Monte Carlo Localization (MCL)

Particle Filter

↳ non parametric filter
↳ sufficient number of samples for better approximation

Samples: $\mathcal{X} = \{\langle x^{[m]}, w^{[m]} \rangle\}_{m=1,\dots,M}$ → M samples ; $\sum_{m=1}^{M} w^{[m]} = 1$

($x^{[m]}$: state (pose); $w^{[m]}$: weight } of each sample)

↑ w ⟷ ↑ probability

$$p(x) = \sum_{m=1}^{M} w^{[m]} \delta_{x^{[m]}}(x)$$

finite number of iteration

The initial set of samples cannot be obtained from an arbitrary function in closed-form, generally.
Specific function, like the gaussian distribution, must be chosen.

↳ In other words, we want to generate samples from a target distribution, $f(x)$, but we can only generate those samples from a proposal distribution, $\pi(x)$, user-defined. → $f(x)>0 \rightarrow \pi(x)>0$

↳ We can do this if we compensate for the mistakes: $w(x) = \frac{f(x)}{\pi(x)}$

This is why it is a non-parametric filter.

The particle filter works with non-linear functions.

Algorithm:

- **Prediction** — Draw samples from $\pi(x)$ (Action model)
- **Correction** — Compensate for mistakes (importance sampling principle) (Observation model)
- **Resampling** — Sample current sample set to a new set, according to the weight of the samples ($\uparrow w \Rightarrow \uparrow$ chance of sample being drawn); we also turn them into uniformly weighted samples
  - ↳ This corresponds to replacing the weights by frequencies
  - ↳ By doing this, we ignore particles with low weight, which don't contribute much to a good approximation
  - ↳ This allows for a more efficient memory (which is a finite resource) management

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

$\overline{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ ($\overline{\mathcal{X}}_t$: bel; temporary sample set; $\mathcal{X}_t$: bel; new sample set)

for $m = 1$ to M do

prediction ← sample $x_t^{[m]} \sim \pi(x_t) \sim p(x_t \mid u_t, x_{t-1})$ ($\pi(x_t)$ is related to previous belief and motion command)

correction ← $w_t^{[m]} \sim \frac{p(x_t^{[m]})}{\pi(x_t^{[m]})} \sim p(z_t \mid x_t)$ (in accordance to the chosen proposal distribution) → pp. 103-104 (124, 125)

Proposal pode ser multi-modal?

$\overline{\mathcal{X}}_t = \overline{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ → re-drawn and re-weighted samples

endfor

resample:
for $m = 1$ to M do
draw $i \in 1,\dots,M$ with probability $\propto w_t^{[m]}$
add $x_t^{[i]}$ to $\mathcal{X}_t$ with $w_t^{[i]} = \frac{1}{M}$ → uniform weight
endfor

return $\mathcal{X}_t$

Note: if we could draw directly from target, we wouldn't need the correction and resampling steps.
(all samples would have the same weight)

Monte Carlo Localization

↳ estimating pose with particle filter

- every particle is a pose hypothesis
- $x_t^{[m]} \sim \underbrace{p(x_t \mid x_{t-1}, u_t)}_{\text{↳ user defined}}$ → draws sample from motion model distribution → this increases uncertainty (i.e., spreads out particles); not all particles move the same
  ↳ move through the environment without observing it
- $w_t^{[m]} = \frac{\text{target}}{\text{proposal}} \propto p(z_t \mid x_t, m)$ ← maps
  ↳ we will normalize weights to 1, in the end

Changes in the particle filter algorithm:

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

$$w_t^{[m]} \sim \frac{p(x_t^{[m]})}{\pi(x_t^{[m]})} \sim p(z_t \mid x_t^{[m]})$$

Resampling:

- Roulette Wheel → binary search; $O(M \log M)$
- Low Variance Resampling → modified roulette wheel with M equispaced arrows; $O(M)$
  ↳ look into the array (no need for binary search)

Problem: if all particles have the same weight after the update step, there is no reason to choose ones over others, but it will still do it; low variance resampling will just return the same sample set over and over again.

