Project: Probabilistic Robot Localization in a 2D Environment

The goal of this project is to implement a simulation of a **mobile robot equipped with a range sensor**, moving in a 2D environment containing obstacles. The robot will maintain a **belief grid of its own pose** (position and orientation) and track itself in the environment using probabilistic methods. Primarily we will implement and compare **probabilistic state estimation using histogram filtering** and **particle filtering during the environment interaction**.

This is a group project with a group of 2 or 3 students. Each group will complete the partially provided architecture by implementing:

1. The **robot motion model** for both the true pose (Chapter 5) and belief prediction (Chapter 4).
2. A **range sensor model** that simulates noisy measurements (Chapter 6).
3. A **control loop** to move the robot and update its belief.
4. A **visualization system** to show the robot's true pose and estimated belief.

Use of AI-code generation tools is perfectly fine as long as you can explain each part of the code and correlate it with the correct section of the book if and when required

Simulation Setup

We will use the following setup to simulate the overall system

- The environment is a 2D rectangular map of size (W x H).
- The map contains multiple **rectangular and circular obstacles**.
- The **Environment** class maintains the **true pose** of the robot and obstacle geometry.
- The **Robot** class contains the robot's **internal belief** (occupancy grid-based map).
- The robot moves according using a single **control command** (i.e. move) and uses a **range sensor** to perceive the environment.
- The robot's belief is updated using **prediction (motion) and measurement (sensor) updates**.

Key Features

Following are the primary features of this simulation

- **Environment**:
  - Consists of the true robot pose and a list of rectangular and circular objects.
  - Provides ray-casting based distance measurements to obstacles.
  - Provides functionality of moving simulation at a given time rate.
- **Robot**:
  - Only maintains a occupancy grid based map and a belief about the actual pose.
  - Executes motion commands and updates belief accordingly.
  - Belief Updated using motion prediction and sensor likelihood from range sensor measurements.
  - **Sensor Model**:
    - Simulate range sensor (laser or sonar) with configurable **FOV**, **number of beams**, **max range**, and **Gaussian noise**.
    - Sensor readings obtained from ray casting against environment obstacles.
- **Control Loop**:
  - Sends motion commands to robot.
  - Invokes prediction and update steps of belief.
  - Generates visualization frames.
- **Visualization**:
  - Shows the environment with obstacles.
  - Shows true robot pose and heading.
  - Displays belief heat-map.

Requirements & Constraints

1. **Programming Language:** Python 3.x
2. **Libraries Allowed:** numpy, matplotlib, optionally scipy for probability computations. No external robotics simulators (e.g., ROS, PyBullet).
3. **Environment**
   - Only store obstacles as objects in a list.
   - **Do not use pre-built distance fields**; sensor distances must come from **ray-casting**.
4. **Belief Grid**
   - Must represent x, y, θ.
   - Support multi-resolution grids (coarse + fine).
5. **Sensor Model**
   - Must simulate **beam-based measurements**.
   - Include configurable **FOV, max range, number of beams, and noise**.

Evaluation Criteria

Component	Weight	Criteria
Environment	20%	Correct obstacle representation, ray-casting measurements.
Robot Motion & Belief	30%	Correct motion model, prediction and update of belief grid.
Range Sensor	20%	Accurate ray-casting, configurable parameters, noise simulation.
Control Loop & Integration	20%	Motion + sensing + belief update executed correctly.
Visualization	10%	Clear display of obstacles, robot true pose, and belief.

Submission

- Python code files (.py).
- Multiple **demonstration maps** showing robot localization.
- Include a detailed **report** explaining:
    - Design choices.
    - Sensor and motion models.
    - Multi-resolution belief implementation.
    - Screenshots of robot trajectory and belief heat-maps.