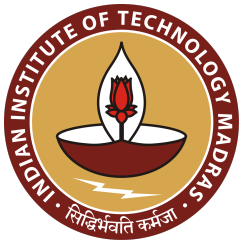
Introduction to Robotics

Week-3

Prof. Balaraman Ravindran

IIT-Madras



- We next focus on motion models, measurement models and maps that allow us to implement the filter algorithms discussed in practice.
  - Motion models comprise the state transition probability $p(x_t|x_{t-1},\ u_t)$ required in the prediction step of filter algorithms.
  - Measurement models comprise the measurement probability $p(z_t|x_t)$ required in the update step of filter algorithms.
  - Maps are specifications of the robot's environment.

Velocity Motion Models

- A velocity motion model assumes that we can control the robot through rotational and translational velocities.

$$u_t = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix}$$

- **Convention:** Positive rotational velocities ω_t induce a counterclockwise rotation (left turns). Positive translational velocities v_t correspond to forward motion.

A **closed form** algorithm for computing the probability $p(x_t|u_t, x_{t-1})$ is shown in the following slides.

It accepts as input:

- An initial pose: $x_{t-1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$
- A control $u_t = \begin{pmatrix} v \\ \omega \end{pmatrix}$
- A hypothesis successor pose $x_t = \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}$

The output of the algorithm is the probability: $p(x_t|u_t,\ x_{t-1})$ of being at x_t after executing control u_t , beginning in state x_{t-1} (assuming that the control is carried out for the fixed duration Δt).

Algorithm motion_model_velocity(x_t, u_t, x_{t-1}):

$$\mu = \frac{1}{2}\frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta}$$

$$x^* = \frac{x+x'}{2} + \mu(y-y')$$

$$y^* = \frac{y+y'}{2} + \mu(x'-x)$$

$$r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2}$$

$$\Delta\theta = \text{atan2}(y'-y^*, x'-x^*) - \text{atan2}(y-y^*, x-x^*)$$

$$\hat{v} = \frac{\Delta\theta}{\Delta t}\, r^*$$

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

$$\hat{\gamma} = \frac{\theta'-\theta}{\Delta t} - \hat{\omega}$$

return $\textbf{prob}(v-\hat{v}, \alpha_1|v| + \alpha_2|\omega|) \cdot \textbf{prob}(\omega-\hat{\omega}, \alpha_3|v| + \alpha_4|\omega|) \cdot \textbf{prob}(\hat{\gamma}, \alpha_5|v| + \alpha_6|\omega|)$

We first calculate the controls of an error-free robot, and then use the function $prob(x, b)$ to model the motion error.

$prob(x, b)$ computes the probability of the parameter x under a zero-centered random variable with variance b

Algorithm prob_normal_distribution(a, b):

return $\frac{1}{\sqrt{2\pi b}} e^{-\frac{1}{2}\frac{a^2}{b}}$

Algorithm prob_triangular_distribution(a, b):

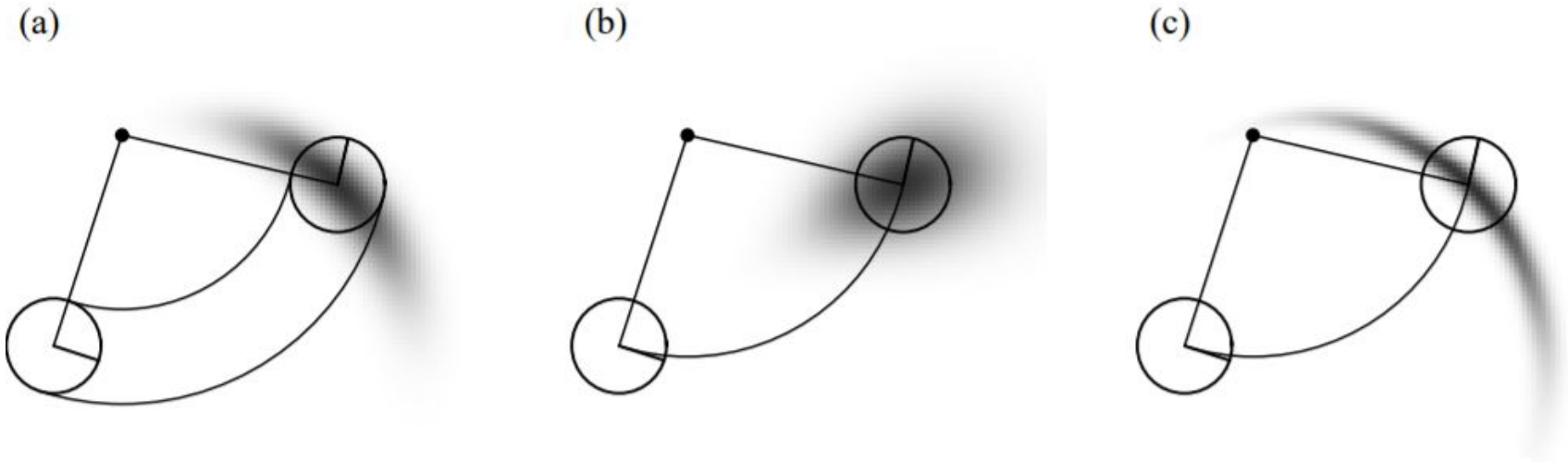
if $|a| > \sqrt{6b}$

return 0

else

return $\frac{\sqrt{6b}-|a|}{6b}$

The parameters $\alpha 1$ to $\alpha 6$ are robot-specific motion error parameters.



- Figure a shows the resulting distribution with moderate error parameters $\alpha 1$ to $\alpha 6$.
- The distribution shown in Figure b is obtained with smaller angular error (parameters $\alpha 3$ and $\alpha 4$) but larger translational error (parameters $\alpha 1$ and $\alpha 2$).
- Figure c shows the distribution under large angular and small translational error.

