[CS-475] Assignment 5: FastSLAM

Changda Tian
cdtian@csd.uoc.gr

Release: 30/04/2025
Deadline: 13/05/2025

1 Overview

In your previous assignments, you implemented two important components of mobile robotics:

- A particle filter for localization (using a provided height map).
- A mapping algorithm with known (ground truth) robot poses, which enabled you to build an accurate map by measuring distances from the robot.

In real-world applications, however, having either a pre-built map or accurate pose measurements is often a luxury. When a robot is equipped only with onboard sensors, it must learn its environment and simultaneously determine its own location. This is the core challenge of **Simultaneous Localization and Mapping (SLAM)**.

In this assignment, we will implement **FastSLAM**, a method that combines:

- A particle filter to represent multiple hypotheses about the robot's pose.
- An occupancy grid mapping algorithm, which generates a map from sensor data.

Each particle maintains a hypothesis of both the robot's pose and a corresponding map. At every time step, the particle's pose is updated with a motion model that includes sensor noise, and its map is updated using new sensor readings. Then, a weight is assigned to each particle according to how well its map explains the sensor measurements. Particles with higher weights (the more "popular" ones) are retained during the resampling step.

For clarification, here are some key definitions:

- X_t: The robot's pose (position and orientation) at time t.
- u_t: The control input (e.g., displacement and change in orientation) between $t-1$ and t.
- z_t: The sensor (e.g., laser scan) measurements at time t.
- $w_t^{[i]}$ and $m_t^{[i]}$: The weight and map associated with the ith particle at time t, respectively.

The following pseudo-code summarizes the FastSLAM algorithm:

Algorithm 1 FastSLAM Algorithm

```
1:  function FastSLAM($X_{t-1}, u_t, z_t$)
2:      $S \leftarrow \emptyset$
3:      for each particle $p \in \{1, \dots, N\}$ do
4:          $x_t^{[p]} \leftarrow$ MotionUpdate($u_t$, $x_{t-1}^{[p]}$)
5:          $w_t^{[p]} \leftarrow$ SensorUpdate($z_t$, $x_t^{[p]}$)
6:          $m_t^{[p]} \leftarrow$ UpdateOccupancyGrid($z_t$, $x_t^{[p]}$, $m_{t-1}^{[p]}$)
7:          $S \leftarrow S \cup \{x_t^{[p]}, w_t^{[p]}, m_t^{[p]}\}$
8:      end for
9:      LowVarianceResample(S)
10: end function
```

Key Equations

Below are some important equations used in FastSLAM:

1. **Weight Calculation:** For each particle, compare the expected sensor measurements $\mathbf{d}$ (generated from the particle's current map) with the actual measurements $\mathbf{z}_t$. The particle weight is given by:

$$w_t^{[i]} = \frac{1}{\|\mathbf{d} - \mathbf{z}_t\| + 1}$$

   where the Euclidean norm is computed as:

$$\|\mathbf{d} - \mathbf{z}_t\| = \sqrt{\sum_{j=1}^{M} (d_j - z_{t,j})^2}$$

   Here, M represents the number of sensor beams (e.g., 360 for a full circle).

2. **Motion Update:** The motion update accounts for the robot's displacement in both translation and rotation. Given the control input (dx, dy, $d\theta$) and incorporating random sensor noise:

$$x_t^{[i]} = x_{t-1}^{[i]} + \Delta r \cos(\theta_{t-1}^{[i]}) + \epsilon_x,$$
$$y_t^{[i]} = y_{t-1}^{[i]} + \Delta r \sin(\theta_{t-1}^{[i]}) + \epsilon_y,$$
$$\theta_t^{[i]} = \theta_{t-1}^{[i]} + d\theta + \epsilon_\theta,$$

   where

$$\Delta r = \sqrt{dx^2 + dy^2},$$

   and ϵ_x, ϵ_y, ϵ_θ are small random noise terms representing measurement errors.

2 Installation

In the provided *.zip* file, you will find a package named *assign5/*. Follow the steps below to prepare your environment:

1. Copy the `assign5/` folder into your catkin workspace.
2. Compile your workspace using: `$ catkin_make`
3. Launch the simulation with: `$ roslaunch assign5 burger.launch`
4. If the new package does not appear immediately, update the ROS package manager using: `$ rospack profile`
5. If everything works, open another terminal and launch the turtlebot3_teleop node:
   `$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch`

After launching, you should see both RViz and Gazebo visualizations, as shown in Figure 1.

