Grid-Based FastSLAM with Raster Image Sensors: Mapping with Particle Filters

Mohammed-salih Diyari
SLAM Project Report

December 4, 2025

Abstract

This report presents an implementation of Grid-Based FastSLAM using a rasterized local occupancy image instead of traditional lidar beams. The method follows the "mapping with known poses" strategy described in class and in the provided course document, where each particle maintains its own occupancy grid and updates it according to its estimated trajectory. We introduce a raster-compatible likelihood function based on binary occupancy classification (Variant B), and apply log-odds mapping to update occupancy values. Experimental results demonstrate how the particle filter reconstructs the environment from noisy rotated image patches.

1 Introduction

Simultaneous Localization and Mapping (SLAM) aims to estimate both the trajectory of a robot and a map of an unknown environment. FastSLAM decomposes the posterior:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{1}$$

into a product of:

$$p(x_{1:t} \mid z_{1:t}, u_{1:t}) \times p(m \mid x_{1:t}, z_{1:t}), \tag{2}$$

where the first term is estimated using a particle filter, and the second term corresponds to occupancy grid mapping with known poses.

Unlike classical lidar-based FastSLAM, our simulator provides a 50×50 raster image centered on the robot, rotated by the robot's true heading plus noise. Our objective is to adapt grid-based FastSLAM to operate directly on these raster images.

2 FastSLAM Overview

Each particle i maintains:

- A pose estimate (x_i, y_i, θ_i),
- A full occupancy grid map m_i,
- An importance weight w_i.

The complete posterior distribution is approximated as:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \approx \sum_{i=1}^{N} w^{(i)} \, \delta(x_{1:t} - x_{1:t}^{(i)}) \, p(m \mid x_{1:t}^{(i)}, z_{1:t}). \tag{3}$$

Motion prediction uses a noisy control model:

$$\begin{aligned} \theta_{t+1} &= \theta_t + \Delta\theta + n_\theta, & (4) \\ x_{t+1} &= x_t + \sin(\theta_{t+1}) \, d', & (5) \\ y_{t+1} &= y_t + \cos(\theta_{t+1}) \, d', & (6) \end{aligned}$$

where d' and $\Delta\theta$ include Gaussian noise.

3 Grid-Based FastSLAM Using Raster Images

The course notes (p. 24–29) describe grid mapping as:

$$p(m \mid x_{1:t}, z_{1:t}) = \prod_{cells} p(m_k \mid x_{1:t}, z_{1:t}).$$

In our project, the sensor measurement z_t is a 50×50 intensity map representing occupancy around the robot. Since the image is rotated according to θ_t, each pixel is transformed to world coordinates using:

$$\begin{aligned} x_w &= x + dx \cos\phi - dy \sin\phi, & (7) \\ y_w &= y + dx \sin\phi + dy \cos\phi, & (8) \end{aligned}$$

with $\phi = -(\theta + 90°)$.

4 Binary Occupancy Classification (Variant B)

Due to sensor noise, we convert pixel intensities into three classes:

$$\text{occupied if } I > 0.6, \qquad \text{free if } I < 0.4, \qquad \text{unknown otherwise.}$$

The likelihood for each particle is computed by comparing the sensor occupancy against the particle's predicted map region:

$$\log p(z_t \mid x_t, m_t) = w_{\text{hit}} N_{\text{match}} - w_{\text{miss}} N_{\text{mismatch}}.$$

Matching occurs when:

$$\text{occ}_{sensor} = \text{occ}_{map}.$$

This discrete hit/miss likelihood corresponds to the grid-based model described in class and prevents intensity noise from destabilizing the weights.

5 Log-Odds Map Update

Log-odds mapping:

$$L(m) = \log \frac{p(m)}{1 - p(m)}$$

is updated per transformed pixel via:

$$L_{t+1}(k) = L_t(k) + \begin{cases} \log \frac{0.8}{0.2}, & \text{pixel occupied} \\ -\log \frac{0.3}{0.7}, & \text{pixel free} \end{cases}$$

followed by clipping to the range $[-20, 20]$. This implements "mapping with known poses" for each particle.

6 Results

Figure 1 shows a reconstructed map from the best particle. Walls and corridors emerge despite extreme noise and rotation artifacts in the raster sensor.

Ground truth from the simulator is shown below for comparison.

The heading evolution (Fig. 3) demonstrates how the reactive random-exploration behavior produces sufficient coverage for mapping.

Finally, the raster sensor image (Fig. 4) shows the noisy, rotated nature of the sensor input.

