Universidad de Buenos Aires
Facultad De Ingeniería
Año 2021 - 1er Cuatrimestre

PRÁCTICA 4: Mapeo y SLAM con Filtro de Partículas (FASTSLAM). Robótica móvil- Un enfoque probabilistico(86.48).

1er. Cuatrimestre
Curso: MAS

INTEGRANTES:
Fuentes Acuña, Brian Alex - #101785
<bfuentes@fi.uba.ar>

Índice

1. Mapeo con poses conocidas

Una forma de mapeo es usando un mapa de grillas en donde las características que este presenta son:

- Discretizamos el mundo en celdas.
- La estructura de grilla es rígida.
- Cada celda puede estar ocupada o libre.
- Es un modelo **no-paramétrico**.
- Requiere considerable memoria de almacenamiento.
- No depende de detección de features.

Entonces teniendo esto en mente cada celda tendrá una probabilidad asignada '1' o '0', ocupada o desocupada respectivamente, por lo que cada celda será una una variable aleatoria binaria que modela la ocupación de la grilla (p = 0.5 si no se sabe nada de la grilla).
Además, se asume un entorno estático en donde las celdas son independientes unas de otras, por lo que la forma de representar el mapa será de la siguiente manera.

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t}) \tag{1}$$

En donde se debe resolver quien es la probabilidad para cada celda. Para ello se usa el filtro de bayes binario de estado estático.

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, z_{1:t}, x_{1:t})p(m_i|z_{1:t-1}, x_{1:t})}{p(z_t|z_{1:t-1}, x_{1:t})} \tag{2}$$

En donde se irá aplicando la suposición de Markov, regla de bayes y simplificaciones teniendo en cuenta el problema de mapeo con poses conocidas.
Llegando así a las nueva expresión de la probabilidad de ocupación condicional de cada celda del mapa de grillas.

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(m_i|z_t, x_t)p(z_t|x_t)p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i)p(z_t|z_{1:t-1}, x_{1:t})} \tag{3}$$

y el caso opuesto será:

$$p(\neg m_i|z_{1:t}, x_{1:t}) = \frac{p(\neg m_i|z_t, x_t)p(z_t|x_t)p(\neg m_i|z_{1:t-1}, x_{1:t-1})}{p(\neg m_i)p(z_t|z_{1:t-1}, x_{1:t})} \tag{4}$$

ya solo queda obtener una expresión más amigable entonces se hace el cociente de los dos casos y se aplica la función Log_Odds.

$$\begin{aligned}
\frac{p(m_i|z_{1:t}, x_{1:t})}{p(\neg m_i|z_{1:t}, x_{1:t})} &= \frac{p(m_i|z_t, x_t)p(m_i|z_{1:t-1}, x_{1:t-1})p(\neg m_i)}{p(\neg m_i|z_t, x_t)p(\neg m_i|z_{1:t-1}, x_{1:t-1})p(m_i)} \\
\frac{p(m_i|z_{1:t}, x_{1:t})}{1 - p(m_i|z_{1:t}, x_{1:t})} &= \frac{p(m_i|z_t, x_t)}{1 - p(m_i|z_t, x_t)} \frac{p(m_i|z_{1:t-1}, x_{1:t-1})}{1 - p(m_i|z_{1:t-1}, x_{1:t-1})} \frac{1 - p(m_i)}{p(m_i)} \\
l(m_i|z_{1:t}, x_{1:t}) &= l(m_i|z_t, x_t) + l(m_i|z_{1:t-1}, x_{1:t-1}) - l(m_i)
\end{aligned} \tag{5}$$

Entonces queda como suma de funciones logarítmicas a las que se llamara chances logarítmicas.
En donde el primer término será el modelo inverso del sensor dado que a través de las mediciones y pose actuales se obtiene una estimación del mapa.
El segundo termino será el termino recursivo dado que representa la belief anterior.
El ultimo termino representa entonces la probabilidad a priori que por lo general es conocidas o se asume Logg_Odds(p = 0.5)
Entonces se tiene un algoritmo recursivo que puede escribirse de la siguiente manera:

$$l_{t,i} = inv\_sensor\_model(m_i, x_t, z_t) + l_{t-1,i} - l_0 \tag{6}$$

en donde la implementacion del mismo para cada medicion z_t será:

