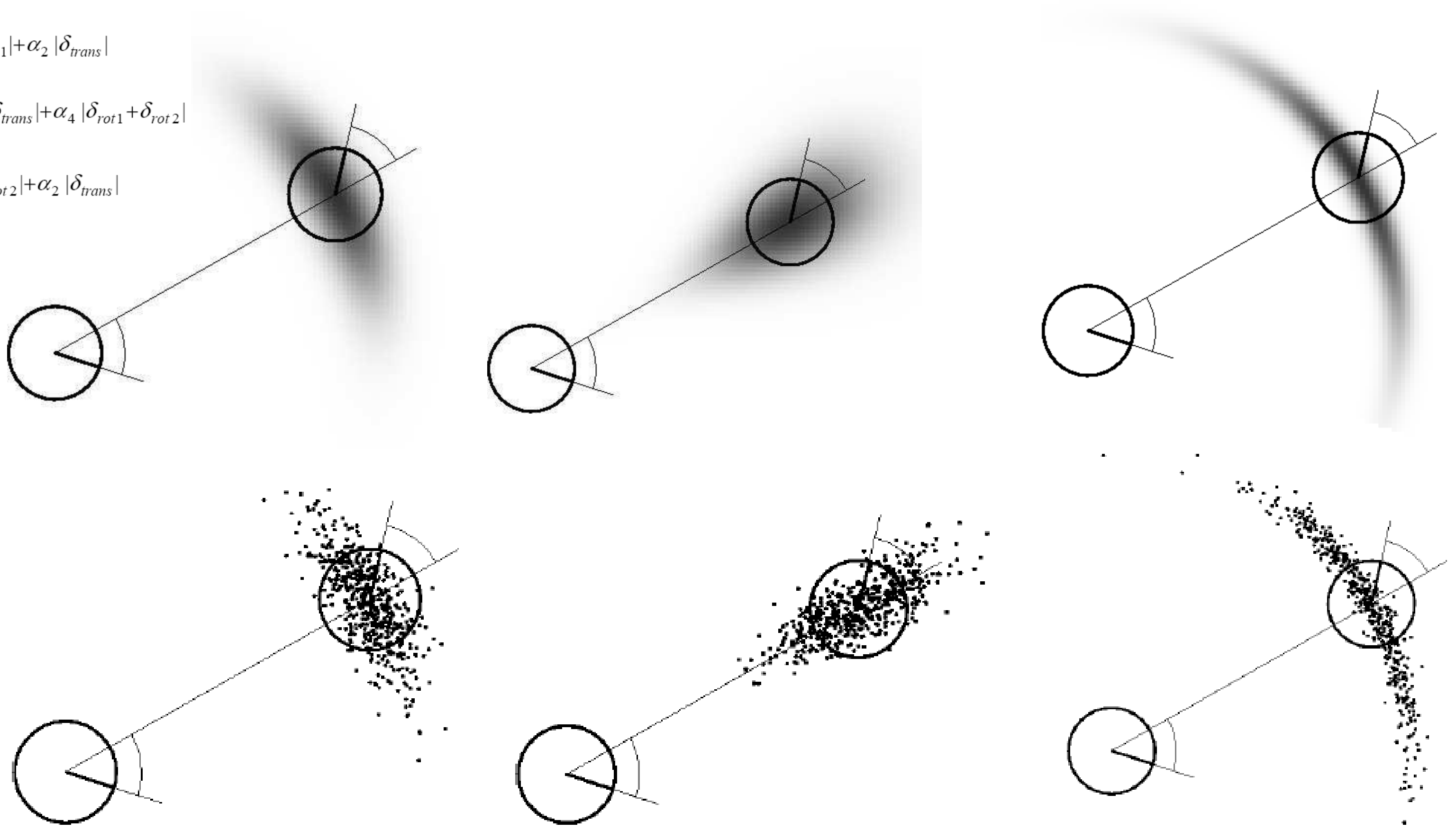
sample_normal_distribution

Examples (Odometry-based)