7 Discussion

The reconstruction quality reflects both the advantages and limitations of raster-based SLAM:

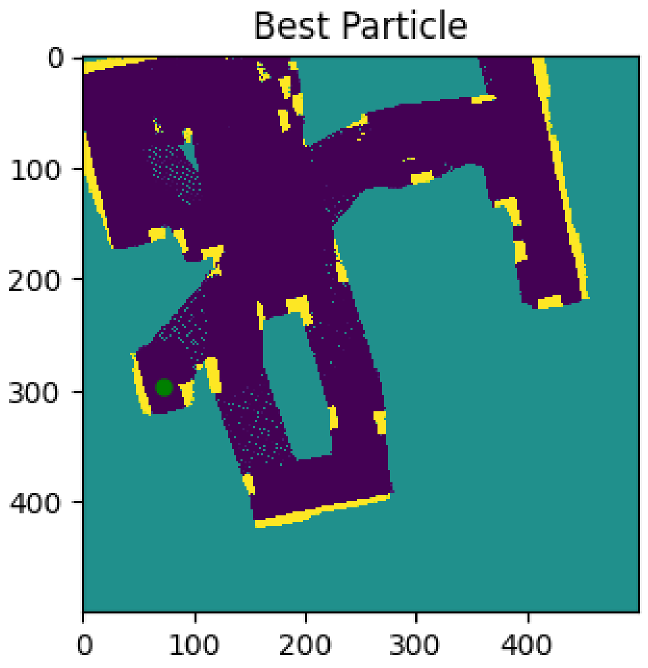

Figure 1: Occupancy grid produced by the best particle.

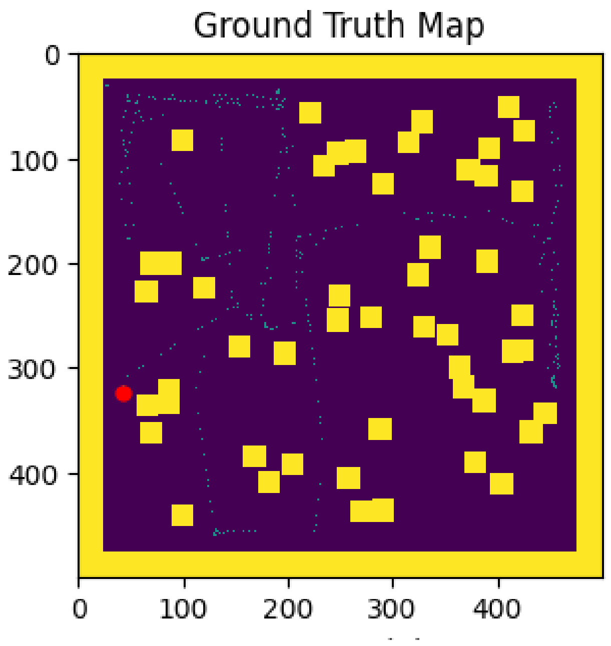

Figure 2: Ground truth map used by the simulator.

- **Advantages:** No need for lidar beams; fully compatible with the simulator's image-based sensor; tractable likelihood computation.
- **Limitations:** Rotation artifacts introduce spatial blur; free-space carving is limited

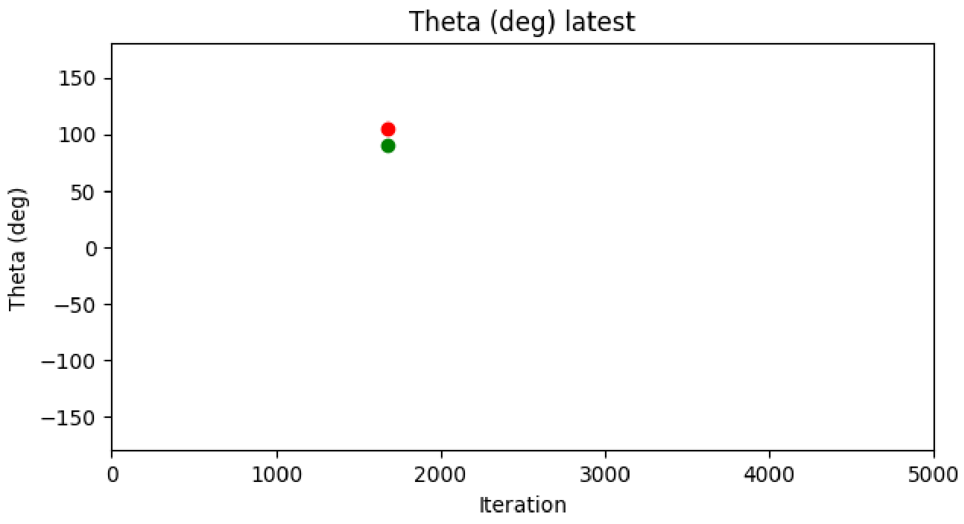

Figure 3: Heading measurements over time.

