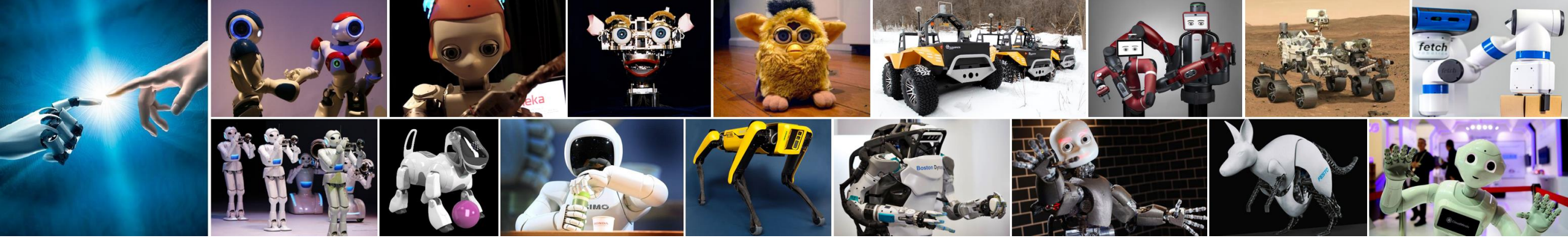
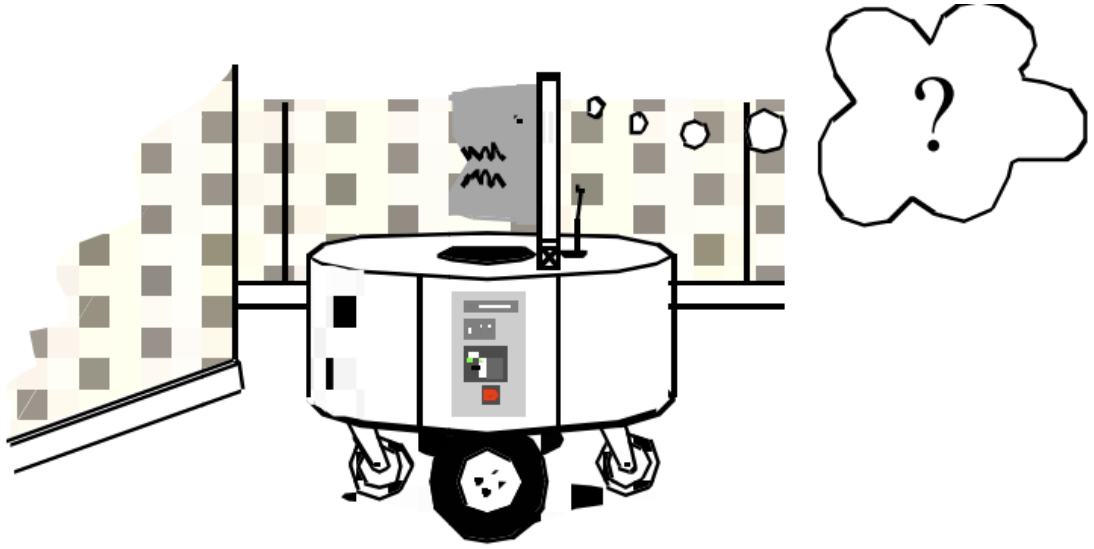


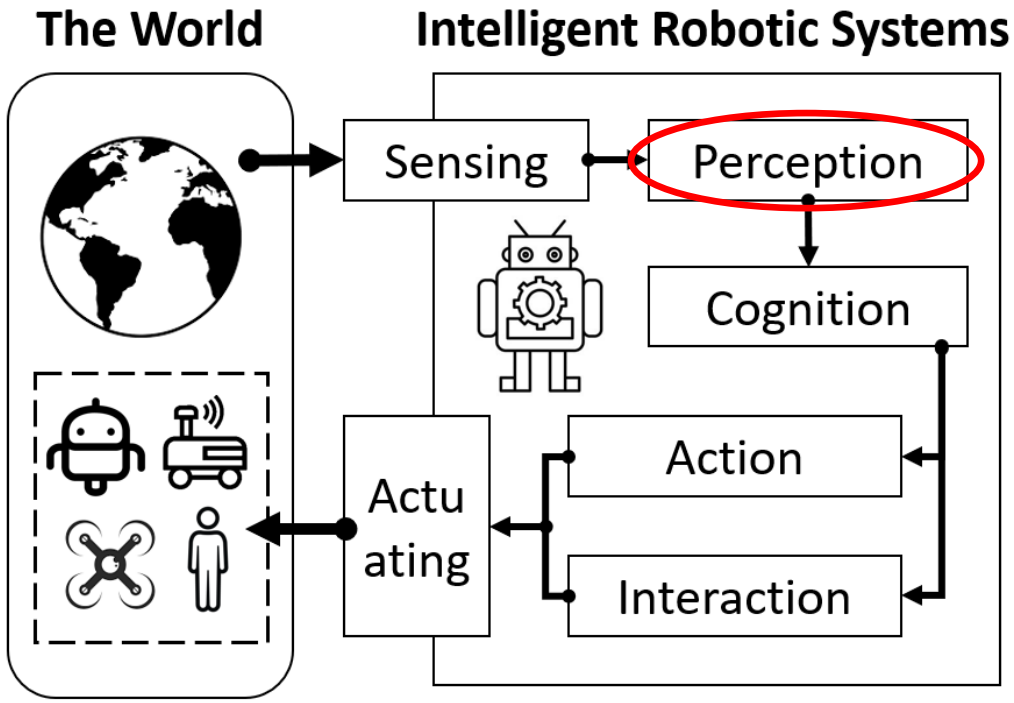
COMPSCI-603: Robotics

Bayes Filters

Partial slide courtesy by C. Stachniss, J. Biswas, and L. Parker

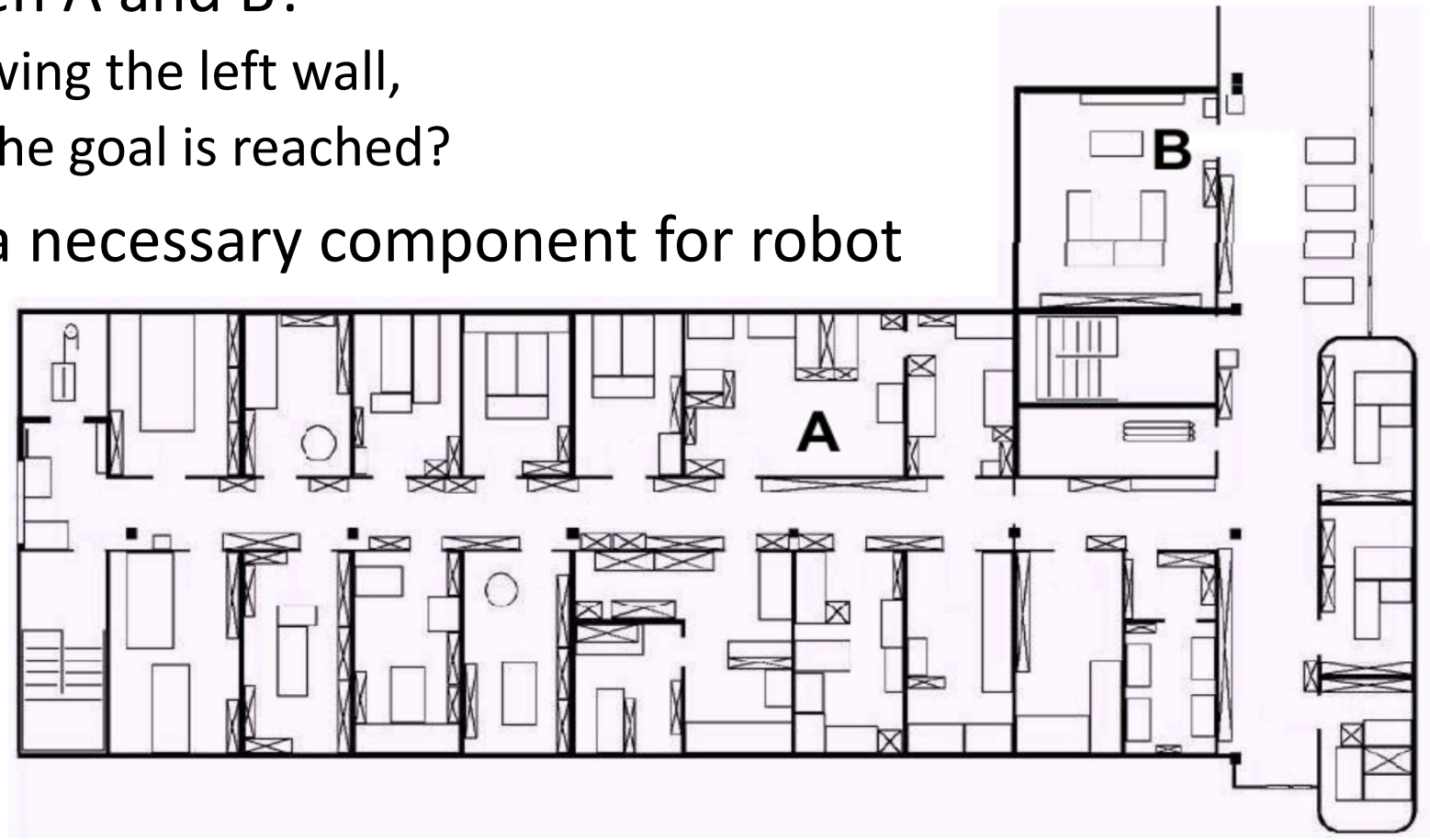
Localization: Where Am I?





Localization and Navigation

- How to navigate between A and B?
  - Possible by always following the left wall,
  - But how to detect that the goal is reached?
- Localization is typically a necessary component for robot navigation:
  - A robot must identify whether it reaches the goal position.
  - A robot often needs to know its location for path planning.

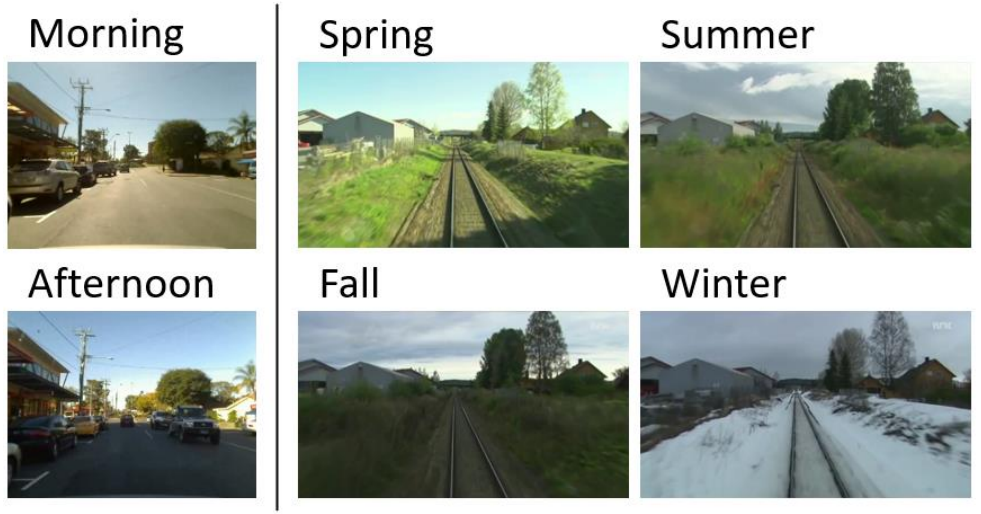


Types of Robot Localization

- “Position tracking” – figure out where the robot is, given that we know where the robot started.
- “Global” localization – figure out where the robot is, but we don’t know where the robot started.
- “Kidnapped robot” – robot is moved by external agent to any arbitrary location.
  - It is more difficult than the global localization problems, in that the robot might believe it knows where it is while it does not.

Challenges of Robot Localization

- Absolute position (e.g., GPS) may be unavailable, unreliable, and in many situations, insufficient.
- Sensors are always noisy and may provide irrelevant information, which may be caused by
  - Environment properties, e.g., mirror reflection or reflective floor surface.
  - Environment changes, e.g., weather, seasonal changes, earthquake, etc.
  - Interference between sensors, e.g., ultrasonic sensors, structured-light sensors.

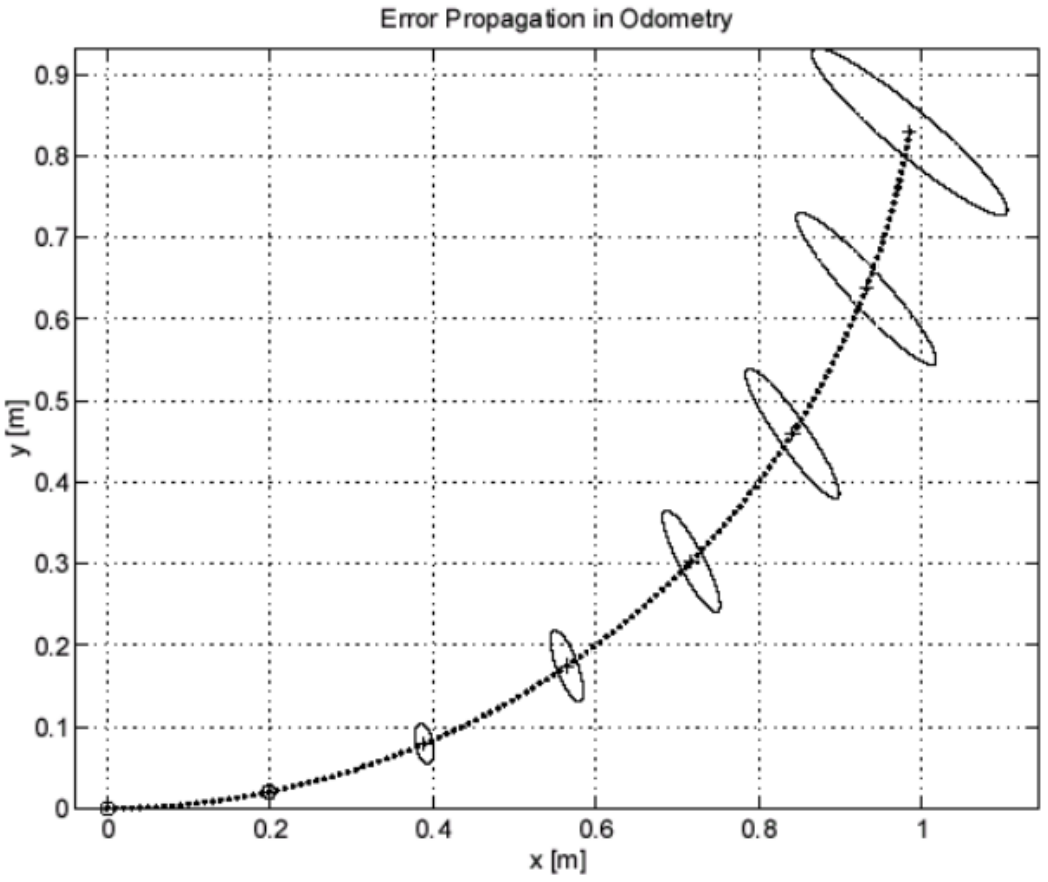


Challenges of Robot Localization

- Perceptual/Sensing Aliasing
  - In robots, non-uniqueness of sensors readings is the norm.
  - Even assuming sensor readings are perfect, robot localization still suffers from the challenge of perceptual aliasing:
    - different places generate a similar visual (or, in general, perceptual) footprint.

Odometry and Dead Reckoning

- These methods update robot position and orientation based on proprioceptive sensors.
  - Odometry: wheel sensors only
  - Dead reckoning: also heading sensors (e.g., gyroscope or compass)
- Pros: Straight forward, easy.
- Cons Errors are unbounded.
  - Limited sensing resolution.
  - Misalignment of motors (e.g., wheels).
  - Unequal floor contact (e.g., slipping).



Place Recognition

- Place recognition aims to identify a place using a map, or templates of places that were previously visited by the robot.
- Landmark-based place recognition is commonly used by humans for localization.
- Pros: Intuitive
- Cons: Easily to suffer from perceptual aliasing and environment changes.

Types of Robot Localization

- “Position tracking” – figure out where the robot is, given that we know where the robot started.
  - Solutions: Odometry, dead reckoning, etc.
- “Global” localization – figure out where the robot is, but we don’t know where the robot started.
  - Solution: GPS, place recognition, etc.
- “Kidnapped robot” – robot is moved by external agent to any arbitrary location.
  - Solution: GPS, place recognition, etc.

Probability Rules and Bayes Theorem

Refresher on Probability Rules and Bayes Theorem

- Discrete Probabilities:

$$
\begin{aligned}
P(X,Y) &= P(X|Y)P(Y) \\
&= P(Y|X)P(X) \\
P(X|Y) &= P(X) \text{ iff. X and Y are independent (Y provides no more information about X)} \\
P(X,Y) &= P(X)P(Y) \text{ iff. X and Y are independent} \\
P(X|Y) &= \frac{P(Y|X)P(X)}{P(Y)} \\
P(X) &= \sum_{Y} P(X,Y) \\
&= \sum_{Y} P(X|Y)P(Y)
\end{aligned}
$$

Slides modified from Prof. Joydeep Biswas

Refresher on Probability Rules and Bayes Theorem

- If x takes on continuous values, $p(x)$ is not a probability, it is the probability density.
- Asking "what is the probability of $x =$ <some specific value>" has no meaning (the most appropriate answer is 0).
  - E.g., if we assume that the tip of a dart is a point, the probability for the dart to land at a specific point on a broad is 0.
- An appropriate question is "what is the probability of x in <some continuous range>":

$$P(x \in [a, b]) = \int_a^b p(x)dx$$

Refresher on Probability Rules and Bayes Theorem

- Conditions:

$$p(x|y)$$

  - Often colloquially read as "probability of x given y“, e.g., probability that the robot is at position x given the sensor input y.
  - But this is not quite right!
  - It also does not mean we know the "value" of y as $p(y)$ is still a distribution!
  - Better way to think about it: If we know nothing about $p(y)$, then $p(x)$ is the best we can infer about x. However, if we do know that y has a distribution $p(y)$, then $p(x|y)$ is a more informative distribution: it is the distribution of x if y has the distribution $p(y)$.

Refresher on Probability Rules and Bayes Theorem

- From discrete to continuous random variables:

Discrete	Continuous
$P(X) \in [0, 1]$	$p(x) \in R^{0+}$
$\sum_X P(X) = 1$	$\int_x p(x)dx = 1$
$P(X) = \sum_Y P(X, Y)$	$p(x) = \int_y p(x, y)dy$

Refresher on Probability Rules and Bayes Theorem

- There are two operations we often would like to perform, for a distribution $p(x)$:
  - Sample a random value: $x \sim p(x)$
  - Evaluate the probability density at a particular value: $p(x)|_{x=x_0}$. It is the limit of the probability of the interval $(x, x+\Delta]$ divided by the length of the interval as the length of the interval goes to 0.

- For a conditional distribution $p(x|y)$ when $y = y_0$:
  - Sample a random value: $x \sim p(x|y)|_{y=y_0}$
  - Evaluate the probability density at a particular value: $p(x|y)|_{x=x_0, y=y_0}$.

Refresher on Probability Rules and Bayes Theorem

- Bayes Rule

$$\underbrace{p(\boldsymbol{x} \mid \boldsymbol{y})}_{\text{posterior}} = \frac{\overbrace{p(\boldsymbol{y} \mid \boldsymbol{x})}^{\text{likelihood}} \overbrace{p(\boldsymbol{x})}^{\text{prior}}}{\underbrace{p(\boldsymbol{y})}_{\text{evidence}}}$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x} \mid \boldsymbol{y}) p(\boldsymbol{y})$$

and

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{x}) p(\boldsymbol{x})$$

so that

$$p(\boldsymbol{x} \mid \boldsymbol{y}) p(\boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{x}) p(\boldsymbol{x}) \iff p(\boldsymbol{x} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{x}) p(\boldsymbol{x})}{p(\boldsymbol{y})} .$$

Bayesian State Estimation

Chapter 2.4, Sebastian Thrun, Wolfram Burgard and Dieter Fox.
“Probabilistic Robotics.” MIT Press. 2005.

The “Belief”

- Belief (in robotics) is defined as the distribution of the robot's state estimates:

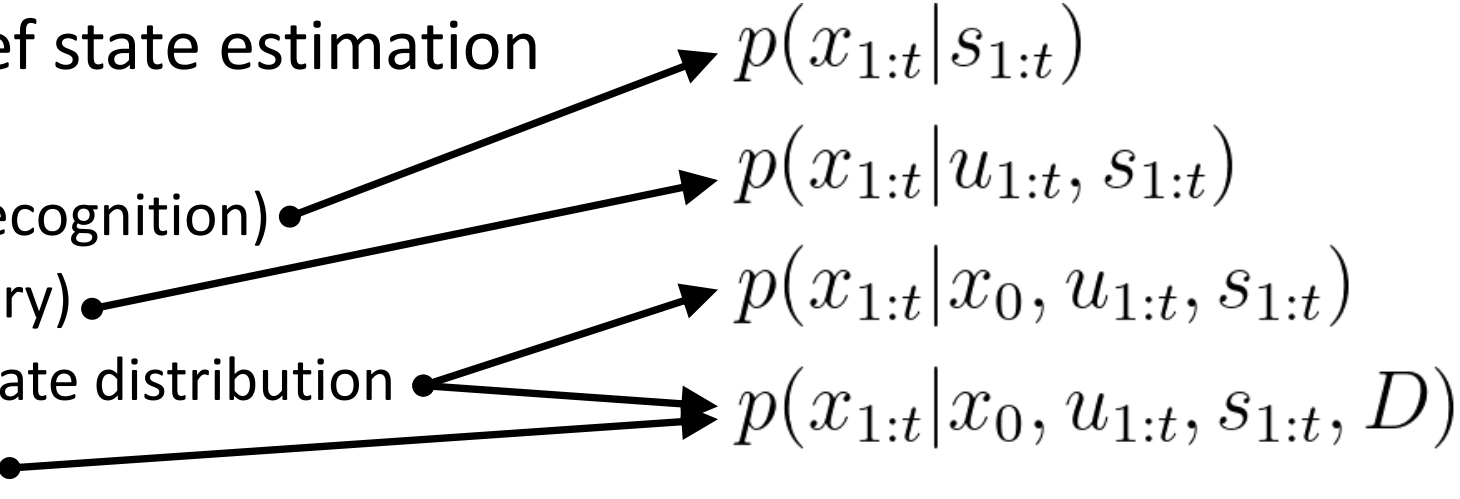
$$\mathrm{Bel}(x_t) = p(x_t | u_{1:t}, s_{1:t})$$
$$\mathrm{Bel}(x_{1:t}) = p(x_{1:t} | u_{1:t}, s_{1:t})$$

x : state (e.g., robot location)
u : control
s : sensing observations

- Characteristics:
  - Belief is a Probability Density Function (pdf).
  - It accounts for all data observed so far from time step 1 to t.
  - “Optimal” estimate is still a pdf.

Belief State Estimation

- In robotics, *State Estimation* is the field that deals with the challenge of using on-board sensors to estimate the state, such as position and orientation, of a robot as it moves through the world.
- *Belief State Estimation* uses belief to represent the distribution of state estimates.
- Variations of the belief state estimation are based on:
  - Sensing (e.g., place recognition)
  - Control (e.g., odometry)
  - Availability of prior state distribution
  - Prior data (e.g., map)



State Estimation

- Given inputs:
  - Motion model / state transition: $p(x_{t+1}|x_t)$ or $p(x_{t+1}|x_t, u_t)$
  - Sensor model / observation model : $p(s_t|x_t, D)$

- Wanted output to compute:

$$p(x_t|x_0, u_{1:t}, s_{1:t}, D)$$

- Note:
  - These are functions of all the variables, not just the first
  - In the state estimation algorithm, we will evaluate them at specific values

Motion Model

- Motion model is mathematically defined as:

$$p(x_{t+1}|x_t, u_{t+1})$$

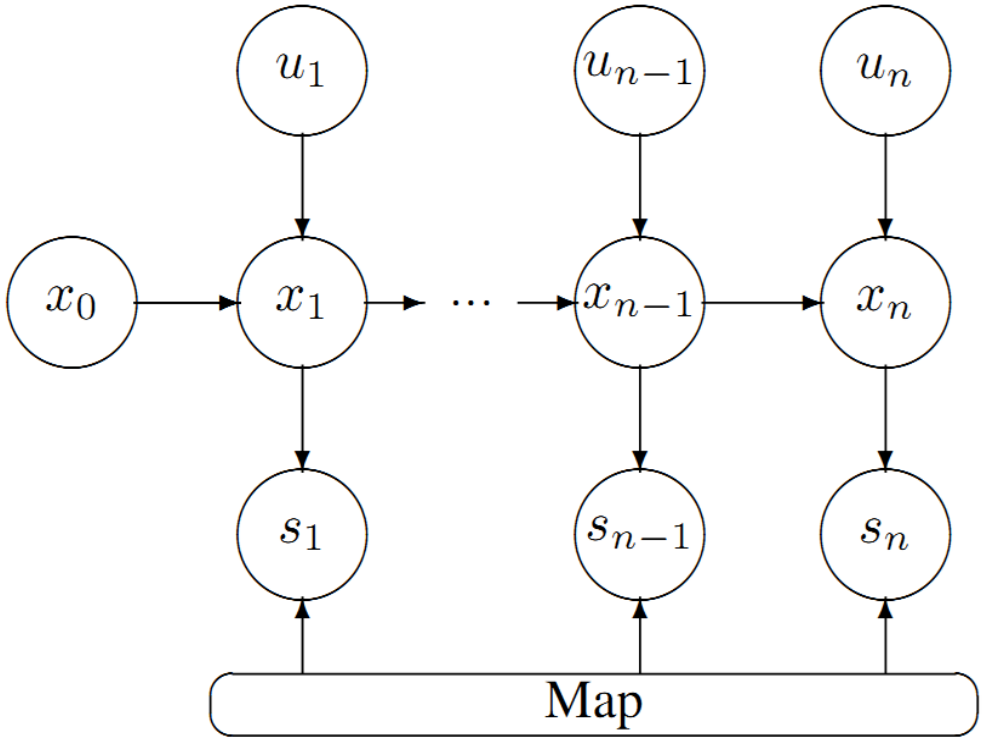
- It predicts how likely for the robot to go from a previous location to the next location based on control (or odometry).
- Variations of motion models:
  - u is based entirely on motion commands sent (velocity).
  - u is from encoder measurements (odometry).
  - u is from IMU (accelerometer & gyroscope) measurements.
  - No u: state estimation of an agent not under our control, e.g., a pedestrian.

Sensor Model

- Sensor model / observation model: $p(s_t|x_t, D)$
- It predicts:
  - What the robot expects to observe at a given x (given an optional domain-specific prior data D, like the map), and
  - Based on the sensor noise model, how likely to observe a specific value of s.
- Factors that contribute to the sensor model:
  - Sensor noise model
  - Sensor limitations (e.g., range, resolution)
  - Environmental / domain factors (e.g., moving obstacles, surface parameters)
  - Other random errors

Markov State Estimation

- Markov property: Past observations/states are independent of future observations/states given the current observations/states.
  - Observations are Markov: given state x_t at time t, observation s_t is independent of all past states and observations.
  - States are Markov: given state x_t at time t and control u_{t+1} at time $t+1$, state x at time $t+1$ is independent of all other past states and observations.



- Markov state estimation can be mathematically formulated with a dynamic Bayesian network.

Derivation: Recursive Belief Update

- Goal: Given the belief at t, the motion model, and the sensor model, represent the belief at time $t+1$ as a function of the belief at t.

