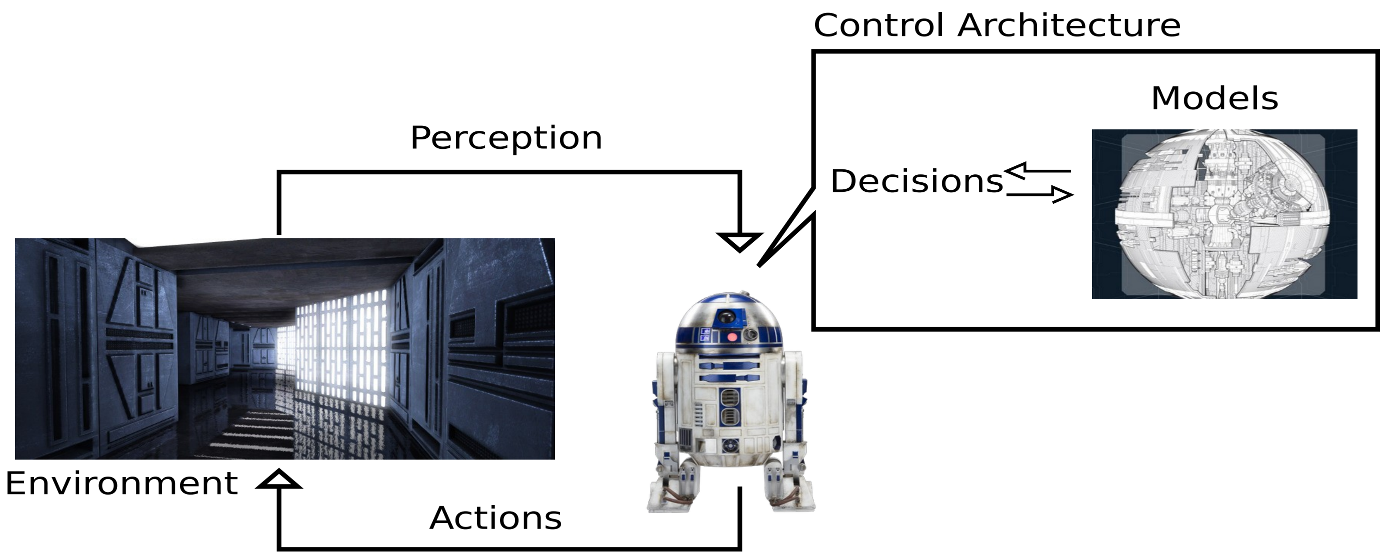
MOBILE ROBOTICS

INTRODUCTION

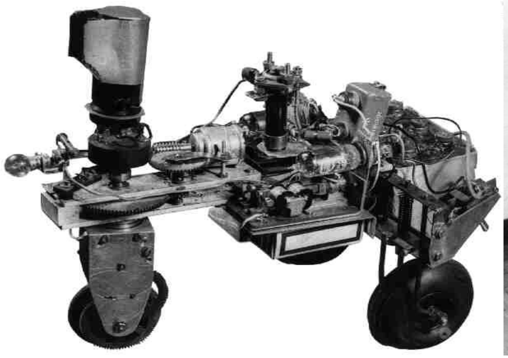
Introduction

- “ A robot is a machine with perception, decision and action abilities. It allows the machine to act autonomously in its environments.”

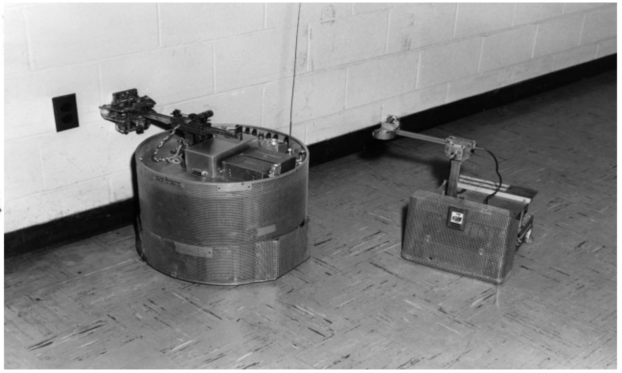


Quick History

- “Rossum’s Universal Robots” - 1920, a Karel Capek play



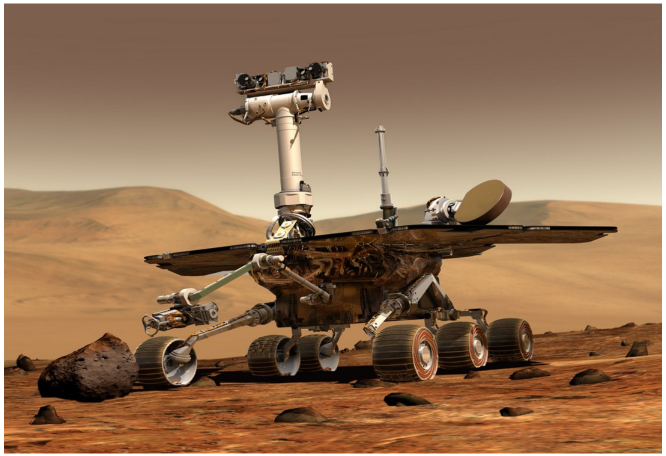
1950 - Grey Walter
“Turtle” robot



1960 - John Hopkins “Beast”
robot

1970 – Stanford
“Shakey” robot

Nowadays

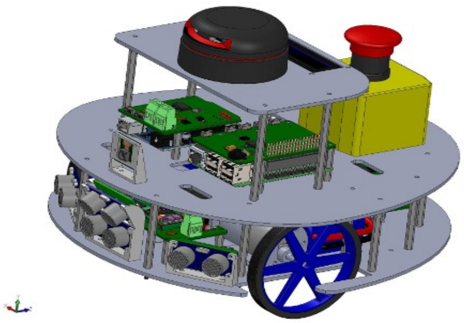


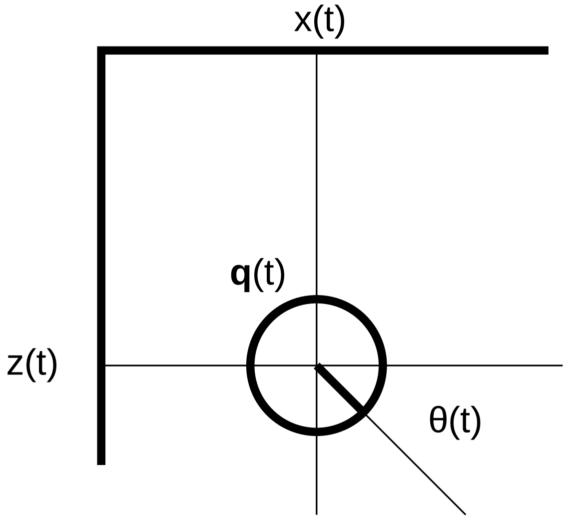
The considered robot

- UGV : Unmanned Ground Vehicle
  - UAV : Unmanned Aerial Vehicle
  - UUV : Unmanned Underwater Vehicle
- Robot pose