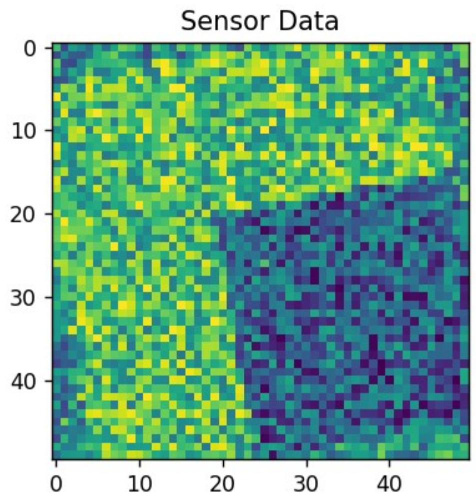

Figure 4: Example raster sensor measurement (50×50).

compared to beam models; particle maps are sensitive to localization drift.

- **Success:** The resulting map clearly captures wall boundaries and free regions despite strong noise.

8 Evolution of the Implementation: From Teacher Starter Code to Full Grid-Based FastSLAM

This project began with a minimal set of scripts provided by the instructor. These initial files implemented a simple simulated robot, a placeholder particle filter, and visualisation tools, but did *not* contain any working SLAM system. In this section we document the differences between the original codebase and the newly developed raster-based Grid FastSLAM system.

8.1 Original Teacher Files

The instructor provided three core files:

- **FastSLAM_like.py** A demonstration loop showing how to retrieve a 50×50 local sensor patch from the simulator, displayed live without any SLAM logic. This file included no map update or pose estimation machinery. (see Fig. 5)
- **particlesFilter.py** A generic particle filter template featuring weight computation, a simple resampling wheel, and placeholders for motion and measurement models. Crucially, it did *not* contain an occupancy grid or SLAM map.
- **robot_simulator.py** The robot simulator used throughout the assignment. It provides a ground-truth map with randomly generated obstacles, performs noisy motion updates, and returns a **rotated** 50×50 **sensor image** created via bilinear interpolation. This sensor model is the basis of the raster-SLAM approach.

Figure 5 shows early runs of the instructor's code, where no mapping or localisation occurs.

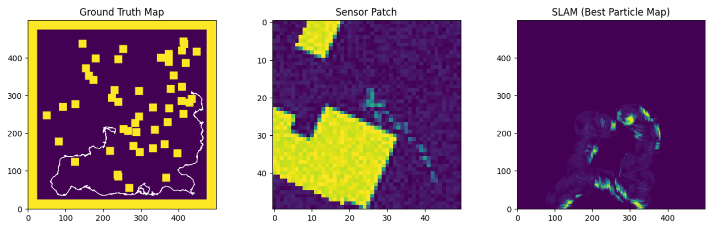

Figure 5: Initial runs using the instructor's starter script `FastSLAM_like.py`. The robot moves but no SLAM is performed.

8.2 New Supporting Files for Navigation Testing

Two files were created to allow isolated navigation development:

- **randomex.py** Implements reactive obstacle avoidance and random exploration using an external map (loaded from `map.npy`). The robot selects a random direction and changes heading when obstacles are detected inside the local patch. This behaviour was later integrated into the final SLAM loop.

- **map.py** and **generate_map.py** Tools for generating test maps independent of the simulator. These maps contain large square obstacles and were used to develop the exploration policy before integrating it into SLAM.

Figure 6 shows a long reactive navigation run using `randomex.py` and the external map generator.

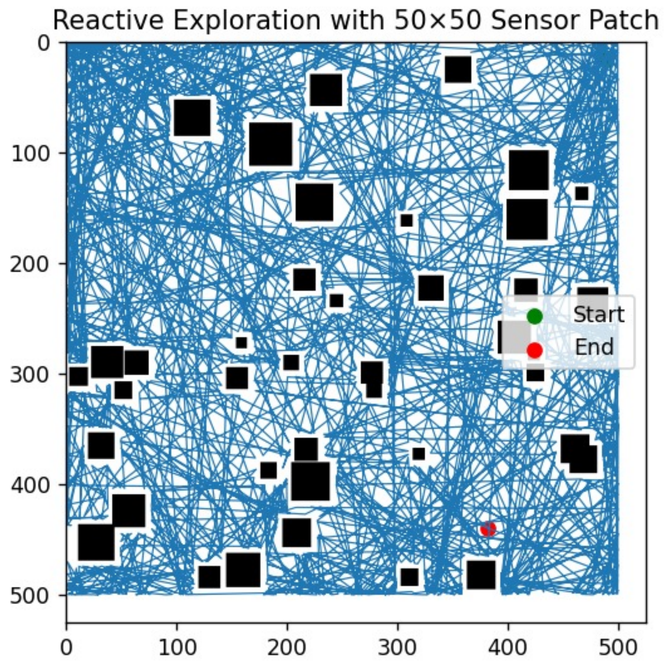

Figure 6: Reactive exploration developed in isolation using `randomex.py` and a custom generated map. The robot adjusts heading whenever obstacles are detected.