Motion Models

In the case of particle filters, it suffices to sample from the motion model $p(x_t|u_t,\ x_{t-1})$, instead of computing the posterior for arbitrary x_t, u_t and x_{t-1}.

A **sample algorithm** to generate random samples from $p(x_t|u_t,\ x_{t-1})$ for a fixed control u_t and pose x_{t-1} is shown in the following slides.

It accepts as input:

- An initial pose: $x_{t-1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$
- A control $u_t = \begin{pmatrix} v \\ \omega \end{pmatrix}$

and generates a random pose x_t according to the distribution $p(x_t|u_t, x_{t-1})$.

1: **Algorithm sample_motion_model_velocity(u_t, x_{t-1}):**

2: $\hat{v} = v + \textbf{sample}(\alpha_1|v| + \alpha_2|\omega|)$

3: $\hat{\omega} = \omega + \textbf{sample}(\alpha_3|v| + \alpha_4|\omega|)$

4: $\hat{\gamma} = \textbf{sample}(\alpha_5|v| + \alpha_6|\omega|)$

5: $x' = x - \frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}} \; sin(\theta + \hat{\omega}\Delta t)$

6: $y' = y + \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}} \; cos(\theta + \hat{\omega}\Delta t)$

7: $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$

8: *return* $x_t = (x', y', \theta')^T$

Lines 2 through 4 "perturb" the commanded control parameters by noise, drawn from the error parameters of the kinematic motion model. The noise values are then used to generate the sample's new pose, in Lines 5 through 7.

1: **Algorithm sample_normal_distribution(b):**

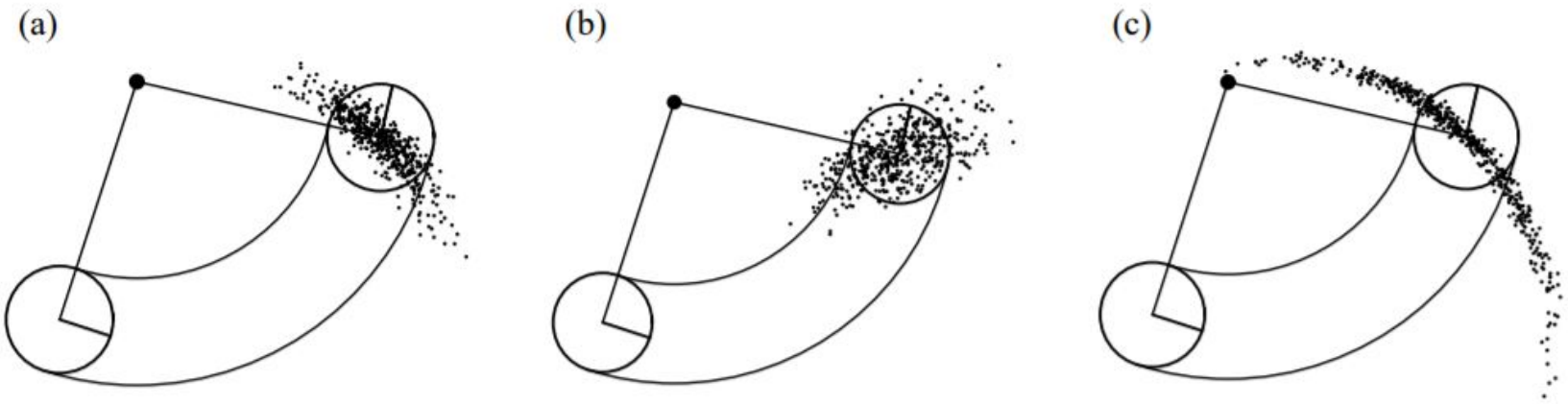
2: *return* $\frac{b}{6}\sum_{i=1}^{12}\textbf{rand}(-1,1)$

3: **Algorithm sample_triangular_distribution(b):**

4: *return* $b \cdot \textbf{rand}(-1,1) \cdot \textbf{rand}(-1,1)$

Note: In many cases, it is easier to sample x_t than calculate the density of a given x_t.

The fact that particle filters rely on sampling makes them specifically attractive from an implementation point of view



Sampling from the velocity motion model, using the same parameters as previously used. Each diagram shows 500 samples.

Odometry Motion Models

- Odometry data is obtained by integrating information from motion sensors of the robot (such as wheel encoders information).

- Alternative to using the robot's velocities to compute posteriors over poses, we can use the odometry measurements as a basis for calculating the robot's motion over time.
- Most commercial robots make such integrated pose estimation available in periodic time intervals

- Odometry, while still erroneous, is generally more accurate than velocity. Both suffer from drift and slippage, but velocity additionally suffers from the mismatch between the actual motion controllers and its mathematical model.

- Technically, odometry are sensor measurements, not controls. To model odometry as measurements, a resulting Bayes filter would have to include the actual velocity as state variables—which increases the dimension of the state space.
- **To keep the state space small, it is therefore common to simply consider the odometry as if it was a control signal.**
- The resulting model is at the core of many of today's best probabilistic robot systems

A **closed form** algorithm for computing the probability $p(x_t|u_t, x_{t-1})$ is discussed in the following slides.

