3.4 Particle Filter

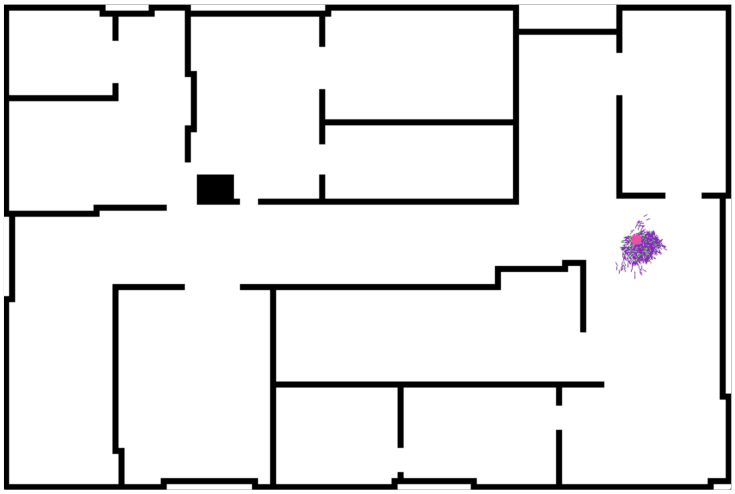


Figure 26: The Particle Filter in action.

Key Concepts

To statistically determine the location of the robot while traversing the map.

Implementation

The implemented Particle Filter makes use of the data from the lidar sensor to estimate the position of the robot. To get a reliable estimation of the robot in the map a *Vision Map* is initially created. The Vision Map consists of a 360° lidar view from all valid positions on the map at a lower angular resolution than the real robot.

The robot uses 200 lidar rays for a field of view of 260° and an angular resolution of 1.3°. The Vision Map would then require $360°/260° \cdot 200 = 276{,}9$. This is not ideal as it would introduce a rounding error, which can be remedied by choosing a more optimal number of rays. A small subset of the available lidar data is applied and therefore fewer rays are used. This will in turn reduce memory usage and time complexity as later comparisons between lidar data and lidar vision will require less time. A subset of 39 rays is chosen, since this will result in a whole number of rays to describe the entire view; $360°/260° \cdot 39 = 54$ resulting in an angular resolution of 6,66°.

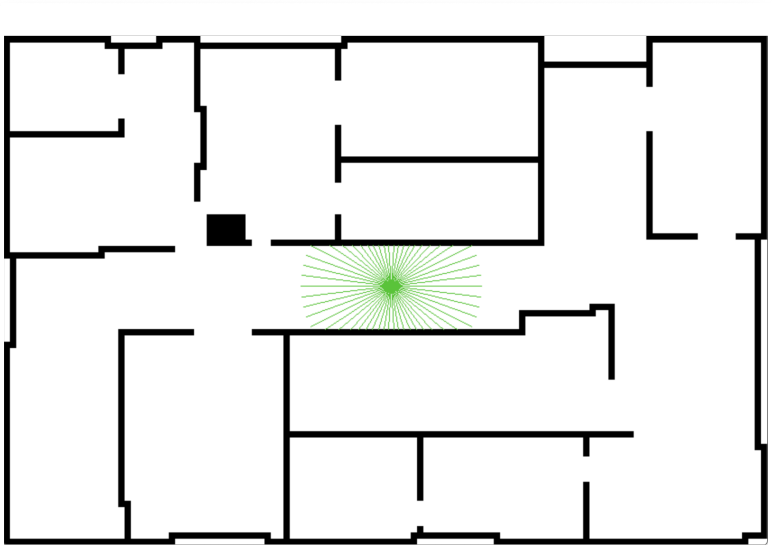


Figure 27: Lidar Vision for a single point on the map with 54 rays spanning 360°.

