Lab05

Localization with Particle Filter

Objectives

- Develop a ROS 2 package to run Particle Filter for robot localization.
- Localize your robot using the landmark measurements model.
- Try the localization system in both simulated and real environments.

Requirements

- The Python scripts explained during lectures (motion model, sensor model, particle filter), present in the portale della didattica (materiale/lecture_notebooks/Discrete_Filters)

Exercise

Main objective: realize a ROS 2 package for Particle Filter localization. The package will be validated in Gazebo simulation and then deployed on the robot.
The lab exercise is divided into 2 Tasks. Tasks 1 can be prepared before the experimental session in the lab using the Gazebo simulation. Task 2 can be addressed by recording trajectory data from the real robot in the lab (a recovery bag is provided in the portale in case time in the lab would not be enough).

Task 1 [3 points]

Starting from the Particle Filter scripts that were presented during lectures, implement inside a ROS 2 node a Particle Filter for tracking the robot position given landmarks measurements.

State of the filter: the state of the filter is the one reported below, where x, y, θ represent the position of the robot in the global reference frame.

$$\mu = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Particles Initialization: sample the initial distribution of the particles using a uniform distribution with the boundaries of the environment where you are navigating in. **Perform this operation in the `__init__` function of your node.**

Prediction: perform the prediction step at a **fixed rate** of 20 Hz using the **velocity motion model**. Use as **command** the most recent v and ω taken from the `/cmd_vel` **topic**. **Create a timer in your node to perform the prediction operation of Particle Filter.**

Update: measurements are provided as **range and bearing** of **landmarks** in the field of view of the robot. Measures are published on **topic** `/landmarks` (or `/camera/landmarks` on the real robot) inside a message of type `landmark_msgs/msg/LandmarkArray`.
Perform the update operation of the Particle Filter inside the subscription callback for each landmark listed in the message.
Once all the updates are done, **normalize weights** and consider **resampling**. Then, **publish** the estimated state in a message of type `nav_msgs/msg/Odometry` on topic `/pf`. Make sure to **fill** the `header.stamp` field with the current time taken from `self.get_clock().now().to_msg()`.

Landmarks coordinates are provided in a yaml file inside the `turtlebot3_perception` package (`turtlebot3_perception/turtlebot3_perception/config/landmarks.yaml`) in the following format:

```yaml
landmarks:
  id: [id0, id1, ..., idN]
  x: [x1, x2, ..., xN]
  y: [y1, y2, ..., yN]
  z: [z1, z2, ..., zN]
```

Resampling: perform the resampling according to one of the strategies presented during lectures (all available in utils.py). You can **try different strategies** and **compare** if any differences appear. Define the moment to perform resampling according to a reasonable condition based on the effective number of particles computed.

Task 2 [2 point]

Run the ROS 2 package for Particle Filter localization on the real robot and record the performance of the filter.

Report requirements

- Provide a concise description and comments on the main structure of your ROS 2 program (do not copy your code in the report, just highlights the main elements and the workflow: nodes, publishers/subscribers to relevant topics and parameters)

- **[Task 1]**: **Run your filter in the simulation environment.** Record the `/ground_truth`, `/odom` and `/pf` topics. Then:
    - **Make plots** comparing the position and orientation reported in the three topics
        - Make a plot for each state (x, y, θ) vs. time and compare the different topics. You shall put the state reported by all the three topics in the same plot. (Examples provided in python-crash-intro repository Module3)
        - Plot the (x, y) trajectory on the 2D plane using the robot's pose data from the three topics. Add the landmarks known poses to the plot.
        - Comment the results
    - **Compute the following metrics** with respect to the ground truth: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) Further instructions will be provided on this point in a separate file.

- **[Task 2]**: **Run your filter on the real robot.** Record the `/odom` and `/pf` topics. Then:
    - Make plots comparing the position and orientation reported in the two topics. For `/odom` you can subtract the first pose to all the other poses to align to the origin. Then:
        - Make a plot for each state component over the time length of the experiments, and compare the different topics.
        - Plot the (x, y) trajectory using the data collected by the two topics. You can also plot the landmarks
        - Comment on the results.