- The odometry model uses the relative information of the robot's internal odometry
- Specifically, in the time interval $(t-1, t]$, if the robot advances from a pose x_{t-1} to pose x_t, the odometry reports back to us a related advance from

$$\overline{x}_{t-1} = \begin{pmatrix} \overline{x} \\ \overline{y} \\ \overline{\theta} \end{pmatrix} \text{ to } \overline{x}_t = \begin{pmatrix} \overline{x}' \\ \overline{y}' \\ \overline{\theta}' \end{pmatrix}$$

  Here the bar indicates that these are odometry measurements, embedded in a robot-internal coordinate whose relation to the global world coordinates is unknown

- The key insight for utilizing this information in state estimation is that the relative difference between $\overline{x}_{t-1}$ and $\overline{x}_t$, is a good estimator for the difference of the true poses x_{t-1} and x_t.
- The motion information u_t is, thus, given by:

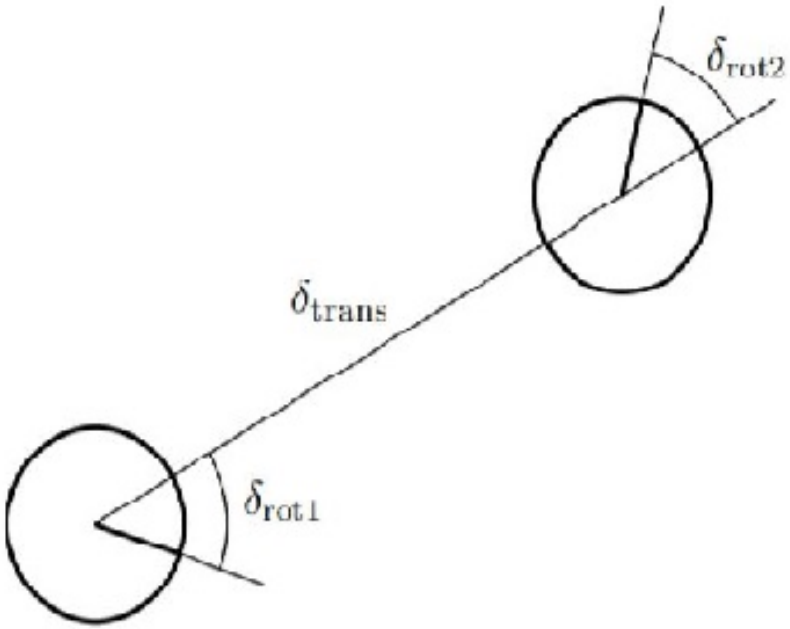
$$u_t = \begin{pmatrix} \overline{x}_{t-1} \\ \overline{x}_t \end{pmatrix}$$

To extract relative odometry, u_t is transformed into a sequence of three steps:

- A rotation $\delta rot1$
- A straight line motion (translation) $\delta trans$
- Another rotation $\delta rot2$

Motion Models

Each pair of positions $(\bar{s}, \bar{s}')$ has a unique parameter vector



$$\begin{pmatrix} \delta rot1 \\ \delta trans \\ \delta rot2 \end{pmatrix}$$

These parameters are sufficient to reconstruct the relative motion between s and s'.

- Our motion model assumes that these three parameters are corrupted by independent noise

The algorithm for calculating the probability density $p(x_t|u_t, x_{t-1})$ in closed form accepts as an input:

- An initial pose: $x_{t-1} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$
- A pair of poses $u_t = \begin{pmatrix} \overline{x}_{t-1} \\ \overline{x}_t \end{pmatrix}$ obtained from the robot's odometry
- A hypothesis successor pose $x_t = \begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix}$

Algorithm motion_model_odometry(x_t, u_t, x_{t-1}):

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

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

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

$\hat{\delta}_{\text{rot1}} = \text{atan2}(y' - y, x' - x) - \theta$

$\hat{\delta}_{\text{trans}} = \sqrt{(x - x')^2 + (y - y')^2}$

$\hat{\delta}_{\text{rot2}} = \theta' - \theta - \hat{\delta}_{\text{rot1}}$

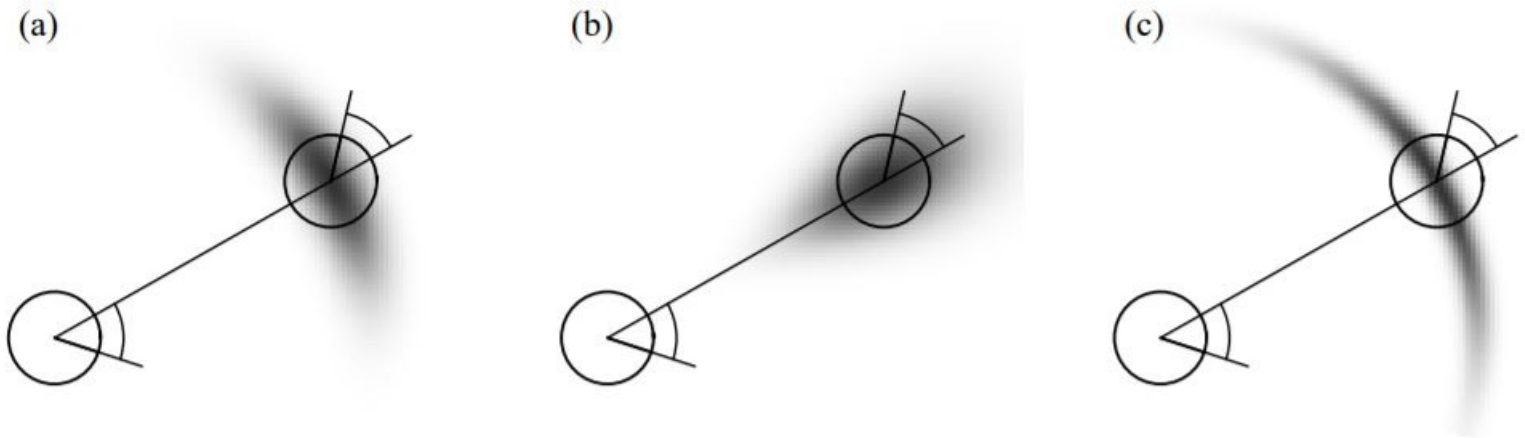
$p_1 = \textbf{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 \hat{\delta}_{\text{rot1}} + \alpha_2 \hat{\delta}_{\text{trans}})$

