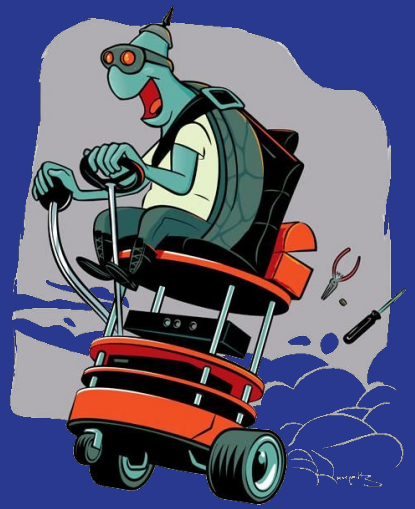
Laboratorio 3: Localización

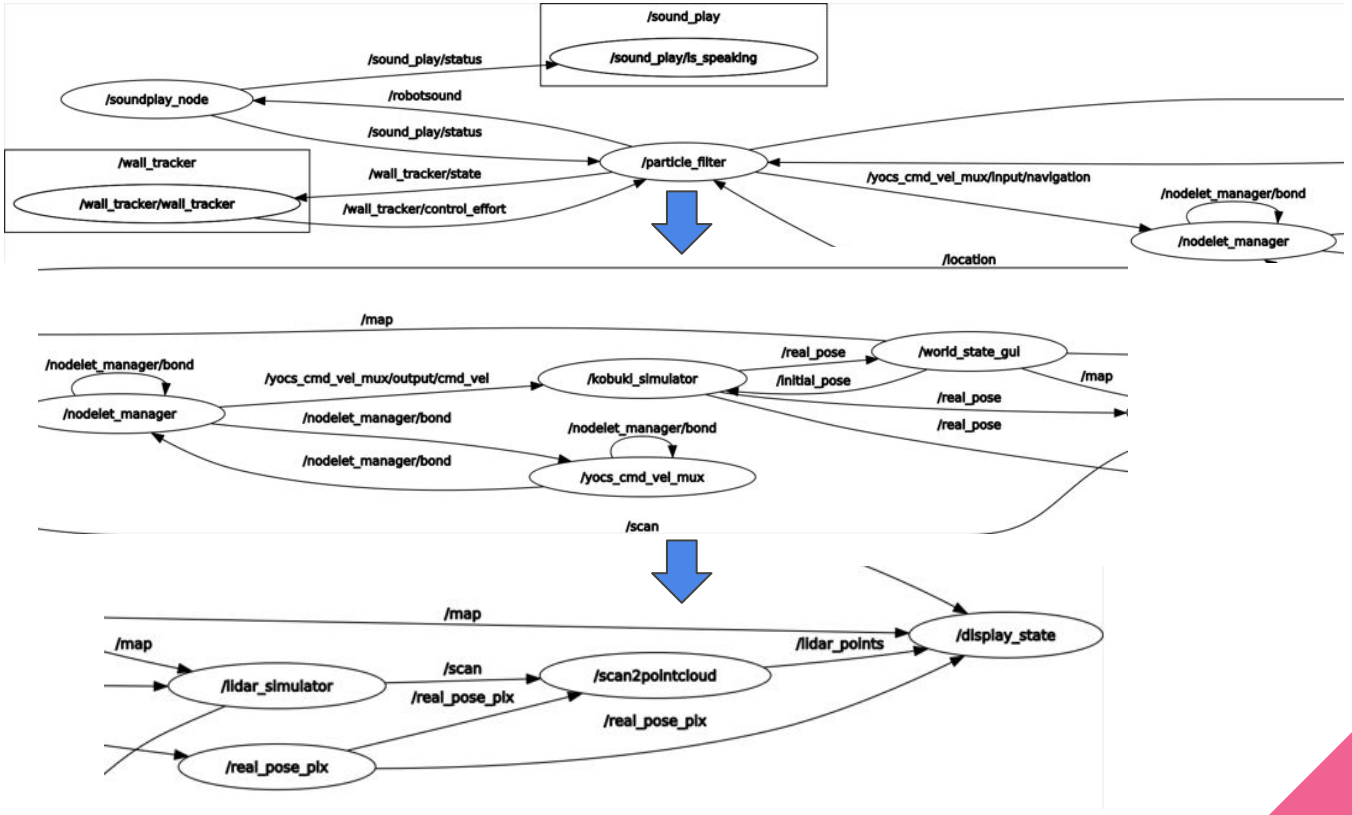
IIC2685 - Robótica Móvil

Equipo:

- Benjamín Farías
- Rafael Fernández
- Lukas Fuenzalida



Diseño del Software



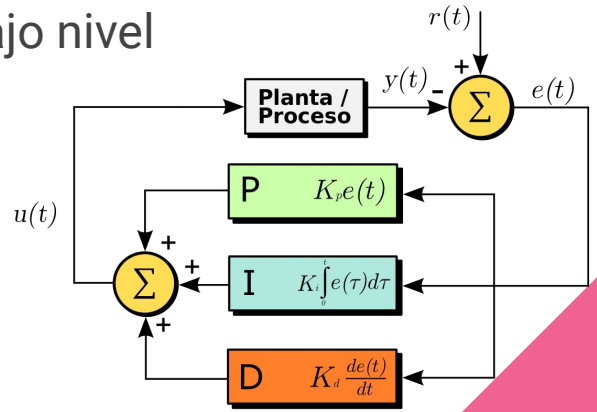
Localización - Componentes

El robot debe ser capaz de localizarse en su entorno, dado que se conoce el **mapa** y se cuenta con un sensor de tipo **LIDAR**. Existen 3 componentes necesarias para lograr este objetivo:

1) Criterio de exploración del entorno
2) Modelo del sensor
3) Filtro de partículas

Criterio de Exploración

- Se navegará con una velocidad lineal constante, intentando mantener una distancia fija de 0.3 [m] a la pared derecha
- Esto se logra mediante un **controlador PID** que regula la velocidad angular
- Si el robot se topa con una pared en frente (o cerca), se pasa a una **subrutina** en la que gira sobre sí mismo hasta encontrar un **camino libre** sobre el que pueda continuar su rutina de bajo nivel



Modelo del Sensor - Algoritmo

- El modelo utilizado para el sensor corresponde al **Likelihood Fields**

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

$q = 1$

for all k *do*

if $z_t^k \neq z_{\max}$

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

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

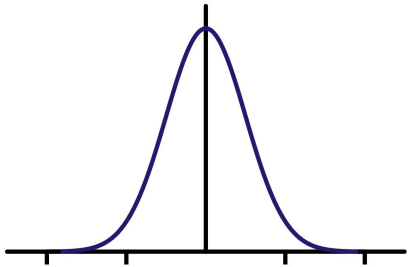
$$dist = \min_{x',y'} \left\{ \sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2} \;\middle|\; \langle x', y' \rangle \text{ occupied in } m \right\}$$

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

return q

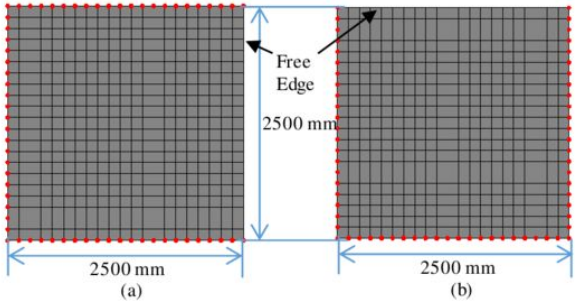
Modelo del Sensor - Implementación

- Una vez calculadas las puntas de los láser, se **descartaron** aquellas mediciones donde se **medía fuera del mapa** o se obtenía ***z_max***
- La distancia al obstáculo más cercano se obtiene de manera eficiente usando un **KDTree**
- No se considera la distribución de las **fallas** ni **mediciones aleatorias**, puesto que estas son ignoradas, o bien, se representan en la **Gaussiana del ruido**
- Mejor valor de ***z_hit* = 10**



Modelo del Sensor - Optimizaciones

- Dado el mapa original, se almacenaron sólo los obstáculos que correspondían a los **bordes** de las paredes (los únicos relevantes)
- Para el **KDTree** se usó la versión **cKDTree** (utiliza código en C por debajo)
- Al calcular las puntas de los láser, aquellos estados que quedan con **menos del 90% de estas coordenadas válidas** son inmediatamente asumidos como erróneos y se les asigna el **likelihood mínimo**