- Recall: conditional are preserved in Bayes rule:

$$p(a|b,c)p(b|c) = p(a,b|c) = p(b|a,c)p(a|c)$$

$$\underbrace{p(\boldsymbol{x} \mid \boldsymbol{y})}_{\text{posterior}} = \frac{\overbrace{p(\boldsymbol{y} \mid \boldsymbol{x})}^{\text{likelihood}}\overbrace{p(\boldsymbol{x})}^{\text{prior}}}{\underbrace{p(\boldsymbol{y})}_{\text{evidence}}}$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{x} \mid \boldsymbol{y})p(\boldsymbol{y})$$

and

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x})$$

so that

$$p(\boldsymbol{x} \mid \boldsymbol{y})p(\boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x}) \iff p(\boldsymbol{x} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{y})}.$$

Derivation: Recursive Belief Update

$$\underbrace{p(x \mid y)}_{\text{posterior}} = \frac{\overbrace{p(y \mid x)}^{\text{likelihood}}\overbrace{p(x)}^{\text{prior}}}{\underbrace{p(y)}_{\text{evidence}}}$$

$$\begin{aligned}
\text{Bel}(x_{t+1}) &= p(x_{t+1}|x_0, s_{1:t+1}, u_{1:t+1}) \\
&= p(x_{t+1}|s_{t+1}, x_0, s_{1:t}, u_{1:t+1}) \\
&= \frac{p(s_{t+1}|x_{t+1}, x_0, s_{1:t}, u_{1:t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1})}{p(s_{t+1}|x_0, s_{1:t}, u_{1:t+1})} \\
&\propto p(s_{t+1}|x_{t+1}, x_0, s_{1:t}, u_{1:t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1})
\end{aligned}$$

Bayes' rule to account for the sensor model

Derivation: Recursive Belief Update

$$
\begin{aligned}
\mathrm{Bel}(x_{t+1}) &= p(x_{t+1}|x_0, s_{1:t+1}, u_{1:t+1}) \\
&= p(x_{t+1}|s_{t+1}, x_0, s_{1:t}, u_{1:t+1}) \\
&= \frac{p(s_{t+1}|x_{t+1}, x_0, s_{1:t}, u_{1:t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1})}{p(s_{t+1}|x_0, s_{1:t}, u_{1:t+1})} \\
&\propto p(s_{t+1}|x_{t+1}, x_0, s_{1:t}, u_{1:t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1}) \\
&\propto p(s_{t+1}|x_{t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1}) \\
&\propto p(s_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}, x_t|x_0, s_{1:t}, u_{1:t+1}) dx_t \\
&\propto p(s_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}|x_t, x_0, s_{1:t}, u_{1:t+1}) p(x_t|x_0, s_{1:t}, u_{1:t+1}) dx_t
\end{aligned}
$$

Bayes' rule to account for the sensor model

~~Markov assumption~~

Introducing term from previous time-step using marginalization

Derivation: Recursive Belief Update

$$
\begin{aligned}
\mathrm{Bel}(x_{t+1}) &= p(x_{t+1}|x_0, s_{1:t+1}, u_{1:t+1}) \\
&= p(x_{t+1}|s_{t+1}, x_0, s_{1:t}, u_{1:t+1}) \\
&= \frac{p(s_{t+1}|x_{t+1}, x_0, s_{1:t}, u_{1:t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1})}{p(s_{t+1}|x_0, s_{1:t}, u_{1:t+1})} \\
&\propto p(s_{t+1}|x_{t+1}, x_0, s_{1:t}, u_{1:t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1})
\end{aligned}
$$

Bayes' rule to account for the sensor model

Markov assumption

$$
\begin{aligned}
&\propto p(s_{t+1}|x_{t+1})p(x_{t+1}|x_0, s_{1:t}, u_{1:t+1}) \\
&\propto p(s_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}, x_t|x_0, s_{1:t}, u_{1:t+1}) dx_t \\
&\propto p(s_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}|x_t, x_0, s_{1:t}, u_{1:t+1})p(x_t|x_0, s_{1:t}, u_{1:t+1}) dx_t
\end{aligned}
$$

Introducing term from previous time-step using marginalization

Markov assumption

$$
\begin{aligned}
&\propto p(s_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}|x_t, u_{t+1})p(x_t|x_0, s_{1:t}, u_{1:t}) dx_t \\
&\propto p(s_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}|x_t, u_{t+1})\mathrm{Bel}(x_t) dx_t
\end{aligned}
$$

Recursive Belief Update

$$\mathrm{Bel}(x_{t+1}) \propto p(s_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}|x_t, u_{t+1}) \mathrm{Bel}(x_t) dx_t$$

- Given the previous belief, the update requires integration over all possible states.
- Note the proportion sign: a re-normalization is required after each recursive belief update, because belief is a distribution.

Bayesian Markov State Estimation

- Given belief $\mathrm{Bel}(x_t)$,
  - If a **motion model** (e.g., odometry) is available, predict as

$$\overline{\mathrm{Bel}}(x_{t+1}) = \int_{x_t} p(x_{t+1}|x_t, u_{t+1})\mathrm{Bel}(x_t)dx_t$$ **"Predict/estimate"**

  - If a **sensor model** (e.g., using LiDAR/camera data) is available, update as

$$\mathrm{Bel}(x_{t+1}) \propto p(s_{t+1}|x_{t+1})\overline{\mathrm{Bel}}(x_t)$$ **“Correct/update"**

- One step can be repeatedly applied when the other is not available.
- Most real robotics applications have both.

Markov Localization

Chapter 7.3-7.4, Sebastian Thrun, Wolfram Burgard and Dieter Fox. “Probabilistic Robotics.” MIT Press. 2005.

Markov Localization

- Markov localization is a classic solution to address probabilistic, map-based localization.
- Consider a mobile robot moving in a known environment:
  - As it starts to move (e.g., from a precisely known location), it might keep track of its location using odometry.
  - However, after a certain movement the robot will get very uncertain about its position. Then the robot will need to update its position using an observation of the environment.
- Odometry information leads to an estimate of the robot's position, which can then be fused with the sensor observations to get the best possible update of the robot's actual position.

Markov Localization

- Markov localization uses an explicit, discrete representation of the positions as states.
- This is usually done by representing the environment by a grid or a topological graph with a finite number of possible states (positions).

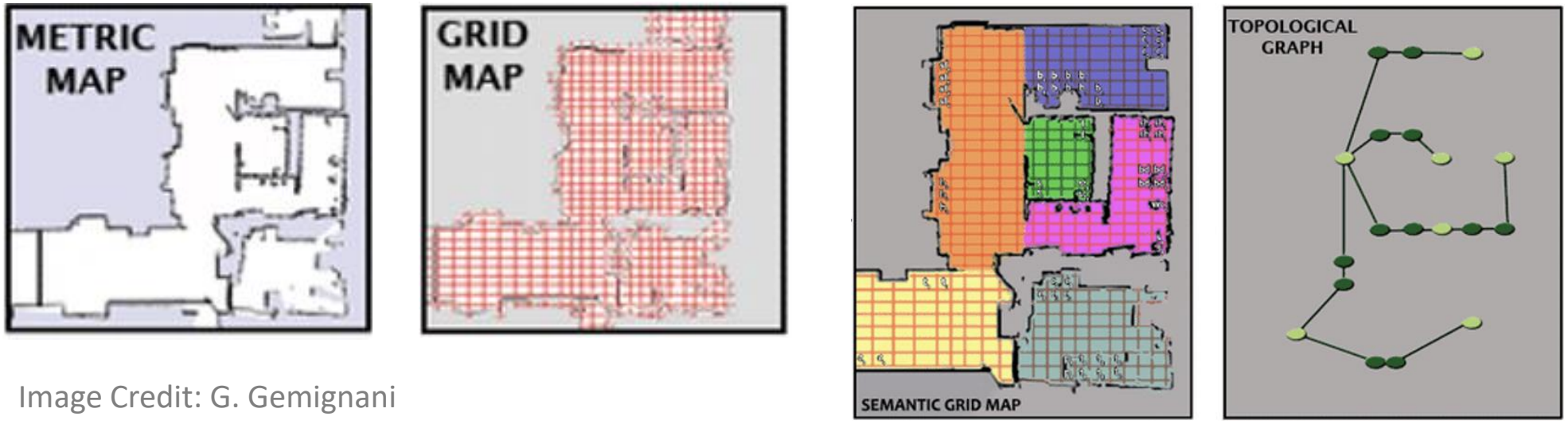
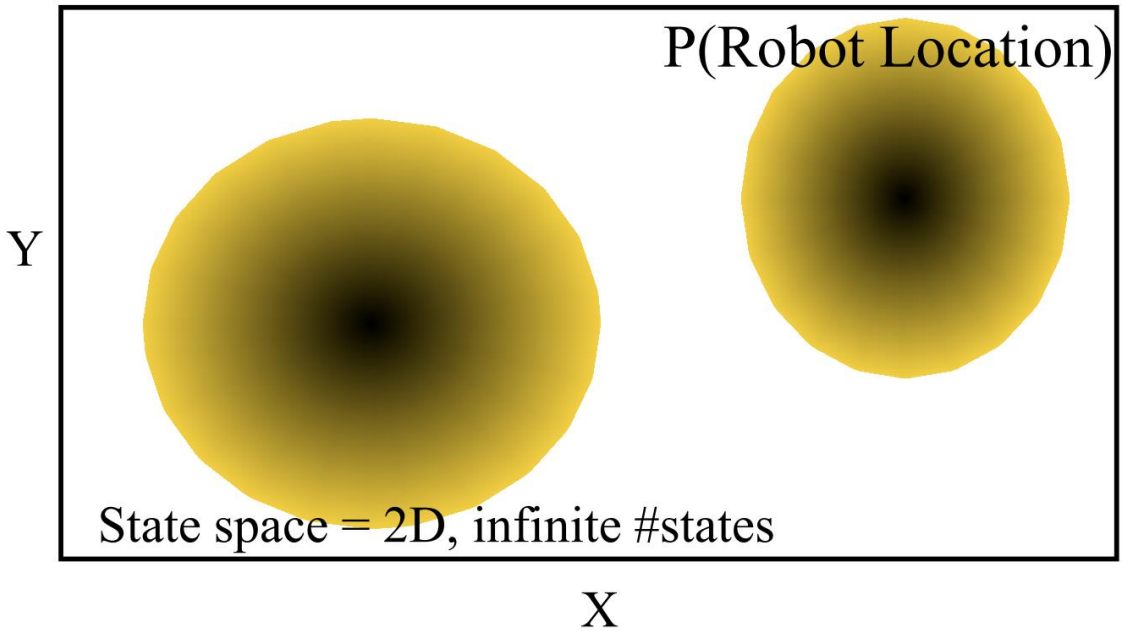


Image Credit: G. Gemignani

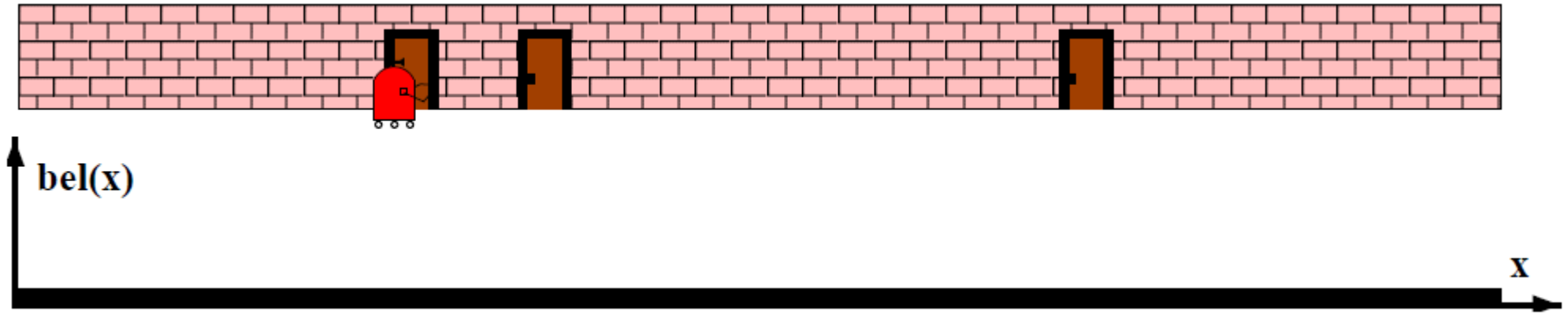
Markov Localization

- Key idea: compute a discrete probability distribution over all possible states (positions) in the environment.
  - Each value in this distribution represents the probability that the robot is in a particular location.
  - Markov localization recursively maintains the estimates of the positions.
  - During each update, the probability for each position of the entire space is updated.



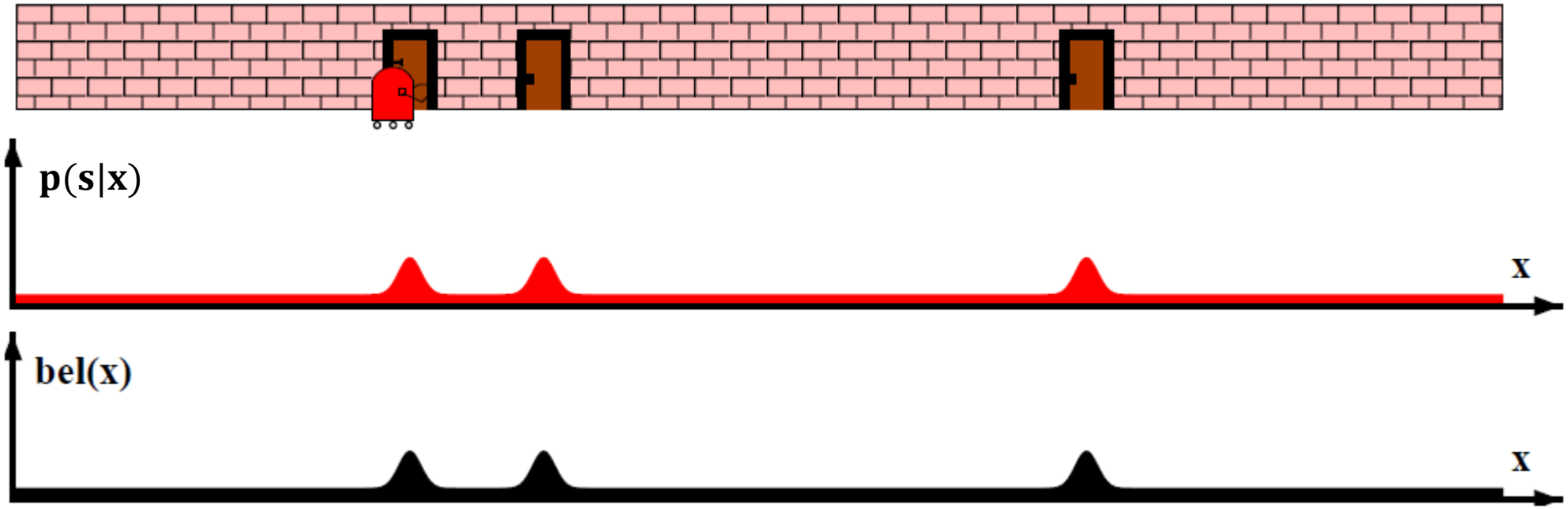
Markov Localization: Example

- The robot doesn’t know where it is. Thus, a reasonable initial believe of its position is a uniform distribution.



Markov Localization: Example

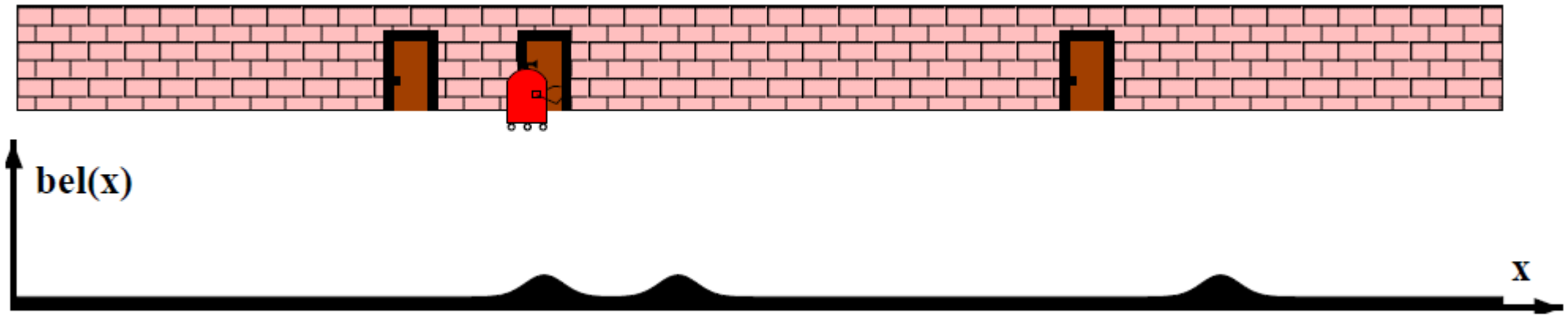
- A sensor reading is made (depending on a sensor model) indicating a door at certain locations (using a map).
- The sensor reading is used to update the belief (using Bayes theorem).



$$\mathrm{Bel}(x_{t+1}) \propto p(s_{t+1}|x_{t+1}, M)\overline{\mathrm{Bel}}(x_t)$$

Markov Localization: Example

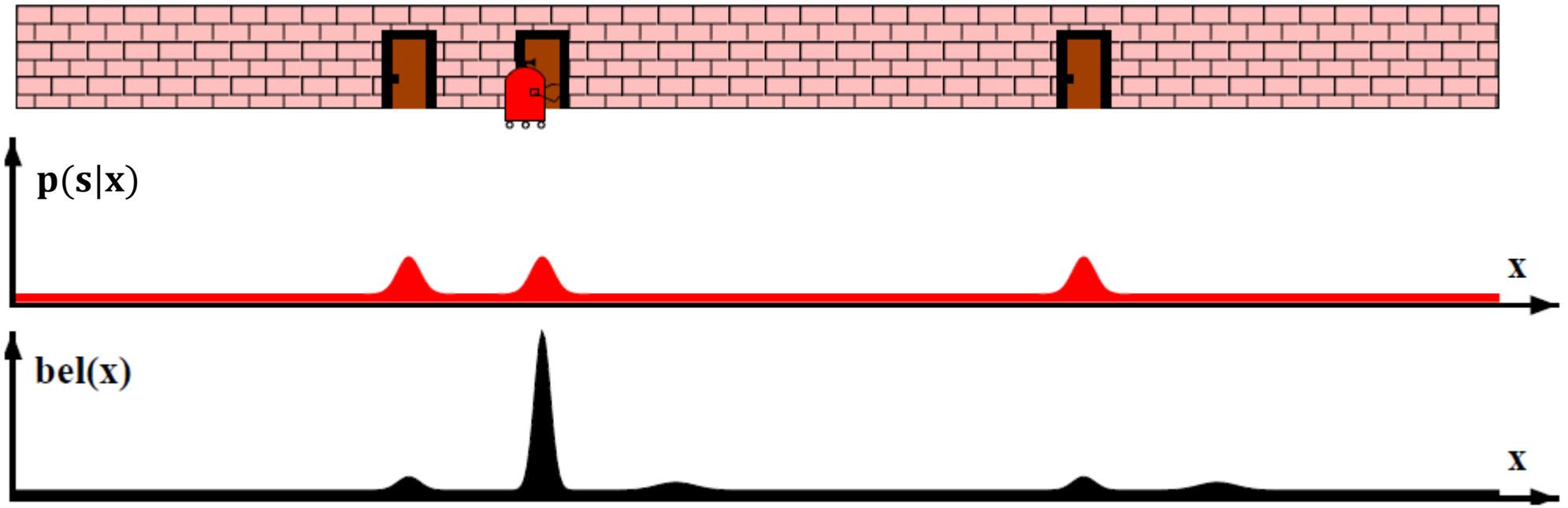
- The robot is moving, which adds noise.
- The robot then can estimate its new belief (using a motion model).



$$\overline{\mathrm{Bel}}(x_{t+1}) = \int_{x_t} p(x_{t+1}|x_t, u_{t+1})\mathrm{Bel}(x_t)dx_t$$

Markov Localization: Example

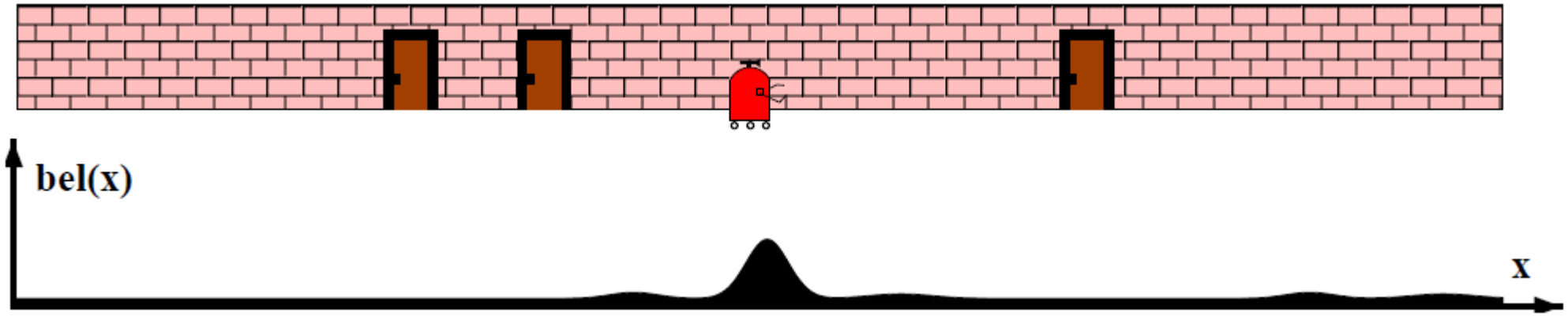
- A sensor reading is made (using a sensor model) indicating a door at certain locations (using a map).
- The sensor reading is used to update the belief (using Bayes theorem).



$$\mathrm{Bel}(x_{t+1}) \propto p(s_{t+1}|x_{t+1}, M)\overline{\mathrm{Bel}}(x_t)$$

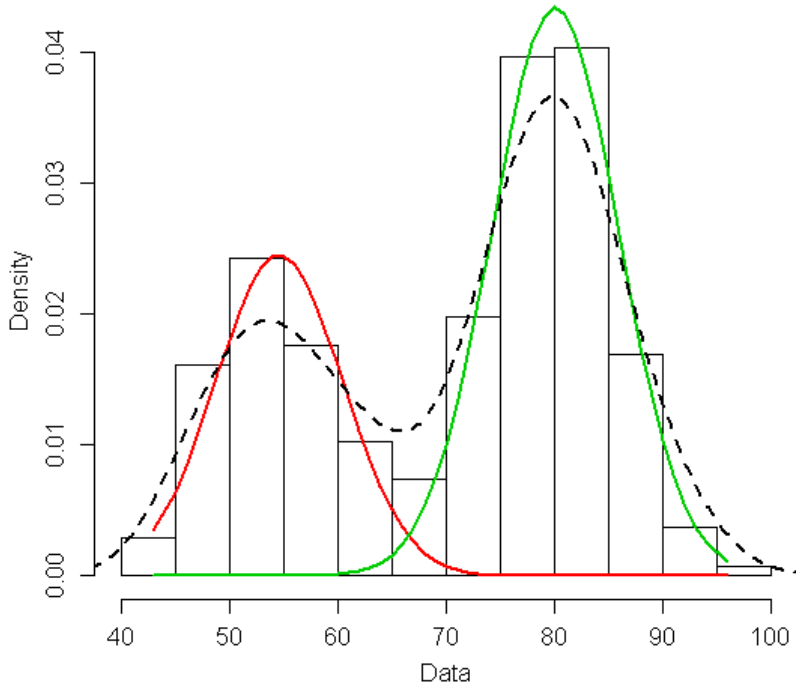
Markov Localization: Example

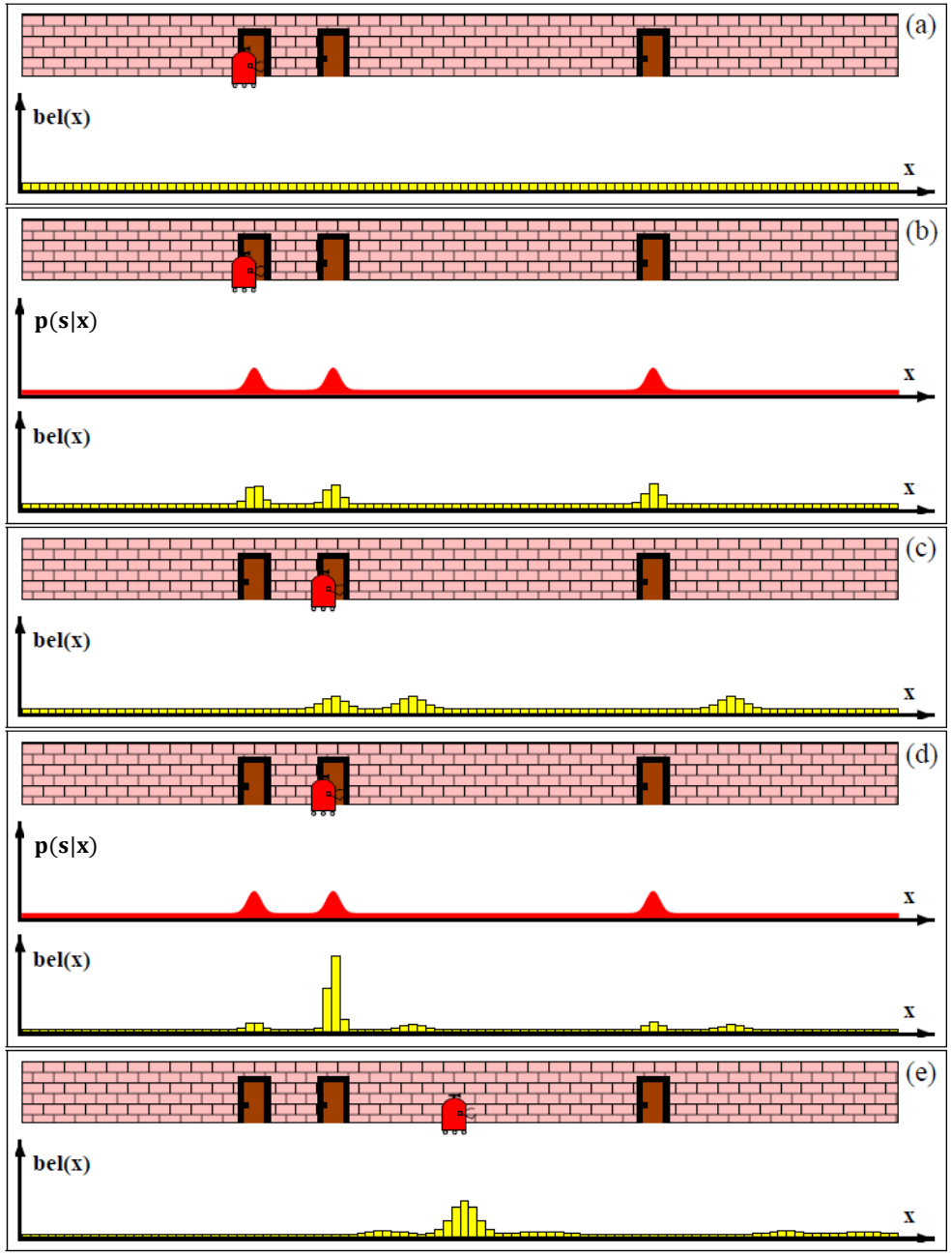
- Repeat the estimation/prediction and correction/update.



Markov Localization: Example

- Markov localization uses discrete beliefs
  - Histogram can be used as a non-parametric model to approximate a distribution



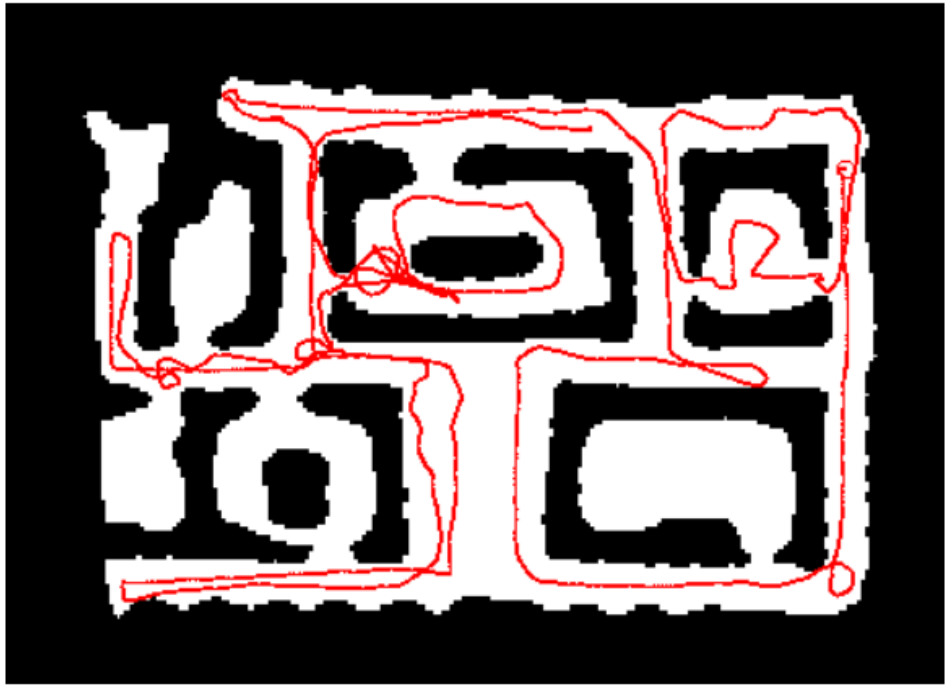


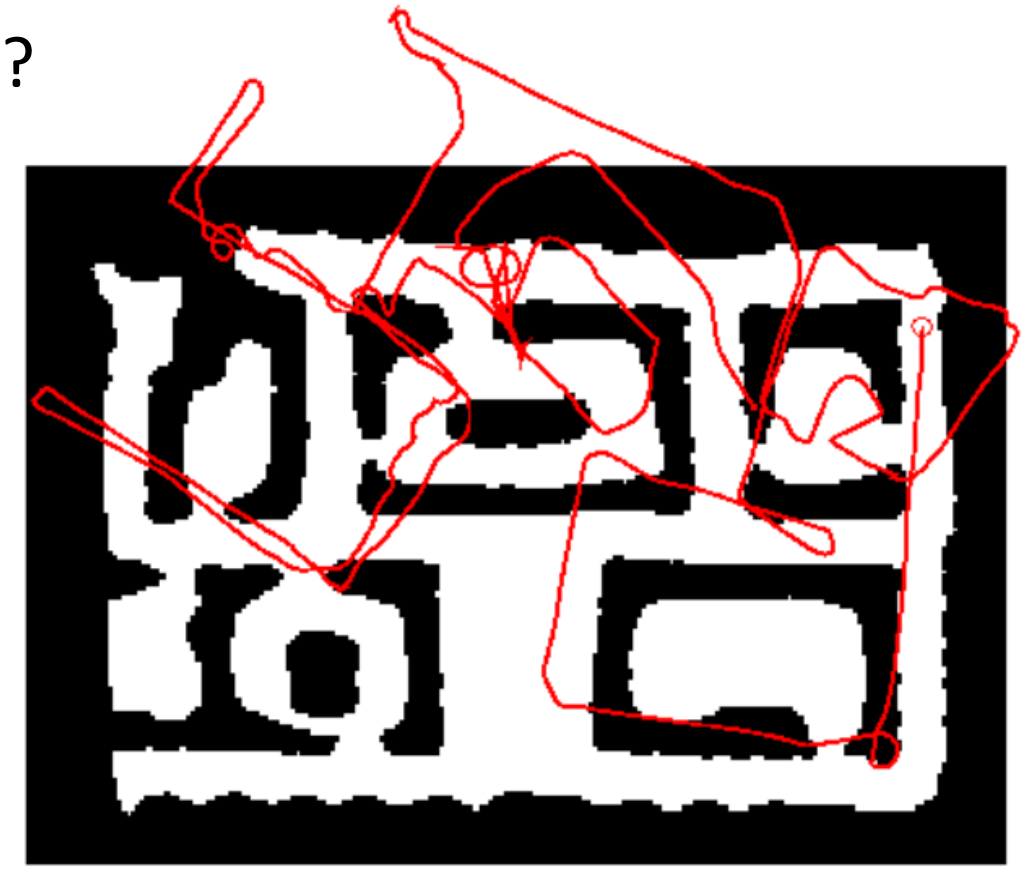
Motion Model

Chapter 5, Sebastian Thrun, Wolfram Burgard and Dieter Fox.
“Probabilistic Robotics.” MIT Press. 2005.