$$\boldsymbol{q}(t) = (\boldsymbol{x}(t), \theta(t))^T$$

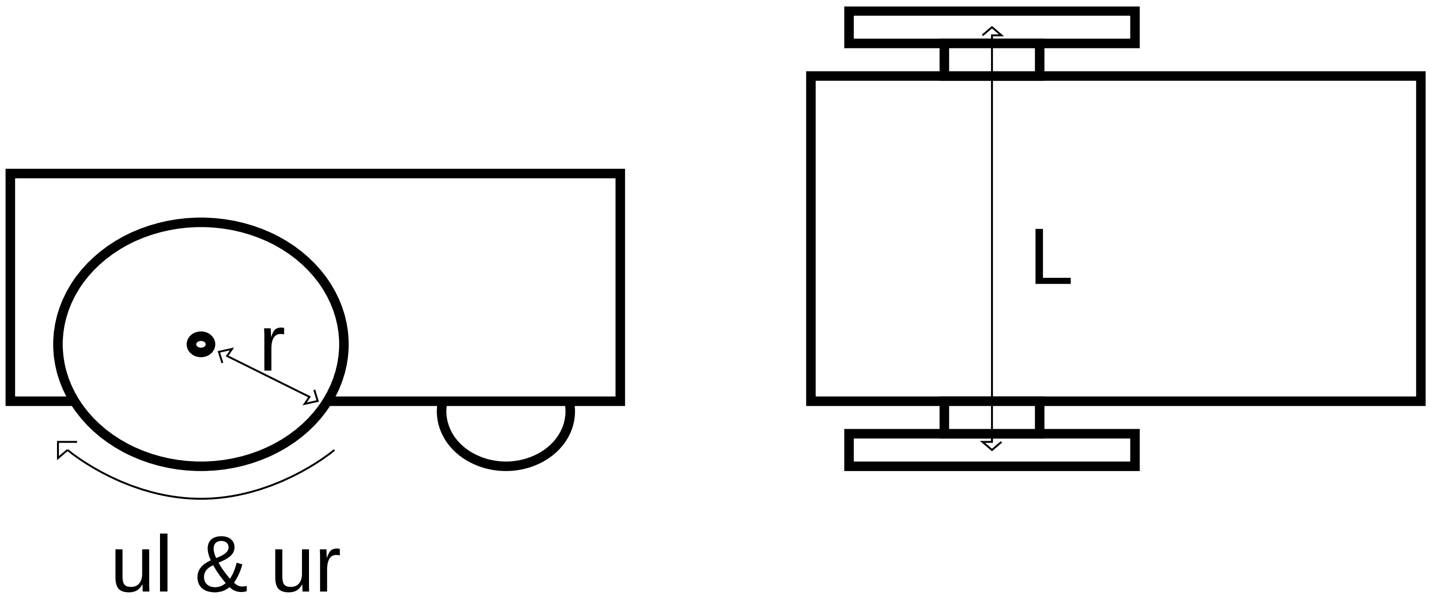
$$\boldsymbol{x}(t) = (x(t), z(t))^T$$





The considered robot

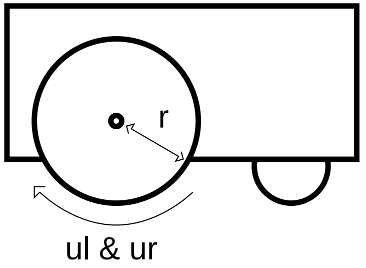
- Differential robot (nonholonomic robot)

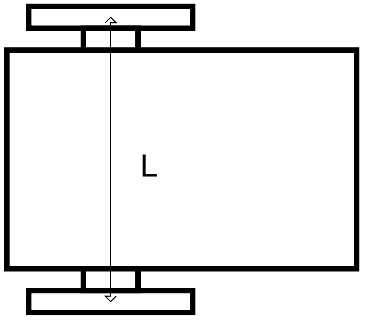


The considered robot

- The differential equations

$$\dot{x}(t)=\frac{r}{2}.(ul(t)-ur(t)).\cos(\theta(t))$$
$$\dot{z}(t)=\frac{r}{2}.(ul(t)-ur(t)).\sin(\theta(t))$$
$$\dot{\theta}(t)=\frac{r}{L}.(ul(t)-ur(t))$$





r : radius of the wheels (m)

L : distance between the wheels (m)

ul : angular speed of left wheel (rad/s)