Filtro de Partículas - Algoritmo

- Se utilizó el algoritmo de **Monte Carlo Localization**

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 $\mathcal{X}_t$

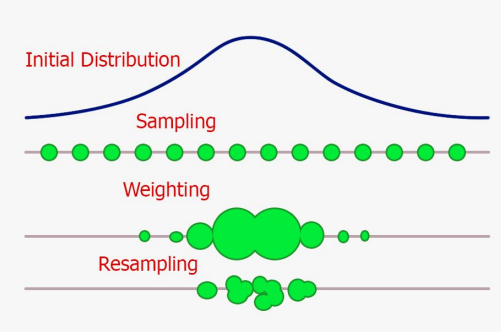
Filtro de Partículas - Implementación

- El modelo de movimiento se simuló con **Gaussianas** ajustadas al comportamiento del robot en el simulador
- Las partículas que quedan fuera del espacio válido al moverse son **descartadas y reemplazadas** por partículas aleatorias
- La cantidad de partículas ideal fue obtenida mediante **prueba y error**, buscando un balance entre cantidad de iteraciones, velocidad de procesamiento y calidad de la solución

N° de Partículas = 2500

Filtro de Partículas - Criterio de Convergencia

- Se revisan las partículas en cada iteración, contando la cantidad de veces que se repiten
- Las **más repetidas y cercanas entre sí** (a una distancia menor a 15 píxeles) son consideradas como el grupo **candidato**
- Si dicho grupo candidato corresponde a **más del 90% de las partículas totales**, se considera como **convergencia**

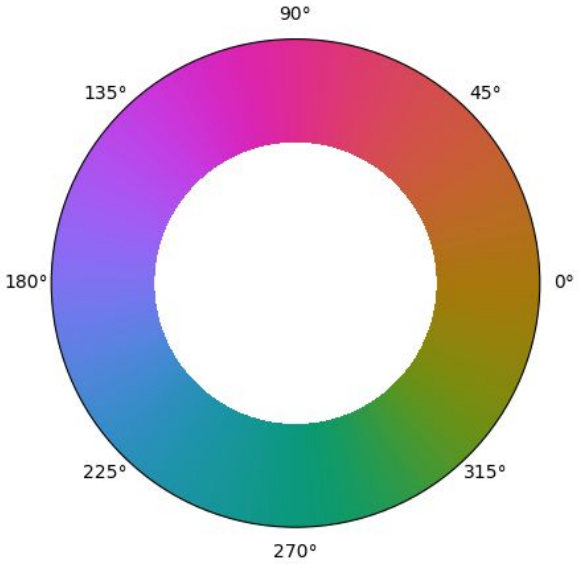


Filtro de Partículas - Optimizaciones

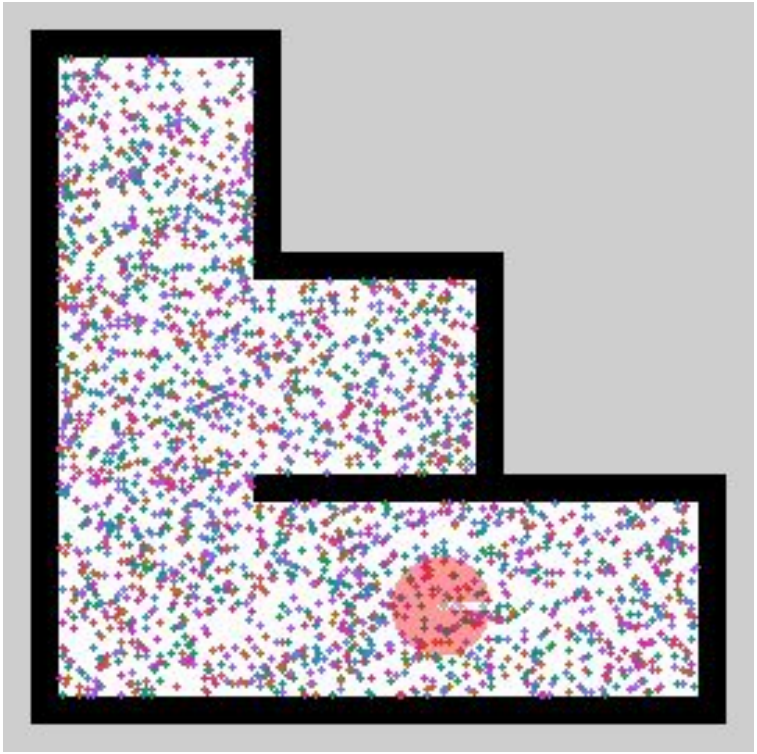
- Las partículas que obtienen un **likelihood bajo** desde el modelo del sensor son **reemplazadas por partículas aleatorias**, al igual que las inválidas tras el desplazamiento
- Al tener un candidato que logra converger, se realiza una **subrutina** en la que el robot da una vuelta completa sobre sí mismo
- Mediante esta subrutina se logra **verificar** que el candidato es efectivamente la **localización real**, y no una zona **simétrica** que se parecía al mirarla desde ciertos ángulos