Robot Motion

- Robot motion is inherently uncertain.
- How can we model this uncertainty?

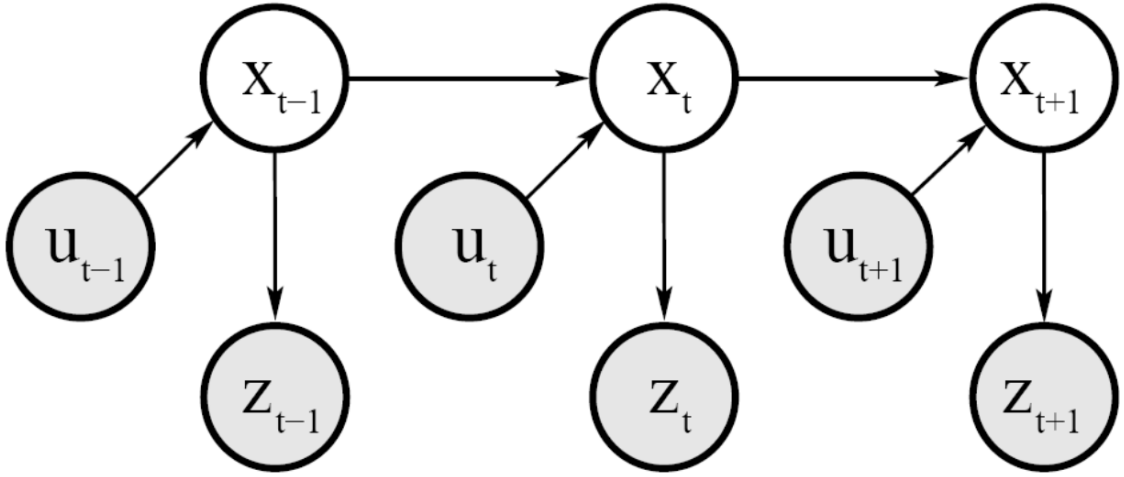




Motion Model

- Motion model specifies a posterior probability that action u carries the robot from x_{t-1} to x_t.

$$p(x_t \mid u_t, x_{t-1})$$



Algorithm Bayes_filter$(bel(x_{t-1}), u_t, z_t)$**:**

for all x_t do

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1})\, dx$$

$$bel(x_t) = \eta\, p(z_t \mid x_t)\, \overline{bel}(x_t)$$

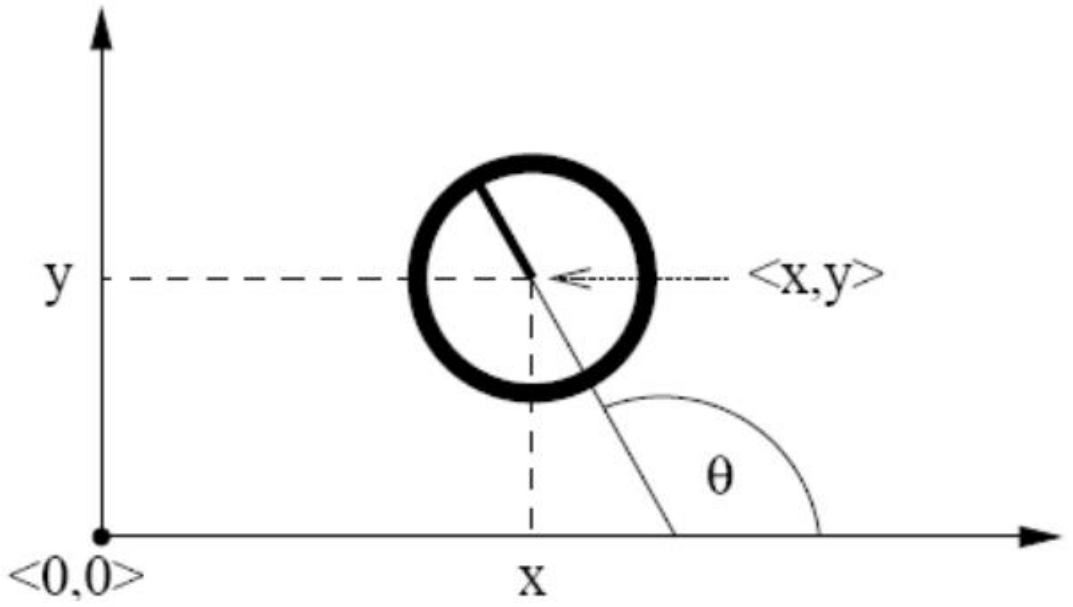
endfor

return $bel(x_t)$

Coordinate Systems

- In 3D space, the pose of a robot can be described by six parameters:
  - Three-dimensional Cartesian coordinates
  - Three Euler angles pitch, roll, and yaw

- In the lecture, we consider robots operating on a planar surface.
- The state space of such coordinate system is three-dimensional (x, y, θ)

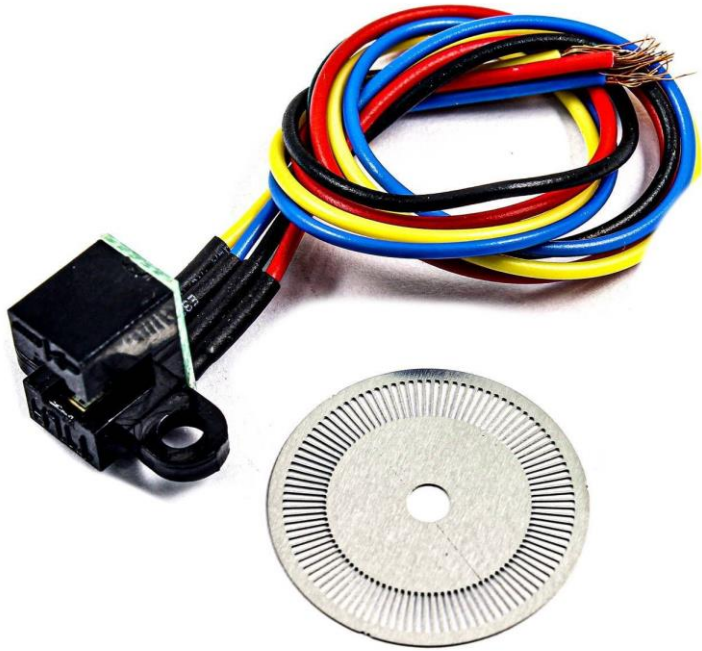


Typical Motion Models

- Two motion models are often used in practice:
  - Odometry-based
  - Velocity-based
- Odometry-based models are used when systems are equipped with wheel encoders (or IMU).
  - They calculate the new pose based on encoder (or IMU) values.
- Velocity-based models are used when no wheel encoders (or IMU measurements) are given.
  - They calculate the new pose based on the velocities and the time elapsed.
- We will focus on the odometry-based motion models.

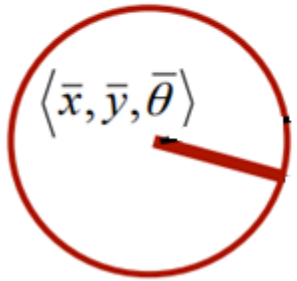
Wheel Encoder Examples

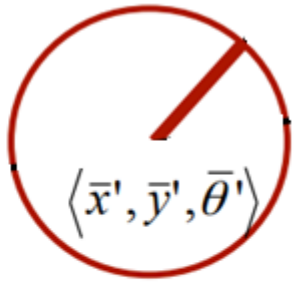
- Typically, these modules require +xV and GND to power them, and provide a 0 to xV output. They provide +xV output when they see a gap, and a 0V output when they do not see a gap.



Odometry-based Motion Model

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





How can a robot move from one pose to another pose?

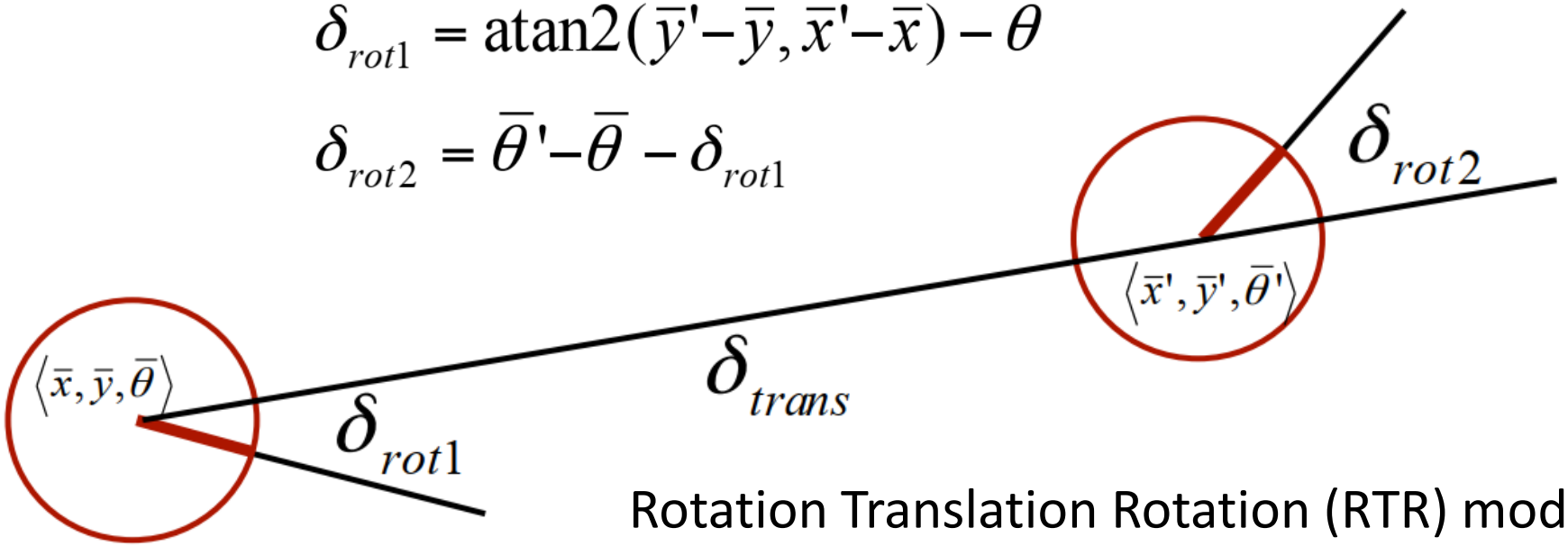
Odometry-based Motion Model

- Robot moves from $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ to $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$
- Odometry information $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle$

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

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

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



Rotation Translation Rotation (RTR) model

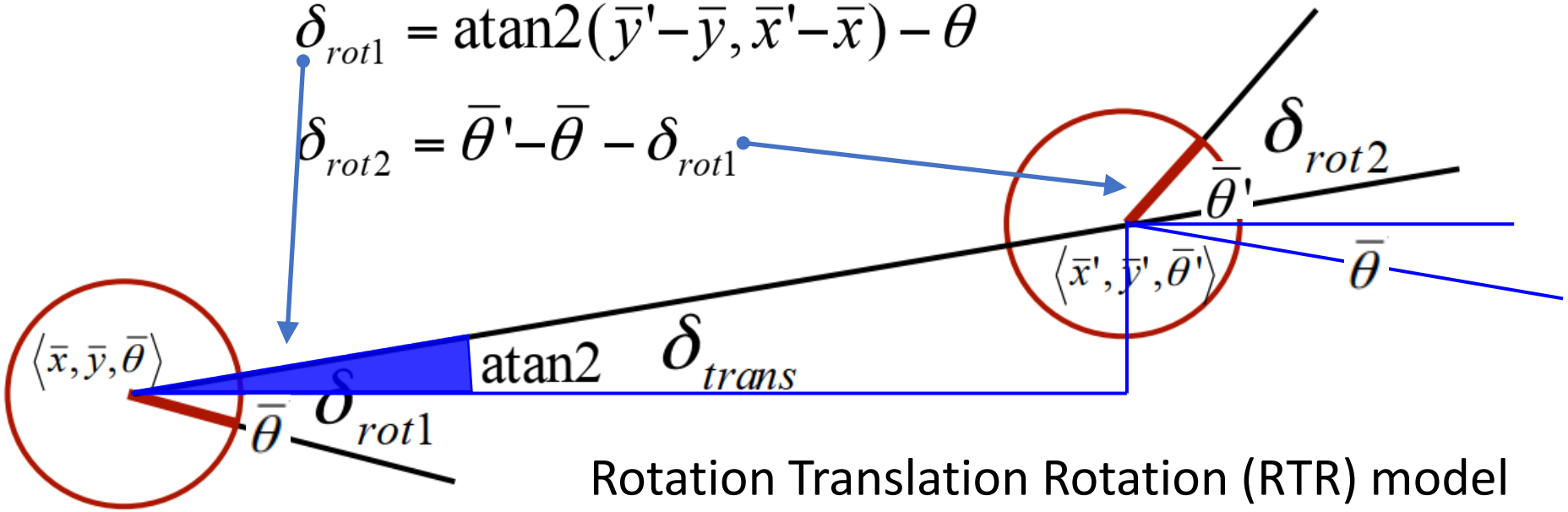
Odometry-based Motion Model

- Robot moves from $\langle \bar{x}, \bar{y}, \bar{\theta} \rangle$ to $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$
- Odometry information $u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle$

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

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

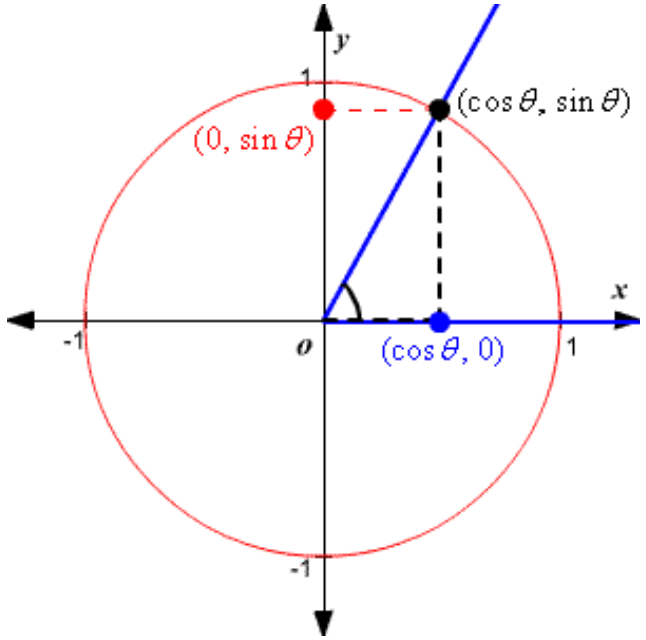
$$\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$$



Rotation Translation Rotation (RTR) model

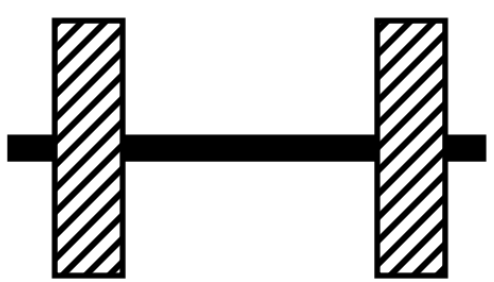
The atan2 Function

- Extends the inverse tangent (or arctangent) and correctly copes with the signs of x and y.

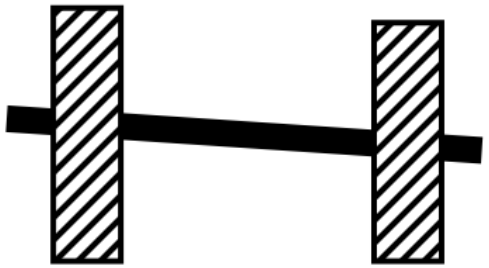


$$\text{atan2}(y,x) = \begin{cases} \text{atan}(y/x) & \text{if } x > 0 \\ \text{sign}(y)\ (\pi - \text{atan}(|y/x|)) & \text{if } x < 0 \\ 0 & \text{if } x = y = 0 \\ \text{sign}(y)\ \pi/2 & \text{if } x = 0, y \neq 0 \end{cases}$$

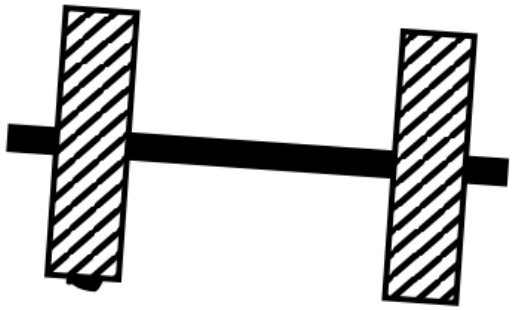
Uncertainty in Motion (of Wheeled Robots)



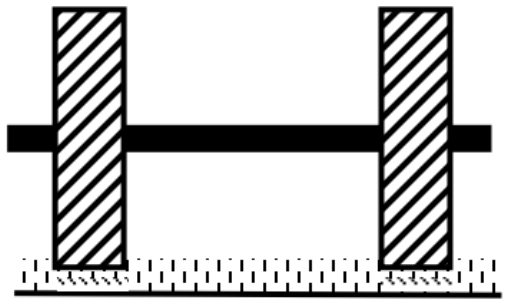
ideal case



different wheel diameters



bump



carpet

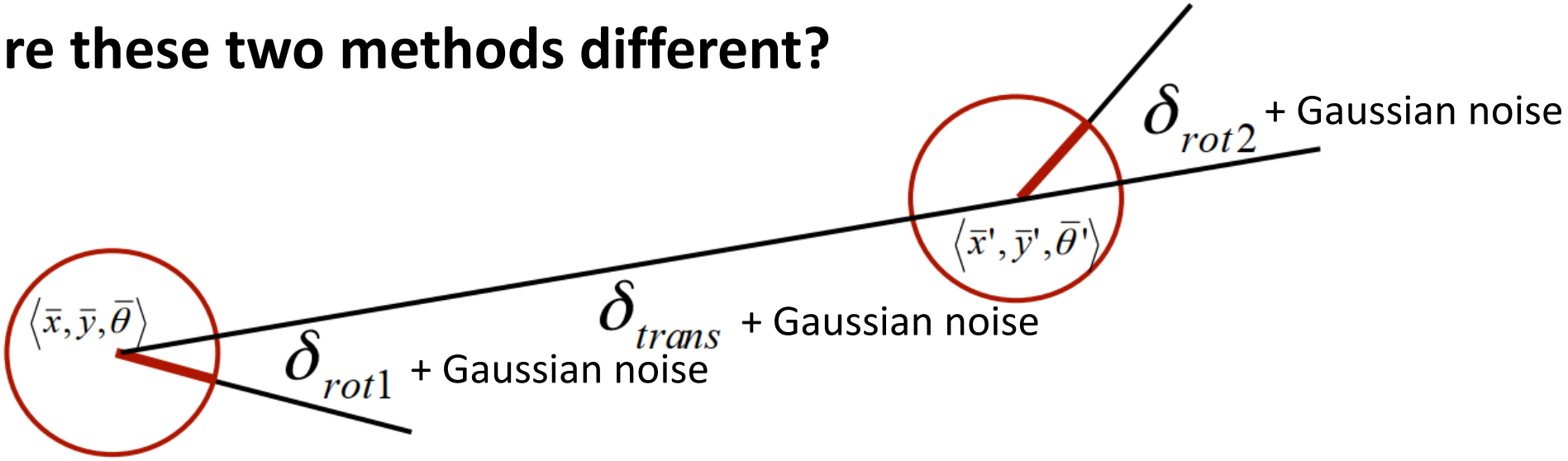
and many more …

Noise Model for Odometry

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

1. One way to model the noise is to simply center a Gaussian noise in the pose $\langle \bar{x}', \bar{y}', \bar{\theta}' \rangle$ only, but this cannot model the reality well.
2. For the RTR model, the noise is introduced by any rotation and translation motions.

- **Why are these two methods different?**

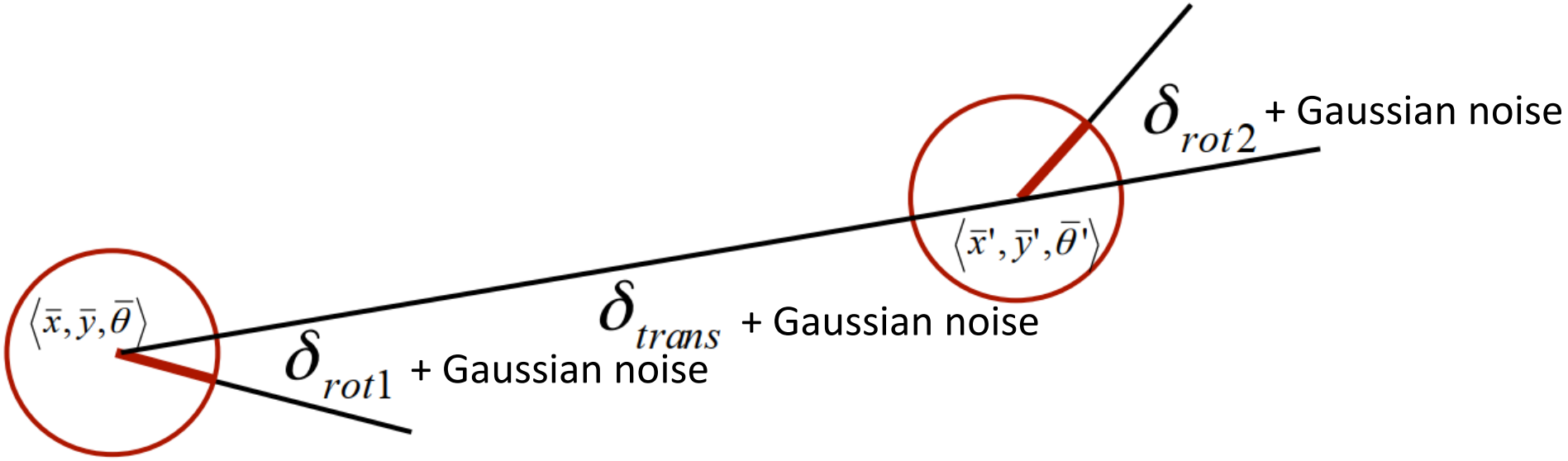


Noise Model for Odometry

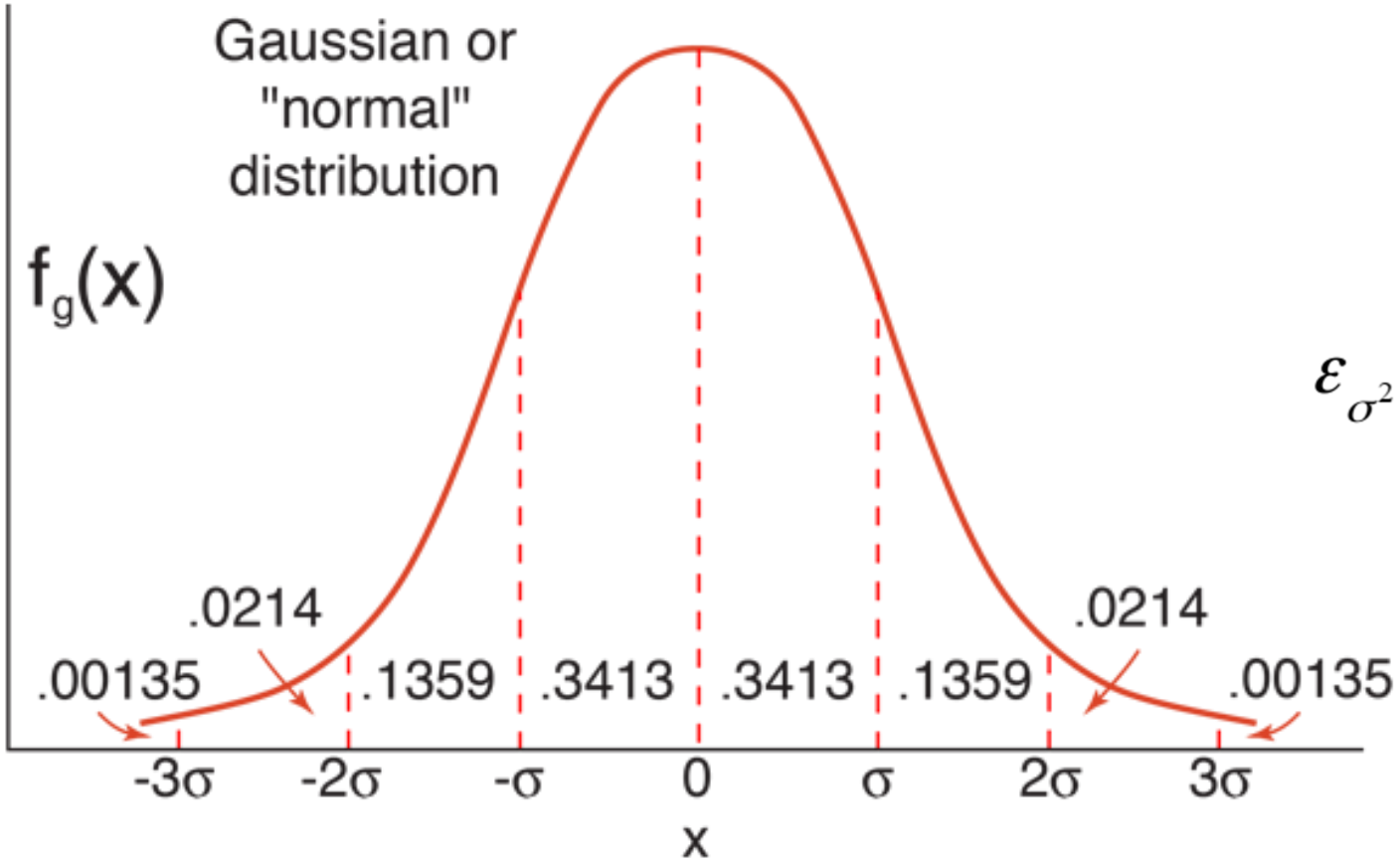
$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1 |\delta_{rot1}| + \alpha_2 |\delta_{trans}|}$$

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

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



Recall: Gaussian Distribution to Model Noise



$$\varepsilon_{\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}}$$

What is the label of the y-axis?