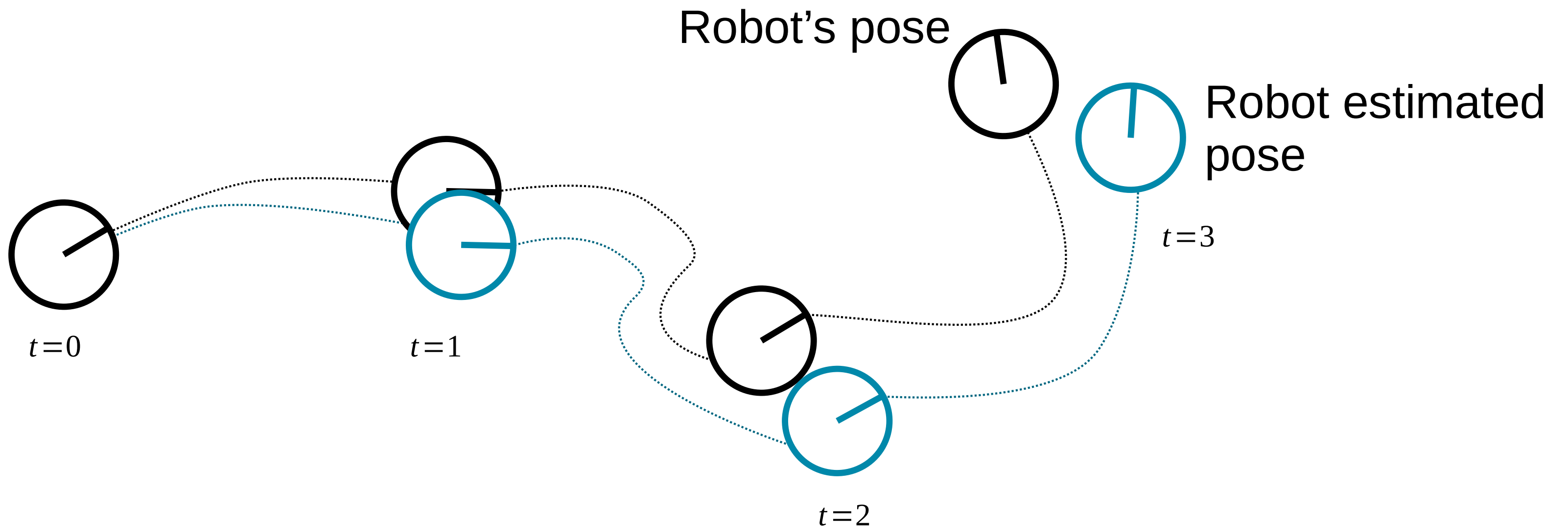
ur : angular speed of the right wheel (rad/s)

MOBILE ROBOTICS

LOCALIZATION PROBLEM

Localization problems

- Pose tracking



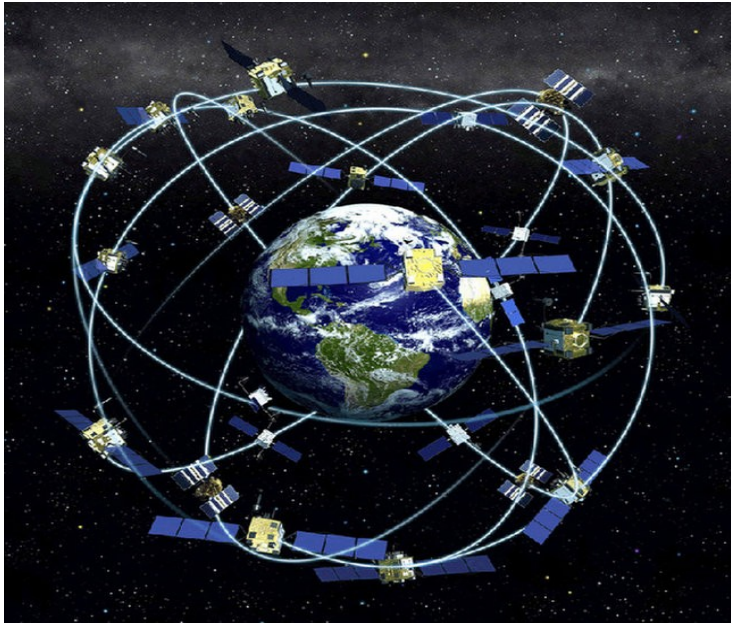
The robot is drifting!

Environments

- Environment types
  - Indoor vs outdoor
  - Static vs dynamic
- Environment representations
  - Topological
  - Metric
  - Hybrid

Environments

- Metric map
  - Landmark map
  - Occupancy grid

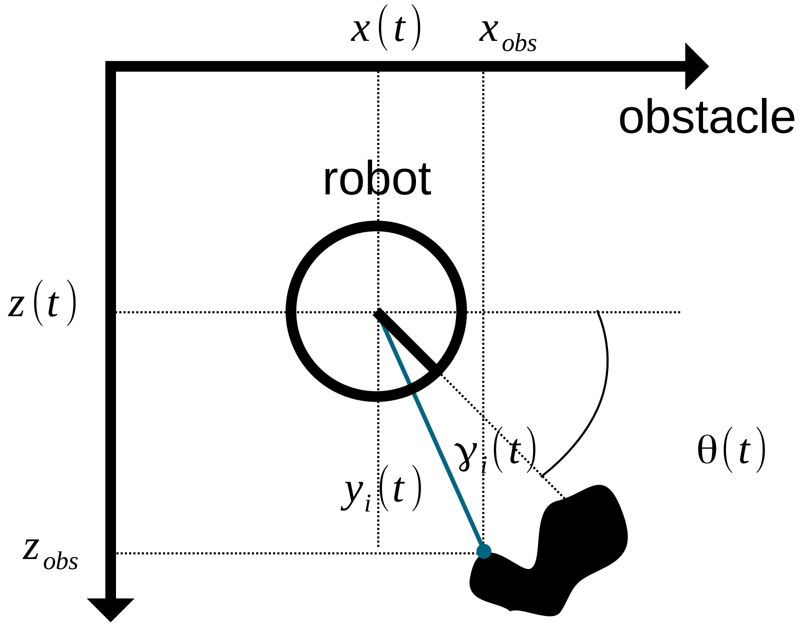


Sensors

- Proprioceptive vs Exteroceptive
- Odometry
- Inertial sensors
- Distance sensors
  - US
  - LiDAR (Light Detection And Ranging)
- Camera
- ...