Filtro de Partículas - Codificación del Ángulo

- La representación visual de la orientación de cada partícula se basa en la siguiente **escala de color**

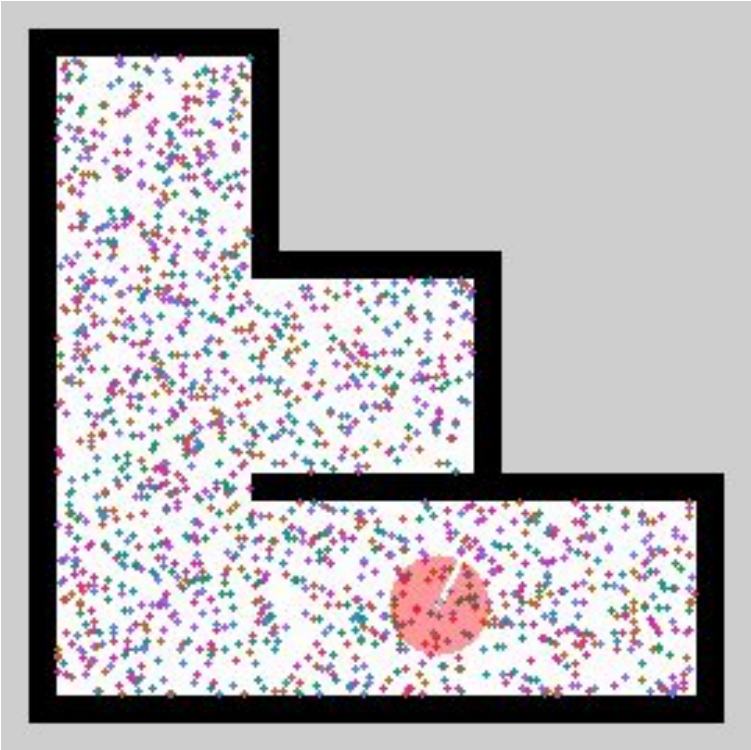


Filtro de Partículas - Ejemplo



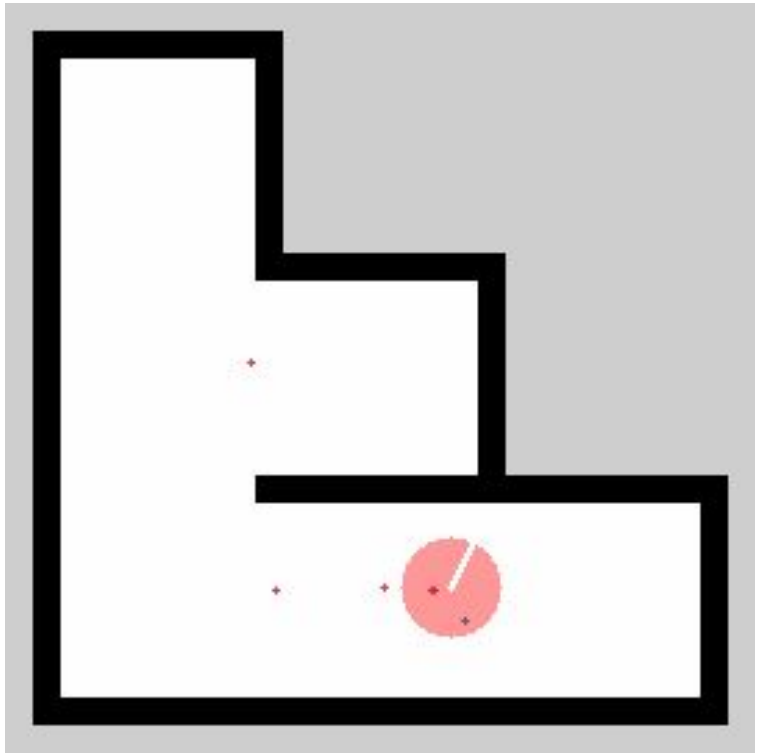
Partículas inicialmente esparcidas de forma uniforme sobre el espacio de poses