Calculating the *Probability Density* (Zero-Centered)

- For a Gaussian/normal distribution:

a: query point
b: std. deviation

1. Algorithm **prob_normal_distribution**(a,b):
2. return $\frac{1}{\sqrt{2\pi\ b^2}}\ \exp\left\{-\frac{1}{2}\frac{a^2}{b^2}\right\}$

Calculating the Posterior

x: pose or state
u: control

1. Algorithm **motion_model_odometry(x,x’,u)**
2. $\delta_{trans} = \sqrt{(\bar{x}'-\bar{x})^2 + (\bar{y}'-\bar{y})^2}$
3. $\delta_{rot1} = \text{atan2}(\bar{y}'-\bar{y}, \bar{x}'-\bar{x}) - \bar{\theta}$
4. $\delta_{rot2} = \bar{\theta}'-\bar{\theta} - \delta_{rot1}$
5. $\hat{\delta}_{trans} = \sqrt{(x'-x)^2 + (y'-y)^2}$
6. $\hat{\delta}_{rot1} = \text{atan2}(y'-y, x'-x) - \bar{\theta}$
7. $\hat{\delta}_{rot2} = \theta'-\theta - \hat{\delta}_{rot1}$
8. $p_1 = \text{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 \mid \hat{\delta}_{\text{rot1}} \mid + \alpha_2 \hat{\delta}_{\text{trans}})$
9. $p_2 = \text{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \hat{\delta}_{\text{trans}} + \alpha_4 (\mid \hat{\delta}_{\text{rot1}} \mid + \mid \hat{\delta}_{\text{rot2}} \mid))$
10. $p_3 = \text{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 \mid \hat{\delta}_{\text{rot2}} \mid + \alpha_2 \hat{\delta}_{\text{trans}})$
11. return $p_1 \cdot p_2 \cdot p_3$

odometry values (u)

What the odometry u tells us

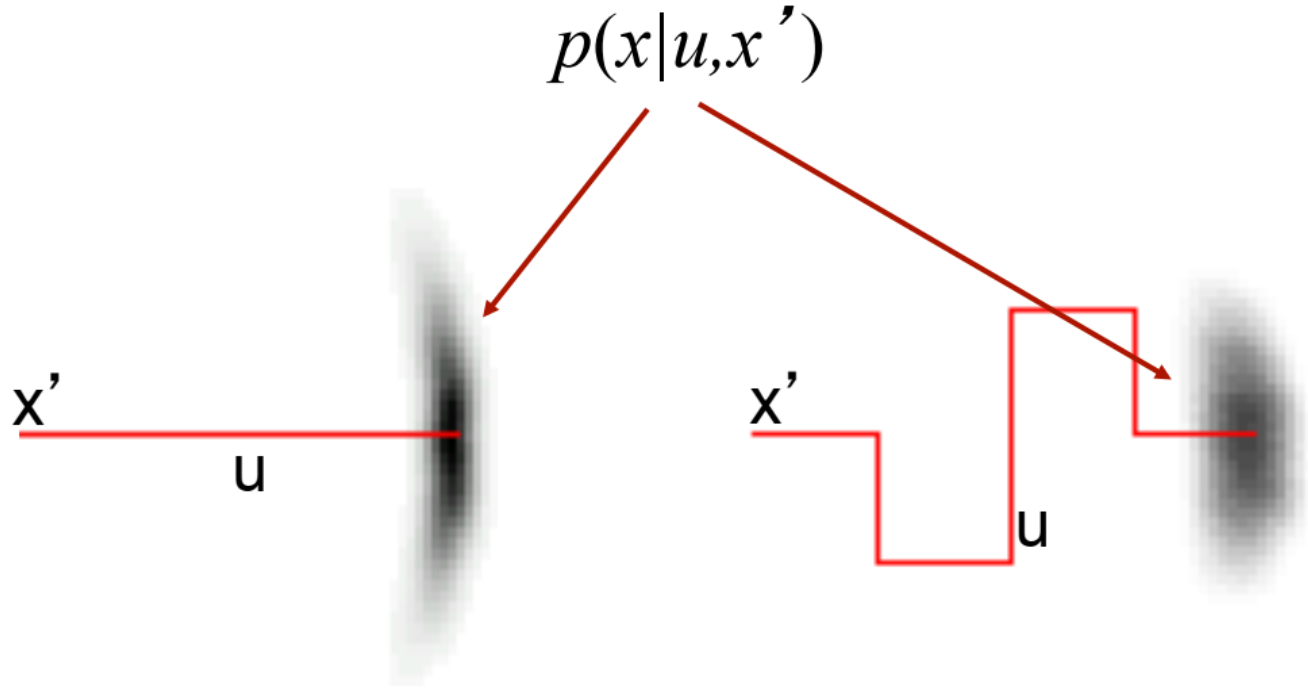
values of interest (x,x’)

What we compute from the input x and x’

This posterior answers: what is the probability density of seeing x’, given x and u under RTR?

Uncertainty Propagation in Motion Models

- Repeated application of the motion model for short movements, we typically obtain banana-shaped distributions



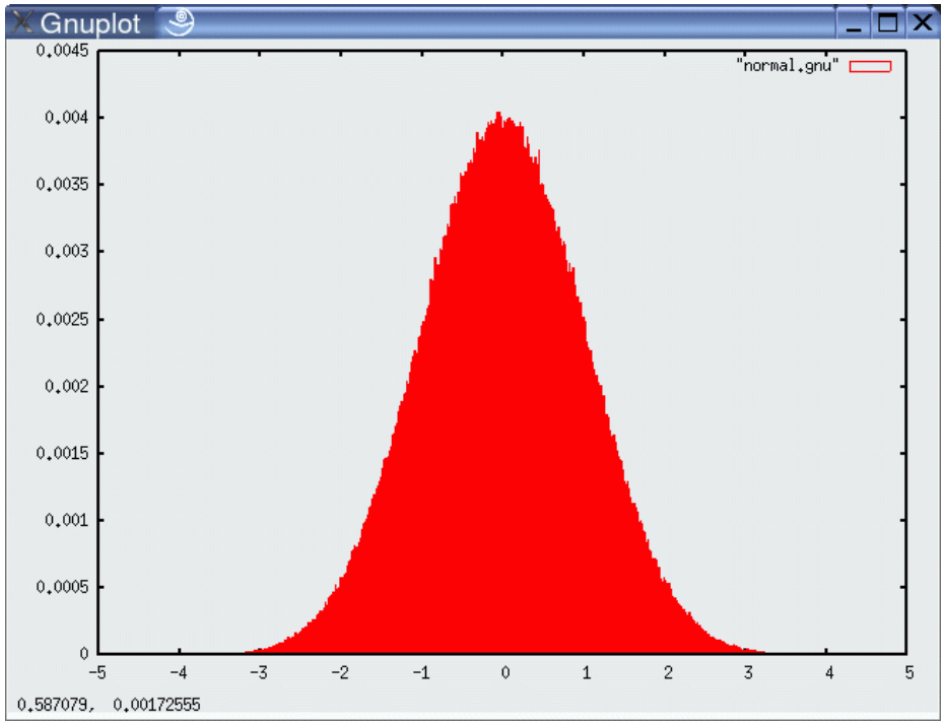
Sampling from a Gaussian Distribution

- Sampling from a Gaussian distribution:

1. Algorithm **sample_normal_distribution**(b):
2. return $\frac{1}{2}\sum_{i=1}^{12} rand(-b, b)$

Central limit theorem:

Let $X_1, X_2, \ldots, X_n$ be n i.i.d. random variables with $E(X_i) = \mu$ and $Var(X_i) = \sigma^2$ and let $S_n = \frac{X_1+X_2+\ldots+X_n}{n}$ be the sample average. Then S_n approximates a normal distribution with mean of μ and variance of $\frac{\sigma^2}{n}$ for large n (i.e. $S_n \approx N(\mu, \frac{\sigma^2}{n})$).



10^6 samples

Source and examples: https://bjlkeng.github.io/posts/sampling-from-a-normal-distribution

Sampling from Odometry Motion Model

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

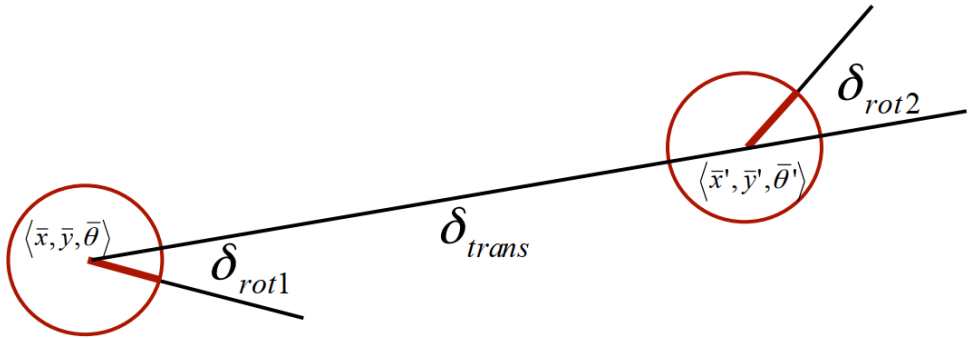
$$u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle, x = \langle x, y, \theta \rangle$$

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. $\hat{\delta}_{rot2} = \delta_{rot2} + \text{sample}(\alpha_1 | \delta_{rot2} | + \alpha_2 \, \delta_{trans})$

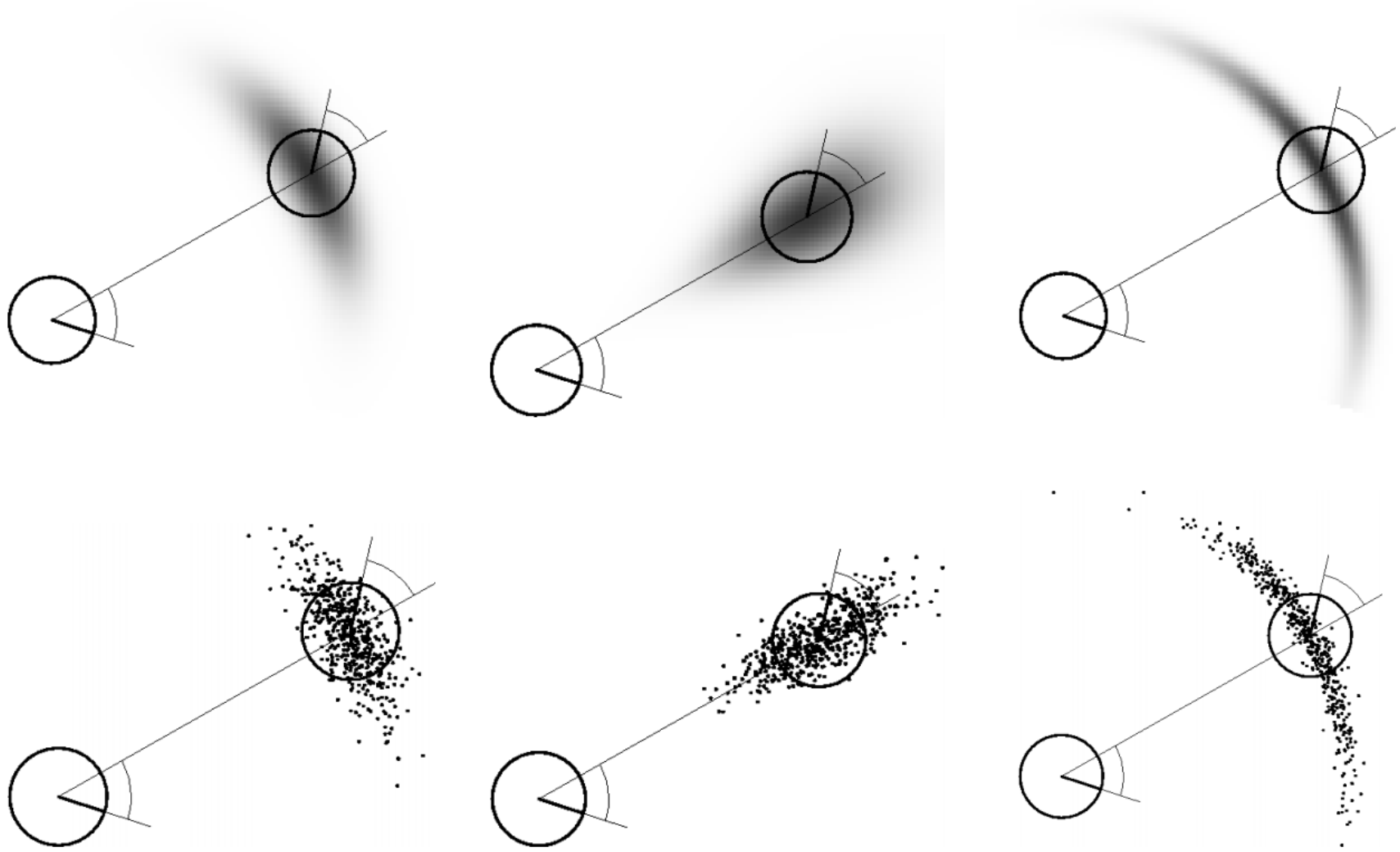
4. $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
5. $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
6. $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

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

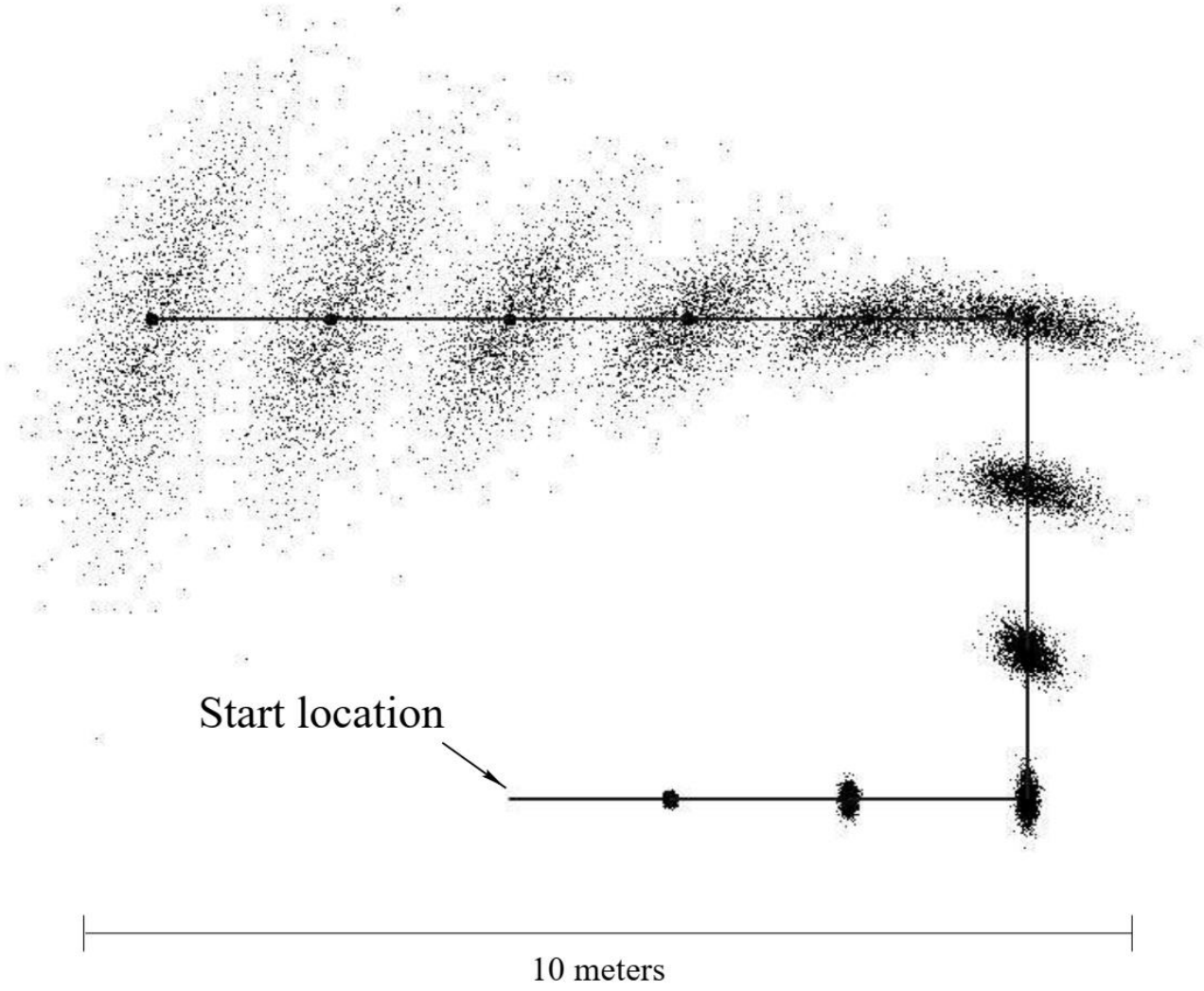
sample_normal_distribution



Example of Sampling from Motion Model



Example of Sampling from Motion Model



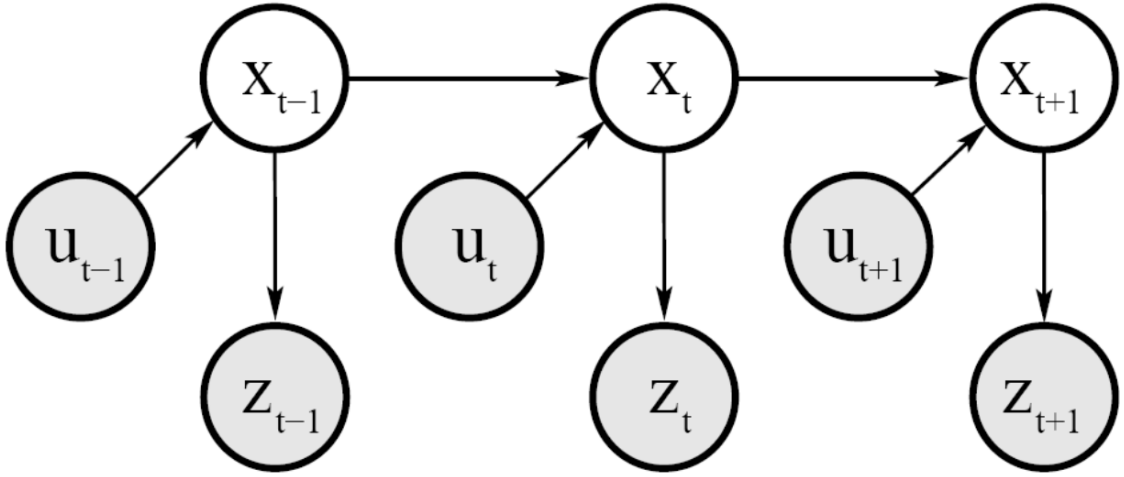
Sensor Model

Chapter 6, Sebastian Thrun, Wolfram Burgard and Dieter Fox.
“Probabilistic Robotics.” MIT Press. 2005.

Sensor Model

- Sensor model (observation model, measurement model) specifies a probability distribution of receiving observation z_t when the robot is in state x_t:

$$p(z_t \mid x_t)$$



Algorithm Bayes_filter$(bel(x_{t-1}), u_t, z_t)$**:**

for all x_t do

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\, bel(x_{t-1})\, dx$$

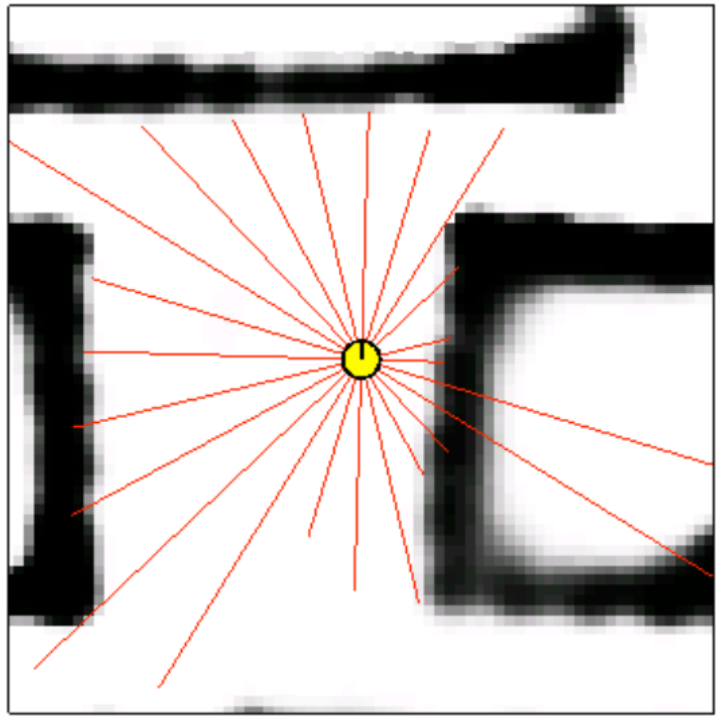
$$bel(x_t) = \eta\, p(z_t \mid x_t)\, \overline{bel}(x_t)$$

endfor

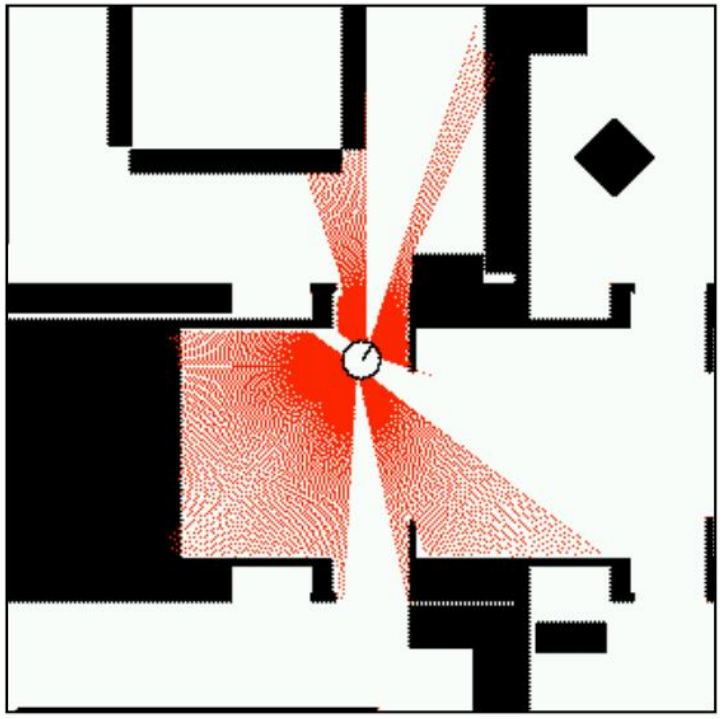
return $bel(x_t)$

Beam-based Model

- Beam-based models can be used to model range-based sensors.



Sonar sensors



Laser sensors

Beam-based Model

- Scan z at time step t consists of K measurements.

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

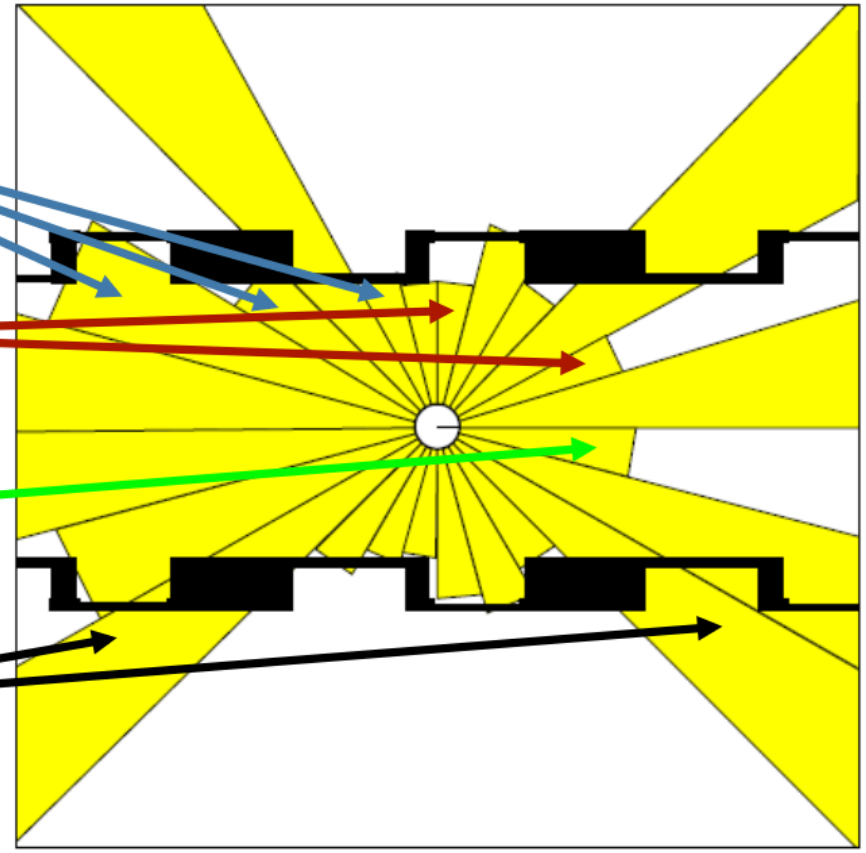
- Individual measurements are independent given the robot position.

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

m: a known map

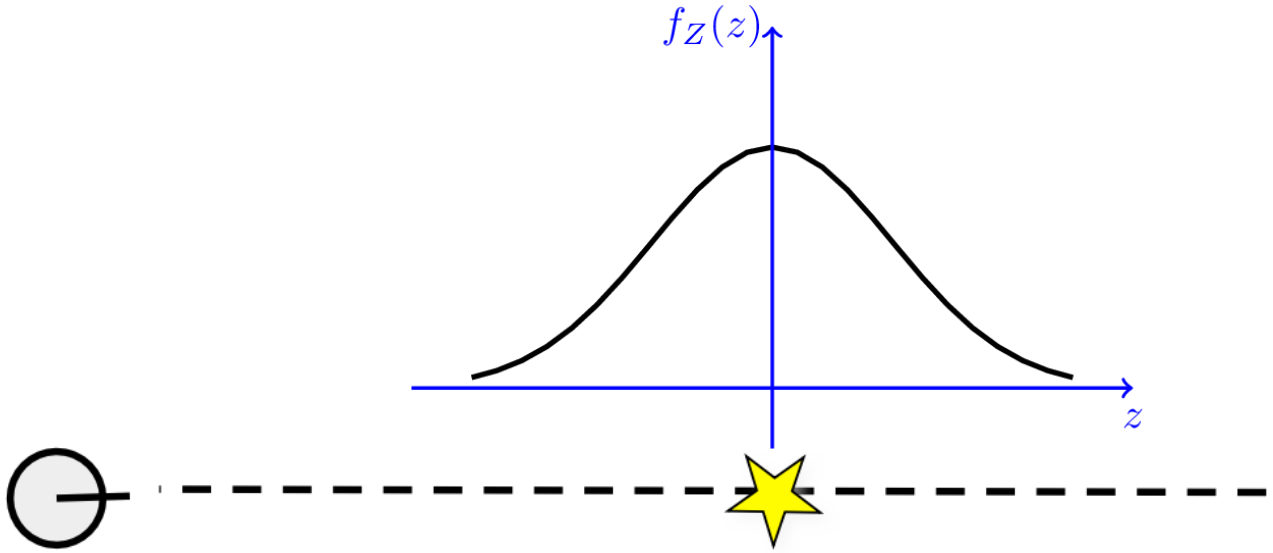
Range Sensor Measurements and Noise

1. Beams reflected by obstacles
2. Beams reflected by persons / caused by crosstalk
3. Random measurements
4. Maximum range measurements



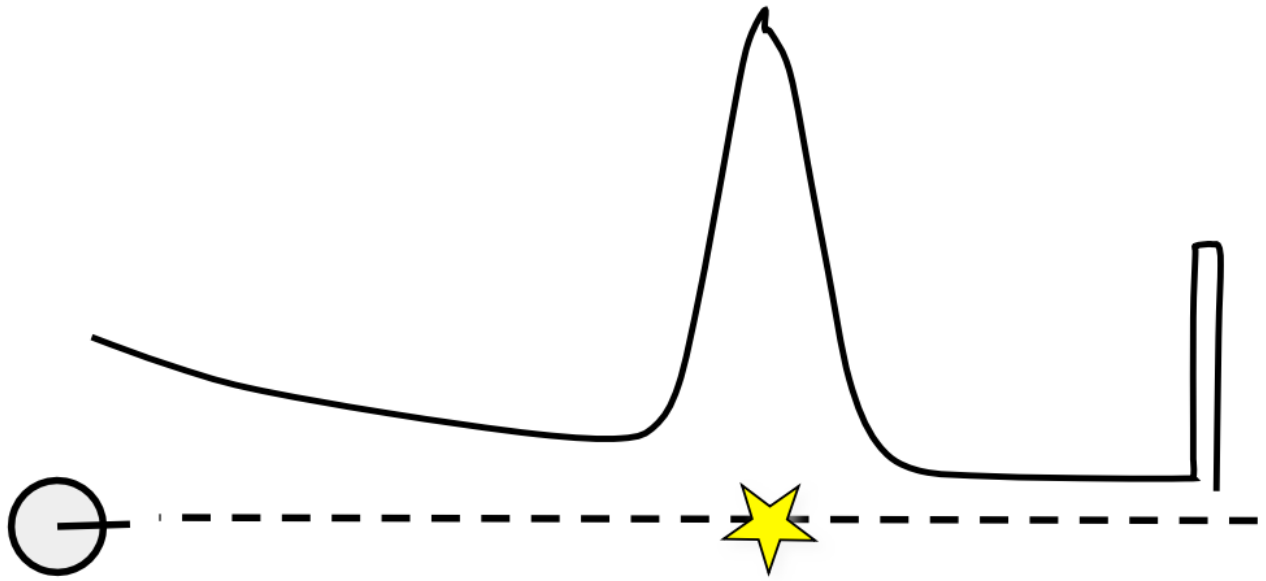
Ray-Casting Model

- Ray-casting model considers the first obstacle along the line of sight.
- Simplest noise modeling: Gaussian noise in the measured distance.