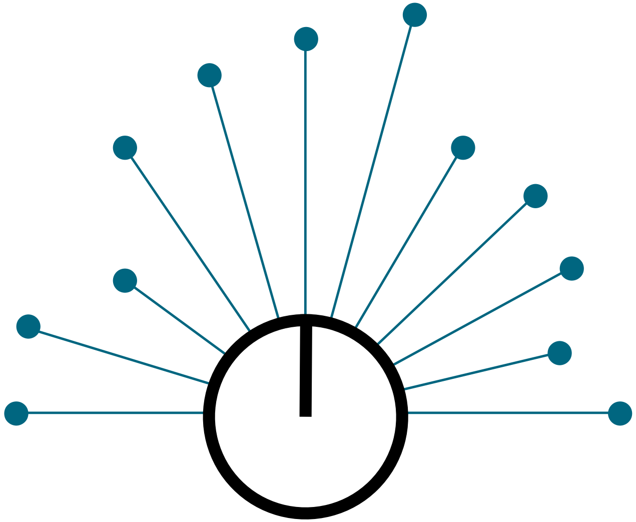
LiDAR sensor

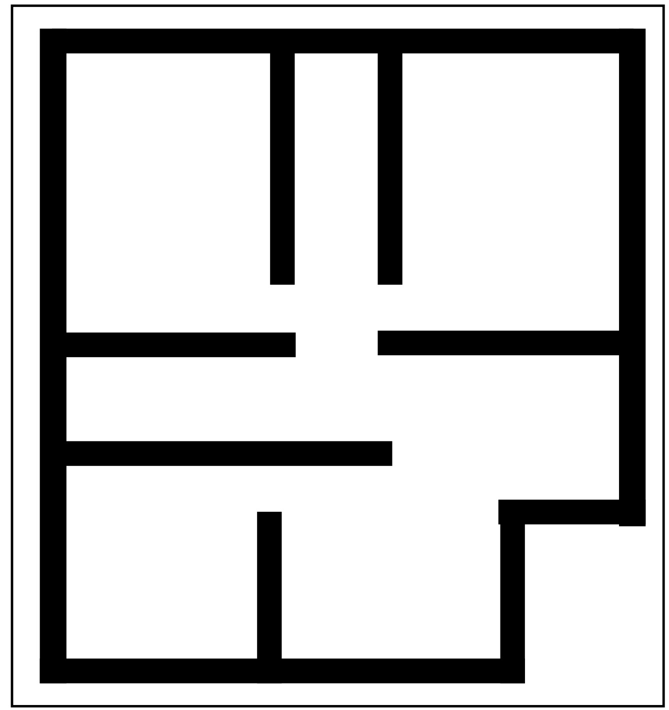
- Distance and range



Global localization

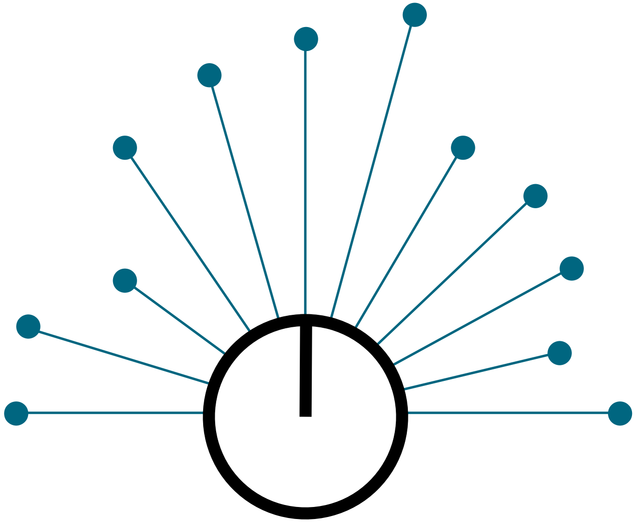
Objective

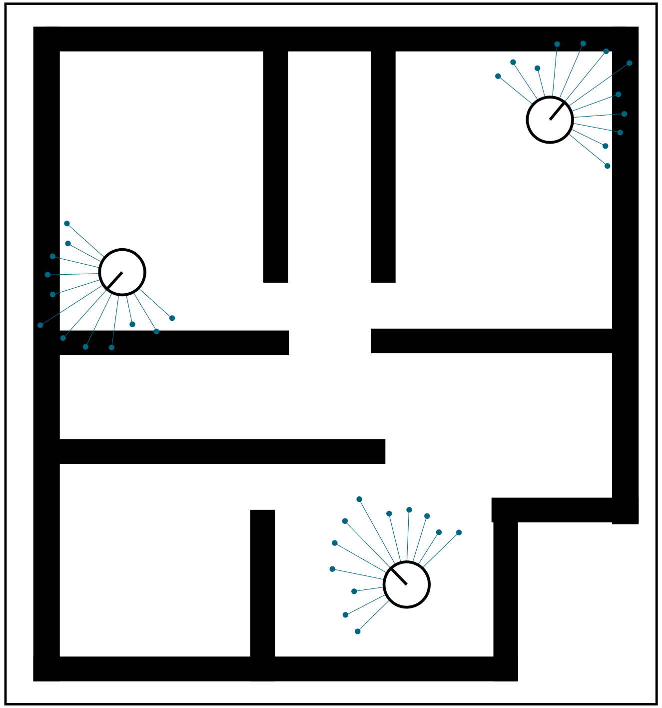




Global localization

Objective





Monte Carlo Localization

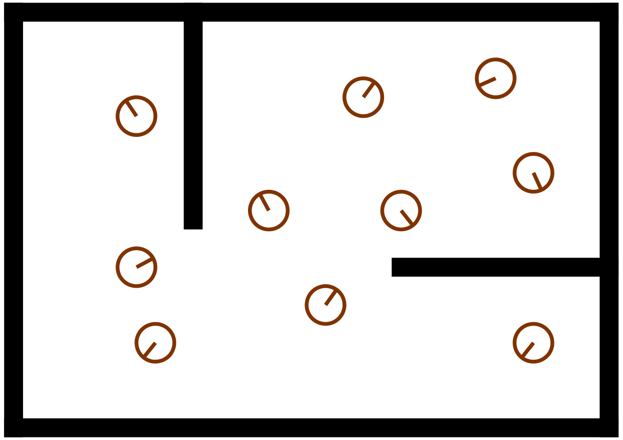
- Belief of the robot by a set of particles

$$X_t = \{ x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]} \}$$

- A particle has a pose and a weight

$$x_t^{[m]} = \langle \boldsymbol{q}_{t,m}, w_{t,m} \rangle$$

- Steps:
  - [] Initialization
  - Evaluation
  - Re-sampling
  - Process Odometry



Monte Carlo Localization

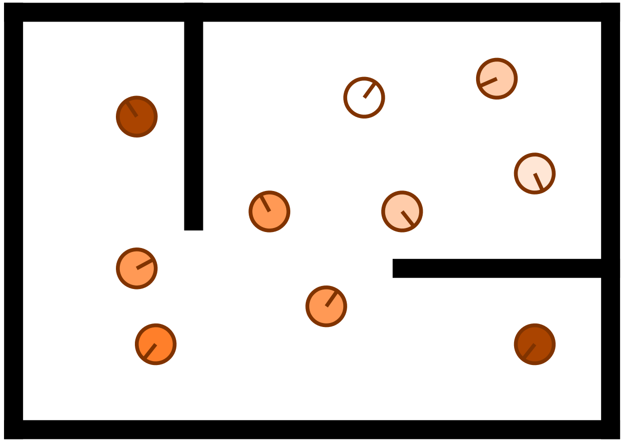
- Belief of the robot by a set of particles

$$X_t = \{ x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]} \}$$

- A particle has a pose and a weight

$$x_t^{[m]} = \langle \boldsymbol{q}_{t,m}, w_{t,m} \rangle$$

- Steps:
  - Initialization
  - [] Evaluation
  - Re-sampling
  - Process Odometry