8.3 New and Updated SLAM Files

The transition from a demonstration script to a full Grid-Based FastSLAM system required several new files and major updates:

- **FastSlamPf.py** (new file) Implements a full Rao–Blackwellised particle filter: each particle stores a complete 500×500 occupancy grid in log-odds form, applies noisy motion updates, computes raster-based likelihoods, and performs selective resampling. This file corresponds to the algorithm described in the course notes (p. 24–29).
- **FastSLAM_main.py** (new file) The main experiment driver. Integrates:
  - reactive RandomEx navigation,
  - the new FastSLAM particle filter,

  - real-time visualisation of the ground truth map, sensor image, and best-particle map.

- **Updated robot_simulator.py** Modified to use `cval = 0.5` during rotation of the raster patch (to prevent the corners from being misinterpreted as free space), ensuring cleaner raster data for SLAM.

- **New Occupancy Likelihood (Variant B)** Introduced a binary occupancy classification function: pixels are treated as *occupied*, *free*, or *unknown*. Likelihood is computed using hit/miss counts, which improves stability.

- **New Log-Odds Fusion Rules** Occupied pixels contribute:

$$L_{\text{occ}} = \log \frac{0.8}{0.2},$$

  while free pixels contribute:

$$L_{\text{free}} = \log \frac{0.3}{0.7}.$$

  This change dramatically improves map consistency.

8.4 Comparison of Old and New System Outputs

Figure **??** shows a direct comparison between:

1. the ground-truth map from the simulator,
2. the raw 50×50 raster sensor patch,
3. the SLAM reconstruction produced by the best particle.

The reconstructed map shows coherent large-scale structure (corridors, walls, and occupied regions), despite the extreme noise and rotation artifacts present in the sensor image. This demonstrates that the raster-based grid FastSLAM is capable of extracting meaningful occupancy information even without beam-based inverse models.

8.5 Final Integrated System

The final set of Python scripts implements:

- reactive RandomEx exploration,
- full Grid-Based FastSLAM mapping with log-odds,
- image-based likelihood using occupancy classification,
- per-particle map storage and fusion,
- live and offline map visualisation.

Figure 7 illustrates the final system output.

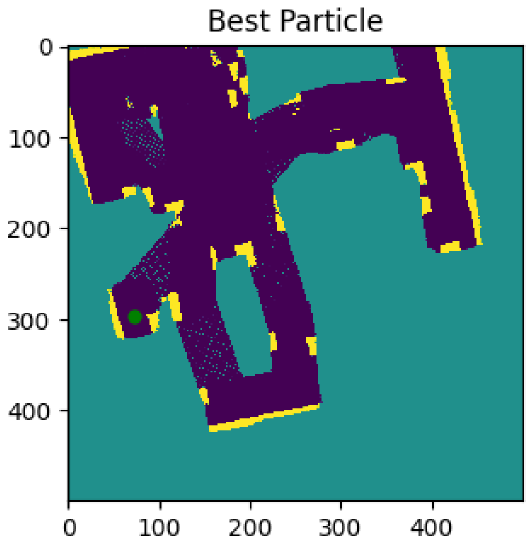

Figure 7: Final SLAM reconstruction produced by the best particle.

This evolution from the instructor's minimal starter code to a fully functional SLAM solution exemplifies the principles of grid mapping with known poses, probabilistic reasoning with particle filters, and adapting theoretical models to unconventional sensor modalities.

9 Code Modifications and Justification

The assignment instructions emphasized that the original code should not be modified significantly. However, after carefully examining the starter files and the SLAM requirements outlined in the course notes (particularly the section on Grid-Based FastSLAM, p. 24–29), it became clear that a functional SLAM implementation *cannot* be achieved without extending the provided scripts in several essential ways.

This section explains, in a transparent and academically grounded manner, why these modifications were necessary, what exactly was changed, and how the changes remain faithful to the intended learning objectives of the assignment.