Filtro de Partículas - Ejemplo



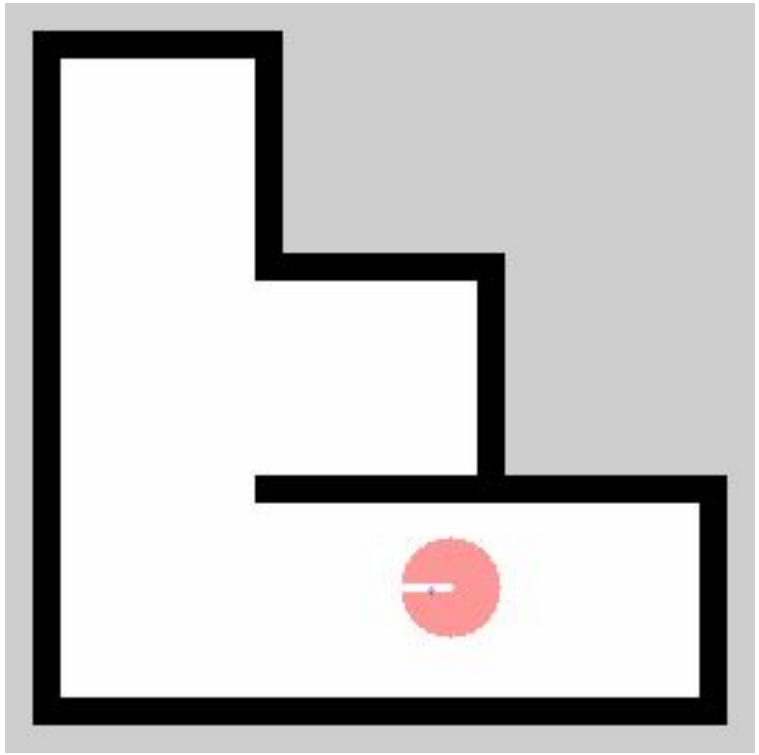
Partículas convergen y aparecen zonas más densas

Filtro de Partículas - Ejemplo



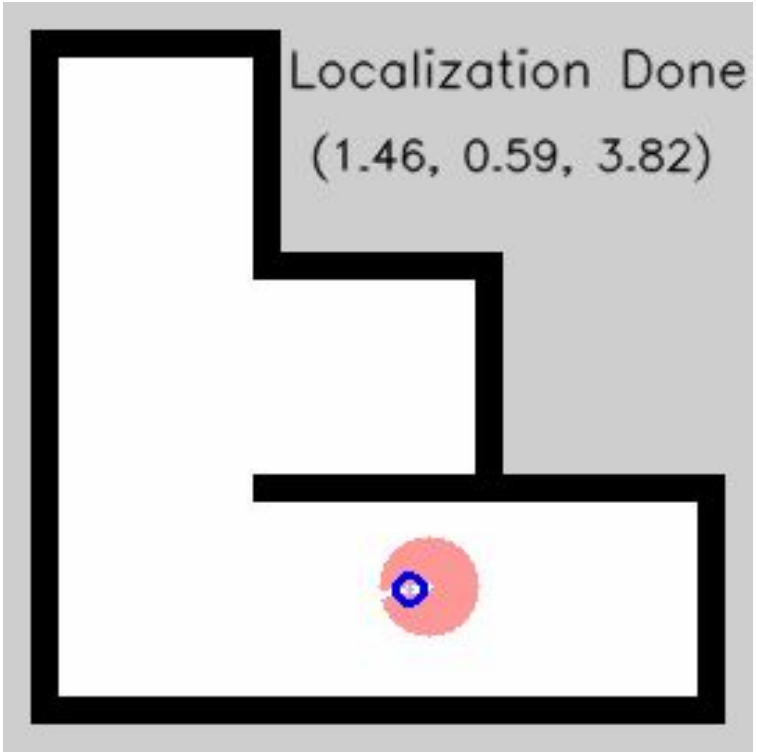
Gran mayoría de partículas se concentran en unas pocas zonas (cúmulos)

Filtro de Partículas - Ejemplo



Sobrevive el cúmulo de mayor importancia

Filtro de Partículas - Ejemplo



El robot determina que dicho cúmulo es efectivamente su localización aproximada

Conclusiones

- Los filtros de partículas son una herramienta muy útil para resolver problemas en los que la cantidad de estados es demasiado grande
- Son altamente customizables, siendo posible optimizarlos según el problema que se desea abordar
- Al ser algoritmos aleatorizados, queda un poco a la suerte su *performance* en ciertos casos (hecho que se puede mitigar con optimizaciones)
- Se puede realizar localización de robots de forma bastante confiable, dado un mapa conocido