Monte Carlo Localization

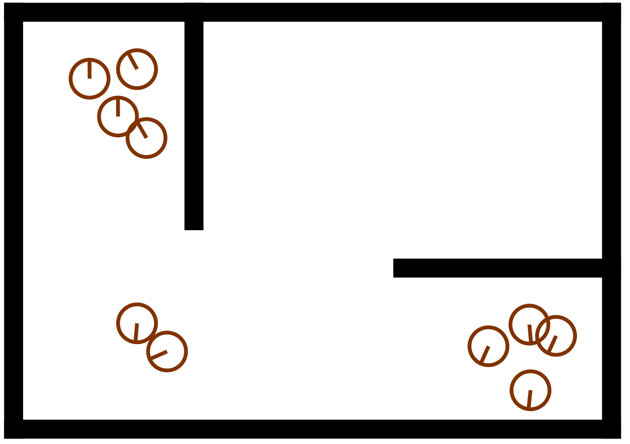
- Belief of the robot by a set of particles

$$X_t = \{ x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]} \}$$

- A particle has a pose and a weight

$$x_t^{[m]} = \langle \boldsymbol{q_{t,m}}, w_{t,m} \rangle$$

- Steps:
  - Initialization
  - Evaluation
  - ▯ Re-sampling
  - Process Odometry



Monte Carlo Localization

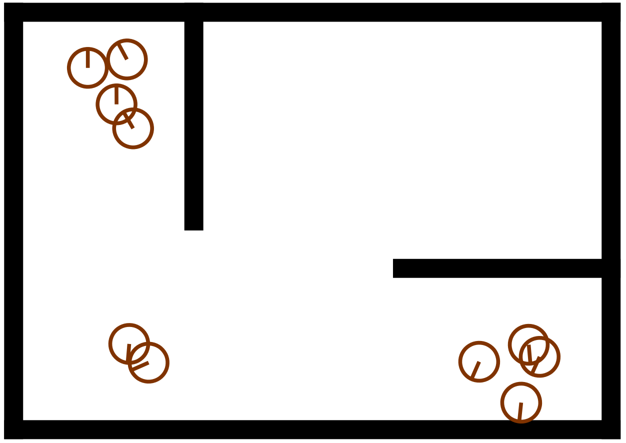
- Belief of the robot by a set of particles

$$X_t = \{ x_t^{[1]}, x_t^{[2]}, \dots, x_t^{[M]} \}$$

- A particle has a pose and a weight

$$x_t^{[m]} = \langle \boldsymbol{q}_{\boldsymbol{t},\boldsymbol{m}}, w_{t,m} \rangle$$

- Steps:
  - Initialization
  - Evaluation
  - Re-sampling
  - [] Process Odometry



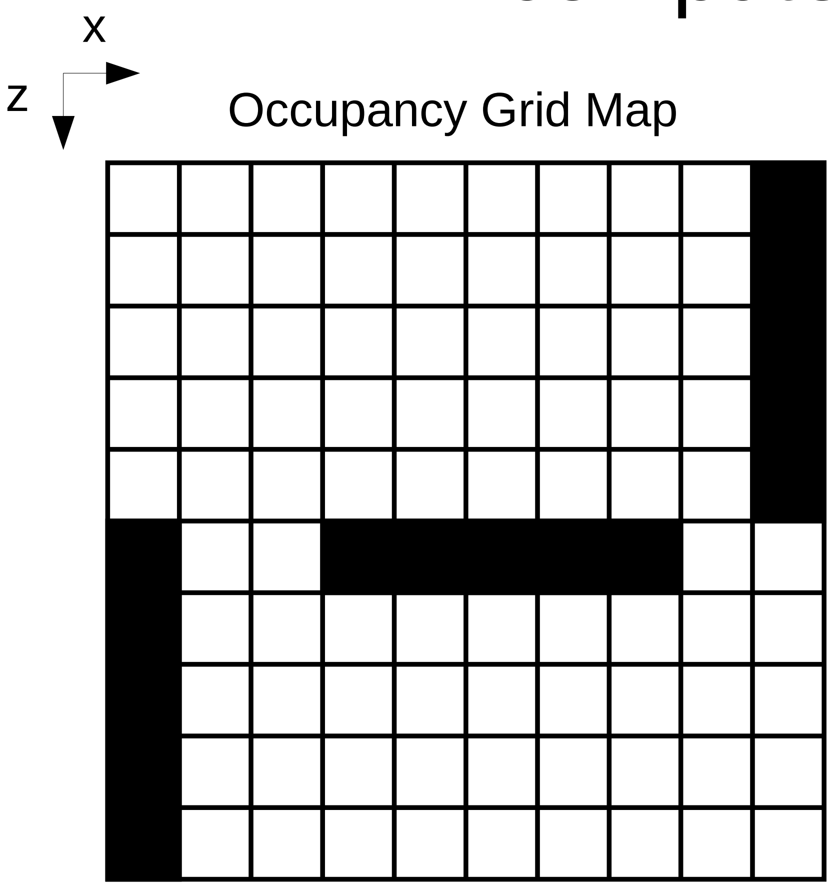
Work to do

- Files to upload in Moodle
  - tp_mcl/monte_carlo.py
  - tp_mcl/cost_map.py
- Warnings
  - Test your code functions after functions, you can use the "Test" button as you want
  - Do not modify the other files
    - You may however modify the btn_test_event() function in tp_mcl/simulator.py for your tests
  - Do not add any library (numpy for instance...)
- You should use run_mcl_tests to run unit tests

tp_mcl/cost_map.py

1. compute_cost_map()