How to test your algorithms

Simulation Environment

The following operations shall be executed on your PC.

A simulation is provided to ease the development and testing of your code. To launch the simulation run the following command in a workspace with both `turtlebot3_simulations` and `turtelbot3_perception` packages. (Mac users should run the `ignition` version of the command in the same way adopted for Lab02)

```
ros2 launch turtlebot3_gazebo lab04.launch.py
```

The simulation environment is the one shown in Figure 1, with the **white columns** acting as **landmarks**. The orange labels are the landmarks IDs.

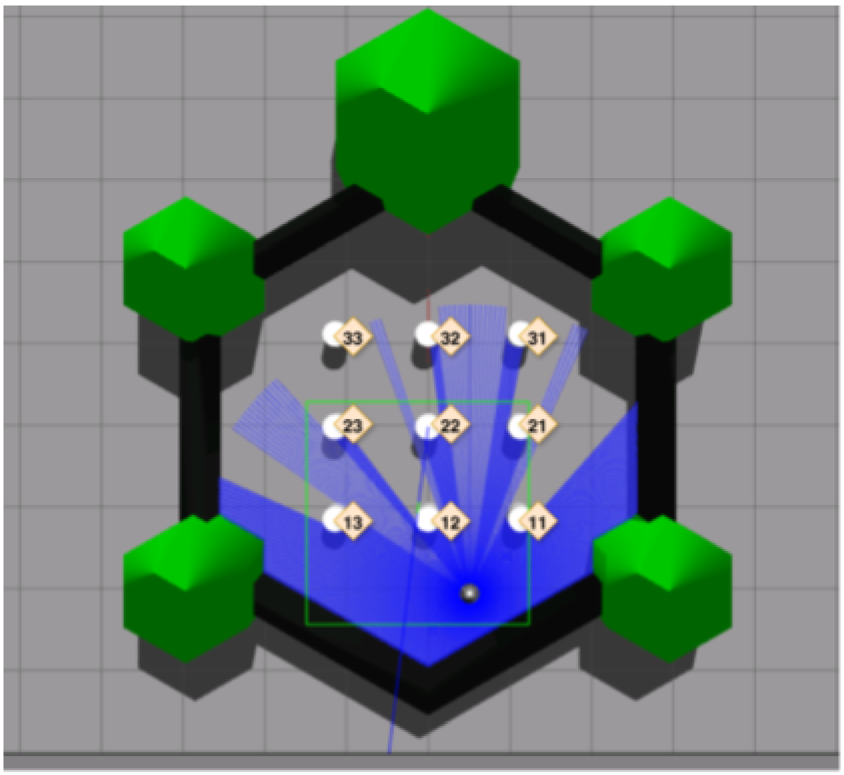

Figure 1. Simulation environment with labelled landmarks

Real Robot

The following operation shall be performed on the robot.

Install the camera

1. Mount the camera on the robot as you did in Lab04.
2. In two different terminals start the camera driver and the landmark detector using the following commands

```
# Terminal 1
ros2 launch turtlebot3_perception camera.launch.py
```

```
# Terminal 2
ros2 launch turtlebot3_perception apriltag.launch.py
```

3. If landmarks are present, they will be published on topic `/camera/landmarks` at approximately 6 Hz.

Record and replay your data

You can choose to **record a rosbag** with the **data needed to run the filter** for testing your algorithm without the robot physically running. For example, you could register with the following command:

```
ros2 bag record /odom /cmd_vel /camera/landmarks
```

To replay the data and run your algorithm you need two terminals. In the first terminal you can replay the rosbag, also publishing it time on the clock. In the second, you shall run your node with the parameter `use_sim_time`, in order to get the time from clock and not from system time. You can find the commands below.

```
# Terminal 1
ros2 bag play --clock 1000 /path/to/rosbag

# Terminal 2
ros2 run your_pkg your_node --ros-args -p use_sim_time:=true
```

To **record the results of your experiments** use the following command to avoid recording useless topics:

```
ros2 bag record /odom /cmd_vel /pf /camera/landmarks
```