$p_2 = \textbf{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \hat{\delta}_{\text{trans}} + \alpha_4 (\hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}))$

$p_3 = \textbf{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 \hat{\delta}_{\text{rot2}} + \alpha_2 \hat{\delta}_{\text{trans}})$

return $p_1 \cdot p_2 \cdot p_3$

- Lines 2 to 4 recover relative motion parameters $\begin{pmatrix} \delta rot1 \\ \delta trans \\ \delta rot2 \end{pmatrix}$ from the odometry readings.
- The corresponding relative motion parameters $\begin{pmatrix} \hat{\delta} rot1 \\ \hat{\delta} trans \\ \hat{\delta} rot2 \end{pmatrix}$ for the given poses x_{t-1} and x_t are calculated in Lines 5 through 7 of this algorithm.
- Lines 8 to 10 compute the error probabilities for the individual motion parameters
- As earlier, the function $prob(a, b)$ implements an error distribution with zero mean and variance b.



The odometry motion model, for different noise parameter settings.

Note: All angular differences must lie in $[-\pi, \pi]$. Hence the outcome of $\delta rot2 - \hat{\delta} rot2$ has to be truncated correspondingly.

Similar to our earlier case, for particle filters, we would like to have a **sample algorithm** to generate random samples from $p(x_t|u_t,\ x_{t-1})$ for a fixed odometry reading u_t and pose x_{t-1} .

Algorithm sample_motion_model_odometry(u_t, x_{t-1})**:**

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

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

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

$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1\delta_{\text{rot1}} + \alpha_2\delta_{\text{trans}})$

$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\ \delta_{\text{trans}} + \alpha_4(\delta_{\text{rot1}} + \delta_{\text{rot2}}))$

$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1\delta_{\text{rot2}} + \alpha_2\delta_{\text{trans}})$

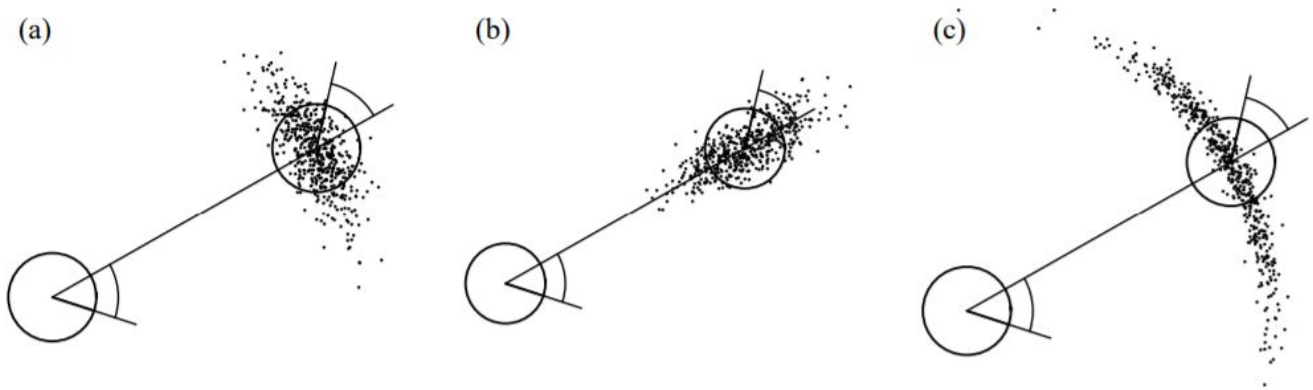
$x' = x + \hat{\delta}_{\text{trans}}\ \cos(\theta + \hat{\delta}_{\text{rot1}})$

$y' = y + \hat{\delta}_{\text{trans}}\ \sin(\theta + \hat{\delta}_{\text{rot1}})$

$\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

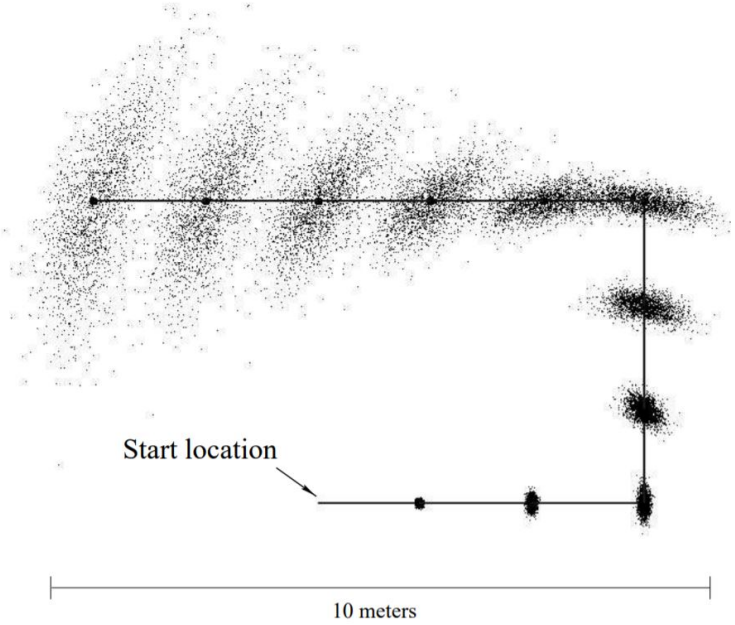
return $x_t = (x', y', \theta')^T$

As before, the sampling algorithm is somewhat easier to implement in practice.



The figure above shows sampling from the odometry motion model, using the same parameters as before, for 500 samples.