Occupancy Grid Map

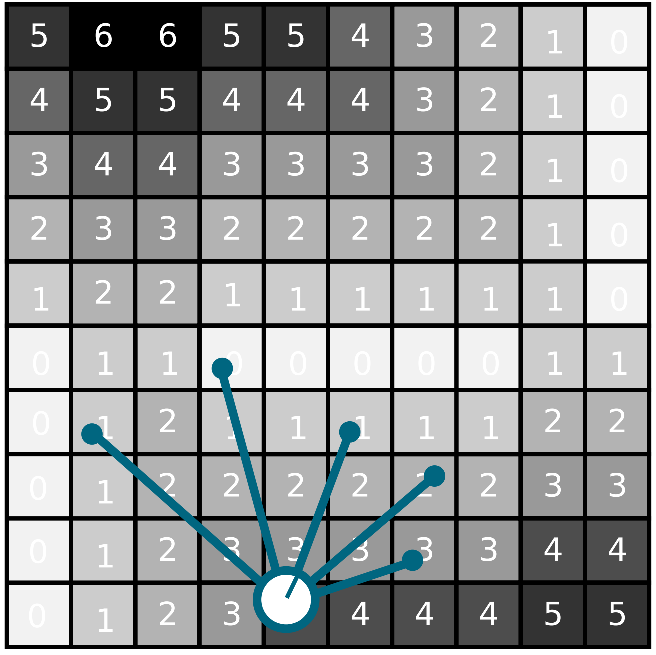


Cost Map

5	6	6	5	5	4	3	2	1	0
4	5	5	4	4	4	3	2	1	0
3	4	4	3	3	3	3	2	1	0
2	3	3	2	2	2	2	2	1	0
1	2	2	1	1	1	1	1	1	0
0	1	1	0	0	0	0	0	1	1
0	1	2	1	1	1	1	1	2	2
0	1	2	2	2	2	2	2	3	3
0	1	2	3	3	3	3	3	4	4
0	1	2	3	4	4	4	4	5	5

1. compute_cost_map()

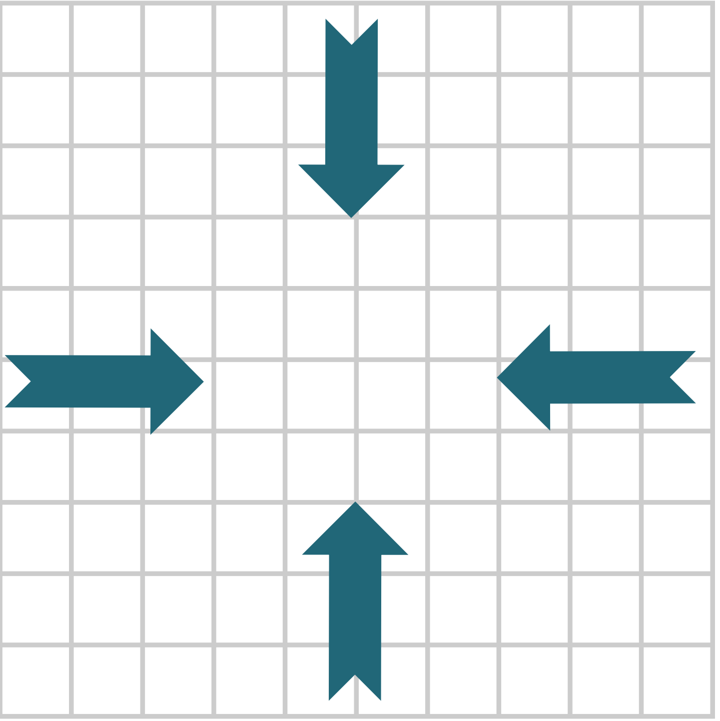
- Ease the evaluation of a particle weight



Cost = 1+0+1+2+3 = 7

1. compute_cost_map()

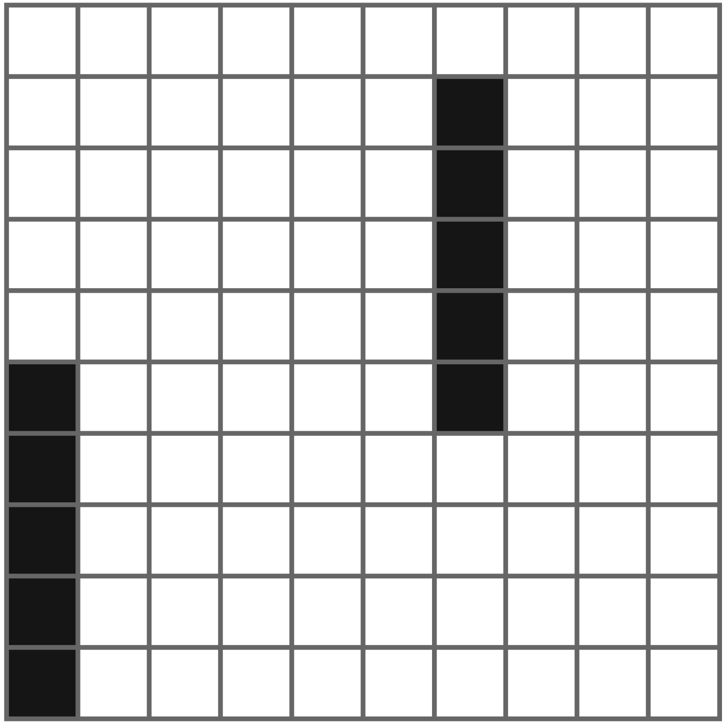
- Four loops over the grid:
  - West to East
  - East to West
  - South to North
  - North to South
- Take the min of the current value and the previous one + 1



- Update max_cost, it is used to display the cost map

1. compute_cost_map()

- Example:
  - Environment



tp_mcl/cost_map.py

1. compute_cost_map()

- Example:
  - Environment
  - Initialization

∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	0	∞	∞	∞
∞	∞	∞	∞	∞	∞	0	∞	∞	∞
∞	∞	∞	∞	∞	∞	0	∞	∞	∞
∞	∞	∞	∞	∞	∞	0	∞	∞	∞
0	∞	∞	∞	∞	∞	0	∞	∞	∞
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	∞	∞	∞	∞	∞	∞	∞	∞	∞

1. compute_cost_map()

- Example:
  - Environment
  - Initialization
  - W-E

∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	0	1	2	3
∞	∞	∞	∞	∞	∞	0	1	2	3
∞	∞	∞	∞	∞	∞	0	1	2	3
∞	∞	∞	∞	∞	∞	0	1	2	3
0	1	2	3	4	5	0	1	2	3
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

tp_mcl/cost_map.py

1. compute_cost_map()

