LOCALIZZAZIONE MONTE CARLO

parte 2

Probabilistics Robotics, cap 8

Indice

- Grid Localization
- Localizzazione Monte Carlo (MCL)
- Grid SLAM
- AMCL
- ROS (Robotic Operating System)

- Fast SLAM (altre slides)
  - EKF
  - GRID

Localizzazione

La localizzazione rappresenta il processo di determinazione della corrispondenza tra il sistema di coordinate mondo ed il sistema di coordinate robot

Diverse categorie di localizzazione, in ordine di difficoltà

- **Position Tracking**: la posizione iniziale del robot è nota, la localizzazione si ottiene grazie al trattamento dell'incertezza di moto.
- **Global Localization**: non si hanno informazioni sulla iniziale posizione del robot.
- **Kidnapped Robot Problem**: il robot può essere «spostato» in una diversa zona dello spazio (e.g. piccolo aspirapolvere sollevato e messo in altra stanza)

Approfondimenti: localizzazione in ambiente dinamico (vedi Prob. Robotics, cap. 8).

Localizzazione 2 - Metodi

- Parametrici es. EKF visti lezioni precedenti
- **Non parametrici es. Grid, MCL**

	EKF	MHT	Coarse (topo-logical) grid	fine (metric) grid	MCL
Measurements	landmarks	landmarks	landmarks	raw measurements	raw measurements
Measurement noise	Gaussian	Gaussian	any	any	any
Posterior	Gaussian	mixture of Gaussians	histogram	histogram	particles
Effi ciency (memory)	++	++	+	−	+
Effi ciency (time)	++	+	+	−	+
Ease of implementation	+	−	+	−	++
Resolution	++	++	−	+	+
Robustness	−	+	+	++	++
Global localization	no	no	yes	yes	yes

In tabella (Prob.Robotics cap.8 ultime pagine) una rassegna comparativa delle diverse tecniche di localizzazione. (info: MHT=Multiple Hypotesis Tracking, estensione EKF, ulteriori dettagli per chi vuole sul libro)

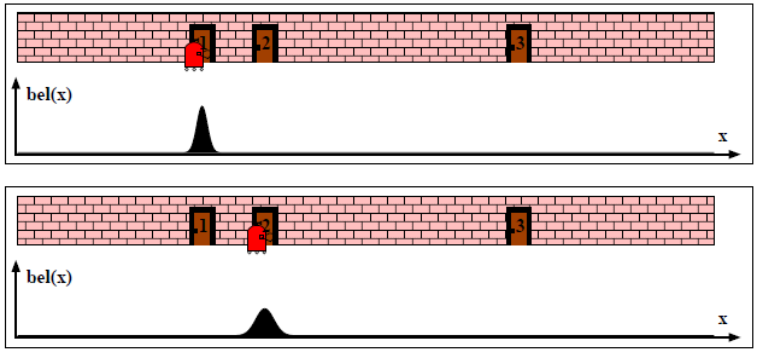
Localizzazione 3 – Perché MCL ?

- Rappresentare distribuzioni multimodali
  - Sono non parametrici e quindi non sono limitati a rappresentare distribuzioni unimodali come nel caso di EKF
- Risolvere problemi di *Localizzazione Globale*
  - Possibilità di localizzarsi globalmente nella mappa e di sopravvivere ai *kidnapped-robot-problems*
- Semplicità di implementazione
  - Si possono utilizzare *direttamente* le misure sensoriali; non è necessario estrarre features
  - In MCL, l'implementazione standard di un PF richiede poche righe di codice

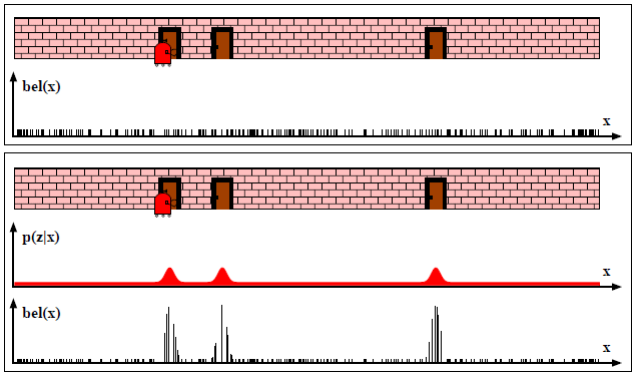
Unimodale vs Multimodale

- **PERCHÉ MULTIMODALE?**

EKF (unimodale)



MCL (multimodale)



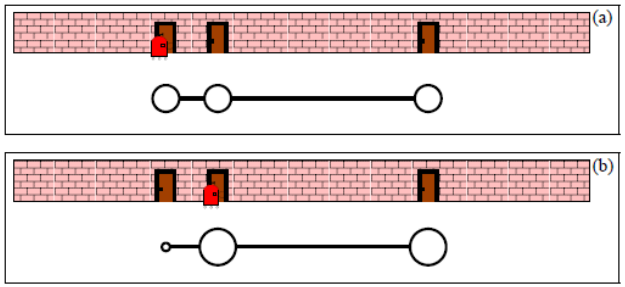
Localizzazione non parametrica con...