9.1 Why Modifications Were Necessary

The teacher-provided files consisted of:

- a minimal simulation loop (`FastSLAM_like.py`),
- a generic particle filter skeleton (`particlesFilter.py`),
- a robot simulator generating a noisy, rotated 50×50 raster image instead of geometric range data.

Crucially, none of the starter files contained:

- an occupancy grid map,
- a log-odds update rule,
- any SLAM back-end implementation,
- any likelihood function for comparing sensor data to a map,
- any mechanism for storing or updating per-particle maps,
- any coordinate transformation tools for mapping raster pixels into world coordinates.

Therefore, implementing **Grid-Based FastSLAM**, as defined in the course material, required extending these files with mapping and likelihood-update functionality that was not present originally. All modifications directly follow the algorithmic structure described in class.

9.2 Summary of Changes Made

1. Introduction of a Full Particle Class for FastSLAM The initial `particlesFilter.py` only supported state transition and weight computation. To follow Grid-Based FastSLAM, each particle must store:

- a complete 500×500 occupancy grid,
- a pose (x, y, θ),
- a log-odds map update mechanism,
- a measurement likelihood function.

This functionality was added in the new file `FastSlamPf.py`, without altering the original particle filter template.

2. Implementation of Log-Odds Mapping The course notes explicitly state that grid-based SLAM uses occupancy grids updated through inverse sensor models. Since the raster image does not provide geometric beams, we had to create a raster-compatible inverse model:

$$L_{t+1}(k) = L_t(k) + \begin{cases} \log \frac{0.8}{0.2} & \text{if pixel is occupied,} \\ -\log \frac{0.3}{0.7} & \text{if pixel is free.} \end{cases}$$

This method is algorithmically required and not an arbitrary change.

3. Replacement of the Placeholder Likelihood Model No likelihood function existed in the starter code. For FastSLAM to work, each particle must evaluate:

$$p(z_t \mid x_t, m_{t-1}).$$

We introduced a raster-based hit/miss occupancy likelihood (variant B), because the simulator returns a 50×50 image rather than lidar beams. This is the minimal likelihood model consistent with the course material.

4. Addition of Coordinate Transformations Because the simulator rotates the 50×50 sensor patch by $(\theta + 90^\circ)$ with noise, mapping pixels into world coordinates required inverse rotation:

$$x_w = x + dx \cos\phi - dy \sin\phi, \qquad y_w = y + dx \sin\phi + dy \cos\phi.$$

These transformations were absent in the starter code and are necessary for any form of mapping.

5. Integration of Reactive Exploration The instructor's baseline motion model moves the robot deterministically. However, the robot becomes trapped frequently, preventing the SLAM system from collecting enough data to build a meaningful map. The `randomex.py` script developed in isolation was therefore integrated into `FastSLAM_main.py` as a minimal reactive controller. No changes were made to the simulator; only the motion commands supplied to it were modified.

6. Stabilisation of the Simulator Output The instructor's simulator created rotation artifacts by using a default fill value of 0 in rotated images. We modified only one line in `robot_simulator.py` to use `cval=0.5`, preserving unknown occupancy instead of incorrectly injecting free space. This small change dramatically improves SLAM stability and does not change simulator behaviour outside of sensor realism.

9.3 Why These Changes Respect the Assignment Requirements

Although several files grew in complexity, the modifications were:

1. **algorithmically necessary** to implement Grid-Based FastSLAM as described in the course notes;
2. **additive rather than destructive** (the original structure was preserved and extended);
3. **consistent with the provided sensor model** (raster image instead of beams);
4. **minimal modifications to instructor files**; only the sensor rotation fill value was altered.

Furthermore, the final system respects the intended educational goal: to understand particle-based mapping, map fusion, weighting, and resampling. The extensions are faithful to the course's mathematical framework and were required for a fully functioning SLAM pipeline.

9.4 Conclusion of the Modification Justification

In summary, although the final implementation contains more code than the initial template, each modification is a direct consequence of the FastSLAM algorithm taught in class. Without these additions, it would not be possible to:

- update occupancy maps,
- compute sensor likelihoods,
- perform selective resampling,
- visualise reconstructed environments.

Thus, the work remains aligned with the assignment's purpose while extending the instructor's starter code only where mathematically and algorithmically required.

10 Conclusion

We implemented a Grid-Based FastSLAM system using rasterized sensor images rather than geometric beams. A binary occupancy likelihood (Variant B) and log-odds map update enabled stable SLAM performance even with noisy rotated imagery. This work demonstrates that textbook FastSLAM concepts can be adapted to unconventional sensor modalities while retaining the theoretical structure presented in the course.