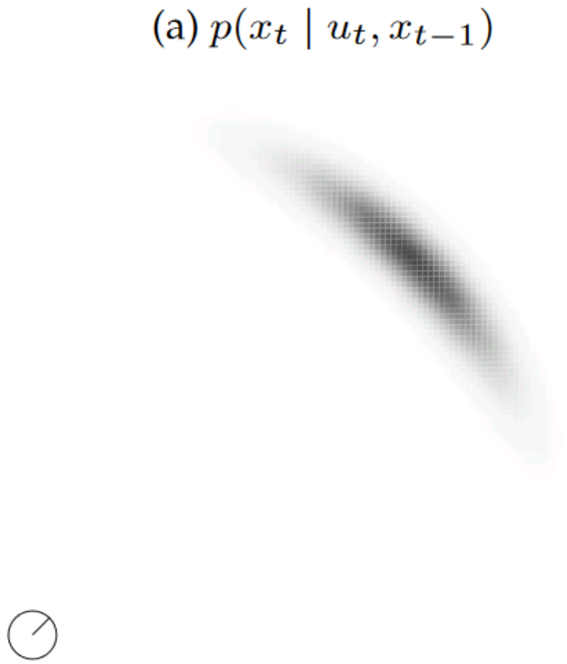
$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}|}$$

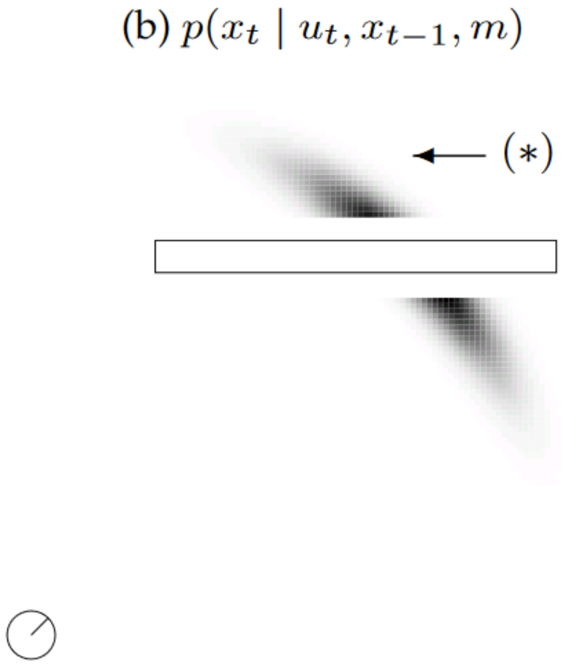
$$\hat{\delta}_{trans} = \delta_{trans} + \varepsilon_{\alpha_3 |\delta_{trans}| + \alpha_4 |\delta_{rot1} + \delta_{rot2}|}$$

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



Map-consistent Motion Model





- We approximate (taking only final pose into account):

$$p(x_t \mid u_t, x_{t-1}, m) \;=\; \eta \, p(x_t \mid m) \, p(x_t \mid u_t, x_{t-1})$$

Sensor Model → $p(z_t|x_t, m)$

- 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

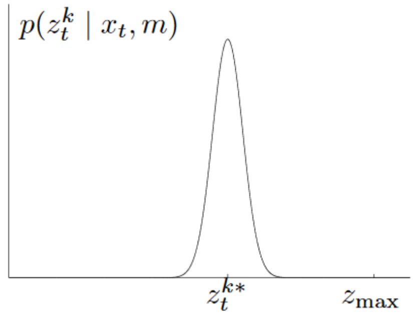
$$z_t = \{z_t^1, \ldots, z_t^K\}$$

- Independence assumption:

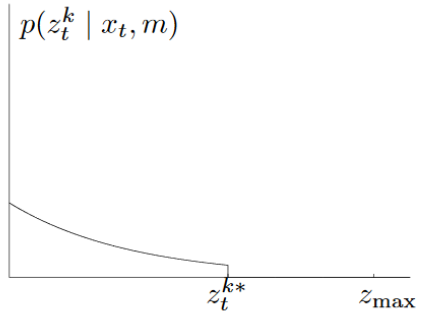
$$p(z_t \mid x_t, m) = \prod_{k=1}^{K} p(z_t^k \mid x_t, m)$$