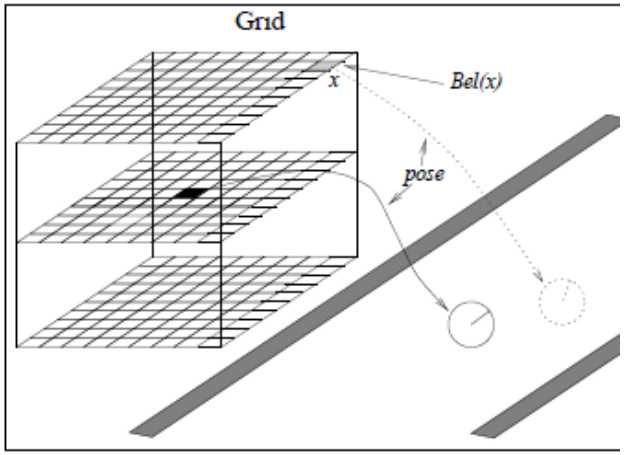
- **Grid Localization**

  Grid resolution: topological?

  metric?

  quanto deve essere granulare la griglia?





- **Monte Carlo Localization**

  Utilizzano filtri a particelle

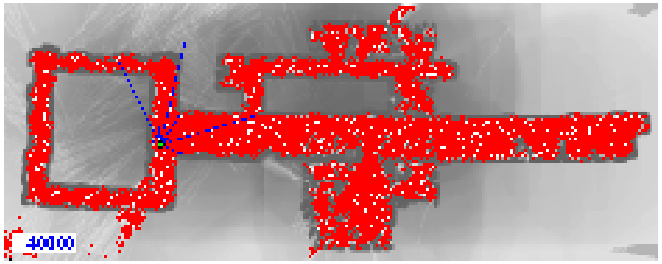
Grid Localization

- I metodi sotto il nome di «grid localization» utilizzano *histogram filters* per rappresentare la distribuzione a posteriori. Le problematiche associate a questo tipo di localizzazione sono legati alla densità della griglia utilizzata: al crescere del numero di celle della griglia crescono le richieste computazionali; viceversa diminuendo la densità della griglia invece potremmo perdere dettagli fondamentali per discriminare correttamente la posizione del robot.

- Keyword: discretizzazione dello spazio

Monte Carlo Localization

- Localizzazione effettuata con l'utilizzo di *Particle Filter*
- Incrementando il numero di particelle aumenta la precisione dell'aprossimazione.
- N# particelle = compromesso tra risorse disponibili e accuratezza localizzazione
- Implementazione «easy» !



website

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

$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$

for $m = 1$ to M do

$x_t^{[m]} =$ **sample_motion_model**$(u_t, x_{t-1}^{[m]})$

$w_t^{[m]} =$ **measurement_model**$(z_t, x_t^{[m]}, m)$

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

endfor

for $m = 1$ to M do

draw i with probability $\propto w_t^{[i]}$

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

endfor

return $\mathcal{X}_t$

Tecniche avanzate di MCL

- Come sopravvivere al kidnapped robot problem?
- Quante particelle servono? Come possiamo adattare il set di particelle all’effettiva necessità?

1. Global loc. failure or Kidnapped pr.

- MCL potrebbe cancellare le particelle «buone» e lasciare la zona di localizzazione corretta scoperta da particelle (fenomeno chiamato *particle deprivation*)

Soluzione

inserimento randomico di particelle al set MCL

- Quando effettuare l'inserimento random?
- Quante particelle inserire?
- Con che distribuzione inserire le particelle?

2. Global loc. failure or Kidnapped pr.

- 1) aggiungere *sempre* un certo numero di particelle

- 2) aggiungere particelle basandosi su una stima della qualità di localizzazione e.g. probabilità della misura sensoriale data la stima di posizione attuale: nei particle filter è facilmente ottenibile da:

$$\frac{1}{M}\sum_{M=1}^{M} w_t^{[m]}$$

ovvero il valore medio dell'importance weight

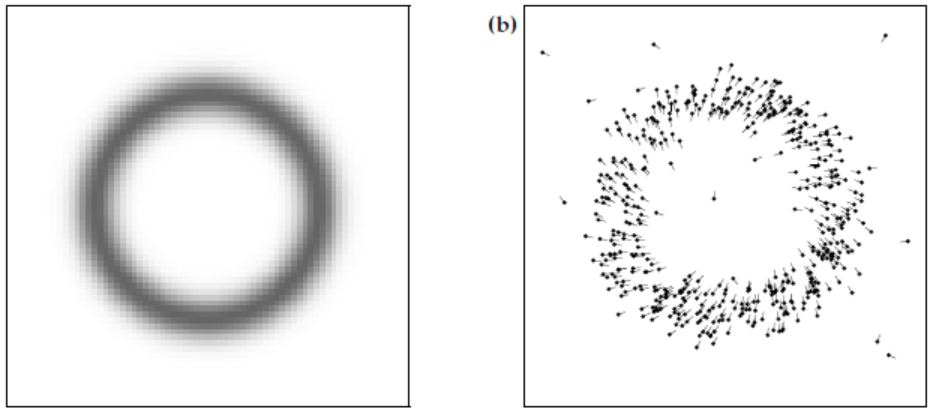
3. Global loc. failure or Kidnapped pr.

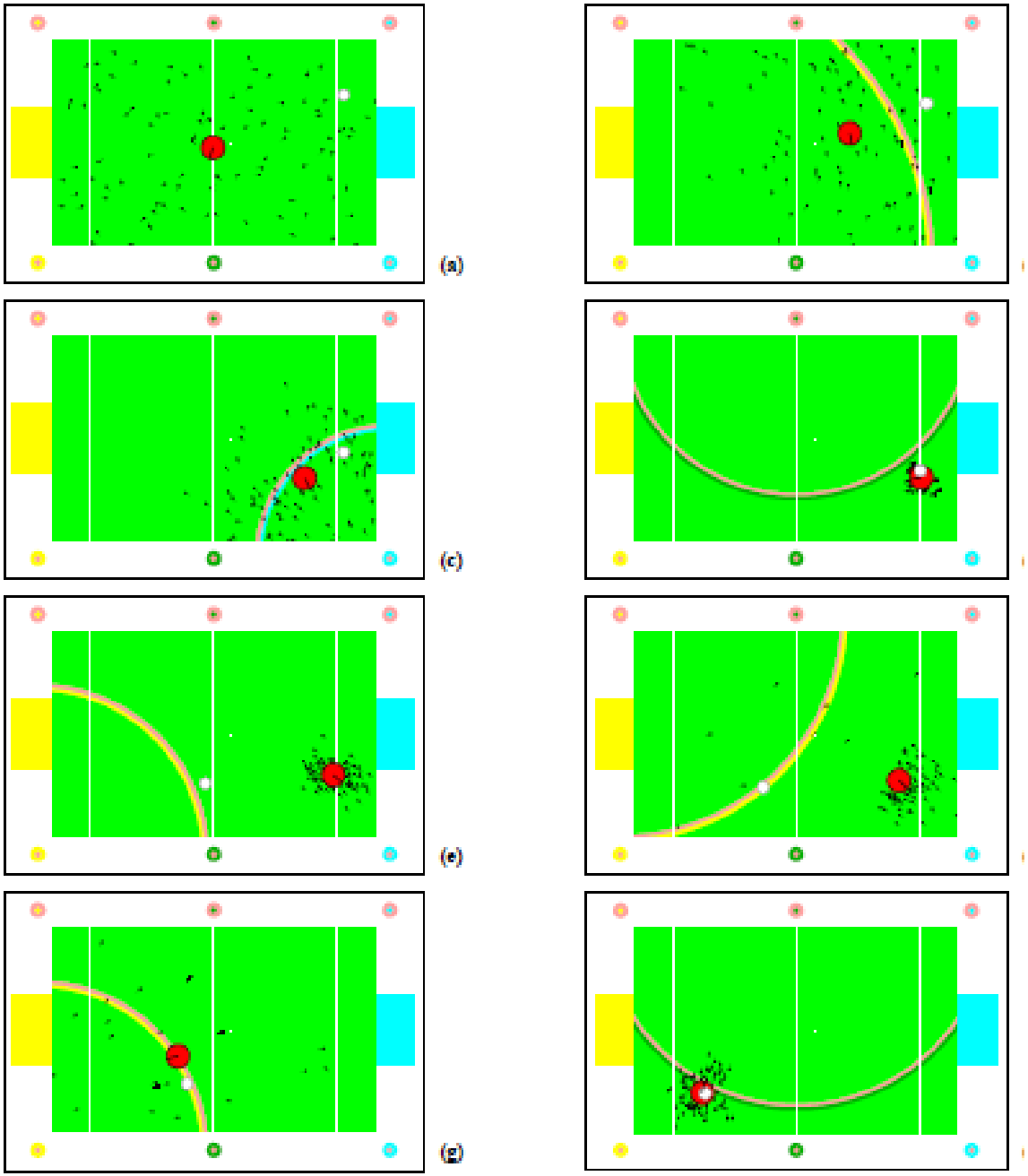
- Metodo elegante: tenere traccia di due medie basandosi sulla verosimiglianza delle misure e determinare quante particelle aggiungere sulla base di queste ultime.
  - Short term average [w_{fast}]
  - Long term average [w_{slow}]

Quale distribuzione usare al momento dell'introduzione di particelle?

