Project Specification

Landmark Detection & Tracking (SLAM)

robot_class.py: Implementation of sense

Criteria	Meets Specifications
Implement the sense function for the robot class.	Implement the sense function to complete the robot class found in the robot_class.py file. This implementation should account for a given amount of measurement_noise and the measurement_range of the robot. This function should return a list of values that reflect the measured distance (dx, dy) between the robot's position and any landmarks it sees. One item in the returned list has the format: [landmark_index, dx, dy].

Notebook 3: Implementation of initialize_constraints

Criteria	Meets Specifications
Initialize constraint matrices.	Initialize the array omega and vector xi such that any unknown values are 0 the size of these should vary with the given world_size, num_landmarks, and time step, N, parameters.

Notebook 3: Implementation of slam

Criteria	Meets Specifications
Iterate through the generated data and update the constraints as you read in sensor measurement data.	The values in the constraint matrices should be affected by sensor measurements *and* these updates should account for uncertainty in sensing.
Update the constraint matrices as you read robot motion data.	The values in the constraint matrices should be affected by motion (dx, dy) *and* these updates should account for uncertainty in motion.
slam returns a list of robot and landmark positions, mu.	The values in mu will be the x, y positions of the robot over time and the estimated locations of landmarks in the world. mu is calculated with the constraint matrices omega^(-1)*xi.

Answer question about the final robot pose.	Compare the slam-estimated and *true* final pose of the robot; answer why these values might be different.
slam passes all provided tests.	There are two provided test_data cases, test your implementation of slam on them and see if the result matches.

Suggestions to Make Your Project Stand Out!

- Create a new version of slam in which omega only keeps track of the latest robot pose (you do not need all of them to implement slam correctly).
- Add visualization code that creates a more realistic-looking display world
- Create a non-random maze of landmarks and see how your implementation of slam performs
- Display your robot world at every time step and stack these image frames to create a short video clip and to see how the robot localizes itself and builds up a model of the world over time
- Take a look at an implementation of slam that uses reinforcement learning and probabilistic motion models, at this Github link