Beam-based Proximity Model *(for completeness)*

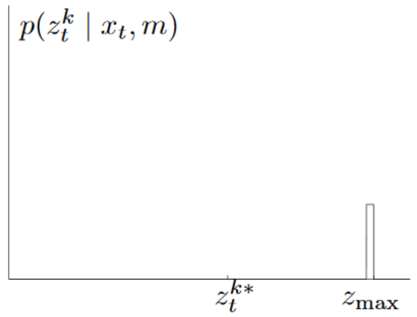
(a) Gaussian distribution p_{hit}



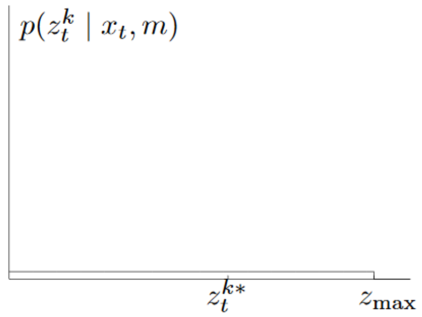
(b) Exponential distribution p_{short}



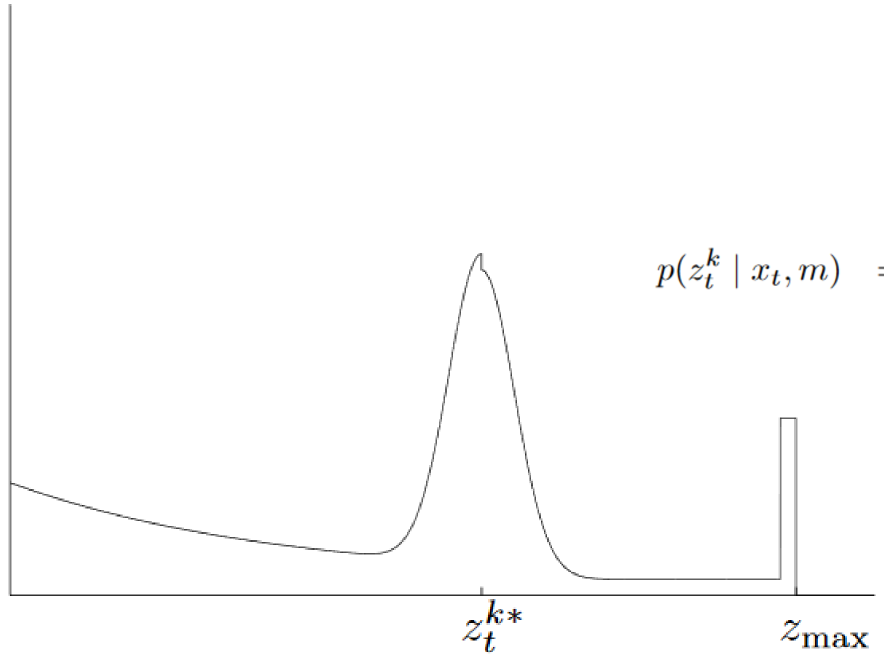
(c) Uniform distribution $p_{\max}$



(d) Uniform distribution p_{rand}



Beam-based Proximity Model *(for completeness)*



$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\max} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix}$$

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

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

"Probabilistic Robotics", p. 169

- 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.

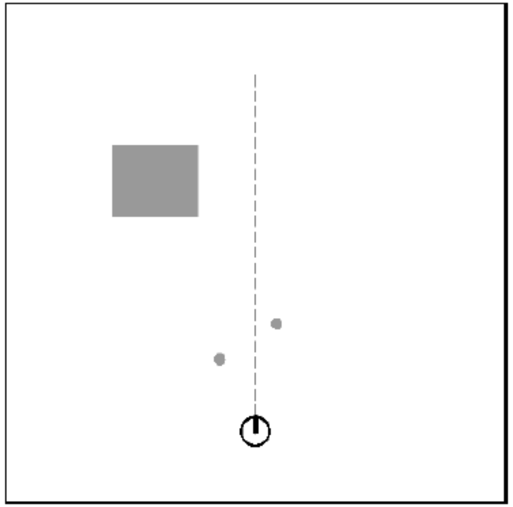
not used in the assignment!