The motion model "in action"



This data has been generated using the motion update equations of the particle filter algorithm, assuming the robot's odometry follows the path indicated by the solid line.

The figure illustrates how the uncertainty grows as the robot moves - the samples are spread across an increasingly larger space.

Motion and Maps

- So far, we have described robot motion in the absence of any knowledge about the nature of the environment.
- However, in many cases, we are also given a map m, which may contain information pertaining to the places that a robot can or can not navigate, before, during, and after executing a control u_t.
- Knowing m gives us further information about the robot pose x_t. For example, with occupancy maps (that distinguish free (traversable) from occupied terrain) the robot's pose must always be in the free space.

To take the environment map into account, we will consider a map-based motion model:

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

If m carries information relevant to pose estimation, then:

$p(x_t|u_t,\ x_{t-1}) \neq (x_t|u_t,\ x_{t-1},\ m)$

- Incorporating map-information to compute this motion model in closed form is difficult. Instead, we consider an approximation for the map-based motion model, which works well if the distance between x_{t-1} and x_t is small.
- The approximation factorizes the map-based motion model into two components:

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

The second term, $p(x_t|m)$ expresses the "consistency" of pose x_t with the map m. In the case of occupancy maps, $p(x_t|m) = 0$ if and only if the robot "collides" with an occupied grid cell in the map. Otherwise it assumes a constant value.

As η can be computed by normalization, this approximation of a map-based motion model can be computed efficiently without any significant overhead compared to a map-free motion model.

Algorithm motion_model_with_map(x_t, u_t, x_{t-1}, m)**:**

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

Algorithm sample_motion_model_with_map(u_t, x_{t-1}, m)**:**

do

$x_t =$ **sample_motion_model**(u_t, x_{t-1})

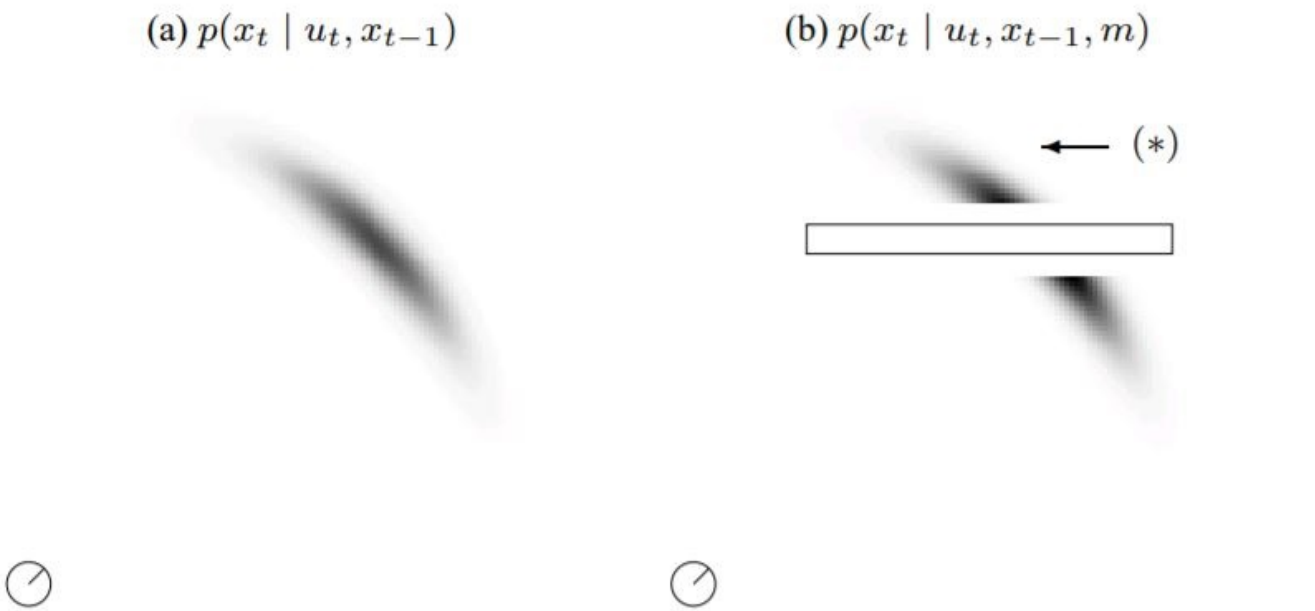
$\pi = p(x_t \mid m)$

until $\pi > 0$

return $\langle x_t, \pi \rangle$

This algorithms bootstraps previous motion models to models that take into account that robots cannot be placed in occupied space in the map *m*.

Motion Models



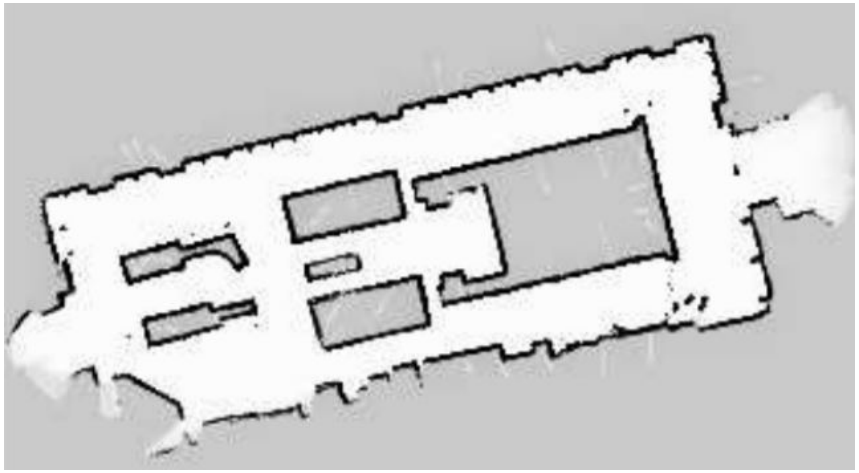
Velocity motion model (a) without a map and (b) conditioned on a map m.