↳ Minute 51:00

↳ There are videos that provide useful motion and observation models

Occupancy Grid Mapping

↳ stores information about which parts of the environment are occupied and which ones are free
↳ black — occupied
white — free

Mapping Task

binary map → in each cell: 1 — occupied, 0 — free

$$m^* = \operatorname{argmax}_m p(m \mid u_{1:t}, z_{1:t})$$

Given robot's pose instead:

$$m^* = \operatorname{argmax}_m p(m \mid x_{1:t}, z_{1:t})$$

→ mapping with known poses → ML? MAP?

Grid Maps

(occupancy grid)

↳ space uniformly discretized in cells
↳ for each cell: is it or occupied or not?
↳ assume the cells are either free or occupied
↳ $p(m) = p(m_1, \dots, m_M)$

Disadvantages: memory consumption (more relevant in 3D); in a sparse environment, a feature-based map could be better
Advantages: doesn't rely on specific feature detector

Assumptions:

- cells are either completely free or completely occupied
  ↳ each cell can be represented by a binary random variable

Note: for the prior, we can say $p(m_i) = 0{,}5$ → maximum uncertainty, maximum entropy

$$p(m_i) = p(M_i = \text{occupied})$$

- the world is static (while being observed)
  ↳ i.e., if a cell is occupied (free), it will be occupied (free) forever

For the implementation, should we make it so that if $p(m_i \mid x, z) = 0{,}5 \Rightarrow$ occupied?
↳ we don't want autonomous vehicles to bump into objects

map (contains various random variables)

- $p(m \mid x_{1:t}, z_{1:t}) = \prod_{i=1}^{M} p(m_i \mid x_{1:t}, z_{1:t})$ → the cells are conditionally independent of each other
  random variable
  can be solved by binary Bayes filter → optimized for binary random variables

↳ by using this approximation, the problem becomes computationally treatable
↳ mention BETTER approximation!

Are we going to use grey cells ???

Static State Binary Bayes Filter

↳ belief is only a function of the robot's motion : $bel(x) = bel(x|z_{1:t}, u_{1:t}) = bel(x|z_{1:t})$
↳ also, no time associated to x

Inverse sensor model depends on the sensor.

For each cell,

$$p(m_i|z_{1:t}, x_{1:t}) \overset{\text{Bayes rule}}{=} \frac{p(z_t|m_i, z_{1:t-1}, x_{1:t})\, p(m_i|z_{1:t-1}, x_{1:t})}{p(z_t|z_{1:t-1}, x_{1:t})} =$$

$$\overset{\text{Markov}}{=} \frac{p(z_t|m_i, x_t)\, p(m_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} =$$

unless we are on top of cell i, our current pose does not influence the state of a cell

$p(z_t|m_i, x_t) = \frac{p(m_i|z_t, x_t)\, p(z_t|x_t)}{p(m_i|x_t)}$

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

no map; no map; no measurements

unless robot is standing on the cell: $p(m_i|x_t) = p(m_i)$

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

$p(m_i)$ ↳ prior

Since M_i are binary random variables, we can do the same for $\neg M_i = \neg m_i$:

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

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

$$= \underbrace{\frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)}}_{z_t} \cdot \underbrace{\frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})}}_{\text{recursive term}} \cdot \underbrace{\frac{1 - p(m_i)}{p(m_i)}}_{\text{prior}}$$

Let's define $odds(x) = \frac{p(x)}{1-p(x)} \Leftrightarrow p(x) = \frac{1}{1 + \frac{1}{odds(x)}}$.