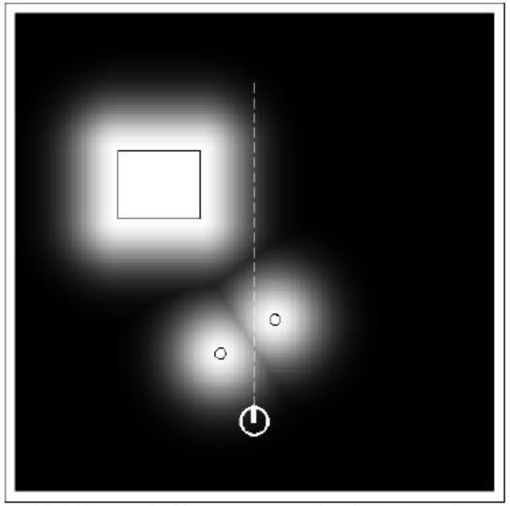
$$p(z_t^k \mid x_t, m) = \underbrace{z_{\text{hit}} \cdot p_{\text{hit}} + z_{\text{rand}} \cdot p_{\text{rand}}}_{\text{we assume } z_{rand}=1 \text{ and } (p_{hit}+p_{rand}=1)} + z_{\text{max}} \cdot p_{\text{max}}$$

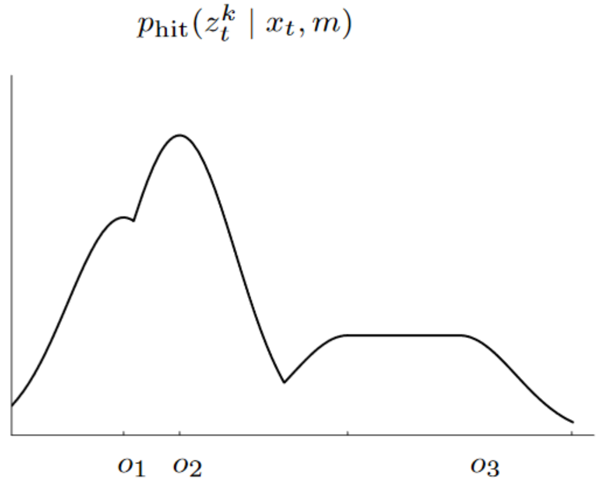
inferred from map

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

Example


Map *m*

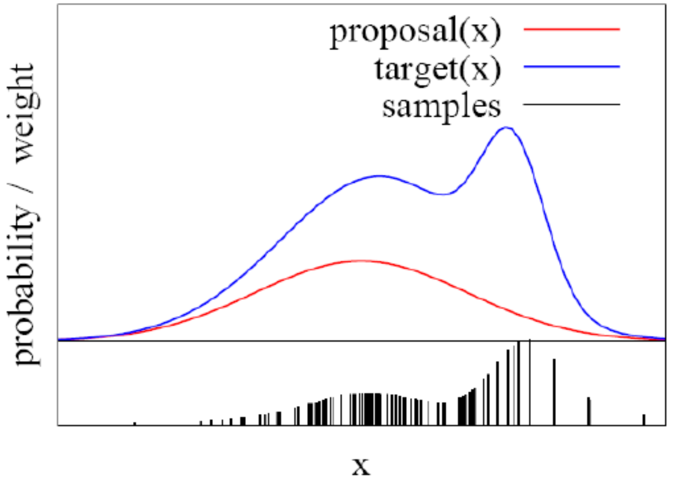

Likelihood field



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)

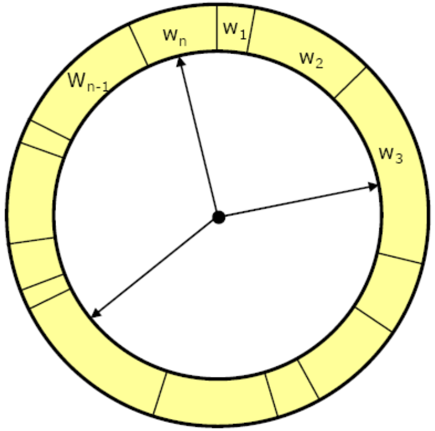
Importance Sampling Principle

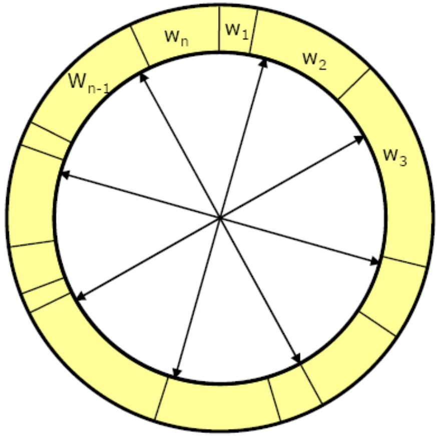


- We can even use a different distribution *g* (*proposal*) *to* generate samples from *f* (*target*)
- By introducing an importance weight w = *f / g*, we can account for the "differences between *g and f* "

