Universidad de San Andrés, Depto. de Ingeniería

Práctica 3 - Filtros de Partículas

I-402 - Principios de la Robótica Autónoma

Prof. Dr. Ignacio Mas, Tadeo Casiraghi y Bautista Chasco

8 de septiembre de 2025

Fecha límite de entrega: 26/09/25, 23:59 hs.
Modo de entrega: Enviar por el Aula Virtual del Campus todo el código comentado y los gráficos (.jpg ó .pdf), todo en una sola carpeta comprimida.

1. Notas preliminares

Antes de poder comenzar con el trabajo práctico deberán agregar lo siguiente al paquete base provisto por la cátedra:

Paquete de turtlebot3_custom_simulations
Descarguen nuevamente el paquete de simulaciones custom ya que ahora contiene el nuevo mundo que utiliarán para este TP. Vuelvan a hacer los pasos que hicieron previamente para instalarlo.

Paquete de custom_code

1. **setup.py**

   Al archivo de setup.py agregar dos entry points nuevos:

   a) `"likelihood = custom_code.likelihood:main"`

   b) `"localization = custom_code.my_localization:main"`

2. **rviz**

   Agregar el archivo ***particulas.rviz*** al directorio de rviz

3. **launch**

   Agregar el archivo `launch_my_particles.launch.py` al directorio de launch

4. **código custom**

   Por último, agregue los siguientes archivos junto con los otros códigos de python:

   - `likelihood.py`
   - `my_localization.py`
   - `robot_functions.py`

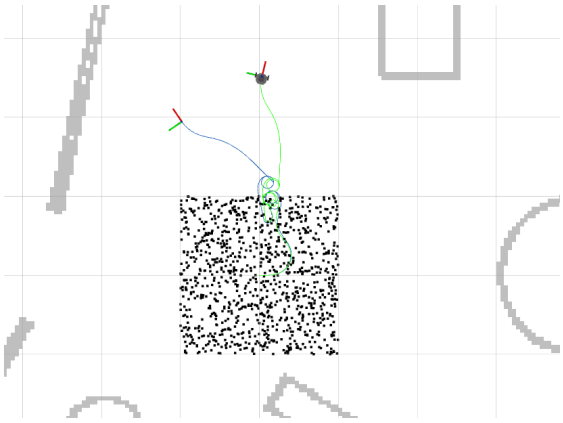
   Tenga cuidado de no eliminar su codigo anterior de `robot_functions.py` cuando agregue el nuevo archivo.

2. Proceso de lanzamiento de nodos

Para correr el código debemos tener 3 terminales (no se olviden del colcon build y el source install/setup.bash)

- Terminal 1: `ros2 launch turtlebot3_custom_simulation custom_room.launch.py`
- Terminal 2: `ros2 launch custom_code launch_my_particles.launch.py`
- Terminal 3: `ros2 run turtlebot3_teleop teleop_keyboard`

Si siguieron todos los pasos correctamente verán lo siguiente en Rviz2:



Notarán que:

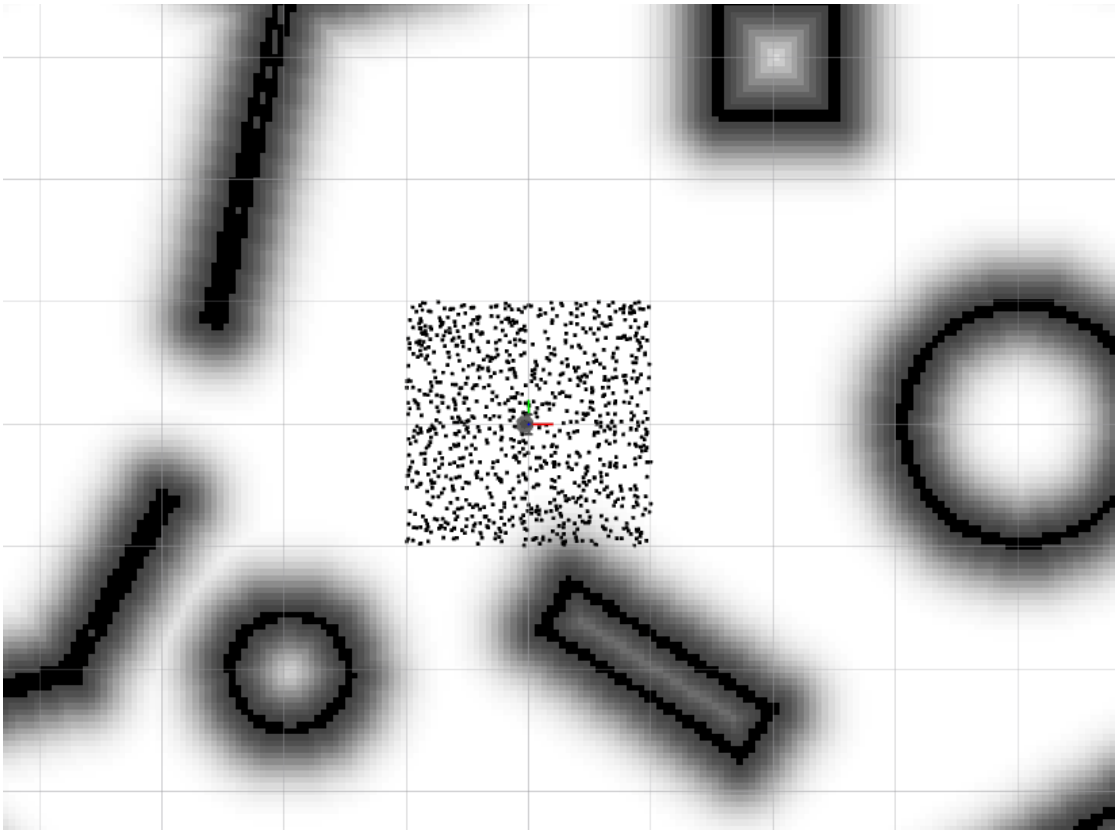
- Está el mapa pero no está el campo de verosimilitud
- Las partículas no se mueven
- Se grafica el camino real del robot (verde) y el calculado en base a la odometría (azul), pero falta el camino (rojo) que calculamos con las partículas

3. Campo de Verosimilitud

Para poder navegar por el entorno y utilizar el sensor LIDAR debemos tener un modelo de sensado implementado. Para este trabajo práctico deberán implementar un modelo de sensor LIDAR basado en el campo de verosimilitud. Esto implica que debemos tener un campo creado. Para eso utilizaremos el nodo de likelihood.py.

El nodo likelihood.py que se encuentra junto a la consigna se subscribe al tópico del mapa y debe publicar el campo de verosimilitud. Para ello deberán completar la función `map_callback` de manera que procese la información recibida y genere esta grila de probabilidades. El mensaje publicado debe contener la data en el campo data, que en ROS se espera como un array unidimensional de enteros de 8 bits con valores entre 0 y 100. La información del mapa llega en msg.data siguiendo un orden específico: comienza en la esquina inferior izquierda del mapa, avanza por cada fila de izquierda a derecha y luego continúa con la fila superior, hasta completar todo el mapa. En este arreglo, las celdas libres tienen valor 0, las ocupadas 100 y las desconocidas -1.

Si completaron correctamente el codigo, deberán ver algo como lo siguiente al lanzar todo nuevamente:



4. Movimiento de Partículas

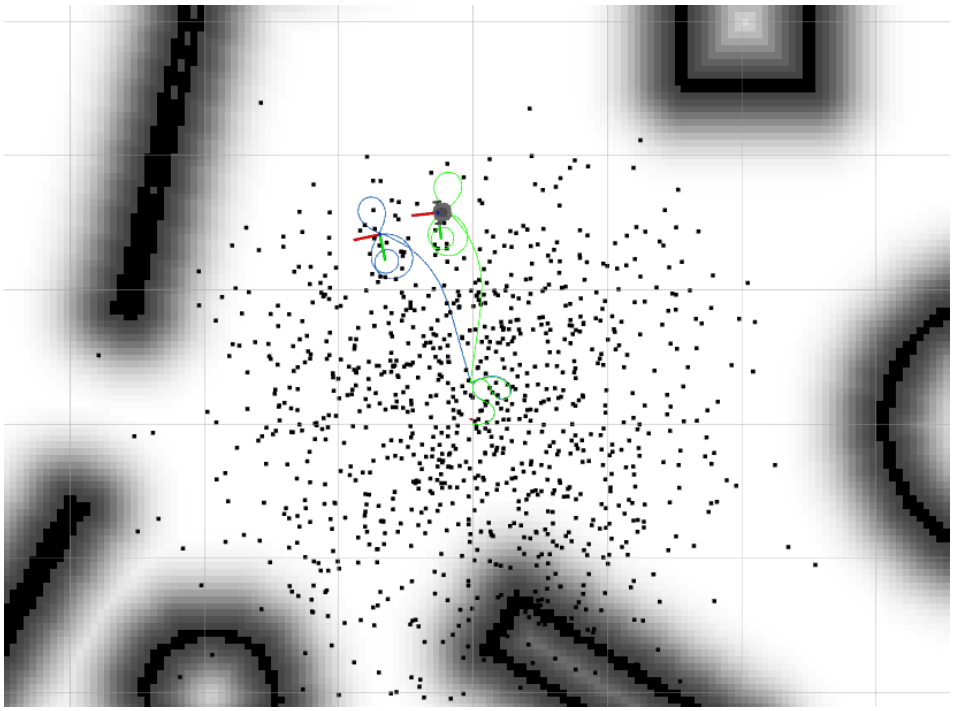
Ahora que tenemos un campo funcional, implementaremos el movimiento de las partículas. El código de funciones de robot provisto en campus crea una clase denominada ***particle***. Esta clase representa una partícula, y tiene valores de posición, orientación y peso. Tiene funciones para modificar el valor de los datos. También tiene la función de movimiento de odometría. Esta función recibe la odometría (deltas) y los valores de ruido a aplicar (los alpha). Deberán implementar un código que calcule las posiciones nuevas incluyendo el error. Pueden usar su código del TP2 para esto.

Deberán también completar la función `get_selected_state`. Esta función debe devolver el valor del estado del robot (x, y, theta) en función a las partículas. El método para hacer eso queda a su criterio.

Si hicieron todo correctamente ahora verán que las partículas se mueven y se grafica la línea roja. Es posible que no se vea claramente la línea roja ya que como las partículas se mueven en orientaciones random no hay una dirección preferencial.

Si con las 1000 partículas default corre muy lento o no se localiza correctamente puede probar cambiando la cantidad de partículas. Para eso puede correr el launch file de la siguiente manera:

```
ros2 launch custom_code launch_my_particles.launch.py num_particles:=500
```



5. Sensado y Resampleo

Por último deberán implementar el modelo de sensor y el resampleo de partículas. Para ello deberán completar la función `update_particles`. Esta funcion recibe:

- **data**: datos del lidar en formato scan (Ver documentacion de ROS sobre tipo de dato LaserScan). Pueden aprovechar la funcion `scan_refererence` del TP1 para convertir los datos crudos en posiciones globales calculadas
- **map_data**: Es el mensaje crudo del campo de verosimilitud. Pueden consultar la documentacion de ROS sobre tipos de dato OccupancyGrid.
- **grid**: Es la representación como matriz de numpy del campo de verosimilitud. Importante:
  - La grilla se indexa como grid[y, x], primero fila (eje Y) y luego columna (eje X)
  - La celda (0,0) corresponde a la esquina inferior izquierda del mapa en coordenadas de ROS.

Esta función debe tomar toda esta data y actualizar el valor de probabilidad (weight) de cada partícula. En base a eso debe resamplear las partículas. Tenga cuidado al resamplear de hacer un deepcopy para que no sean el mismo objeto de python. El resampleo deberá ser guardado en self.particles, que es una lista de partículas.

Si se completó todo correctemante verán que, al mover el robót con el nodo del teclado, ahora las partículas convergen a la posición **real** del robot, y que el camino rojo sigue el camino verde.