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: $l_{t,i} = l_{t-1,i} + \mathbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$
5: else
6: $l_{t,i} = l_{t-1,i}$
7: endif
8: endfor
9: return $\{l_{t,i}\}$

Figura 1: Algoritmo: Mapeo con poses conocidas.

En Este caso el mapa de grillas es 1-D.
Entonces siguiendo el algoritmo se itera para cada medición z_t y siempre recibe las chances logarítmicas anteriores.
Luego dentro del algoritmo se itera para cada grilla del mapa y si se percibe la medición entonces se calcula la nueva chance logarítmica con las datos obtenidos. En caso contrario se lo deja tal como estaba.
En la implementación se hace al revés, primero se calcula si la medición es **no visible** $if(m_i > z_t+20)$ de esa manera si no es alcanzable el algoritmo termina rápidamente. en caso contrario calcula la nueva chance logarítmica y se actualizara recursivamente.
El modelo inverso está dado por sera $(z_t < m + 10)?Log\_Odds(p = 0{,}6) : Log\_Odds(p = 0{,}3)$. y los datos usados son:

Resolución de la grilla	$10cm$
Logitud del mapa (1-D)	$2m$
posición del robot	c_0
orientación del sensor	mirando hacia c_n (ver figura)
mediciones (en cm)	101, 82, 91, 112, 99, 151, 96, 85, 99, 105
prob. *a priori*	0.5

Figura 2: Datos del problema.

Los resultados son los siguientes:

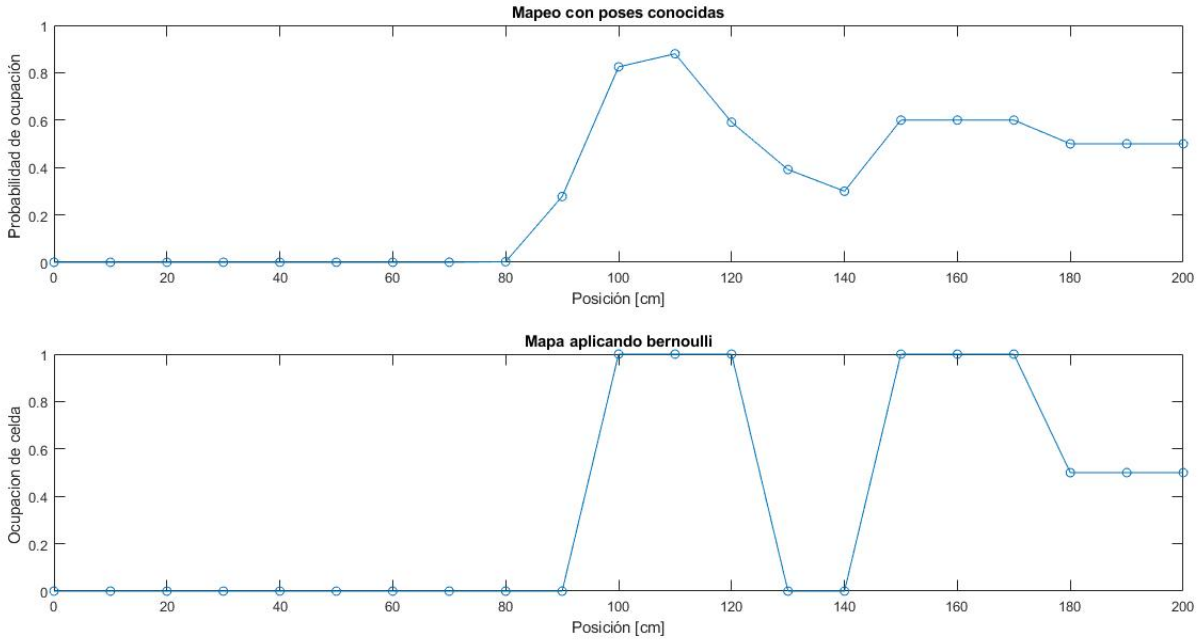


Figura 3: Probabilidad de ocupacion

En donde se puede ver cómo quedan las probabilidades para cada grilla y abajo se usa una Bernoulli con la probabilidad correspondiente a cada celda, esto hace que la celda este vacía o ocupada según la probabilidad y 0.5 en caso de no tener información.
Se puede ver que el robot está en c_0 y se actualizan las grillas del mapa para cada medición z_t pero esta mediciones son distintas e incluso pareciera que mide dos celdas ocupadas pero hay que tener en cuenta que las mediciones obtenidas pueden ser de un solo bloque ocupado pero obtener distintas mediciones por el ruido de la medición.

2. Implementación de algoritmo FASTSLAM

En este ejercicio se implementará el algoritmo FASTSLAM basado en landmarks. Se asume que los landmarks son identificables por lo que el problema de asociación de datos está resuelto.

Entonces se implementara un Fast-Slam con filtro de partículas y así resolver el problema de localización y mapeo con landmarks $< x, y, z, l_1, l_2, ..., l_N >$.
Pero esto es algo que resolvía SLAM entonces la justificación del Fast-Slam es porque el número de partículas necesarias para representar la belief crecería exponencialmente con la dimensión del espacio de estado.
Entonces habrá que buscar alguna forma de reducir la complejidad del sistema para ello se busca dependencias entre ciertas dimensiones del espacio de estado y así poder resolver el problema eficientemente.
Entonces se usa las dependencias de las poses del robot con el mapa, dado que se sabe construir un mapa conociendo las poses del robot. Se usa entonces la distribución posterior factorizada para separar la posterior del trayecto del robot y las posiciones de los landmarks.

$$\begin{aligned} p(x_{1:t}, l_{1:m}|z_{1:t}, u_{0:t-1}) &= p(x_{1:t}|z_{1:t}, u_{0:t-1})p(l_{1:m}|x_{1:t}, z_{1:t-1}) \\ [Post.deSLAM] &= [Post.Tray.Robot][POSEsLandmarks] \end{aligned} \tag{7}$$

La idea sería usar la Rao-Blackwellizacion, calcular $p(l_{1:m}|x_{1:t}, z_{1:t-1})$ de forma cerrada y entonces poder representar solo $p(x_{1:t}|z_{1:t}, u_{0:t-1})$ con muestras del filtro de partículas y después calcular $p(l_{1:m}|x_{1:t}, z_{1:t-1})$ para cada una de las muestras