Resampling

"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

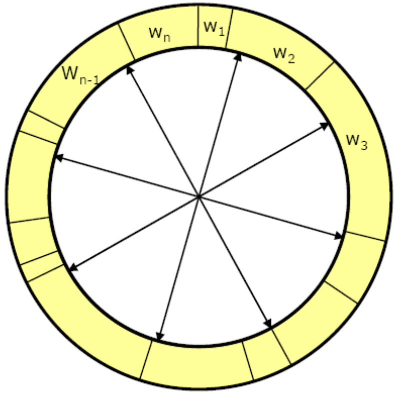
Stochastic Universal Sampling

```
1.  Algorithm systematic_resampling(S,n):

2.  $S' = \emptyset, c_1 = w^1$
3.  For $i = 2 \ldots n$                       Generate cdf
4.      $c_i = c_{i-1} + w^i$
5.  $u_1 \sim U]0, n^{-1}], i = 1$             Initialize threshold

6.  For $j = 1 \ldots n$                       Draw samples …
7.      While ( $u_j > c_i$ )                  Skip until next threshold reached
8.          $i = i + 1$
9.      $S' = S' \cup \{< x^i, n^{-1} >\}$     Insert
10.     $u_{j+1} = u_j + n^{-1}$               Increment threshold


11. Return S'
```



Running assignment code – preliminaries

- Download assignment workspace from ISIS
- Build the nodes required for localization
  (if not done already during assignment 4.2):

```
$ cd  /<path_to_dir>/ws_assignment4/
$ catkin_make
```

- Source the workspace [better: put into .bashrc]

```
$ source devel/setup.bash [or zsh]
```

Running assignment code – localization

```
$ roslaunch localization mcl.launch
```

- Launches:
  - **particle_filter**: Node that you must 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
- You implement your code in **`ws_assignment4`**`/src/localization/src/ParticleFilter.cpp`

Visualization / Debugging Tool

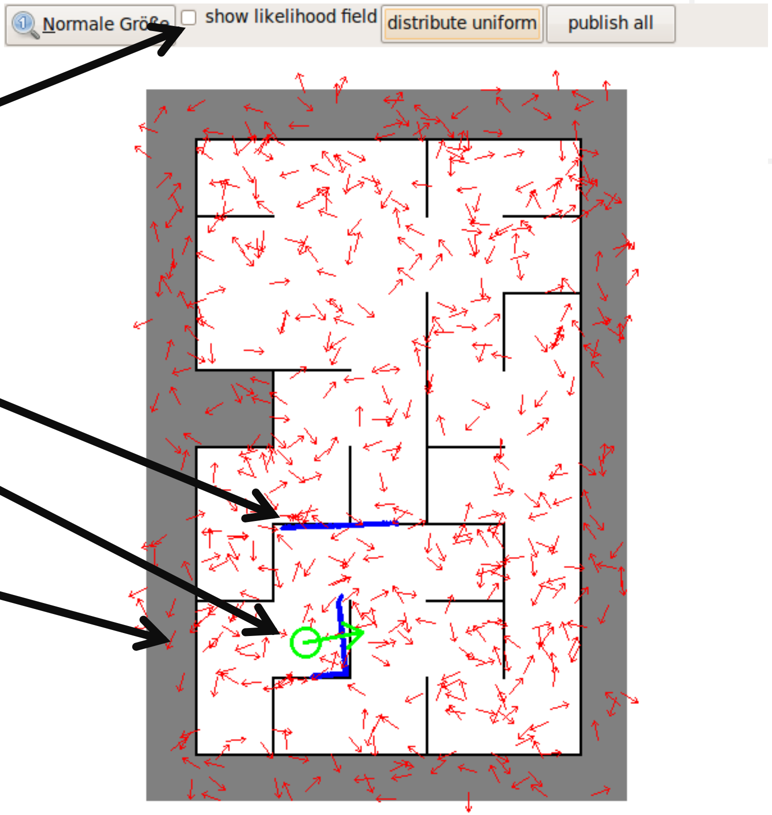
```
$ rosrun create_gui map_view
```

- 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 must press „publish all“ to get the data from the localization.

Visualization / Debugging Tool

Displays:

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



On the real robot