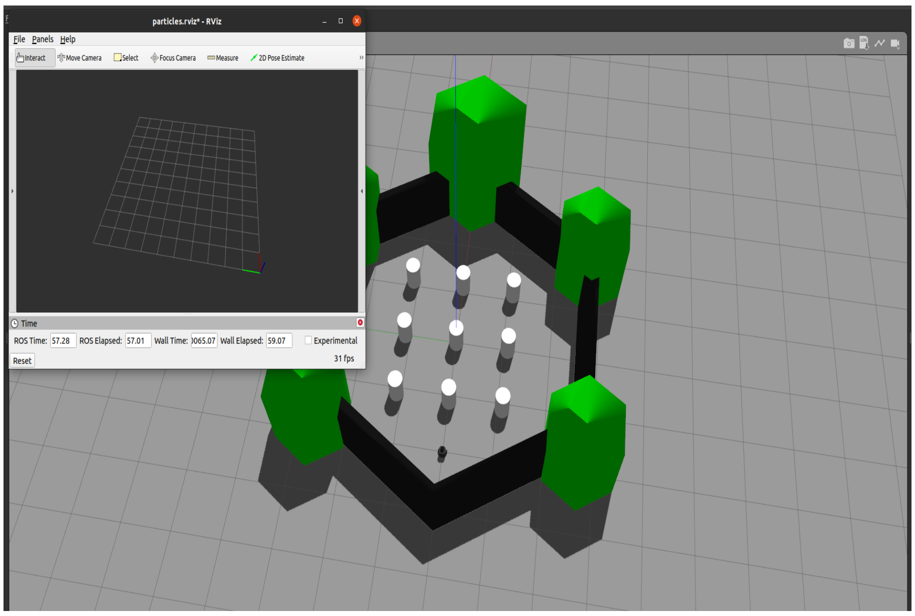
particles.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate
Time
ROS Time: 57.28 ROS Elapsed: 57.01 Wall Time:)065.07 Wall Elapsed: 59.07
Experimental
Reset
31 fps

Figure 1: Environment

3 Implementation Details

You are required to implement a ROS node (located at `src/slam.py`) that carries out the FastSLAM algorithm. The following sections explain each module in detail.

3.1 Initialization

Map Initialization:

- The workspace dimensions are approximately $6\,m \times 6\,m$. This information is used to initialize an occupancy grid map.
- For example, the map might initially be defined over an area of $10\,m \times 10\,m$ (allowing for some flexibility), with each grid cell initialized to an "unknown" value (commonly 50).

Particle Initialization:

- Initialize N particles (recommended: ~ 20).
- Each particle is assigned a pose (x, y, θ), which should be near the center of the occupancy grid. This is crucial because particles starting at the map's boundaries may not receive sensor measurements that are within the grid.
- Each particle also maintains its own copy of the occupancy grid.

3.2 Motion Update

The motion update adjusts each particle's pose based on the control inputs and includes random noise to simulate real-world uncertainty.

- Obtain the displacement inputs dx, dy and change in orientation $d\theta$ from odometry or GPS. These values describe the movement relative to a fixed global frame.
- Each particle's new position is computed by:

  $$\begin{aligned} x_t^{[i]} &= x_{t-1}^{[i]} + \Delta r \cos(\theta_{t-1}^{[i]}) + \epsilon_x, \\ y_t^{[i]} &= y_{t-1}^{[i]} + \Delta r \sin(\theta_{t-1}^{[i]}) + \epsilon_y, \\ \theta_t^{[i]} &= \theta_{t-1}^{[i]} + d\theta + \epsilon_\theta, \end{aligned}$$

  where

  $$\Delta r = \sqrt{dx^2 + dy^2}.$$

- The inclusion of ϵ_x, ϵ_y, and ϵ_θ (small random noise values) ensures that the particles do not move in perfect synchrony, which is a realistic representation of sensor and actuator noise.
- Visualize the particles in RViz. If implemented correctly, you should see a slight "wiggling" of particles reflecting the inherent uncertainty in the motion measurements.

3.3 Occupancy Grid Update

Each particle updates its respective occupancy grid based on the latest laser scan measurements.

- For each laser scan beam:
  1. Start from the particle's current position and "raytrace" along the beam's direction.
  2. For grid cells along the beam path (from the minimum scan range up to but not including the hit cell), update the log-odds to indicate free space.
  3. For the cell corresponding to the endpoint where an obstacle is detected, update with a log-odds value that indicates occupancy.