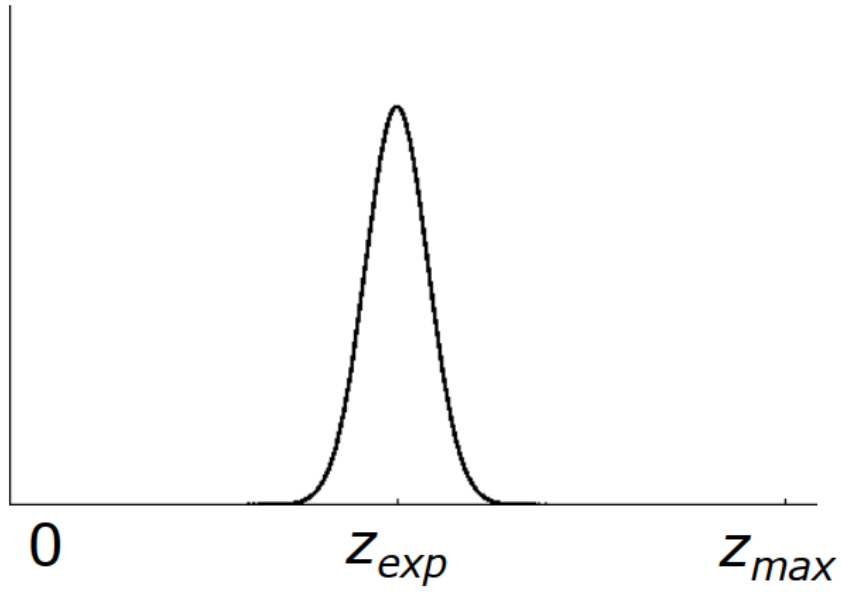
Ray-Casting Model

- More realistic ray-casting model includes a mixture of four models.
- It considers dynamic objects, randomness, max rage, and noise.



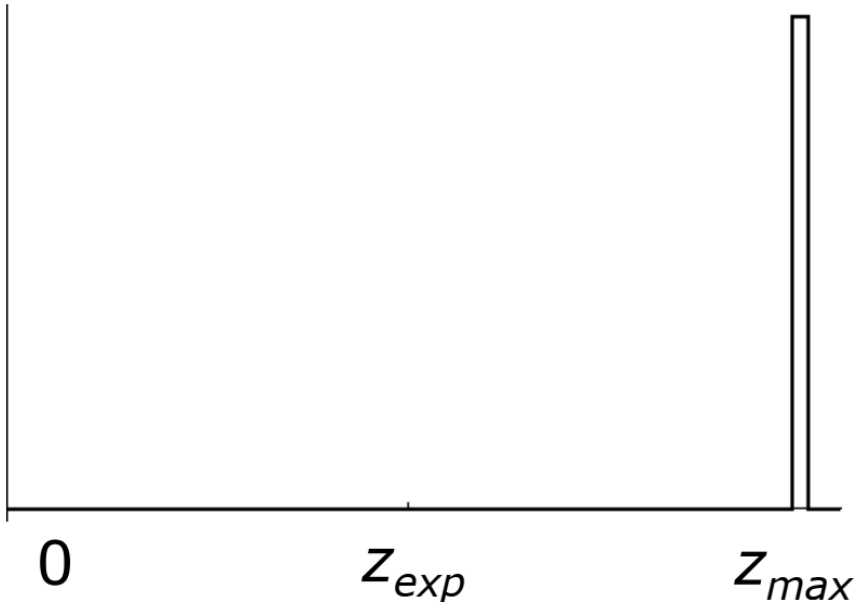
Ray-Casting Proximity Model

Measurement noise



$$P_{hit}(z \mid x, m) = \eta \frac{1}{\sqrt{2\pi b}} e^{-\frac{1}{2}\frac{(z-z_{\exp})^2}{b}}$$

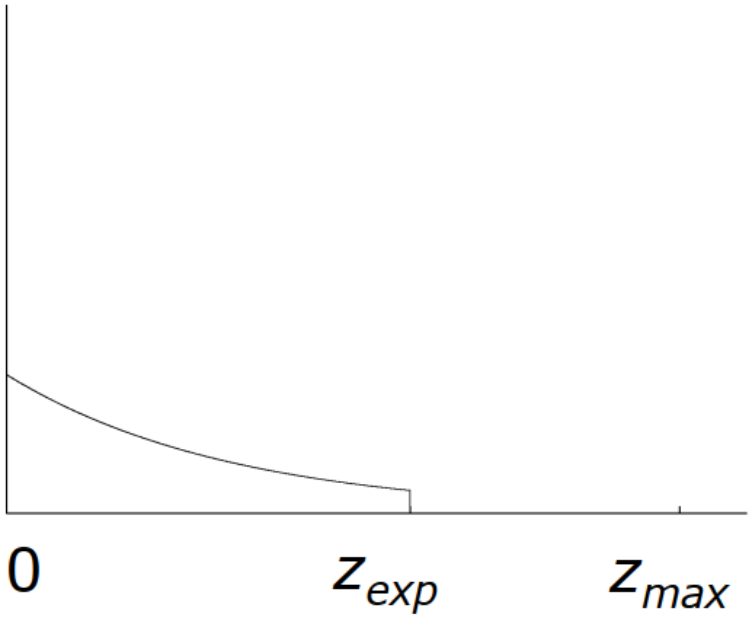
Max range



$$P_{\max}(z \mid x, m) = \eta \frac{1}{z_{small}}$$

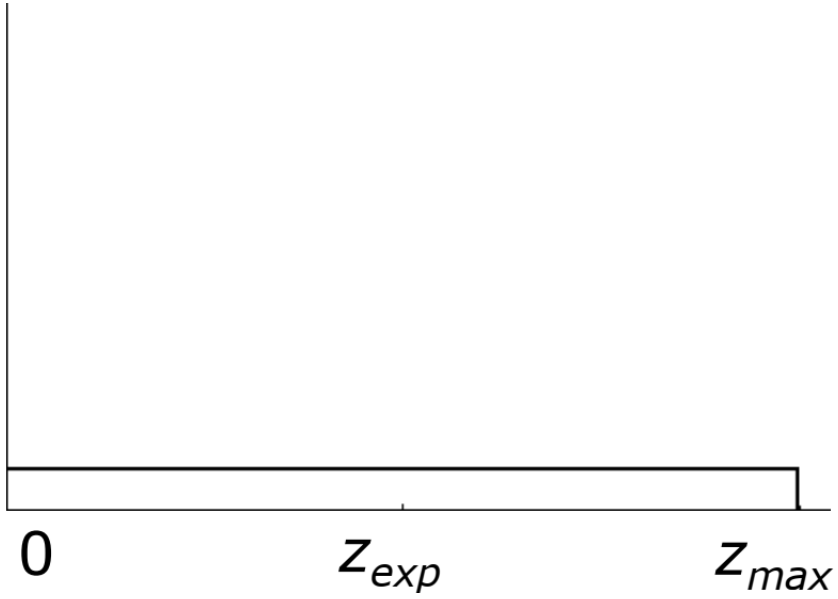
Ray-Casting Proximity Model

Unexpected obstacles



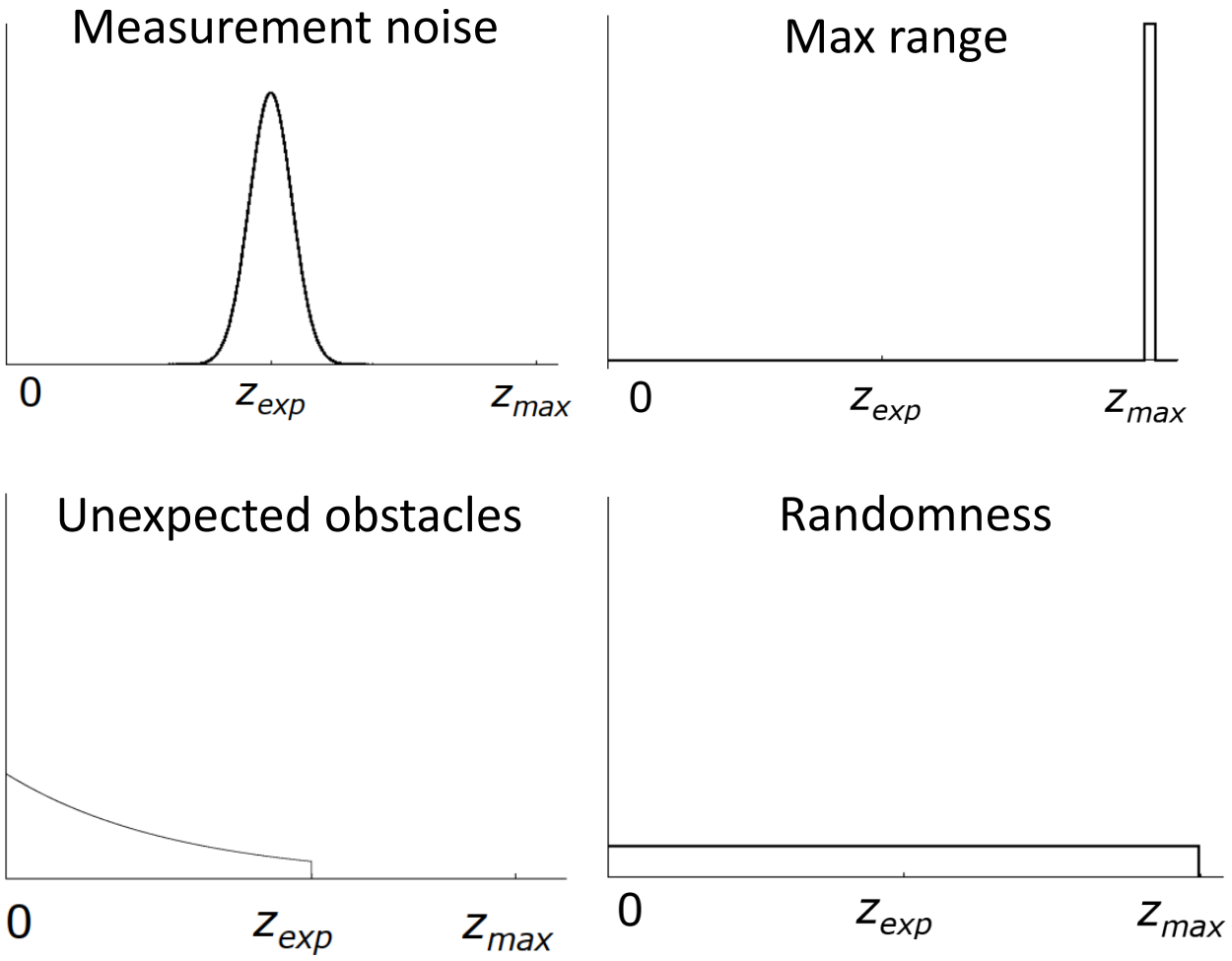
$$P_{\text{unexp}}(z \mid x,m) = \begin{cases} \eta\ \lambda\ \mathrm{e}^{-\lambda z} & z < z_{\text{exp}} \\ 0 & otherwise \end{cases}$$

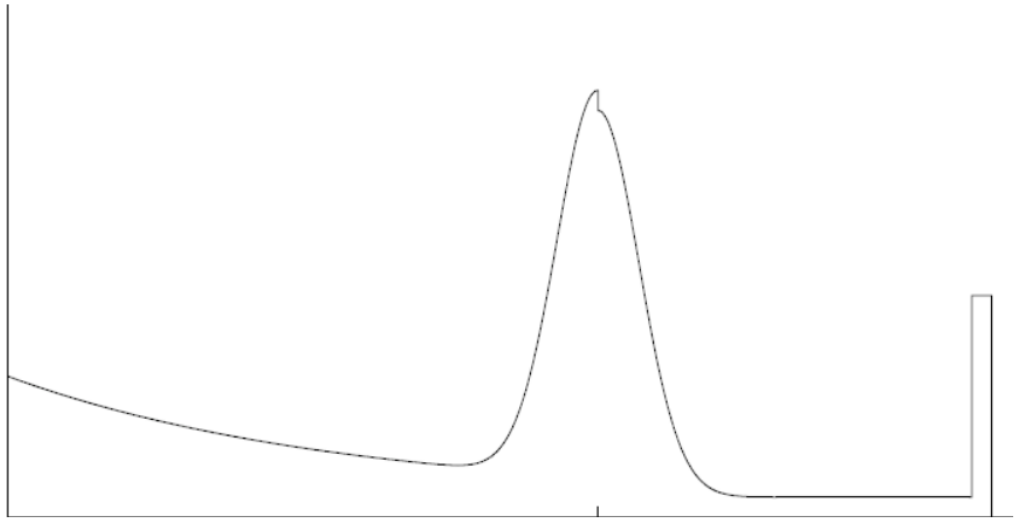
Randomness



$$P_{rand}(z \mid x,m) = \eta \frac{1}{z_{\max}}$$

Ray-Casting Proximity Model

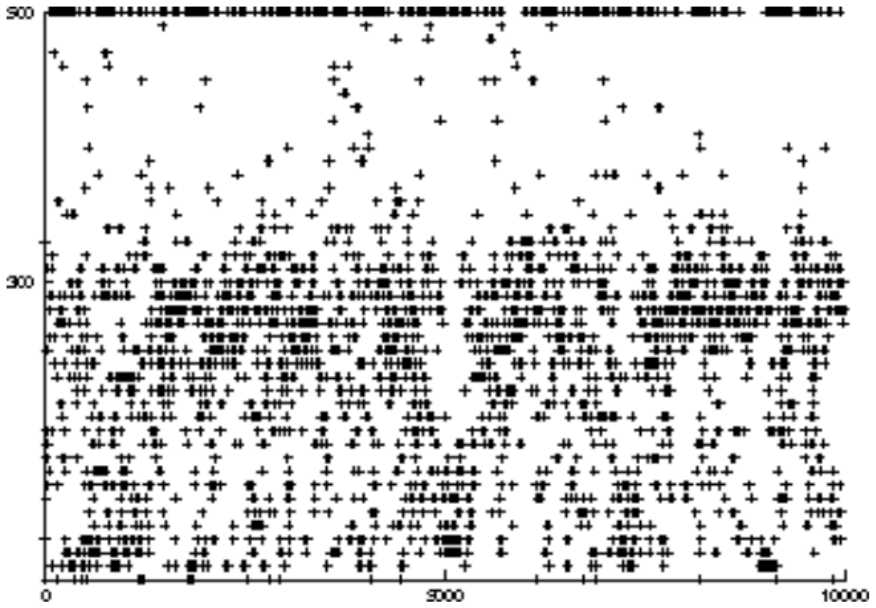




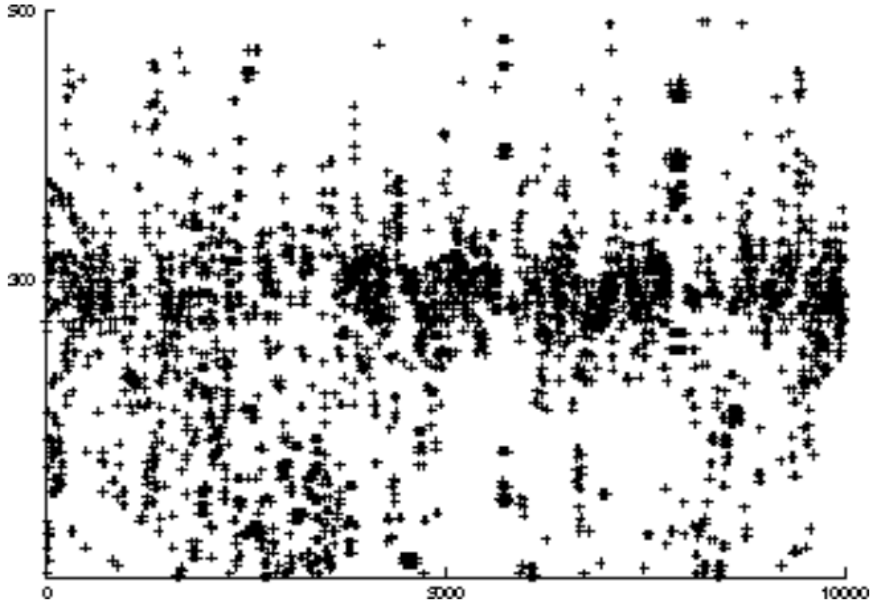
$$P(z \mid x,m) = \begin{pmatrix} \alpha_{\text{hit}} \\ \alpha_{\text{unexp}} \\ \alpha_{\text{max}} \\ \alpha_{\text{rand}} \end{pmatrix}^{T} \cdot \begin{pmatrix} P_{\text{hit}}(z \mid x,m) \\ P_{\text{unexp}}(z \mid x,m) \\ P_{\text{max}}(z \mid x,m) \\ P_{\text{rand}}(z \mid x,m) \end{pmatrix}$$

Examples: Raw Data

- Measured distances for expected distance of 300 cm.
- Parameters can be estimated using Maximum Likelihood Estimation (MLE).

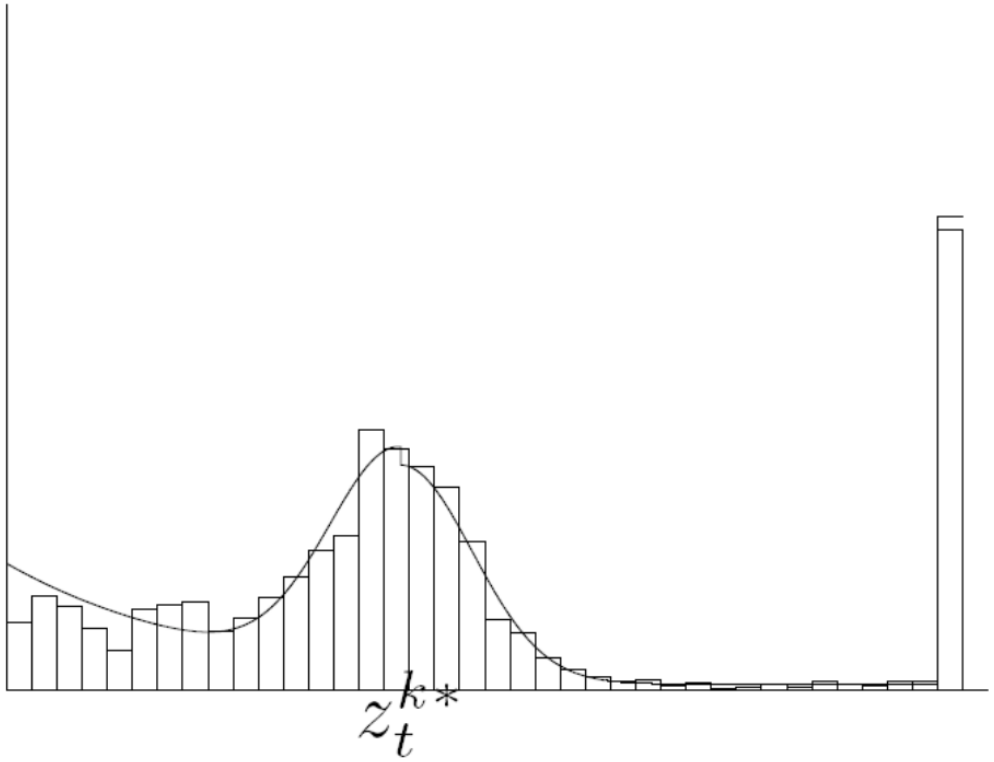


Sonar sensors

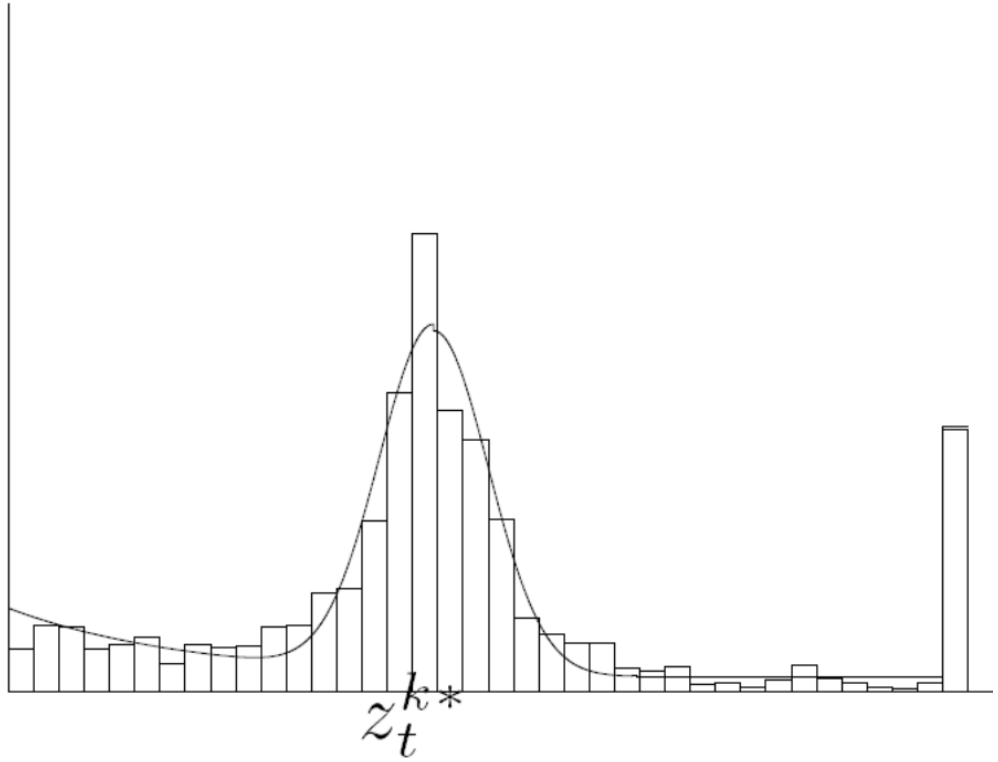


(old) Laser sensors

Examples: Approximation Results



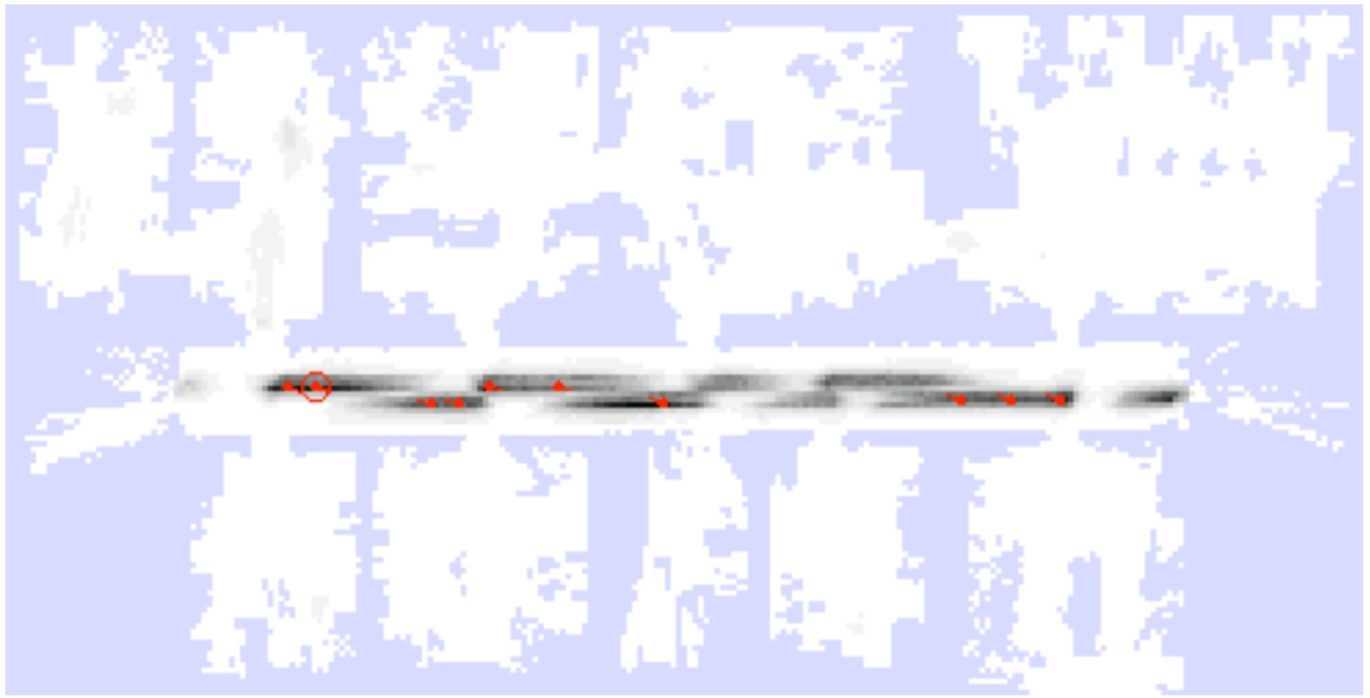
Sonar sensors



Laser sensors

Examples

z



$P(z|x,m)$

Beam-Endpoint Model

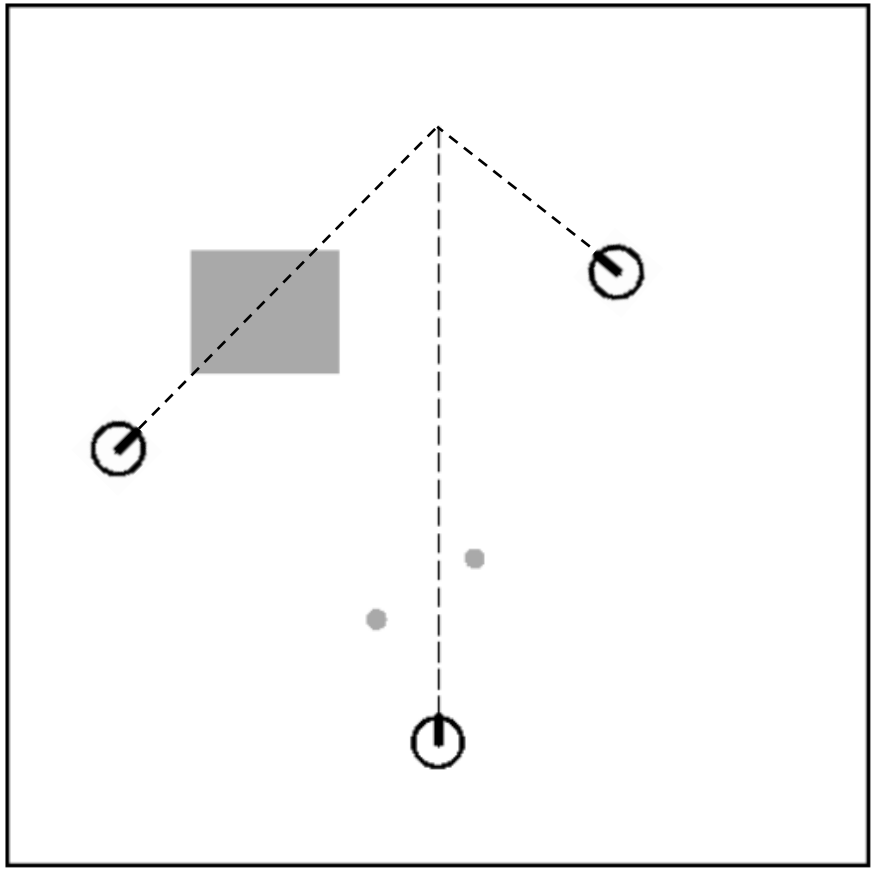
- Pure Beam-Based Model is
  - not smooth for small obstacles and at edges.
  - not easy and efficient to compute.

- Idea for **Beam-Endpoint Model** to solve this:
  - Instead of following along the beam, just check the end point (of the beam).
  - Simple version: whether there is an obstacle at the end point.
  - More sophisticated version: what is the distance from the end point to the nearest obstacle.