The map m possesses a long rectangular obstacle, as indicated in the figure above. The probability $p(x_t|m)$ is zero at all poses x_t where the robot would intersect the obstacle.

- The figure also illustrates a problem with our approximation. The region marked $(*)$ possesses non-zero likelihood, since both $p(x_t|u_t, x_{t-1})$ and $p(x_t|m)$ are nonzero in this region. However, for the robot to be in this particular area it must have gone through the wall, which is impossible in the real world.
- In particular, we need to pay attention to the update frequency. In practice, such errors only occur for relatively large motions u_t, and it can be neglected for higher update frequencies.
- A Bayes filter that is updated frequently might yield fundamentally different results from one that is updated only occasionally.

Maps

- Mapping is one of the core competencies of truly autonomous robots.



- Formally, a map m is a list of objects in the environment along with their properties:

$$m = \{m_1, ..., m_N\}$$

- Maps are usually indexed in one of two ways:
  - **Feature-Based**: In feature-based maps, n is a feature index. The value of m_n contains, next to the properties of a feature, the Cartesian location of the feature.

    Feature-based maps only specify the shape of the environment at the specific locations, namely the locations of the objects contained in the map.

    Feature representation makes it easier to adjust the position of an object, e.g., as a result of additional sensing

- **Location-Based**: In location-based maps, the index n corresponds to a specific location.

  Location-based maps are volumetric, in that they offer a label for any location in the world. Volumetric maps contain information not only about objects in the environment, but also about the absence of objects (e.g., free-space).

  Occupancy maps are location-based: They assign to each x-y coordinate a binary occupancy value which specifies whether or not a location is occupied with an object.

The Occupancy Grid Mapping Algorithm

- The goal is to calculate the posterior over maps given the data:

  $p(m|z_{1:t}, x_{1:t})$

- The controls $u_{1:t}$ play no role in occupancy grid maps, since the path is already known.
- Let m_i denote the grid cell with index i. An occupancy grid map partitions the space into finitely many grid cells:

  $m = \sum_i \boldsymbol{m_i}$

- Each m_i has attached to it a binary occupancy value, which specifies whether a cell is occupied or free. We will write "1" for occupied and "0" for free.
- The notation $p(\boldsymbol{m_i} = 1)$ or $p(\boldsymbol{m_i})$ will refer to a probability that a grid cell is occupied.

Using the ideas on the previous slide, the standard occupancy grid approach breaks down the problem of estimating the map into a collection of separate problems, namely that of estimating:

$p(\boldsymbol{m_i}|z_{1:t}, x_{1:t})$ $\forall$ grid cells $\boldsymbol{m_i}$

Thanks to our factorization, the estimation of the occupancy probability for each grid cell is now a binary estimation problem with static state.
A filter for this problem has already been discussed - **The Binary Bayes filter**

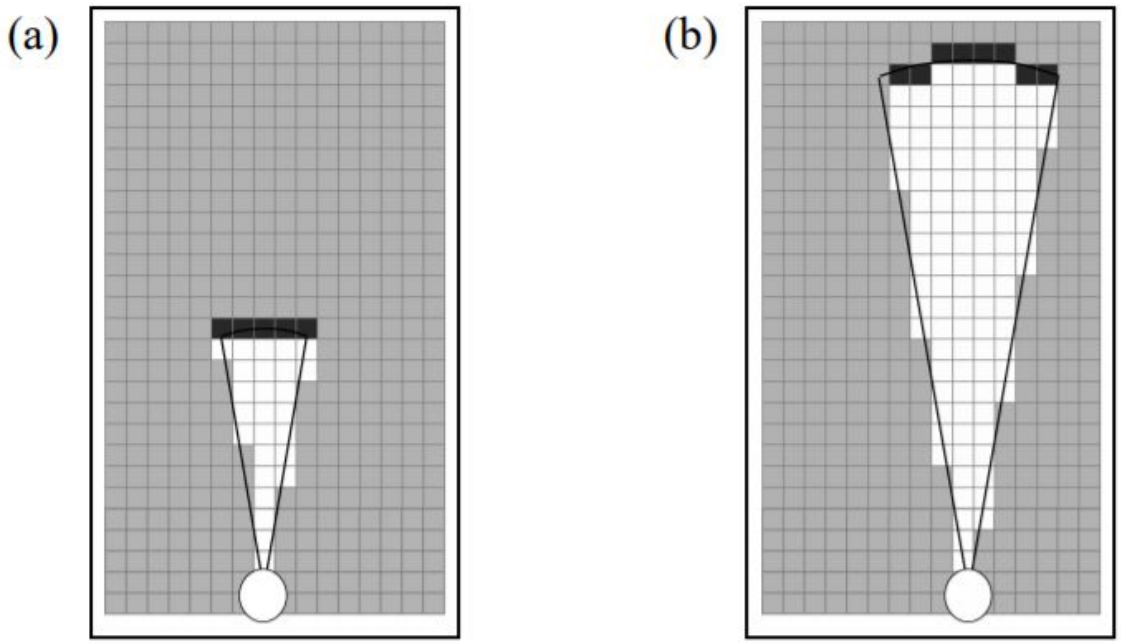
As in the original Binary-Bayes filter, our occupancy grid mapping algorithm uses the log-odds representation of occupancy:

$l_{t,i} = log \frac{p(\boldsymbol{m_i}|z_{1:t},x_{1:t})}{1-p(\boldsymbol{m_i}|z_{1:t},x_{1:t})}$

1: **Algorithm occupancy_grid_mapping**($\{l_{t-1,i}\}, x_t, z_t$):
2: for all cells $\mathbf{m}_i$ do
3: if $\mathbf{m}_i$ in perceptual field of z_t then
4: $l_{t,i} = l_{t-1,i} + \textbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$
5: else
6: $l_{t,i} = l_{t-1,i}$
7: endif
8: endfor
9: return $\{l_{t,i}\}$

- The algorithm **occupancy-grid-mapping** loops through all grid cells i, and updates those that fall into the sensor cone of the measurement z_t. For those where it does, it updates the occupancy value by virtue of the function **inverse-sensor-model**.

