Workflow:

1. Initialize particles across the map in grid
2. Take in current robot data
3. Process to find closest occupancy
4. Cull particles for those that fall below a threshold, weight the survivors
5. Move the robot
6. Update the surviving particles positions. If any moved through an obstacle or ended out of bounds, cull.
7. Disperse new particles around surviving particles. Number around the particles based on weight and keeping below a certain number of particles to keep computation down.
8. Repeat steps 2 through 7

Questions:

How much will we implement to rviz? Just the best hypothesis?
Can we do a top down 2D map with matplotlib like Paul's test code?

Particle Manager

Attributes

Current_Particles - list of hypotheses currently being evaluated

Max_Particles - greatest number of particles on map at any given time. Set too low and it will take very long to converge. Set too high and computation time will take forever

Threshold/Percent keep: A threshold for when to keep a particle given how far from the readings that particle is. **Or** the top percent of particles with maximum likelihood.

Init Particles -

Most likely called in the initializer. Will be used to initialize the particles given a map (or will initialize them spread around a point)

Input:

Map - map of the area the robot is attempting to localize in
Offset - *optional* - move initialized particles by some offset based on robot movement

Output:

None → Modifies attributes

Get Particles -

Return the current particles we have

Input:

None

Output:

List of particles with weights, directions, and x,y position

Add particles -

Create additional particles to begin next iteration of sensor intake. New positions based on when called remaining particle's weights

Input:

None - uses object attributes

Output:

None - updates object attribute Current_Particles with new particles

Delete Particles -

Given a set of particles, a transformation and LIDAR scan, delete extra particles

Input:

Transformation →An object containing the amount turnt and the distance traveled
LIDAR → LIDAR scans after the movement

Output:

None → Will modify the object attributes

Update Particles -

Given a set of movements and turns, update the particles' positions

Input:

Particle - the particle to be modifying position.
Map - map to compare ensure new position of particle is valid
DeltaYaw - Change in the actual robot's heading
DeltaLinear - Change in the actual robot's position

Output:

None → Modifies particle in Current_Particles

Update Weights -

Called by delete Particles. Given a set of LIDAR measurements, update the particles' likelihoods. Particle's who are out of bounds get set to 0.
Uses Distance to Nearest Object from Occupancy Field

Input:

None

Output:

None → Will modify the object attributes

Particle

Attributes

X - particle's x position on the map
Y - particles y position on the map
Yaw - particle's heading
Weight - particle's likelihood to be the robot's location. Between 0 and 1

Init -

Input:

X → X position
Y → Y position
Yaw → Heading

Output:

Particle Object

Helper Functions

Angle_normalize

Map angle to range [-pi,pi]

Angle_diff

Find difference between two angles while dealing with angle wrapping

Normalize

Deal with converting weights consistently

Get_closest_obstacle_distance

From the Occupancy Field class, finds the closest occupancy. Will run each particle through this class for culling