- Più semplice: distribuzione uniforme (usata nell'algoritmo AMCL, Dieter Fox et al. disponibile su ROS)

- Altro tipo di distribuzione che tenga in considerazione la distribuzione di probabilità delle misure sensoriali





Localizzazione Monte Carlo con aggiunta di particelle random. Nelle figure è visualizzato il particle-set che rappresenta la stima di posizione del robot (le piccole linee/frecce). In BIANCO la vera posizione del robot. In ROSSO la **media** delle particelle. La sequenza di immagini mostra una localizzazione globale (fig. a-d), seguita da un kidnapping ed una successiva relocalizzazione (fig. e-h).

Dalle particelle alla posizione….

Nel caso un algoritmo esterno (o il semplice utente) abbia la necessità di conoscere **«una»** posizione, non un insieme di particelle (che rappresentano una distribuzione di probabilità), come ci si comporta?

Ovvero, *come estrarre **una** posizione da un insieme di particelle?*

- **Particella con probabilità maggiore**: estremamente sensibile a disturbi, basta che casualmente ed erroneamente una particella abbia peso/prob. alta e il raffinato sistema viene meno
- **Media di tutte le particelle (pesate)**: potrebbe essere più sensato fare una media di particelle, ma la media potrebbe essere ugualmente irrealistica (riuscite a capire perché?)
- **Clustering**: una soluzione più elegante è creare dei cluster di particelle, fare la media per cluster e quindi scegliere la media associata al cluster!

AMCL

- Algoritmo di localizzazione Monte Carlo con:
  - Adaptive KLD: adatta il numero di particelle utilizzate (AMCL [ROS] [Player {Dieter Fox}], KLD-Sampling [Dieter Fox])
  - Augmented: inserisce particelle random per evitare particle deprivation

Algorithm Augmented_MCL$(\mathcal{X}_{t-1}, u_t, z_t, m)$:
static w_{slow}, w_{fast}
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for $m = 1$ to M do
$x_t^{[m]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
$w_t^{[m]} = \textbf{measurement\_model}(z_t, x_t^{[m]}, m)$
$\mathcal{X}_t = \mathcal{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
$w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M} w_t^{[m]}$
endfor
$w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$
$w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$
for $m = 1$ to M do
with probability $\max(0.0,\ 1.0 - w_{\text{fast}}/w_{\text{slow}})$ do
add random pose to $\mathcal{X}_t$
else
draw $i \in \{1, \ldots, N\}$ with probability $\propto w_t^{[i]}$
add $x_t^{[i]}$ to $\mathcal{X}_t$
endwith
endfor
return $\mathcal{X}_t$

Algorithm MCL$(\mathcal{X}_{t-1}, u_t, z_t, m)$:
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for $m = 1$ to M do
$x_t^{[m]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
$w_t^{[m]} = \textbf{measurement\_model}(z_t, x_t^{[m]}, m)$
$\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
endfor
for $m = 1$ to M do
draw i with probability $\propto w_t^{[i]}$
add $x_t^{[i]}$ to $\mathcal{X}_t$
endfor
return $\mathcal{X}_t$

Algorithm KLD_Sampling_MCL$(\mathcal{X}_{t-1}, u_t, z_t, m, \varepsilon, \delta)$:
$\mathcal{X}_t = \emptyset$
$M = 0$, $M_\chi = \infty$, $k = 0$
for all b in H do
$b = empty$
endfor
do
draw i with probability $\propto w_{t-1}^{[i]}$
$x_t^{[M]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[i]})$
$w_t^{[M]} = \textbf{measurement\_model}(z_t, x_t^{[M]}, m)$
$\mathcal{X}_t = \mathcal{X}_t + \langle x_t^{[M]}, w_t^{[M]} \rangle$
if ($x_t^{[M]}$ *falls into empty bin b*) *then*
$k = k + 1$
$b =$ *non-empty*
if ($k > 1$) *then*
$M_\chi := \frac{k-1}{2\varepsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta}\right\}^3$
endif
$M = M + 1$
while ($M < M_\chi$)
return $\mathcal{X}_t$

In questi tre blocchi: in centro l'algoritmo base MCL. A sinistra la versione puramente *augmented* mentre a destra la versione *KLD.* AMCL unisce questi ultimi due metodi. Si vedano i link qui sopra e il libro probabilistic robotics cap 8

ROS

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

Il framework è scritto in C++ ed offre possibilità di creare nodi sia in C++ che Python. Offre una notevole mole di software pronto all'uso, driver per svariate tipologie di sensori, strutture dati, algoritmi, suite di visualizzazione, software di simulazione, and much more!