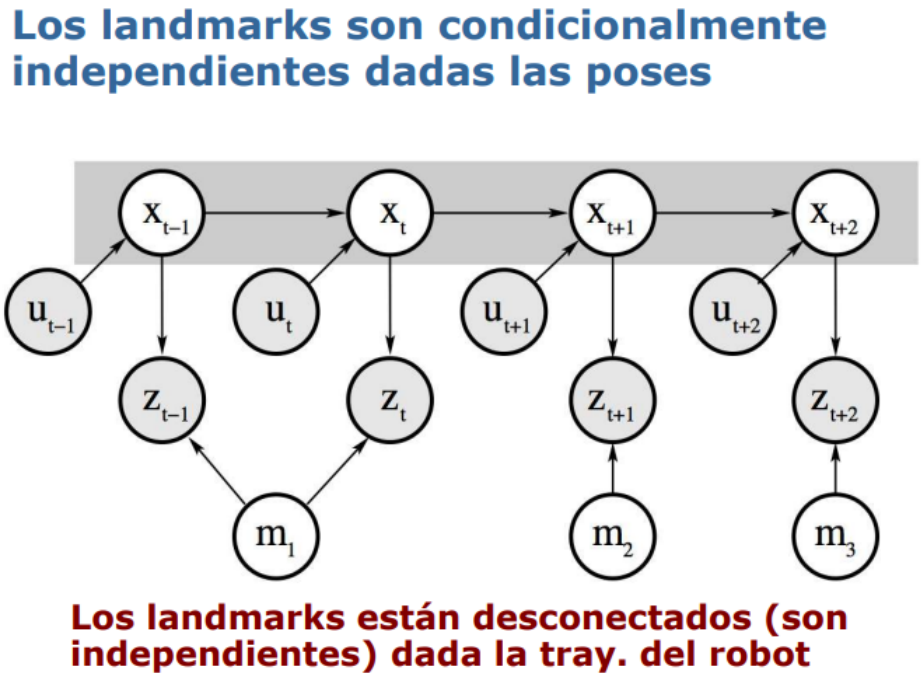


Figura 4: Modelo grafico

Viendo la la Figura 4 se puede observar que entonces si se suponen las poses conocidas entonces se puede asumir que los landmarks(mapa) son independientes entre sí dado que para que un landmark dependa del otro debería seguir el camino que marca el modelo gráfico, pero pasando por las poses las cuales ya son conocidas.
esto nos sirve para factorizar $p(l_{1:m}|x_{1:t}, z_{1:t-1})$ quedando:

$$p(x_{1:t}, l_{1:m}|z_{1:t}, u_{0:t-1}) = p(x_{1:t}|z_{1:t}, u_{0:t-1}) \prod_{i=1}^{M} p(l_i|x_{1:t}, z_{1:t}) \tag{8}$$

Entonces dado que los landmarks se pueden representar eficientemente se podra utilizar el filtro de partículas en la primer parte (poses) y agregarle la segunda parte(landmarks).
Esto se implementará de la siguiente manera.

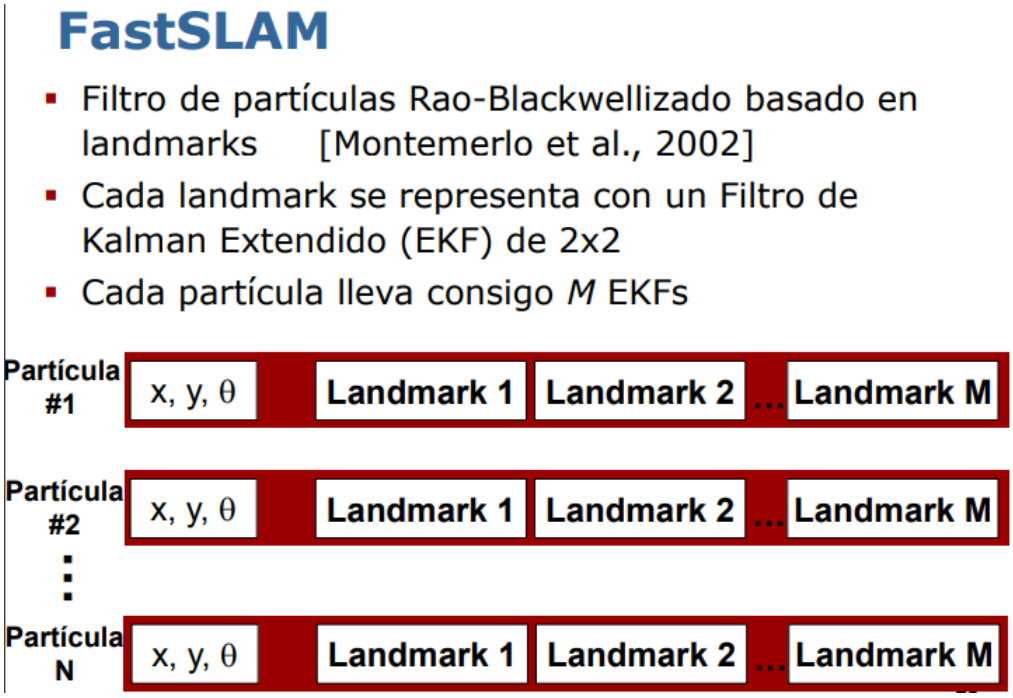


Figura 5: Fast Slam

Para ello se hace uso del siguiente algoritmo:

1: **FastSLAM1.0_known_correspondence**$(z_t, c_t, u_t, \mathcal{X}_{t-1})$**:**

2: *for* $k = 1$ *to* N *do* // *loop over all particles*

3: *Let* $\left\langle x_{t-1}^{[k]}, \left\langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]} \right\rangle, \ldots \right\rangle$ *be particle* k *in* $\mathcal{X}_{t-1}$

4: $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$ // *sample pose*

5: $j = c_t$ // *observed feature*

6: *if feature* j *never seen before*

7: $\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ // *initialize mean*

8: $H = h'(\mu_{j,t}^{[k]}, x_t^{[k]})$ // *calculate Jacobian*

9: $\Sigma_{j,t}^{[k]} = H^{-1}\, Q_t\, (H^{-1})^T$ // *initialize covariance*

10: $w^{[k]} = p_0$ // *default importance weight*

11: *else*

(a) inicializacion

11: *else*

12: $\hat{z}^{[k]} = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // *measurement prediction*

13: $H = h'(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ // *calculate Jacobian*

14: $Q = H\, \Sigma_{j,t-1}^{[k]}\, H^T + Q_t$ // *measurement covariance*

15: $K = \Sigma_{j,t-1}^{[k]}\, H^T\, Q^{-1}$ // *calculate Kalman gain*

16: $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z}^{[k]})$ // *update mean*

17: $\Sigma_{j,t}^{[k]} = (I - K\, H)\Sigma_{j,t-1}^{[k]}$ // *update covariance*

18: $w^{[k]} = |2\pi Q|^{-\frac{1}{2}}\ \exp\left\{-\frac{1}{2}(z_t - \hat{z}^{[k]})^T Q^{-1}\,(z_t - \hat{z}^{[k]})\right\}$ // *importance factor*

19: *endif*

20: *for all unobserved features* j' *do*

21: $\langle \mu_{j',t}^{[k]}, \Sigma_{j',t}^{[k]} \rangle = \langle \mu_{j',t-1}^{[k]}, \Sigma_{j',t-1}^{[k]} \rangle$ // *leave unchanged*

23: *endfor*

24: *endfor*

25: $\mathcal{X}_t = \text{resample}\left(\left\langle x_t^{[k]}, \left\langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \right\rangle, \ldots, w^{[k]} \right\rangle_{k=1,\ldots,N}\right)$