Since $odds(m_i|z_{1:t}, x_{1:t}) = \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} \cdot \frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})} \cdot \frac{1 - p(m_i)}{p(m_i)}$,

$$p(m_i|z_{1:t}, x_{1:t}) = \left[1 + \frac{1 - p(m_i|z_t, x_t)}{p(m_i|z_t, x_t)} \cdot \frac{1 - p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i|z_{1:t-1}, x_{1:t-1})} \cdot \frac{p(m_i)}{1 - p(m_i)}\right]^{-1}$$

We can make it even more efficient.

$$l(x) = \log\left[\frac{p(x)}{1-p(x)}\right] \Leftrightarrow \exp[l(x)] = \frac{p(x)}{1-p(x)} \Leftrightarrow p(x) = 1 - \frac{1}{1 + \exp[l(x)]}$$

Thus,

$$l(m_i|z_{1:t}, x_{1:t}) = \underbrace{l(m_i|z_t, x_t)}_{\text{inverse sensor model}} + \underbrace{l(m_i|z_{1:t-1}, x_{1:t-1})}_{\text{recursive term}} - \underbrace{l(m_i)}_{\text{prior}} \Leftrightarrow$$

$$\Leftrightarrow l_{t,i} = \text{inverse\_sensor\_model}(m_i, z_t, x_t) + l_{t-1,i} - l_0$$

$$p(m_i|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp[l(m_i|z_{1:t}, x_{1:t})]}$$

A MELHORAR:

- grid não fixa
- inverse sensor model → → forward sensor model

occupancy_grid_mapping($\{l_{t-1,i}\}, x_t, z_t$):

for all cells m_i do

if m_i in perceptual field of z_t then

$l_{t,i} = inverse\_sensor\_model(m_i, z_t, x_t) + l_{t-1,i} - l_0$

else

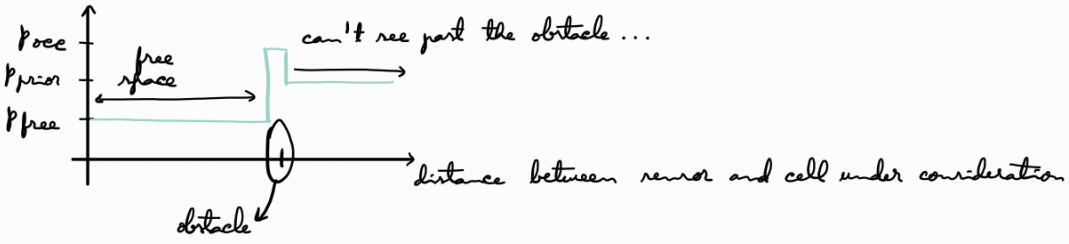
$l_{t,i} = l_{t-1,i}$

endif

endfor

return $\{l_{t,i}\}$

Inverse Sensor Model for laser:



We can return maximum likelihood map by rounding probability of each cell to 0 or 1.

We need to determine which cells are affected by laser beam.

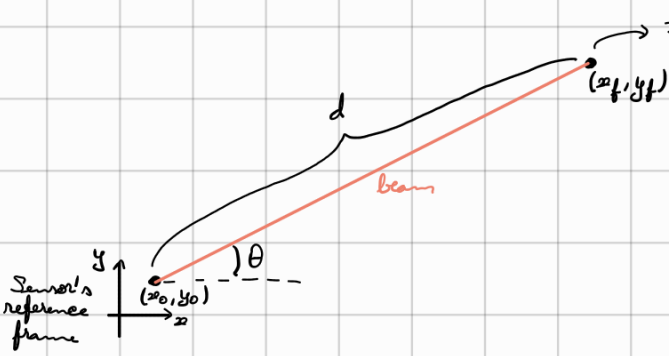
↳ Bresenham's line algorithm

↳ primeiro, vamos ignorar $p = 0,5$, para ver o que dá:

$p > 0,5 \Rightarrow p = 1$

$p < 0,5 \Rightarrow p = 0$

Determine occupied and free cells:



Because of AMCL, we know the robot's current location. Because of /tf, we then know sensor's current location.

Since $c = \frac{d}{TOF/2}$, $d = c\,\frac{TOF}{2}$. → time of flight (measured)

We also know θ (robot's pose → /tf).

x_f = convert_to_grid [round $(d \cdot \cos\theta)$]
y_f = convert_to_grid [round $(d \cdot \sin\theta)$]

↳ e.g.: (0,0) (1,0) — 50cm

$\Delta x = x_f - x_0$
$\Delta y = y_f - y_0$, it's likely that $(x_0, y_0) = (0,0)$