To have a well-working particle filter, we need some particles. Each particle has some attributes; an x,y-coordinate, a weight, an `indexOfRightmostRay` and an orientation. The `indexOfRightmostRay` is used to determine the orientation of the particle. The index can be any number from 0 to the number of vision rays, here 54. This will determine which lidar ray the rightmost ray translates to and therefore the subsequent rays as well. For a down-scaled example see figure 28. This way of indexing particles with different orientations heavily reduces memory costs as there is no need to generate all the different lidar data that exists at different orientations. One lidar *Vision* is simply created at each point on the map and any given particle's lidar data is solely dependant on the `indexOfRightmostRay`.

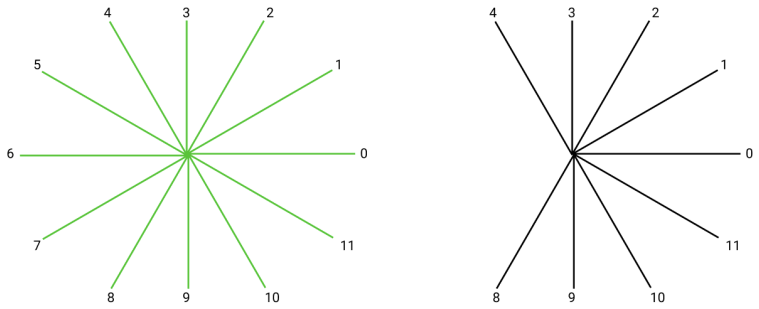


Figure 28: *Left*: Lidar Vision for a single point on the map. *Right*: Lidar data for a particle at set point with `indexOfRightmostRay` being 8.

At initialization an initial set of particles with random orientations, which cover the entire map, are generated. The weight of each particle with its given orientation is determined by comparing the lidar data of the particle with the lidar data from the robot. The weight is the sum of the distances, where each distance is weighted based on a Gaussian distribution. The closer the two data values are to each other the higher the weight (see figure 29).

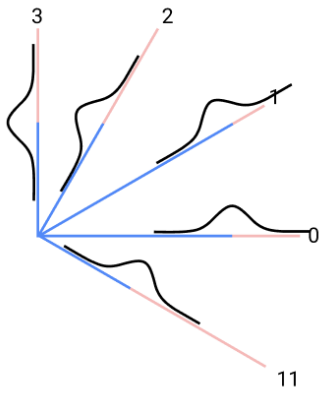
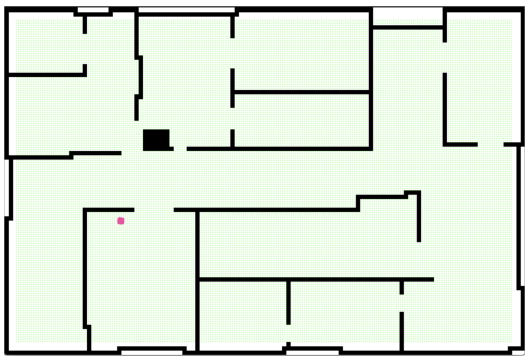


Figure 29: Lidar data for a particle (pink) compared to lidar data from the robot (blue).

From this initial set of particles a fixed number (e.g. 1000, 2000) of new particles are resampled each time step. Resampling happens by choosing at random from a weighted distribution; so a particle with a larger weight compared to other particles has a higher probability of being a resampled more times.

Since many particles have a probability of spawning in the same position on the map their states are updated based on a model describing the motion of the robot. This is done to avoid sample impoverishment.

If the particles drift away from the robot or the robot were to 'teleport' across the map an extra functionality has been implemented to handle this problem. This is done by looking at the summed total of weights, η, of all the particles. If η drops below a certain lower bound the initial map-covering particles are generated anew to relocate the robot (see figure 30).



(a) Particles covering the entire map.

(b) Resampled 1000 particles with highest probability of same location as robot.

(c) Particles cluster at locations of highest probability.

(d) Particles settle on position where the robot is located.

Figure 30: Relocating the robot.