Likelihood Field Model

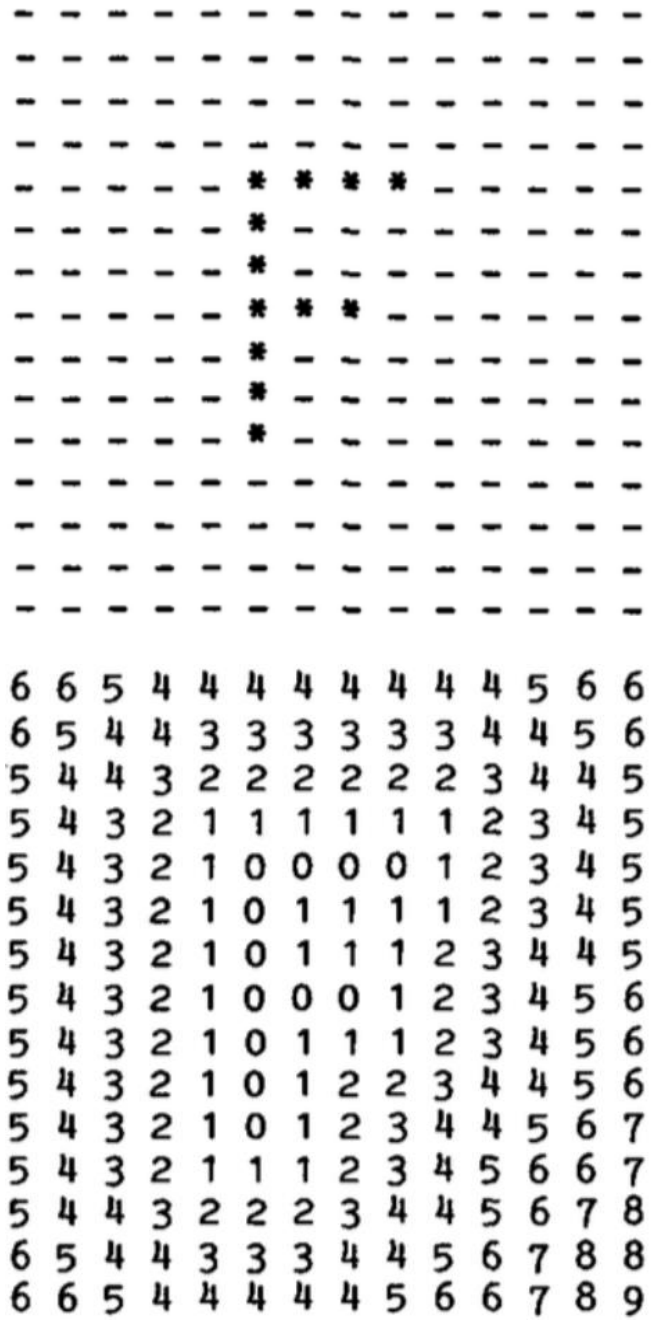
1. Given the robot's location and the (penetrating) beam, compute the location of the beam end $(x_{z_t^k} \quad y_{z_t^k})$.
2. Compute the Euclidean distance $dist$ between the end point and the end point's nearest object in map m.
3. The probability of a LiDAR sensor measurement is given by a zero-centered Gaussian:

$$p_{\text{hit}}(z_t^k \mid x_t, m) \quad = \quad \varepsilon_{\sigma_{\text{hit}}^2}(dist^2)$$



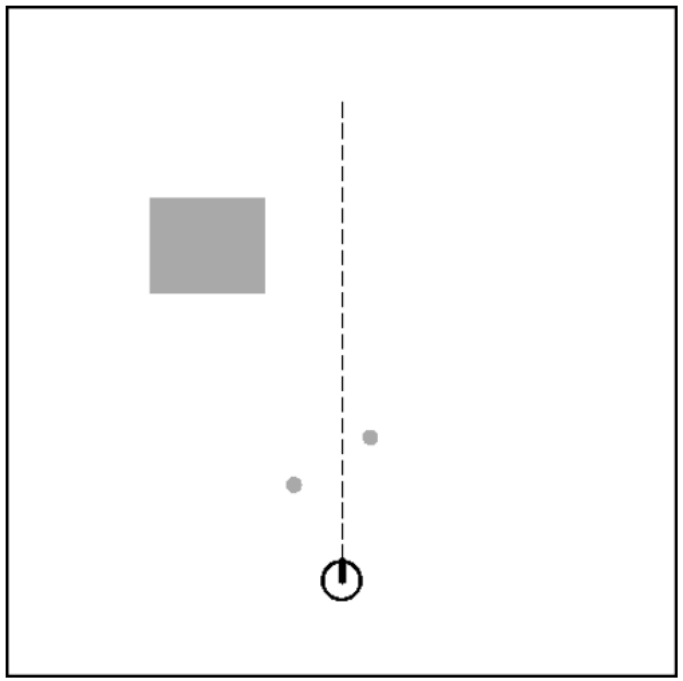
Likelihood Field Model

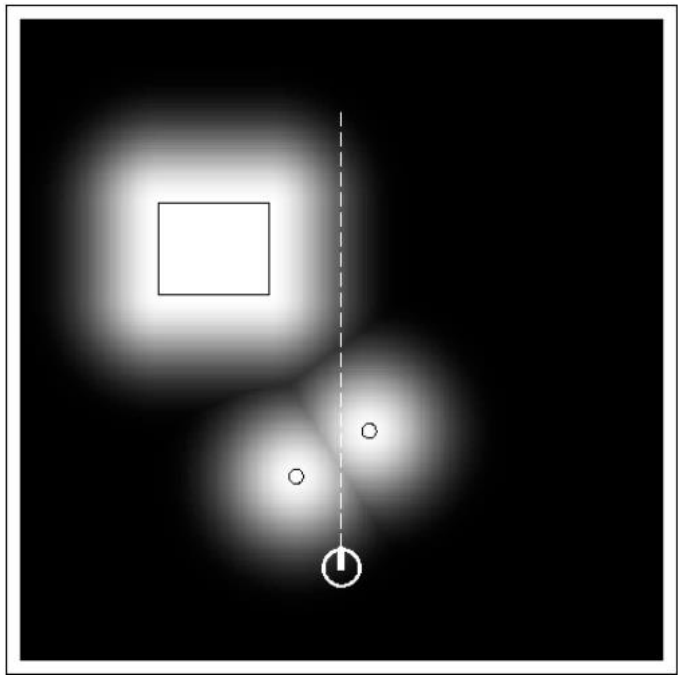
- The distance at each point to the nearest obstacle can be pre-computed, and stored into a distance matrix or field as a lookup table.
- The value at each point in this field indicates the distance to the nearest obstacle:
  - Brute force calculation
  - Distance transformation (from CV)
  - Brushfire algorithm (from planning)



Likelihood Field Model

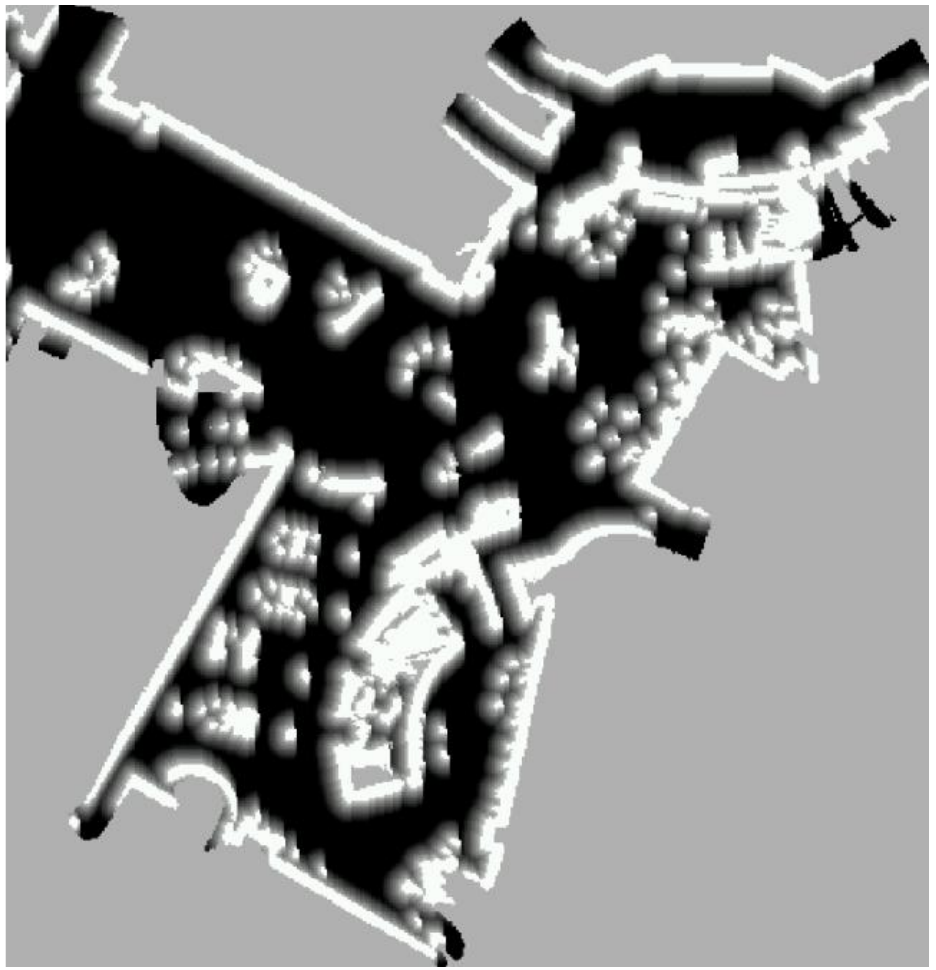
- The distance matrix or field can be saved as an image, which can also be corrupted by Gaussian noise centered at each point.





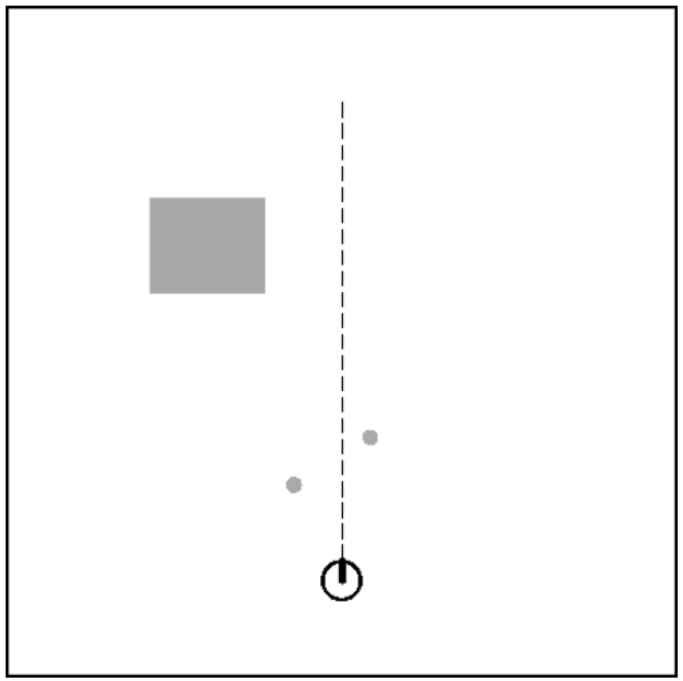
- The brighter a location:
  - The closer it is to the nearest obstacle.
  - The more likely it is to measure an obstacle with a range finder.

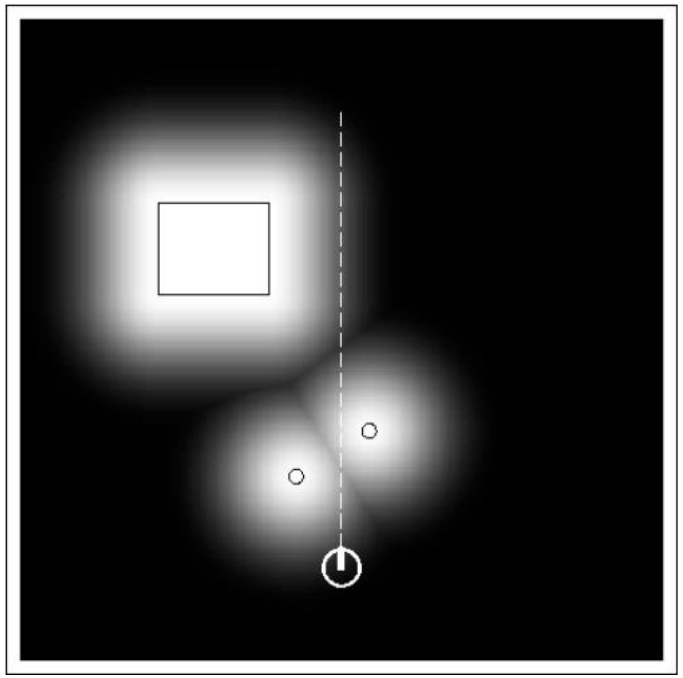
Likelihood Field Model

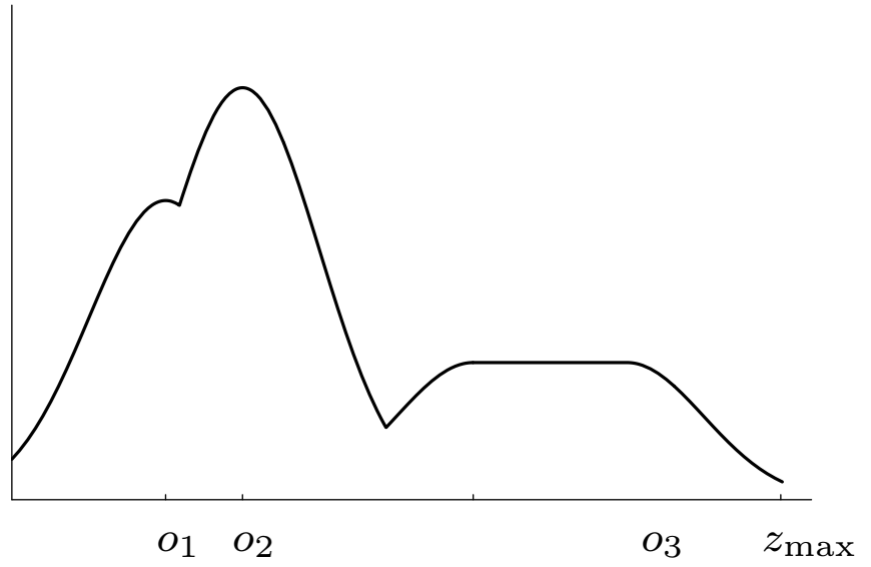


Likelihood Field Model

- The density p_{hit} of a *beam-based* model can be obtained by intersecting (and normalizing) the likelihood field by the sensor axis (e.g., indicated by the dashed line)



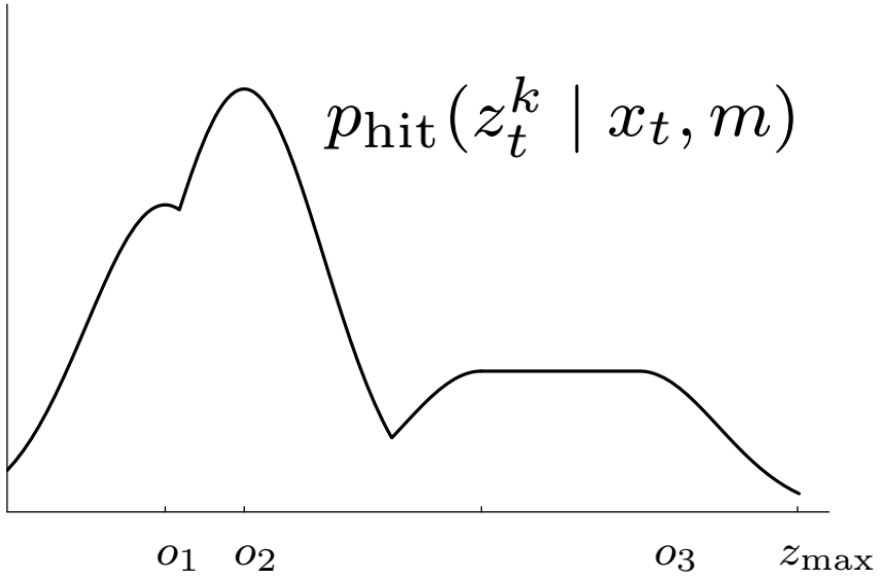




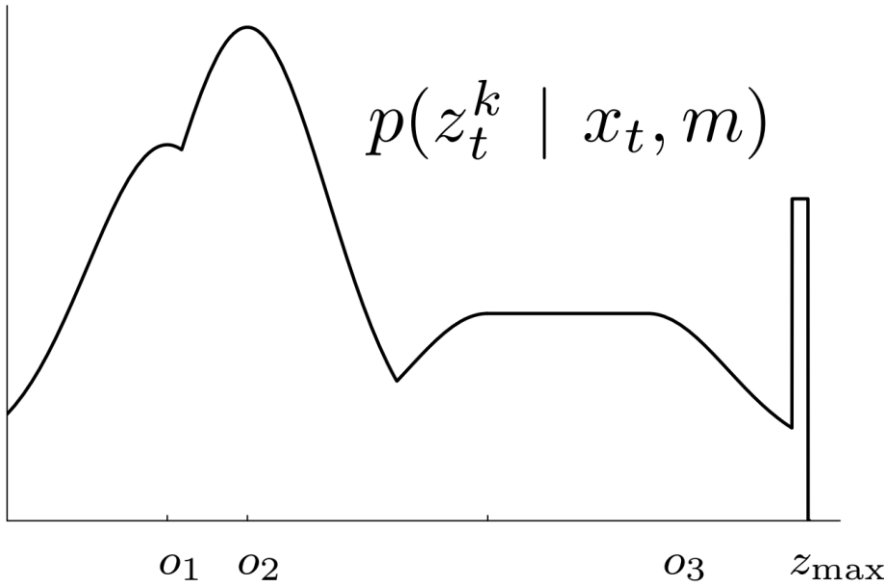
$$p_{\text{hit}}(z_t^k \mid x_t, m)$$

Likelihood Field Model

- So far, we only considered the measurement noise model.
- Two other components: maximum range $p_{\max}$, and randomness p_{rand}.



Measurement only



Full model

Do we need to consider the unexpected obstacle model?

Likelihood Field Model

- Given robot pose x and LiDAR reading $\boldsymbol{z}$ with k beams, compute $p(\boldsymbol{z}|x)$:

1: **Algorithm likelihood_field_range_finder_model**(z_t, x_t, m)**:**

2: $q = 1$

3: *for all* k *do* — Multiply the individual values of $p(z_t^k \mid x_t, m)$

4: *if* $z_t^k \neq z_{\max}$ — If the sensor reading is a max range reading, ignore it

Compute end point location

5: $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$

6: $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$

Check lookup table (dist matrix)

7: $dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \;\middle|\; \langle x', y' \rangle \text{ occupied in } m \right\}$

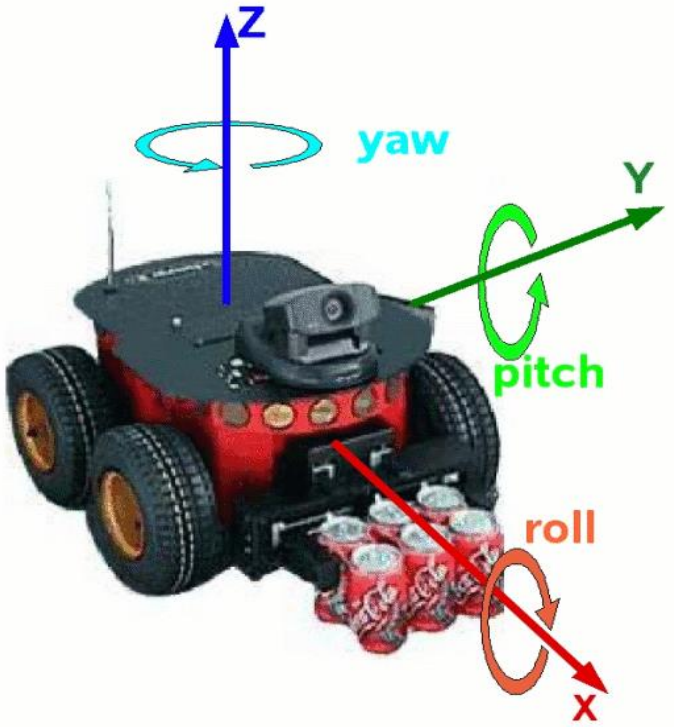
Compute probability density

8: $q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$

9: *return* q

Summary and an Open Discussion

- Robot coordinate frame
- Odometry from proprioceptive sensors



nav_msgs/Odometry Message

File: nav_msgs/Odometry.msg

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Calculating the Posterior Using Motion Model

x: pose or state
u: control

1. Algorithm **motion_model_odometry(x,x',u)**
2. $\delta_{trans} = \sqrt{(\bar{x}'-\bar{x})^2 + (\bar{y}'-\bar{y})^2}$
3. $\delta_{rot1} = \text{atan2}(\bar{y}'-\bar{y}, \bar{x}'-\bar{x}) - \bar{\theta}$
4. $\delta_{rot2} = \bar{\theta}'-\bar{\theta} - \delta_{rot1}$

odometry values (u)

What the odometry u tells us

5. $\hat{\delta}_{trans} = \sqrt{(x'-x)^2 + (y'-y)^2}$
6. $\hat{\delta}_{rot1} = \text{atan2}(y'-y, x'-x) - \bar{\theta}$
7. $\hat{\delta}_{rot2} = \theta'-\theta - \hat{\delta}_{rot1}$

values of interest (x,x')

What we compute from the input x and x'

8. $p_1 = \text{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 \mid \hat{\delta}_{\text{rot1}} \mid + \alpha_2 \hat{\delta}_{\text{trans}})$
9. $p_2 = \text{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \hat{\delta}_{\text{trans}} + \alpha_4 (\mid \hat{\delta}_{\text{rot1}} \mid + \mid \hat{\delta}_{\text{rot2}} \mid))$
10. $p_3 = \text{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 \mid \hat{\delta}_{\text{rot2}} \mid + \alpha_2 \hat{\delta}_{\text{trans}})$
11. return $p_1 \cdot p_2 \cdot p_3$

This posterior answers: what is the probability density of seeing x', given x and u under RTR?

Sampling from Odometry Motion Model

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

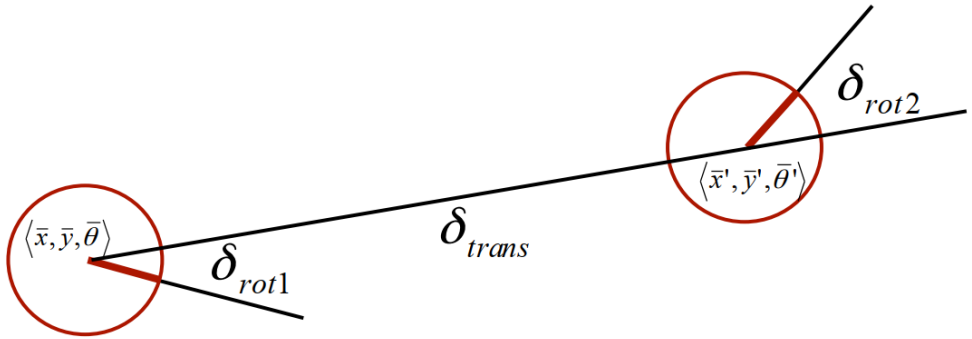
$$u = \langle \delta_{rot1}, \delta_{rot2}, \delta_{trans} \rangle, x = \langle x, y, \theta \rangle$$

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



Likelihood Field Model

- Given robot pose x and LiDAR reading $\boldsymbol{z}$ with k beams, compute $p(\boldsymbol{z}|x)$:

1: **Algorithm likelihood_field_range_finder_model**(z_t, x_t, m)**:**

2: $q = 1$

3: *for all* k *do* — Multiply the individual values of $p(z_t^k \mid x_t, m)$

4: *if* $z_t^k \neq z_{\max}$ — If the sensor reading is a max range reading, ignore it

Compute end point location

5: $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$

6: $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$

Check lookup table (dist matrix)

7: $dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \;\middle|\; \langle x', y' \rangle \text{ occupied in } m \right\}$

Compute probability density

8: $q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$

9: *return* q

Particle Filters

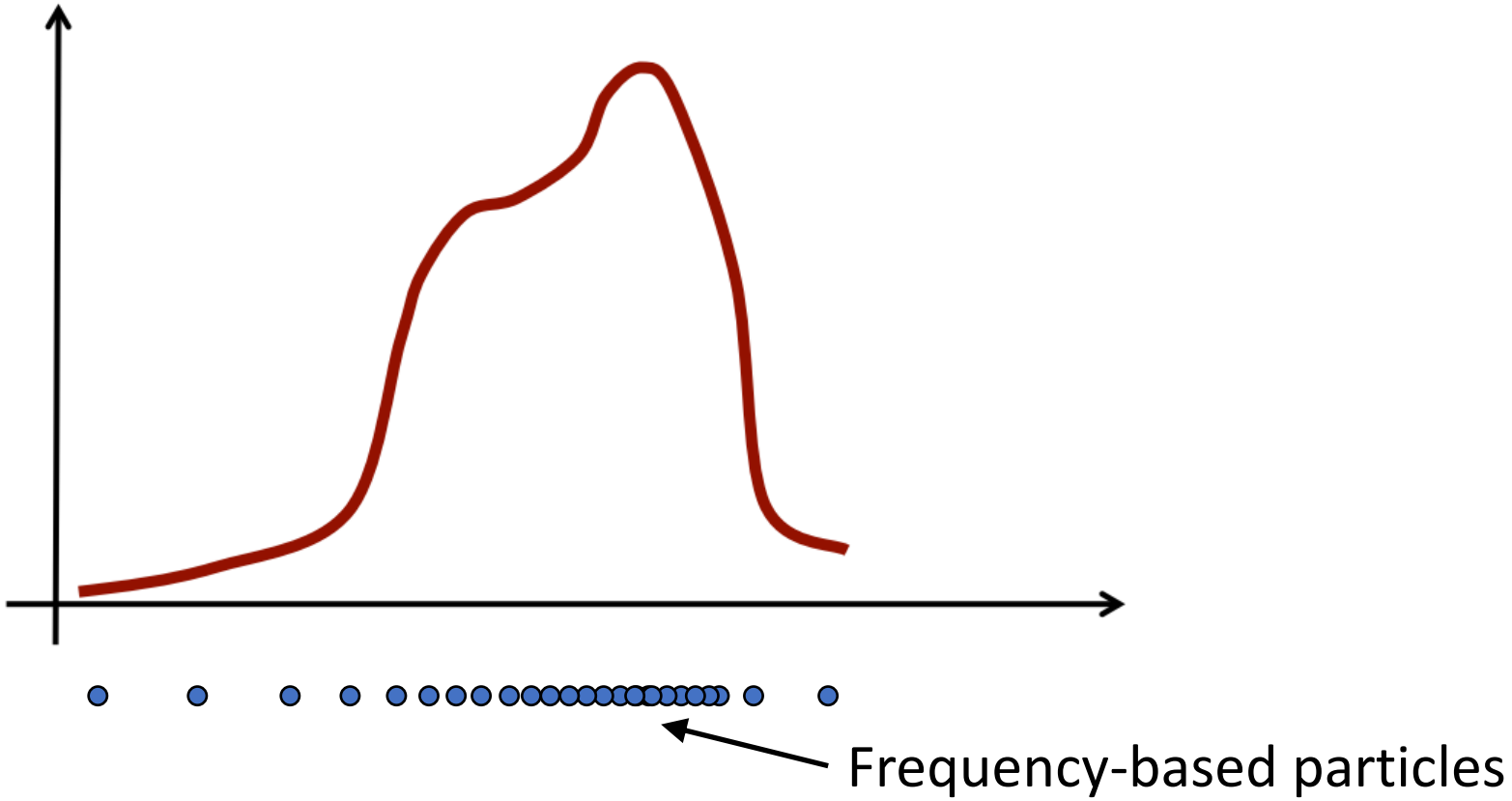
Chapter 4.2, Sebastian Thrun, Wolfram Burgard and Dieter Fox. "Probabilistic Robotics." MIT Press. 2005.

Particle Filter

- Definition:
  - Particle filter is a Bayesian filter that samples the whole hypothesis space by a weight function derived from the previous belief (and motion/sensor models).
  - Particle filter is a Monte Carlo method – a computational method that relies on sampling to obtain numerical results.
- In robot localization,
  - Particle filter is used to estimate robot poses that are non-Gaussian and non-linear in general.
  - Samples or “particles” are used to represent state hypothesis (i.e., poses) and belief (as a distribution).
  - Particle filter updates its belief through survival of the fittest particles (best fit to sensor observations).