- Example:
  - Environment
  - Initialization
  - W-E
  - E-W

∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
6	5	4	3	2	1	0	1	2	3
6	5	4	3	2	1	0	1	2	3
6	5	4	3	2	1	0	1	2	3
6	5	4	3	2	1	0	1	2	3
0	1	2	3	2	1	0	1	2	3
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

1. compute_cost_map()

- Example:
  - Environment
  - Initialization
  - W-E
  - E-W
  - S-N

5	6	5	4	3	2	1	2	3	4
4	5	4	3	2	1	0	1	2	3
3	4	4	3	2	1	0	1	2	3
2	3	4	3	2	1	0	1	2	3
1	2	3	3	2	1	0	1	2	3
0	1	2	3	2	1	0	1	2	3
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

tp_mcl/cost_map.py

1. compute_cost_map()

- Example:
  - Environment
  - Initialization
  - W-E
  - E-W
  - S-N
  - N-S

5	6	5	4	3	2	1	2	3	4
4	5	4	3	2	1	0	1	2	3
3	4	4	3	2	1	0	1	2	3
2	3	4	3	2	1	0	1	2	3
1	2	3	3	2	1	0	1	2	3
0	1	2	3	2	1	0	1	2	3
0	1	2	3	3	2	1	2	3	4
0	1	2	3	4	3	2	3	4	5
0	1	2	3	4	4	3	4	5	6
0	1	2	3	4	5	4	5	6	7

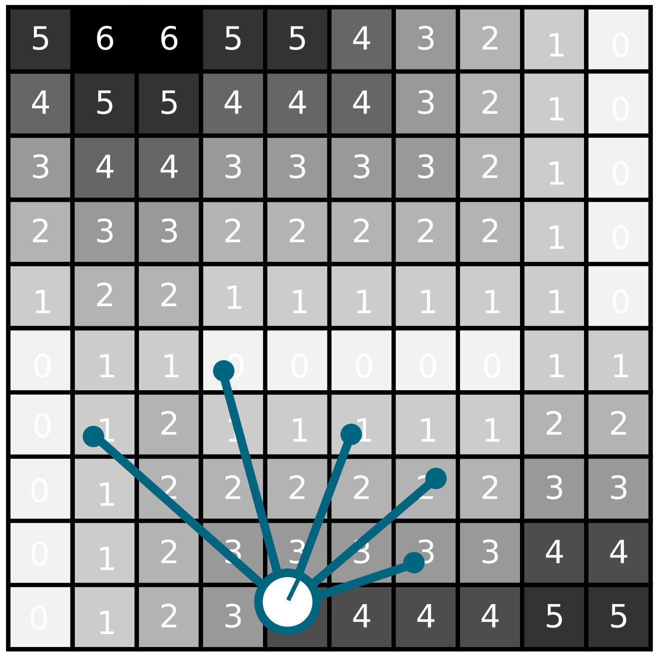
1. compute_cost_map()

- Remark :
  - max_cost needs to be set to be able to display the costmap!

tp_mcl/cost_map.py

2. evaluate_cost()

- Evaluate the cost of a measurement set according to a pose (x, z, theta)
- Each measurement **m** corresponds to a LiDARMeasurement object
  - m.distance
  - m.angle



Cost = 1+0+1+2+3 = 7

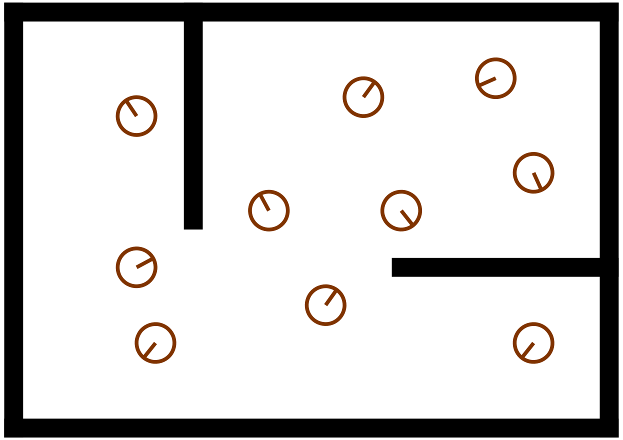
2. evaluate_cost()

- Remarks:
  - When computing the position of an obstacle in the costmap you should test if the obstacle is in the costmap…
  - If the detected obstacle is outside the environment (i.e. the costmap) the cost of it should be **maxcost**
  - When displaying the costmap, infinity values are drawn in yellow color (for debugging purpose)

3. init_particles()

- Initialization of each particles (M particles in total):

$$x_0^{[m]} = \langle \boldsymbol{q_{0,m}}, w_{0,m} \rangle$$
$$\boldsymbol{q_{0,m}} = (x_{0,m}, z_{0,m}, \theta_{0,m})$$
$$x_{0,m} \sim U(0, width)$$
$$z_{0,m} \sim U(0, height)$$
$$\theta_{0,m} \sim U(0, 2.\pi)$$
$$w_{0,m} = 1/M$$



- A particle must be in an empty cell (obstacle free)
- Update the max_weight, used to display the particles

4. evaluate_particles()

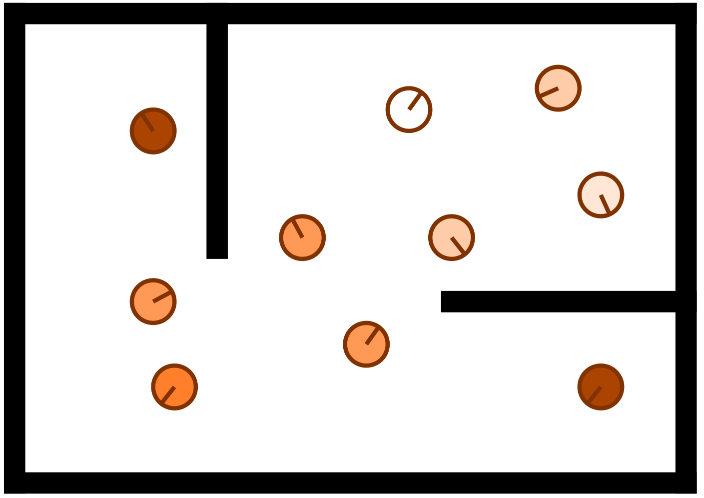
- Evaluate the cost for each particle (using the costmap and the evaluate_cost function)
  - Cost => sum of distances
  - Weight => value between 0 and 1
  - The higher the cost the lower the weight
- Normalize the particle weights