- The log-odds update can be represented as:

$$L(cell) = \log \left(\frac{p(\text{occupied} \mid z)}{1 - p(\text{occupied} \mid z)} \right)$$

- To verify this module, first test with a single particle and disable the noise (i.e., set `ToggleNoise = 0`). A nearly perfect map should emerge. Once verified, add additional particles and compare their individual maps.

3.4 Sensor Weight Update

The weight of each particle is a measure of how well its map matches the actual sensor data.

- For each particle, simulate an expected laser scan using its current map. This creates a vector $\mathbf{d}$, where each element d_j is the distance from the particle to the nearest obstacle along the beam direction j.

- Compare this expected measurement $\mathbf{d}$ with the actual measurement $\mathbf{z}_t$ obtained from the sensor:

$$\|\mathbf{d} - \mathbf{z}_t\| = \sqrt{\sum_{j=1}^{M} (d_j - z_{t,j})^2}.$$

- Calculate the particle's weight as:

$$w_t^{[i]} = \frac{1}{\|\mathbf{d} - \mathbf{z}_t\| + 1}.$$

- Finally, normalize the weights so that:

$$\sum_{i=1}^{N} w_t^{[i]} = 1.$$

3.5 Low Variance Resampling

This resampling step refines the particle set by keeping only the more probable hypotheses.

- Construct a cumulative distribution function (CDF) from the normalized particle weights.

- Generate a uniformly distributed random number $u_0 \in [0, \frac{1}{N}]$.

- For each particle index $j \in \{0, 1, \ldots, N-1\}$, choose a particle based on the CDF using:

$$u_j = u_0 + \frac{j}{N}.$$

- Select the particle corresponding to u_j and replicate it into the new particle set. This process favors particles with higher weights.

For further details on low variance resampling, see Table 4.4 in *Probabilistic Robotics* (p.110).

4 Results

When your FastSLAM implementation is working correctly, you should observe:

- Each particle's occupancy grid map will gradually converge towards an accurate representation of the environment.
- The individual maps may be shifted or rotated relative to each other due to the particle's different estimated poses, but the local representation (i.e., the relationship between the robot and surrounding obstacles) should be correct.

For instance, when particles are initialized near the robot's true pose, you might see occupancy grids similar to those in Figure 2:

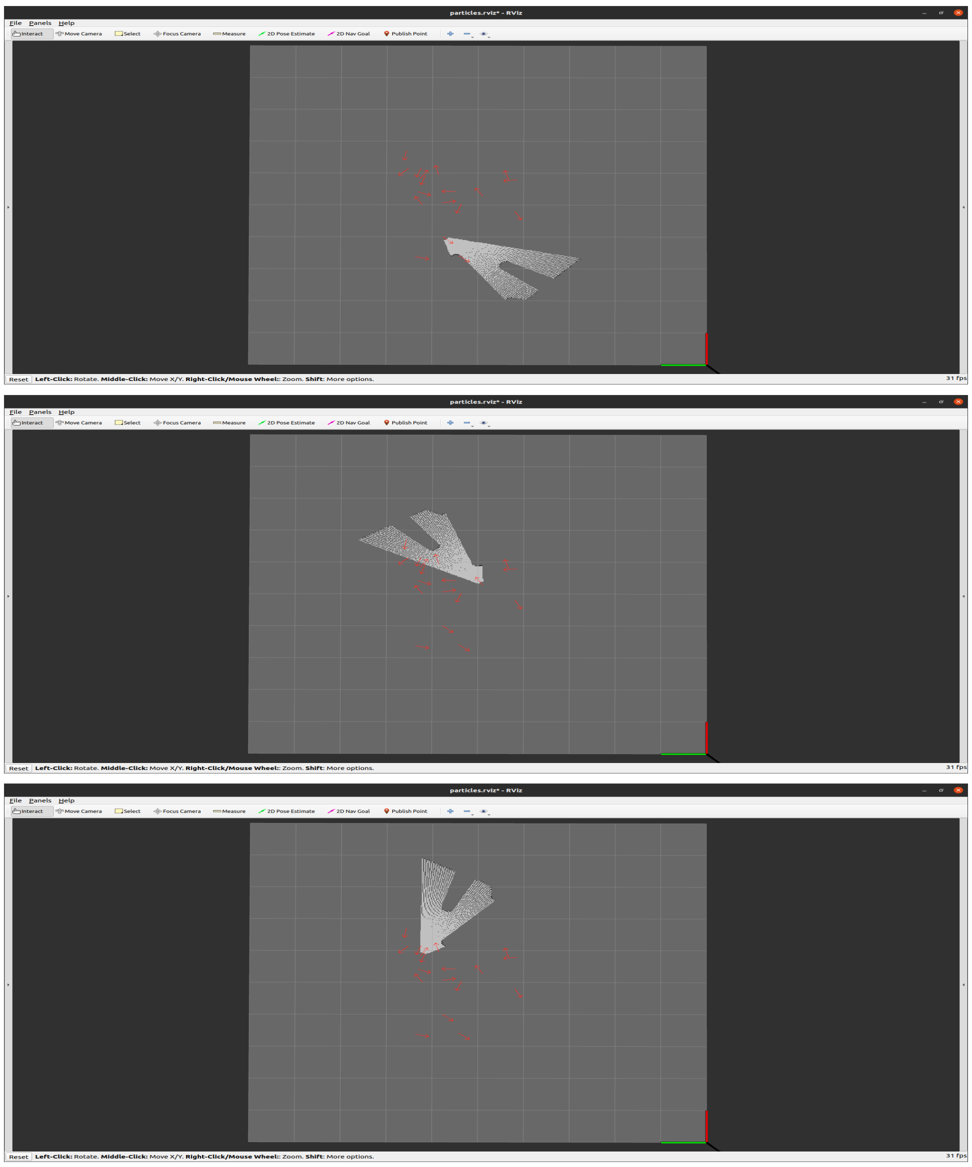

Figure 2: Example occupancy grid maps generated by three individual particles. Although each map's global frame may differ, the relative configuration between the robot and obstacles is correctly captured.

Finally, the particle with the highest weight, indicating the best match between predicted and actual sensor data, will have its occupancy grid published as the final map.

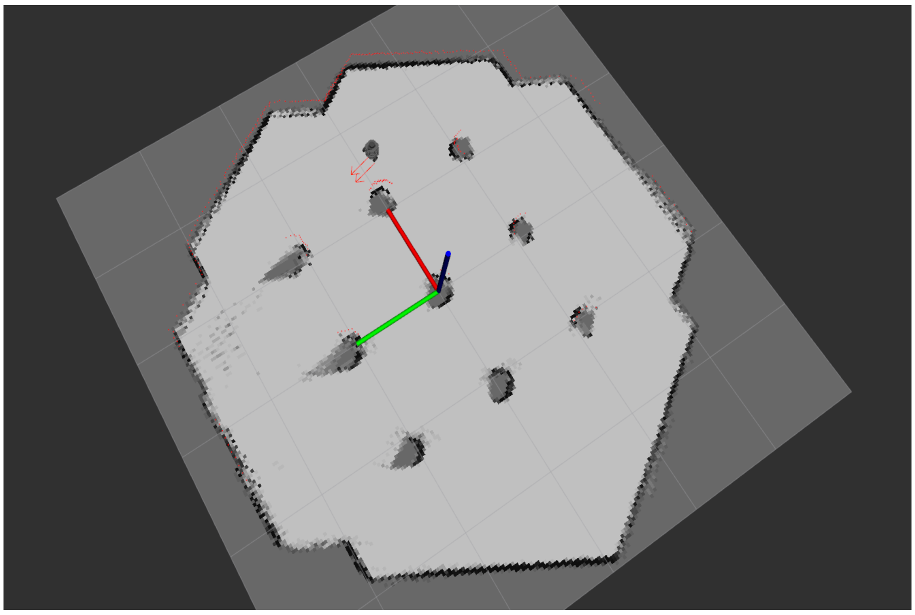

Figure 3: Final Result: Occupancy grid map published from the highest-weight particle.

5 Tips for Success

1. **Subsampling the Scan:** Processing an entire 360° scan can be computationally expensive. Consider using a smaller field-of-view (e.g., from -30° to 30°) to reduce processing time.
2. **Map Resolution:** Tweak the `map_resolution` parameter to balance the level of detail and real-time performance.
3. **Step-by-Step Validation:** Implement and validate each module (motion update, occupancy grid update, weight update, and resampling) independently before integration.
4. **Visualization:** Regularly publish both the particle poses and occupancy grids to RViz. This will help you identify and correct errors in real time.

6 Submission

Email your ROS node file (`slam.py`) with subject: `[CS-475] Assignment 5 Submission` to:

`cdtian@csd.uoc.gr`

Clearly mention your **name and registration number**. Submission deadline is strictly:

13/05/2025, 23:59