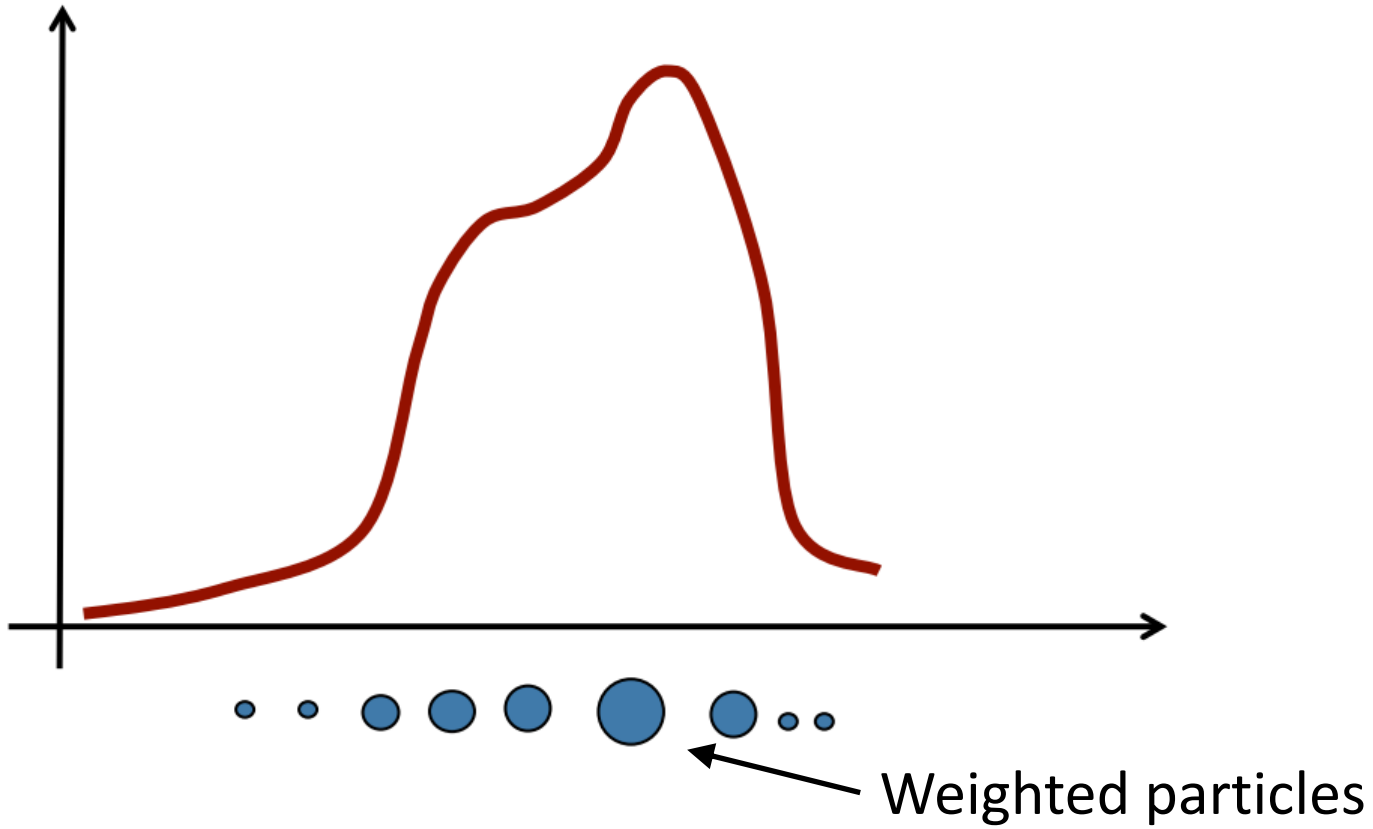
Weighted Samples

- Goal: dealing with arbitrary distributions using a smaller number of samples, or particles in the context of particle filters.



Weighted Samples

- Key idea: multiple **weighted particles** to represent an arbitrary distribution with a smaller number of particles.



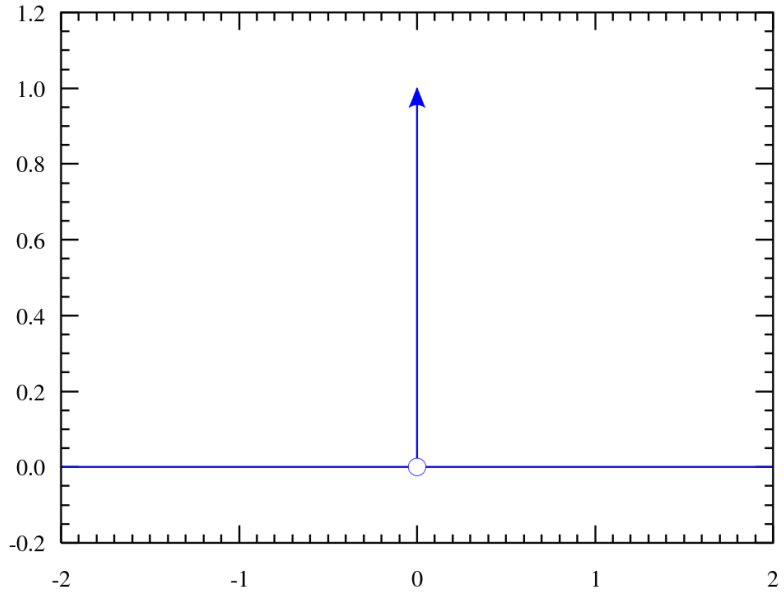
Particle Set

- Set of weighted particles:

$$\mathcal{X} = \left\{ \left\langle x^{[j]}, w^{[j]} \right\rangle \right\}_{j=1,\ldots,J}$$

state hypothesis

importance weight



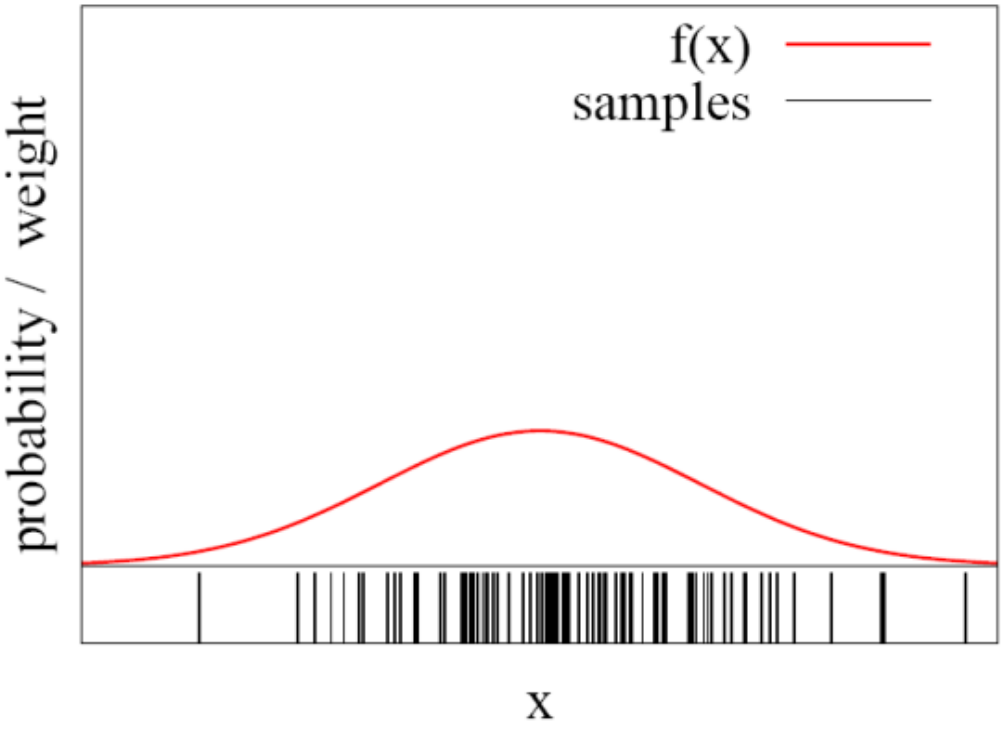
- The particles represent the posterior

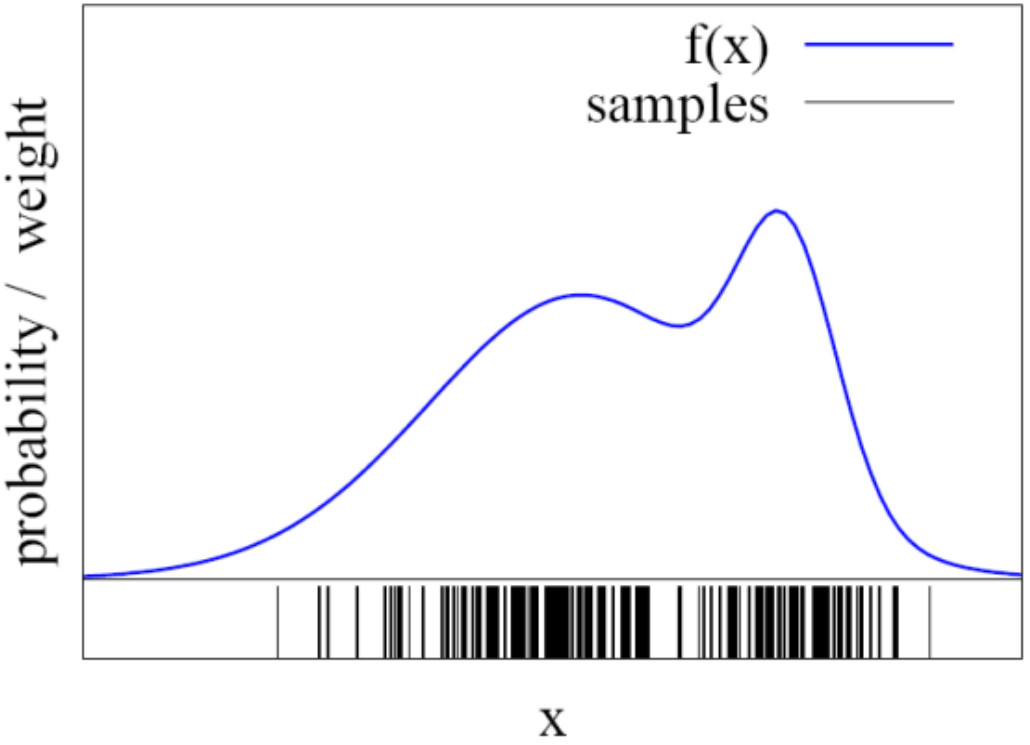
$$p(x) = \sum_{j=1}^{J} w^{[j]} \, \delta_{x^{[j]}}(x)$$

Delta (or Dirac) function

Particles for Approximation

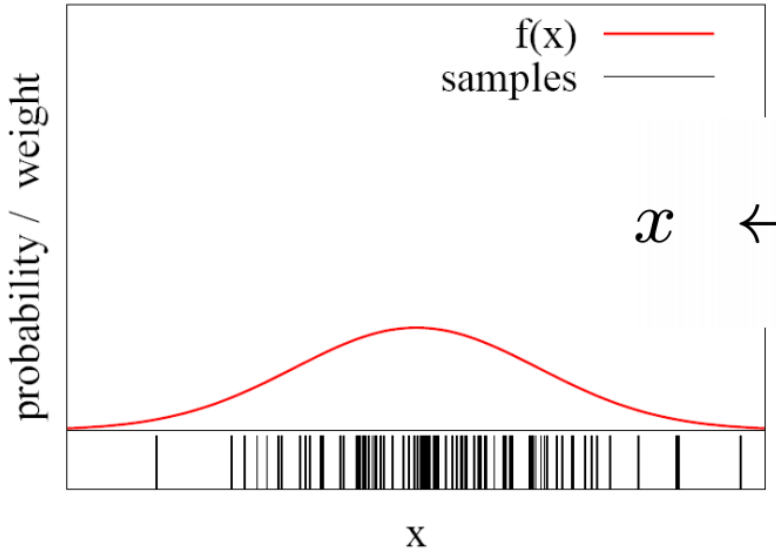
- We know that particles can be used to approximate a PDF.
- But how to draw particles form a distribution?





Closed Form Sampling

- Closed form sampling is only possible for a few distributions.
- Sampling from a Gaussian distribution:



$$x \leftarrow \frac{1}{2}\sum_{i=1}^{12} \text{rand}(-\sigma, \sigma)$$

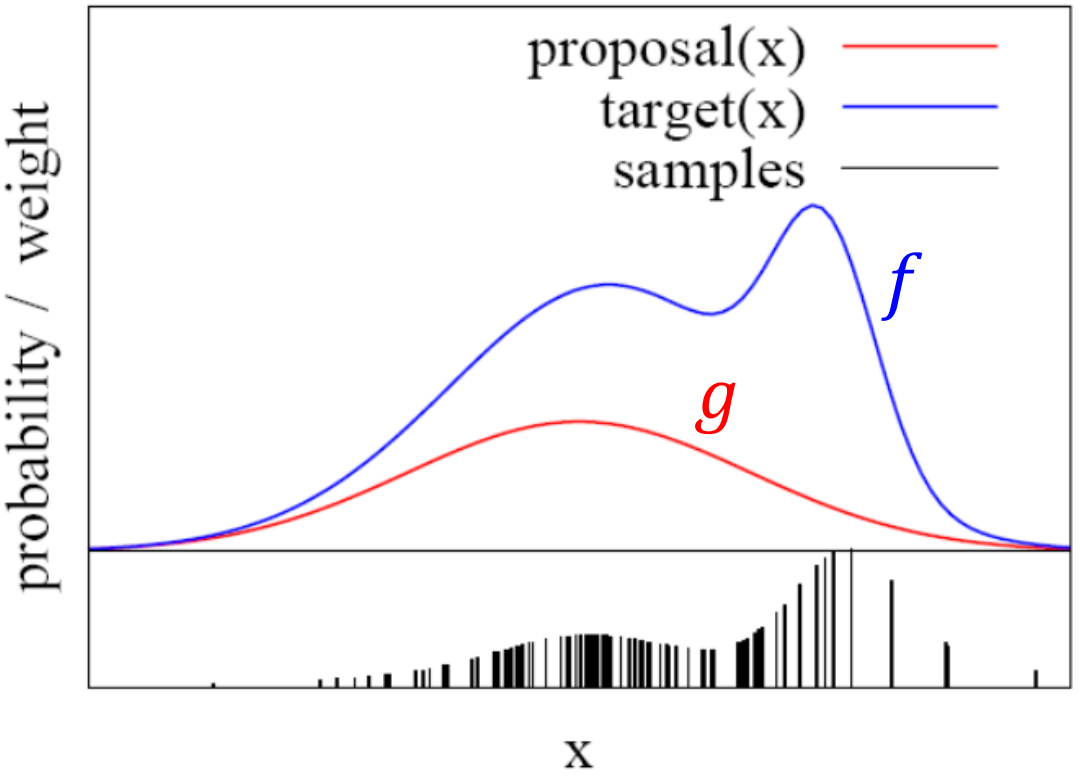
How to sample from other or arbitrary distributions?

Importance Sampling Principle

- We can use a different distribution g to generate samples from f.
- Target distribution f (blue).
- Proposal distribution g (red).
- Pre-condition:

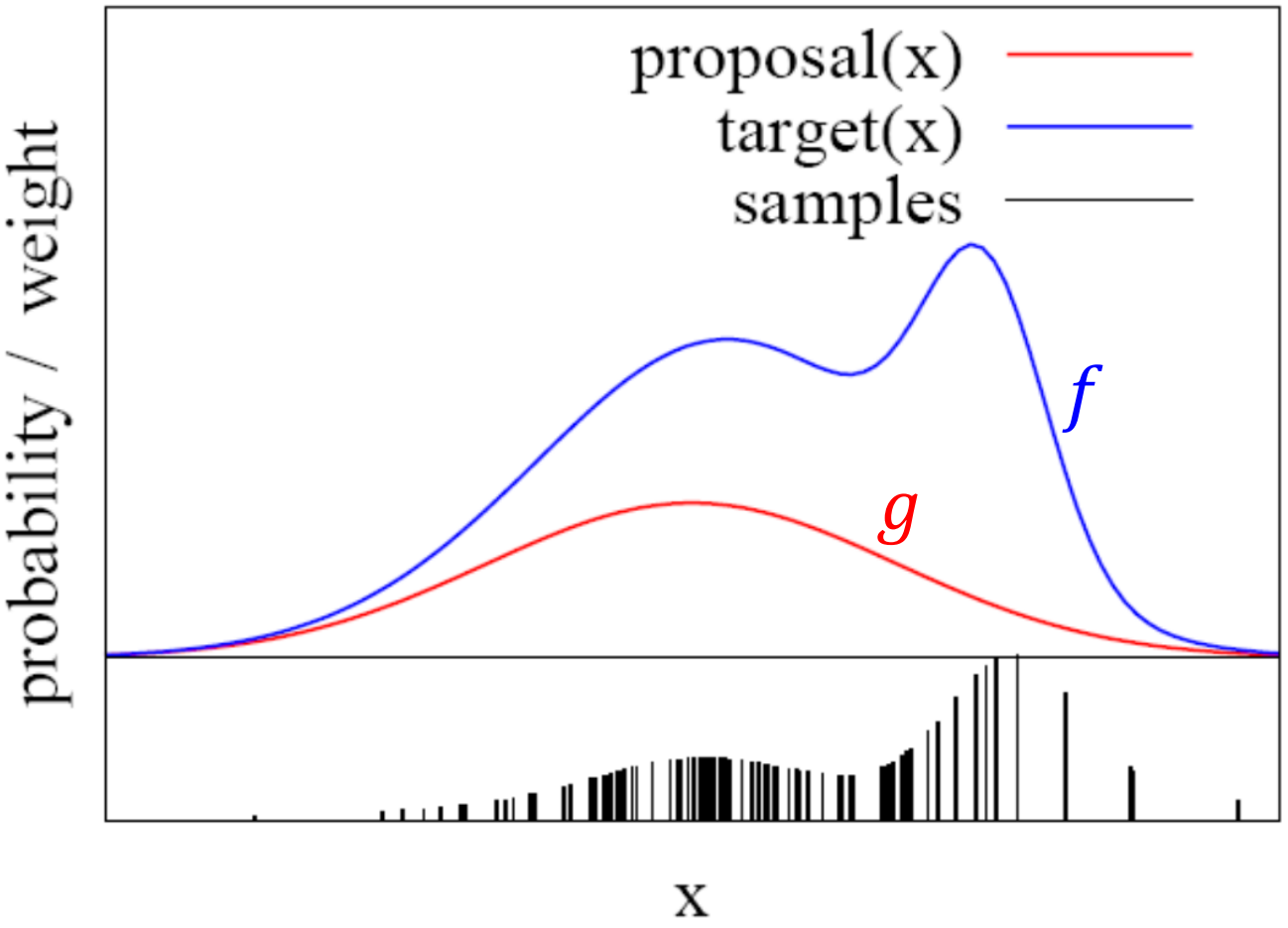
$$f(x) > 0 \rightarrow g(x) > 0$$

- Account for the "differences between g and f " using a weight $w = f(x) / g(x)$ at a particular x.



Importance Sampling Principle

- Draw particles from g.
- Compute weight for each particle:
  - In the figure, the taller the particle is, the higher weight the particle has.
- Use the weighted particles to represent f.



Particle Filter for Bayesian State Estimation

- Particle filter is a non-parametric method to implement a recursive Bayes filter:
  - when distributions are not Gaussian.
  - (often) when models are non-linear, e.g., motion models.
- The more particles we use, the better is the estimate.
- Prediction <-> drawing from the proposal.
- Correction <-> weighting by the ratio of target and proposal.

Prediction: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx$

Correction: $bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$

x : state (e.g., pose)
u : control
z : observation

Particle Filter Algorithm

1. Sample particles using the proposal distribution:

$$x_t^{[j]} \sim \pi(x_t \mid \ldots)$$

2. Compute the importance weights:

$$w_t^{[j]} = \frac{target(x_t^{[j]})}{proposal(x_t^{[j]})}$$

The proposal distribution is user defined. Then the math equation of the weight must be manually derived.
It is a design question!

3. Resampling: Draw sample i with probability $w_t^{[i]}$ and repeat J times:
   - The newly sampled particles have equal weight.
   - This allows us to convert the weighted particles to frequency-based particles to represent the same distribution.

Particle Filter Algorithm

Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t$

2: *for* $j = 1$ *to* J *do*

3: *sample* $x_t^{[j]} \sim \pi(x_t)$

4: $w_t^{[j]} = \frac{p(x_t^{[j]})}{\pi(x_t^{[j]})}$

5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$

6: *endfor*

7: *for* $j = 1$ *to* J *do*

8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$

9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$

10: *endfor*

11: *return* $\mathcal{X}_t$

x : state (e.g., pose)

u : control (e.g., odometry)

z : observation (e.g., from LiDAR)

$\bar{\mathcal{X}}$: temporary particle set

$\mathcal{X}$: particle set after resampling

π : proposal distribution

p : target distribution

Monte Carlo Localization

- As particle filter is a Monte Carlo method, robot localization using particle filters is also called Monte Carlo Localization.
- Proposal is the motion model:

$$x_t^{[j]} \sim p(x_t \mid x_{t-1}, u_t)$$

- Correction via the observation model to compute weights:

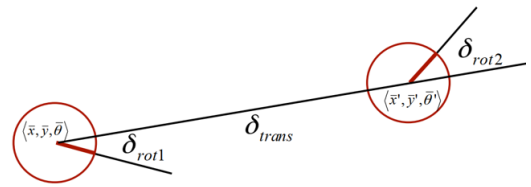
$$w_t^{[j]} = \frac{target}{proposal} \propto p(z_t \mid x_t, m)$$

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

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

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



1: **Algorithm likelihood_field_range_finder_model**(z_t, x_t, m)**:**

2: $q = 1$

3: *for all* k *do* — Multiply the individual values of $p(z_t^k \mid x_t, m)$

4: *if* $z_t^k \neq z_{\max}$ — If the sensor reading is a max range reading, ignore it

Compute end point location

5: $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$

6: $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$

Check lookup table (dist matrix)

7: $dist^2 = \min_{x',y'} \left\{ (x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2 \;\middle|\; \langle x', y' \rangle \text{ occupied in } m \right\}$

Compute probability

8: $q = q \cdot \left(z_{\text{hit}} \cdot \textbf{prob}(dist^2, \sigma_{\text{hit}}^2) + \frac{z_{\text{random}}}{z_{\max}} \right)$

9: *return* q

Monte Carlo Localization

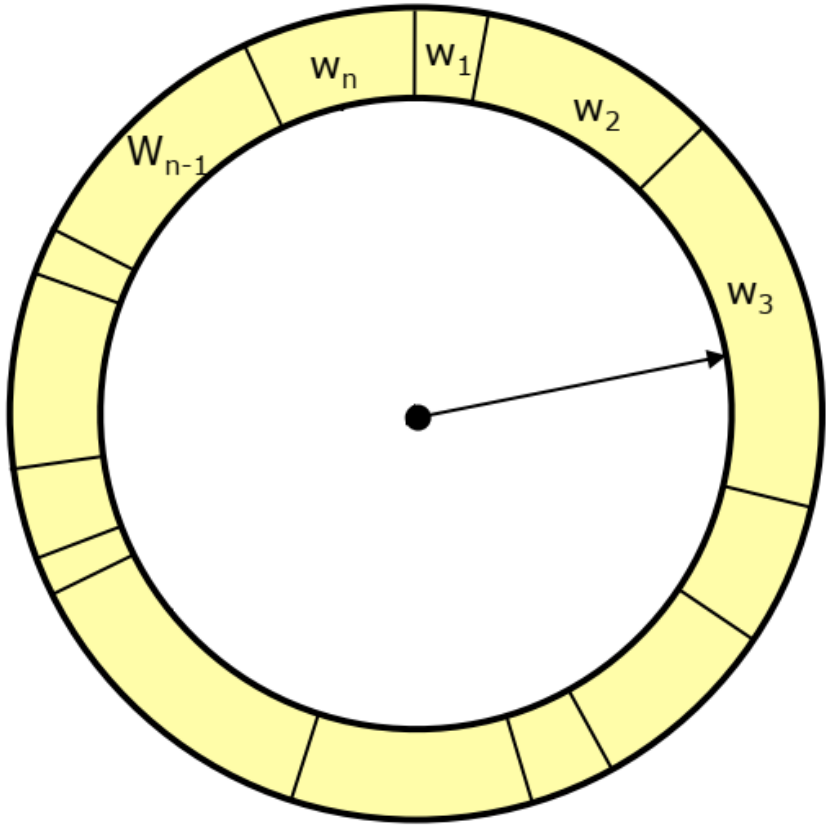
Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t$
2: *for* $j = 1$ *to* J *do*
3: *sample* $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$
4: $w_t^{[j]} = p(z_t \mid x_t^{[j]})$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6: *endfor*
7: *for* $j = 1$ *to* J *do*
8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$

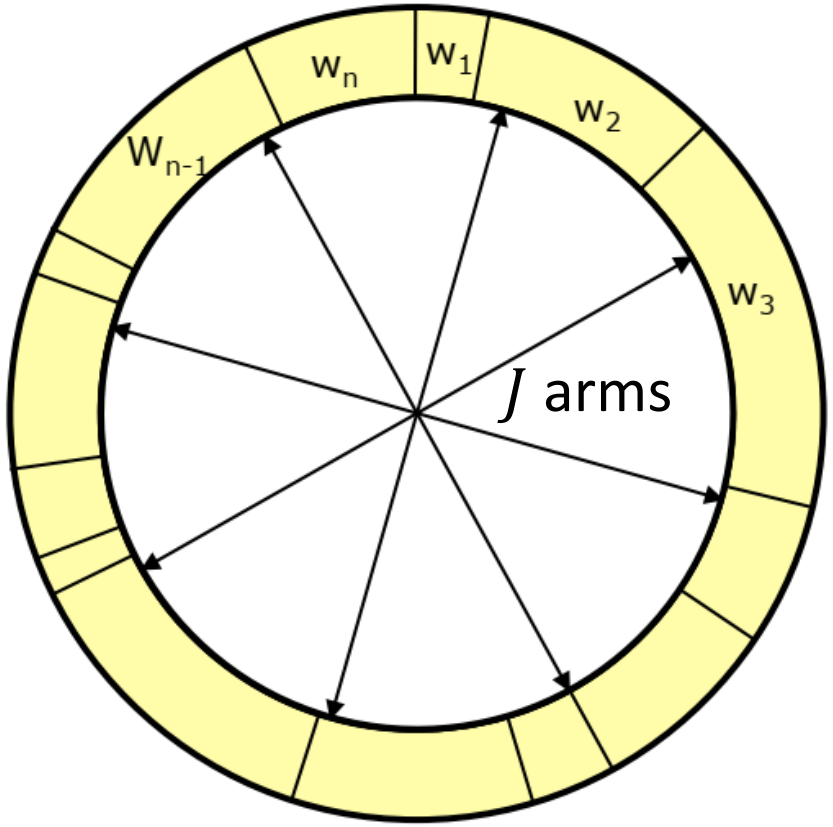
Resampling

- Draw sample i with probability $w_t^{[i]}$, repeat J times.
- Informally: “Replace unlikely samples by more likely ones”
- Survival of the fittest
- “Trick” to avoid that many samples cover unlikely states
- Needed as we have a limited number of samples

Resampling



- Roulette wheel with n bins
- Brute force $O(nJ)$, or binary search $O(nlogJ)$



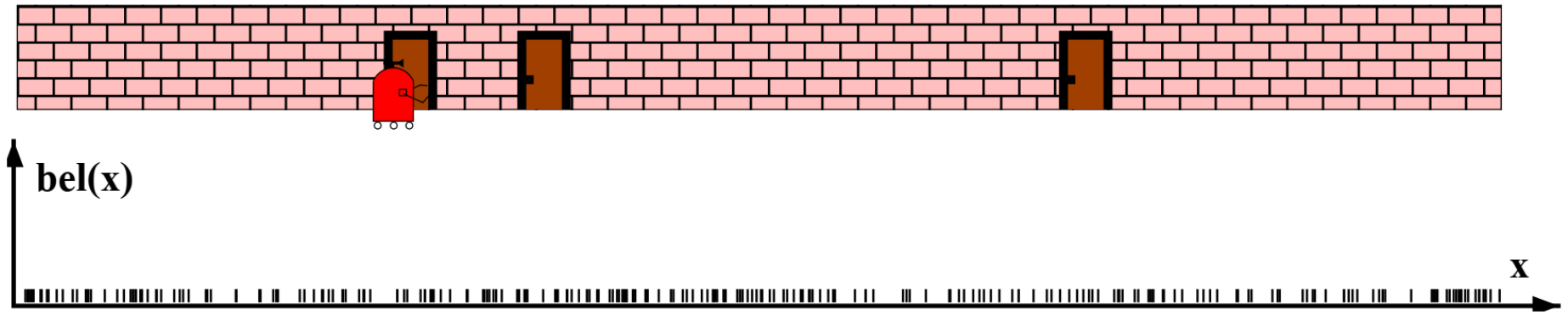
- Stochastic universal sampling
- Also called low variance sampling
- Complexity: $O(n)$

Monte Carlo Localization Example

- Solving “where am I?” using particles:

Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $j = 1$ *to* J *do*
3: *sample* $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$
4: $w_t^{[j]} = p(z_t \mid x_t^{[j]})$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6: *endfor*
7: *for* $j = 1$ *to* J *do*
8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$



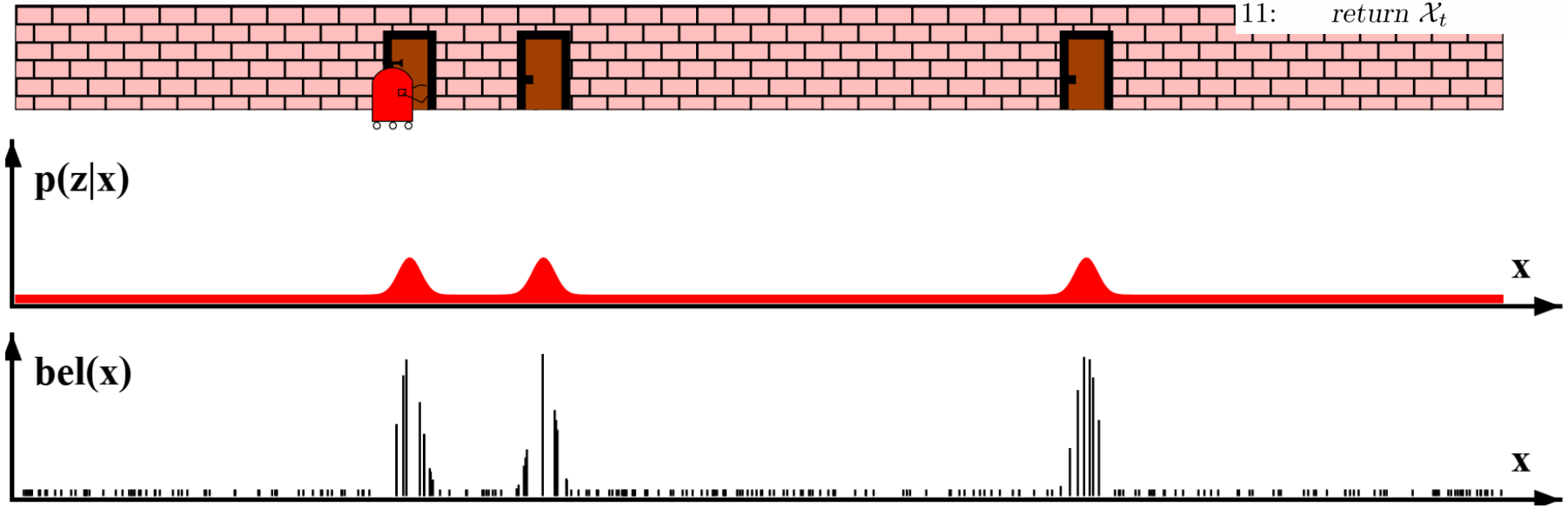
- Initialization: (*in the first step*) robot pose particles are drawn randomly and uniformly over the entire pose space.