- The function **inverse-sensor-model** implements the inverse measurement model $p(\boldsymbol{m_i}|z_t,\ x_t)$ in its log-odds form.



A somewhat simplistic illustration of such a function for range finders for two different measurement ranges. The darkness of each grid cell corresponds to the likelihood of occupancy.

The algorithm corresponding to the previous slide's illustration is shown below:

Algorithm inverse_range_sensor_model(i, x_t, z_t)**:**

Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$

$r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$

$\phi = \text{atan2}(y_i - y, x_i - x) - \theta$

$k = \text{argmin}_j \, |\phi - \theta_{j,\text{sens}}|$

if $r > \min(z_{\max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{k,\text{sens}}| > \beta/2$ then

  return l_0

if $z_t^k < z_{\max}$ and $|r - z_{\max}| < \alpha/2$

  return l_{occ}

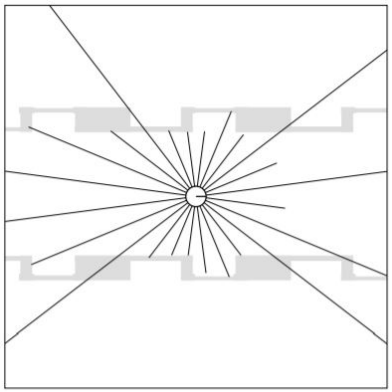
if $r \leq z_t^k$

  return l_{free}

endif

Measurement Models

We mainly address range-sensors in the following slides. However, the underlying principles can be applied to any kind of sensor, such as a camera or a bar-code operated landmark detector.



As an example of a typical range-scan, consider a mobile robot in a corridor, equipped with ultrasound sensors. Most of these measurements correspond to the distance of the nearest object in the measurement cone. However, some measurements, fail (sensor noise).

Many sensors generate more than one numerical measurement value when queried. For example, range finders usually generate entire scans of ranges. We will denote these measurement values as:

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

We use z_t^k to refer to an individual measurement(e.g., one range value).

Further, we will **assume independence** between individual measurement likelihoods to approximate $p(z_t|x_t, m)$:

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

- Our model incorporates four types of measurement errors, all of which are essential to making this model work:
  - small measurement noise
  - errors due to unexpected objects
  - errors due to failures to detect objects
  - random unexplained noise
- The model $p(z_t|x_t,\ m)$ is a mixture of four densities, each of which corresponds to a particular type of error.

Notation: Use z_t^{k*} to denote the "true" range of the object measured by z_t^k.

Small Measurement Noise

Even if the sensor correctly measures the range to the nearest object, the value it returns is subject to error. This error arises from:

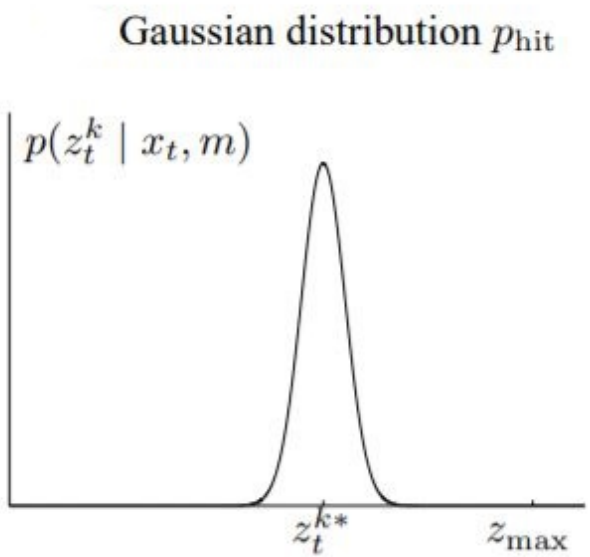
- limited resolution of range sensors
- atmospheric effect on the measurement signal
- etc.

This noise is usually modeled by a narrow Gaussian with mean z_t^{k*} and standard deviation σ_{hit}.

We will denote the Gaussian by p_{hit}.

In practice, the values measured by the range sensor are limited to the interval $[0;\ z_{\text{max}}]$, where z_{max} denotes the maximum sensor range. Thus, the measurement probability is given by:

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



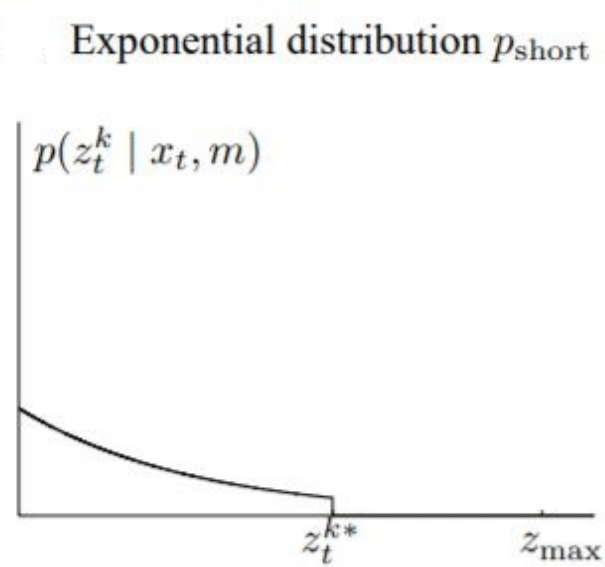
Unexpected Objects

Environments of mobile robots are dynamic, whereas maps m are static. As a result, objects not contained in the map can cause range finders to produce surprisingly short ranges—at least when compared to the map.