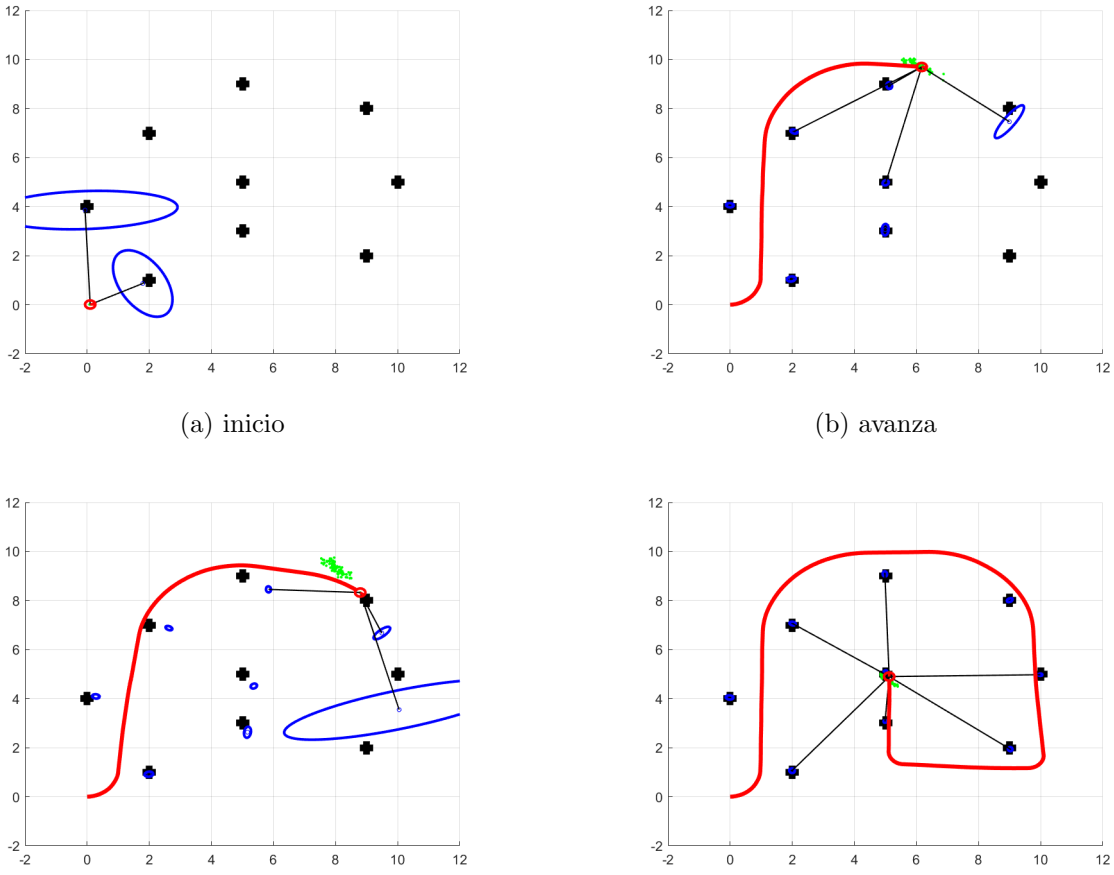
26: *return* $\mathcal{X}_t$

(b) Correccion

Figura 6: Algortimo de Fast-Slam con filtro de particulas

En donde se itera para todas las partículas. en la línea 3 se carga la pose con su EKF y en la 4 se calcula la odometria.
Luego de 6 a 10 si el landmark no fue visto anteriormente se inicializa el EKF para ese landmarks.
En la línea 12 a 17 actualiza el EKF para el landmark en caso de que ya se haya visto luego en la 18 se actualiza el peso teniendo en cuenta la covarianza de medición y el error entre el landmark observado con el esperado

y finalmente se hace un remuestreo de las partículas teniendo en cuenta el numero eficaz de partículas para remuestrear dado que en este caso se utilizarán pocas partículas, porque remuestrear frecuentemente impide la diversidad de partículas cuando son pocas, es decir el filtro de partículas carecería de sentido, cosa que no sucedía en el PF por que se tenían muchas partículas en comparación a este caso.



(a) inicio

(b) avanza

(c) Pocas mediciones

(d) fin

Figura 7: Algortimo de Fast-Slam con filtro de particulas

Entonces corriendo la simulación se obtiene la Figura 7 en donde se puede ver cómo evoluciona la trayectoria del robot a medida que se va localizando su pose y la ubicación de los landmarks.
También es importante recalcar que lo que se grafica es el mapa asociado a la partícula más probable por eso es que se ven distintas trayectorias.
En la simulación se pudo ver cómo funcionaba el Fast Slam y en realidad lo hacía bastante bien considerando que solo se usaban 100 partículas pero es importante notar con la figura 7-c) la estimación de los landmarks y pose del robot difiere de los landmarks reales, esto es porque en esa ubicación el robot tiene pocas mediciones para hacer la corrección por lo que las partículas no serán buenas hipótesis en comparación a las anteriores y esto hace que aumente la varianza en la ubicación del robot y el mapa.