Monte Carlo Localization Example

Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $j = 1$ *to* J *do*
3: *sample* $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$
4: $w_t^{[j]} = p(z_t \mid x_t^{[j]})$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6: *endfor*
7: *for* $j = 1$ *to* J *do*
8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$

- Solving “where am I?” using particles:



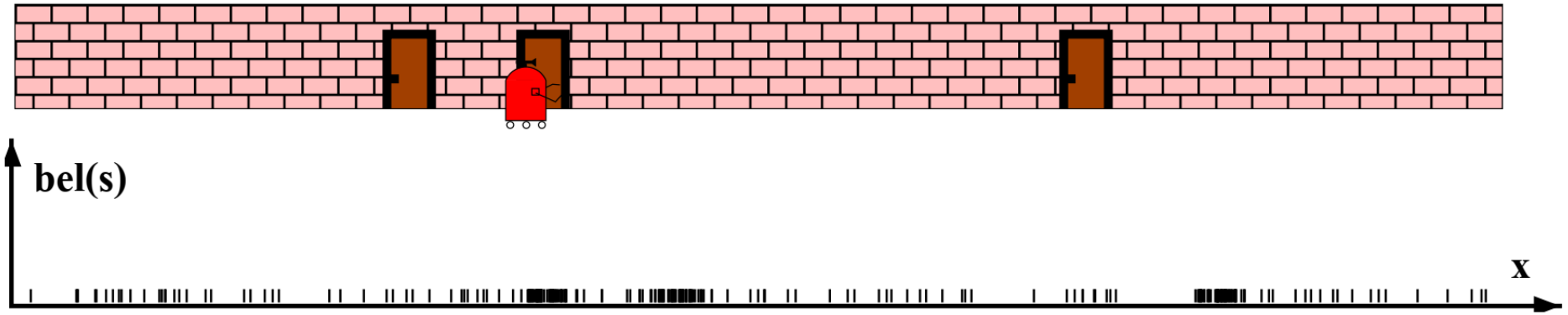
- Correction: use the observation model, Monte Carlo Localization assigns a weight to each particle.
- Resampling must be performed after each correction step.

Monte Carlo Localization Example

- Solving “where am I?” using particles:

Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $j = 1$ *to* J *do*
3: *sample* $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$
4: $w_t^{[j]} = p(z_t \mid x_t^{[j]})$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6: *endfor*
7: *for* $j = 1$ *to* J *do*
8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$



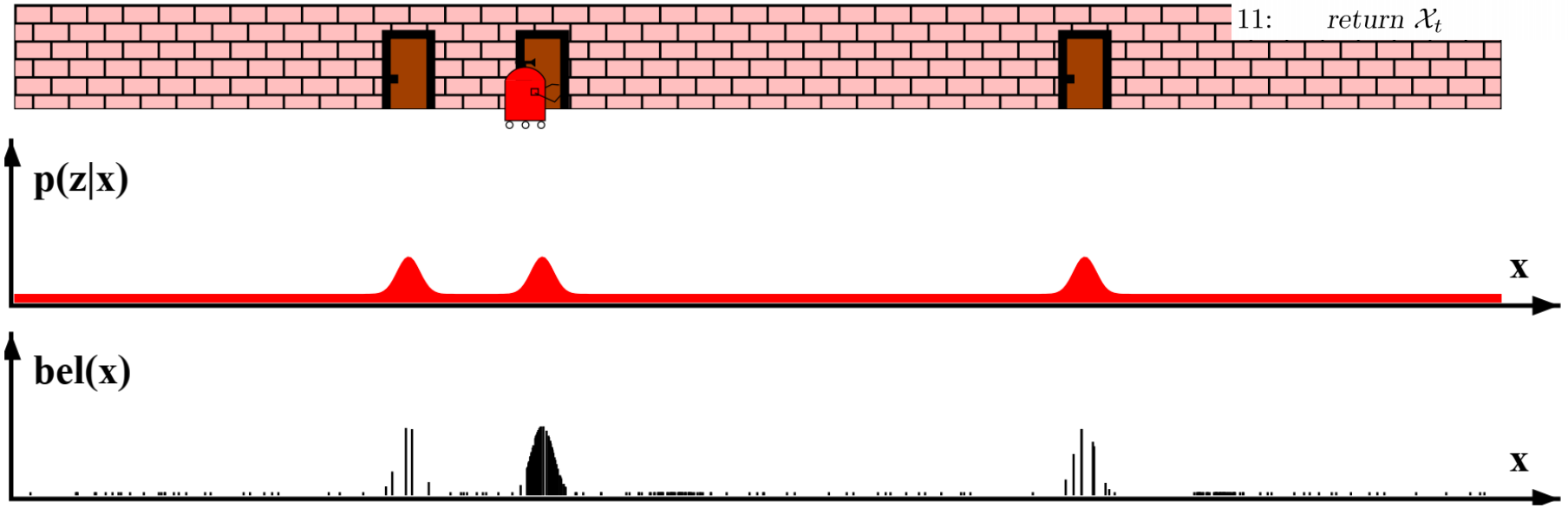
- Prediction: use the motion model to sample particles that represent the next robot poses.

Monte Carlo Localization Example

Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $j = 1$ *to* J *do*
3: *sample* $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$
4: $w_t^{[j]} = p(z_t \mid x_t^{[j]})$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6: *endfor*
7: *for* $j = 1$ *to* J *do*
8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$

- Solving "where am I?" using particles:



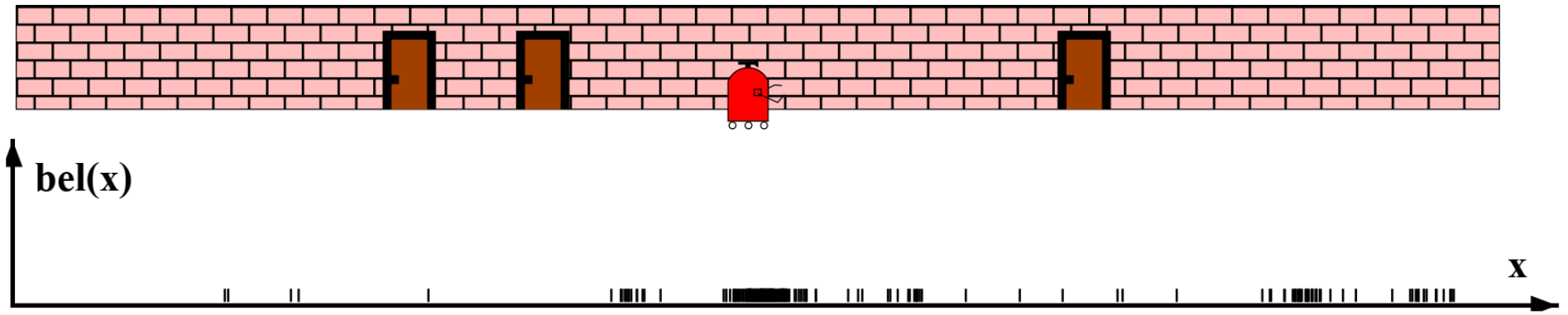
- Correction: use the observation model, Monte Carlo Localization assigns a weight to each particle.
- Resampling must be performed after each correction step.

Monte Carlo Localization Example

- Solving “where am I?” using particles:

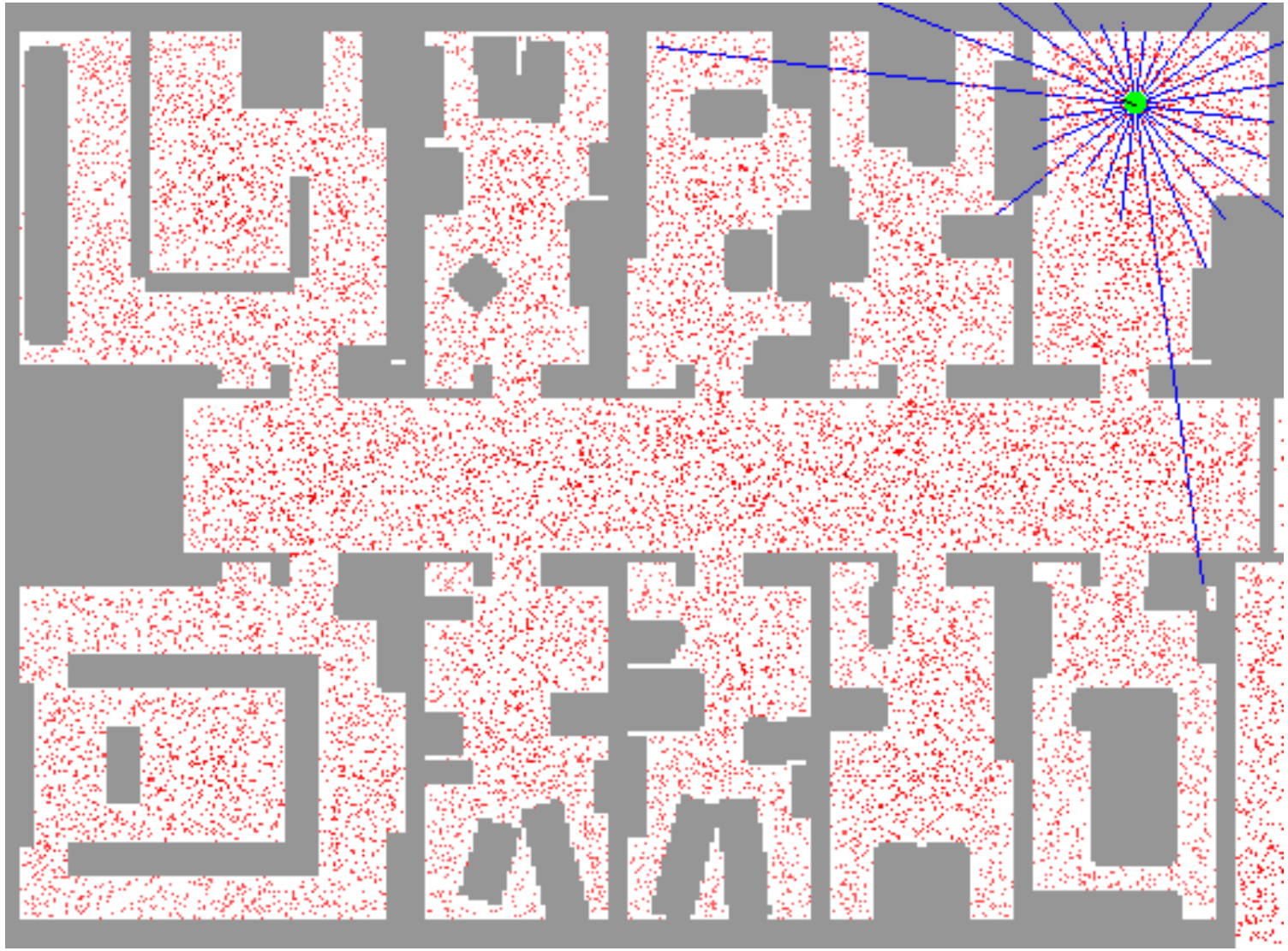
Particle_filter$(\mathcal{X}_{t-1}, u_t, z_t)$**:**

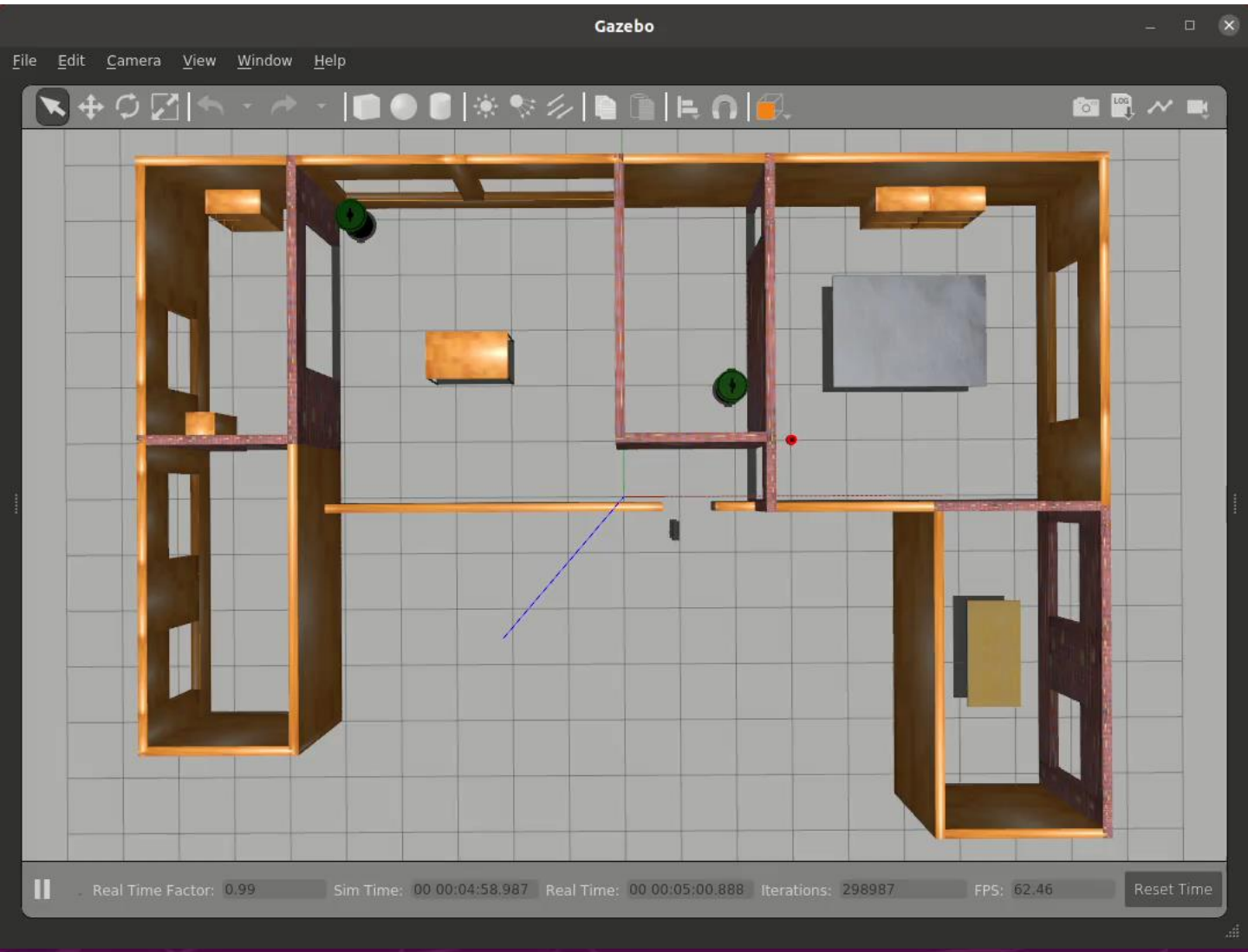
1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
2: *for* $j = 1$ *to* J *do*
3: *sample* $x_t^{[j]} \sim p(x_t \mid u_t, x_{t-1}^{[j]})$
4: $w_t^{[j]} = p(z_t \mid x_t^{[j]})$
5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[j]}, w_t^{[j]} \rangle$
6: *endfor*
7: *for* $j = 1$ *to* J *do*
8: *draw* $i \in 1, \ldots, J$ *with probability* $\propto w_t^{[i]}$
9: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
10: *endfor*
11: *return* $\mathcal{X}_t$

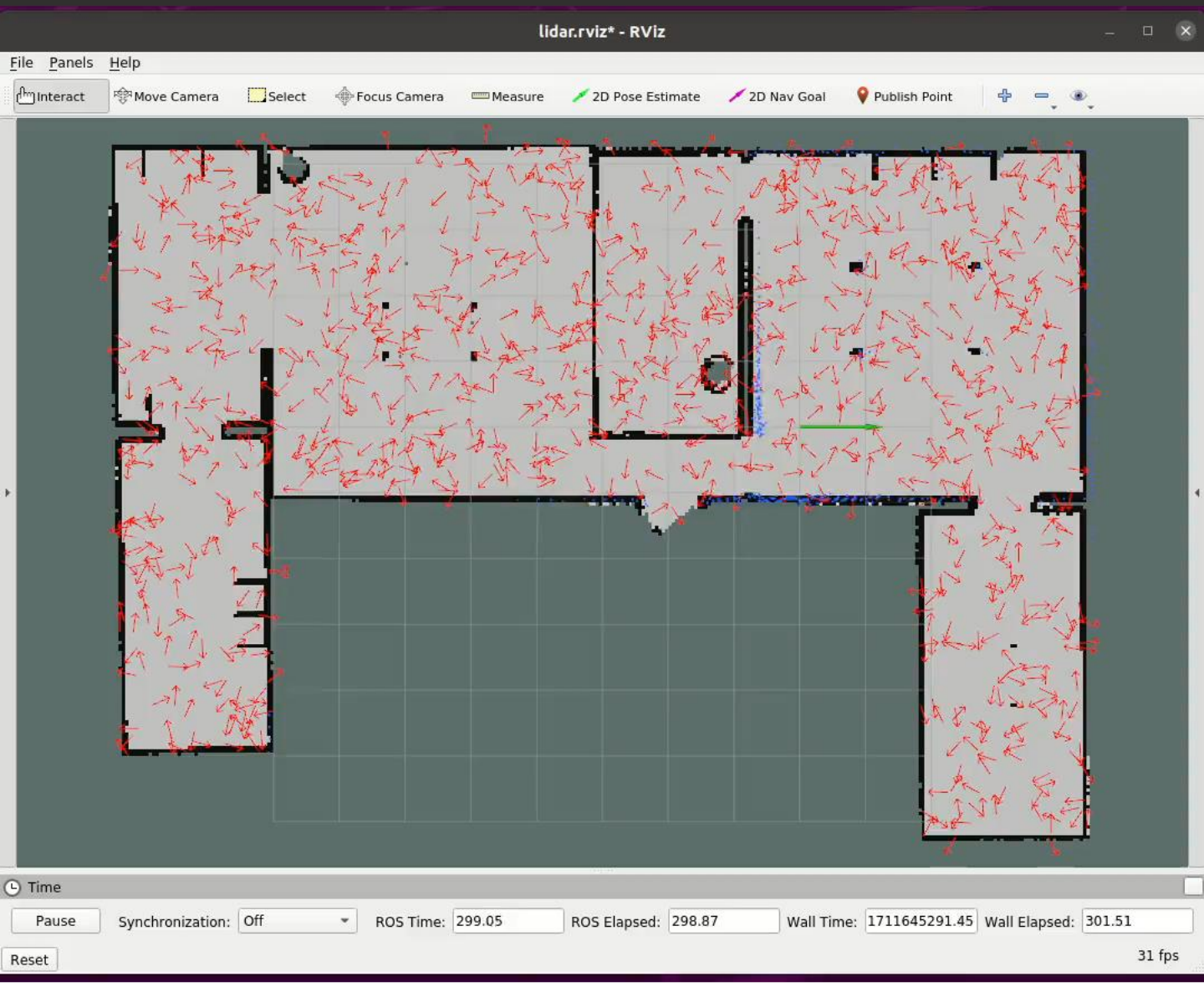


- Repeat the prediction by sampling from the motion model, correction by sampling from the observation model, and the resampling procedures.

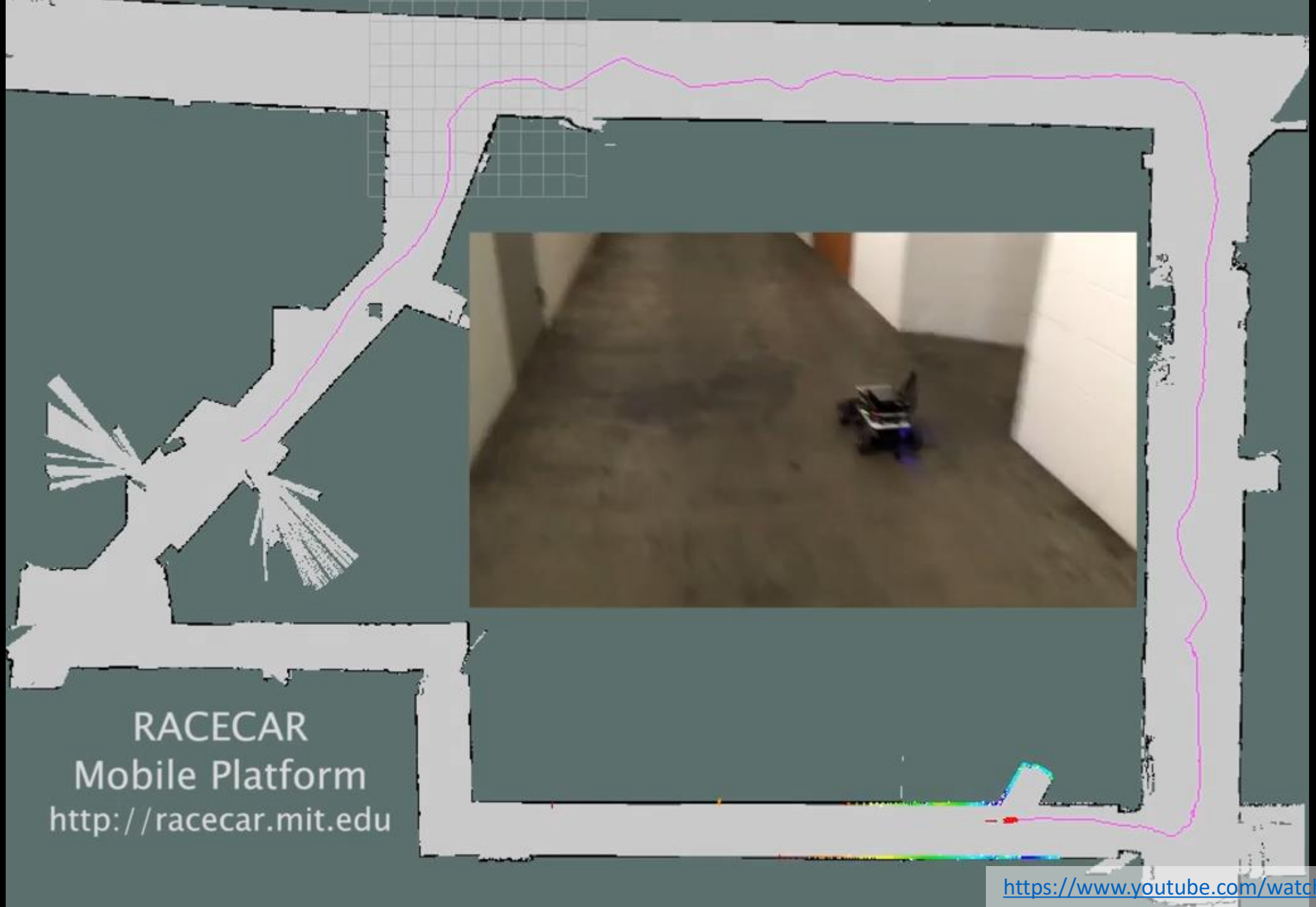
Monte Carlo Localization Example



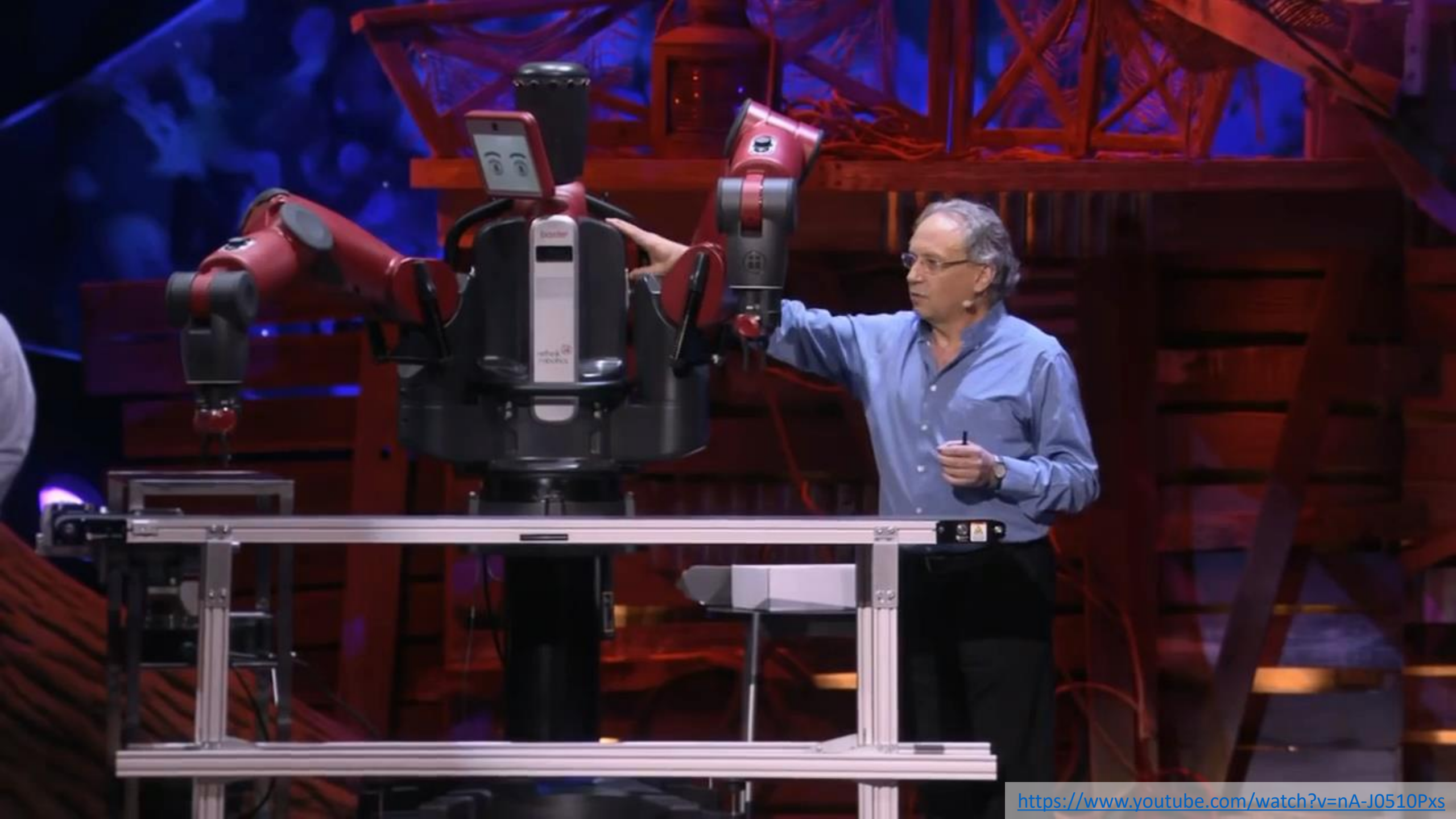

Gazebo
File Edit Camera View Window Help
Real Time Factor: 0.99
Sim Time: 00 00:04:58.987
Real Time: 00 00:05:00.888
Iterations: 298987
FPS: 62.46
Reset Time


lidar.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
Pause
Synchronization: Off
ROS Time: 299.05
ROS Elapsed: 298.87
Wall Time: 1711645291.45
Wall Elapsed: 301.51
Reset
31 fps

https://www.youtube.com/watch?v=0XbKZvXt5c4



https://www.youtube.com/watch?v=-c_0hSjgLYw



https://www.youtube.com/watch?v=nA-J0510Pxs