$$\forall x_t^{[m]}, w_{t,m} \in [0,1]$$

- The sum of the weights must be 1

$$\sum w_{t,m} = 1$$

- Update max_weight and bestParticle

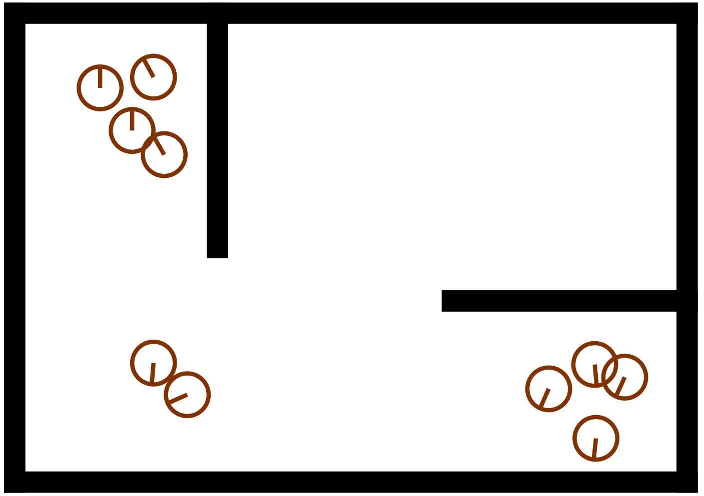


4. evaluate_particles()

- To find the weight of a particle, you first have to compute its cost (with the cost of all the particles) and then normalize it so that:
  - The addition of all the particles weight must be 1
  - 0 means a 0 probability for the robot to have the same state as the particle
  - 1 means a 100% probability that the robot has the same state as the particle

5. re_sampling()

- Randomly re-sample particles around the best ones according to a Gaussian distribution



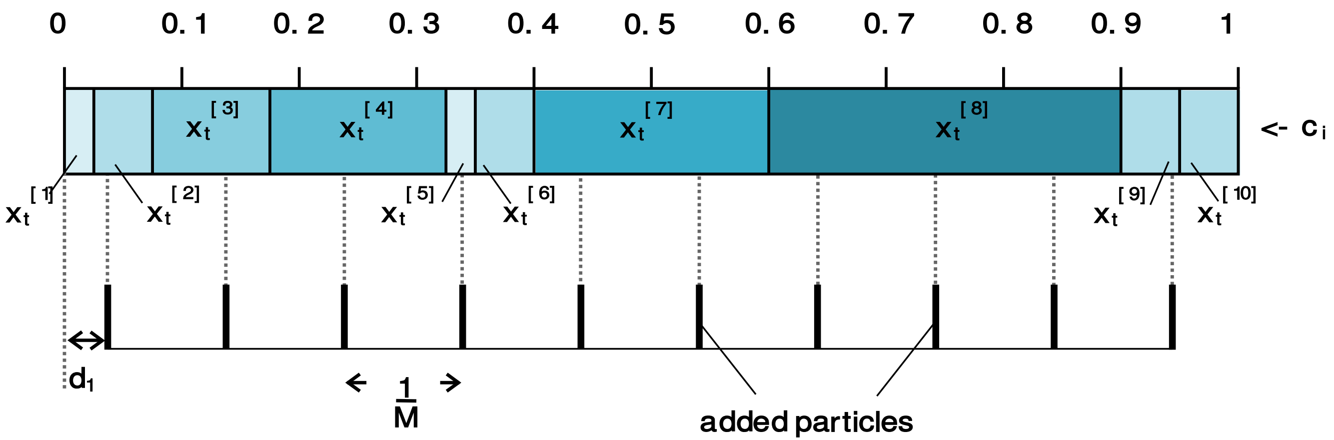
$$x_{new} \sim N(x_t, \sigma^2_{x,z}),$$
$$z_{new} \sim N(z_t, \sigma^2_{x,z}),$$
$$\theta_{new} \sim N(\theta_t, \sigma^2_\theta),$$
$$w_{new} = 0,$$
$$with\, \sigma^2_{x,z} = 0{,}05 \quad \text{and} \quad \sigma^2_\theta = \% \frac{pi}{180}.$$

- The best particle has to be kept to stabilize the robot’s pose evaluation

5. re_sampling()

- Systematic re-sampling

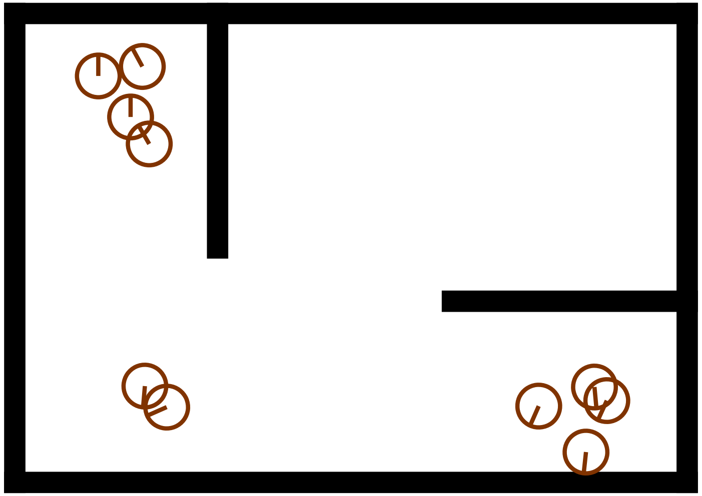
i	1	2	3	4	5	6	7	8	9	10
wt[i]	0.025	0.05	0.1	0.15	0.025	0.05	0.2	0.3	0.05	0.05
ci	0.025	0.075	0.175	0.325	0.350	0.4	0.6	0.9	0.95	1



6. estimate_from_odometry()

- Move the particles according to the odometry data

$$\text{odometry data} = \{\Delta_{dst}, \Delta_\theta\}$$
$$\forall x_t^{[m]},$$
$$x_{t+1,m} = x_{t,m} + \cos(\theta_{t,m}).\Delta_{dst}$$
$$z_{t+1,m} = z_{t,m} + \sin(\theta_{t,m}).\Delta_{dst}$$
$$\theta_{t+1,m} = \theta_{t,m} + \Delta_\theta$$



7. add_random_particles()

- To handle a kidnapping recovery
- Should not modify the total number of particles
  - Randomly modify some particles of the set
  - Should avoid to change the best particle as a random particle