- One way to deal with such objects is to treat them as part of the state vector and estimate their location.
- Another, much simpler approach, is to treat them as sensor noise.
- **Notice:** Treated as sensor noise, unmodeled objects have the property that they cause ranges to be shorter than z_t^{k*}, not longer - More generally, the likelihood of sensing unexpected objects decreases with range.

Mathematically, the probability of range measurements in such situations is described by an exponential distribution. The parameter of this distribution, λ_{short}, is an intrinsic parameter of the measurement model.

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



Failure to Detect Objects

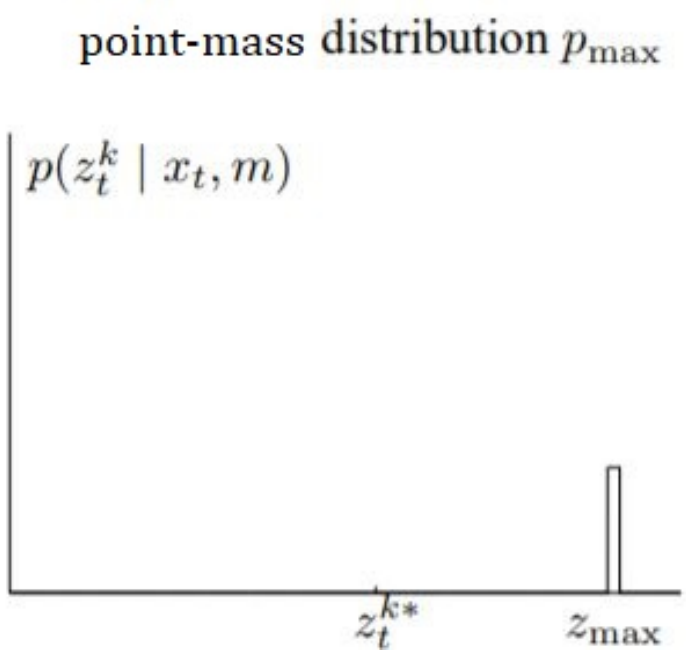
Sometimes, obstacles are missed altogether.

- For example, failures also occur with laser range finders when sensing black, light-absorbing objects, or when measuring objects in bright light.
- A typical result of sensor failures are max-range measurements: the sensor returns its maximum allowable value zmax.

Since such events are quite frequent, it is necessary to explicitly model max-range measurements in the measurement model.

We will model this case with a point-mass distribution centered at $z_{\max}$.

$$p_{\max}(z_t^k | x_t,\ m) = \mathbb{I}(z = z_{\max}) = \begin{cases} 1, & \text{if } z = z_{\max} \\ 0, & \text{otherwise} \end{cases}$$



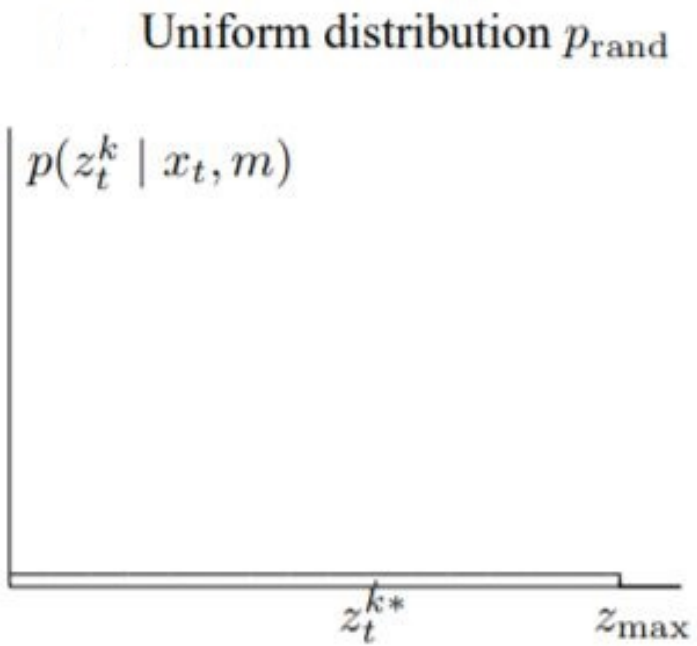
Random Measurements

Finally, range finders occasionally produce entirely unexplained measurements.

- sonars often generate phantom readings when they bounce off walls
- sensor readings may be subject to cross-talk with other different sensors.
- etc.

To keep things simple, such measurements will be modeled using a uniform distribution spread over the entire sensor measurement range $[0;\ z_{\max}]$:

$$p_{\text{rand}}(z_t^k | x_t,\ m) = \begin{cases} \frac{1}{z_{\max}}, & \text{if } 0 \leq z_t^k \leq z_{\max} \\ 0, & \text{otherwise} \end{cases}$$



These four different distributions are now mixed by a weighted average, defined by the parameters z_{hit}, z_{short}, z_{max}, and z_{rand} with:

$$z_{\text{hit}} + z_{\text{short}} + z_{\text{max}} + z_{\text{rand}} = 1$$

Algorithm beam_range_finder_model(z_t, x_t, m)**:**

$q = 1$

for $k = 1$ *to* K *do*

compute z_t^{k*} *for the measurement* z_t^k *using ray casting*

$p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$

$+ z_{\text{max}} \cdot p_{\text{max}}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$

$q = q \cdot p$

return q

After iterating through all sensor measurements z_t^k in z_t, the algorithm returns the desired probability $p(z_t|x_t, m)$.