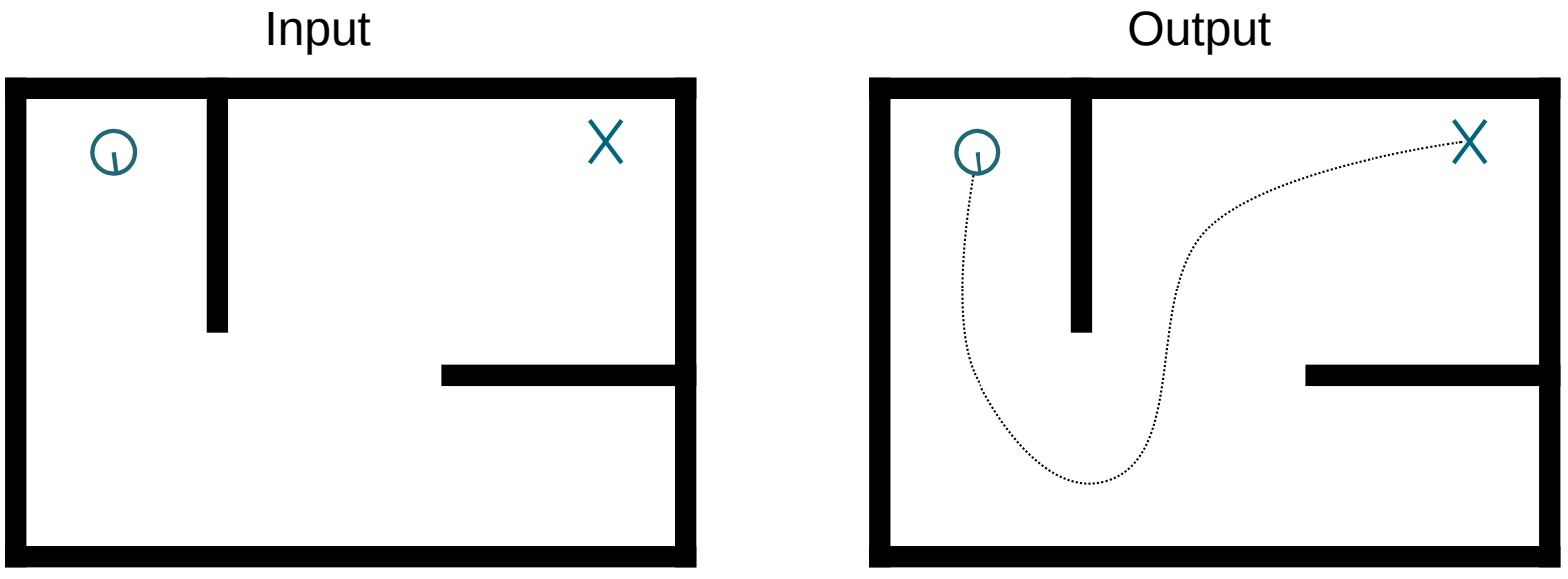
MOBILE ROBOTICS

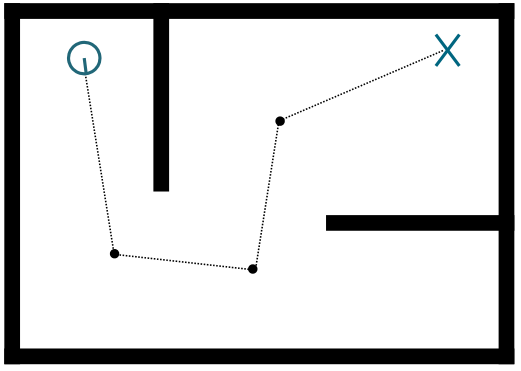
PATH PLANNING – Rapidly-explored Random Tree

Introduction



Assumptions

- Vectorized map (set of segments)
- Known pose at each step
- Nonholonomic constraints are not considered
  - The Robot is assumed to rotate or move forward : no curve!



Configuration Space

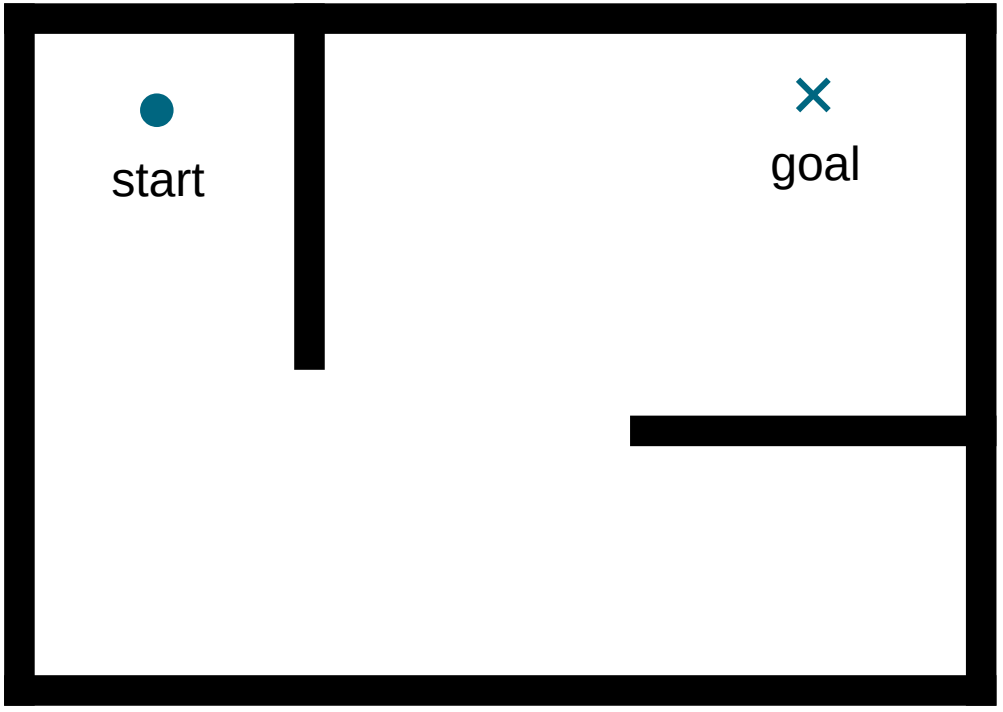
- Robot state : $\boldsymbol{q}(t)=(x(t),z(t),\theta(t))^T$
- Relaxing the nonholonomic constraint:
  - No constraint over θ
  - Relaxed configuration space : (x(t), z(t))

RRT – Rapidly-explored Random Tree

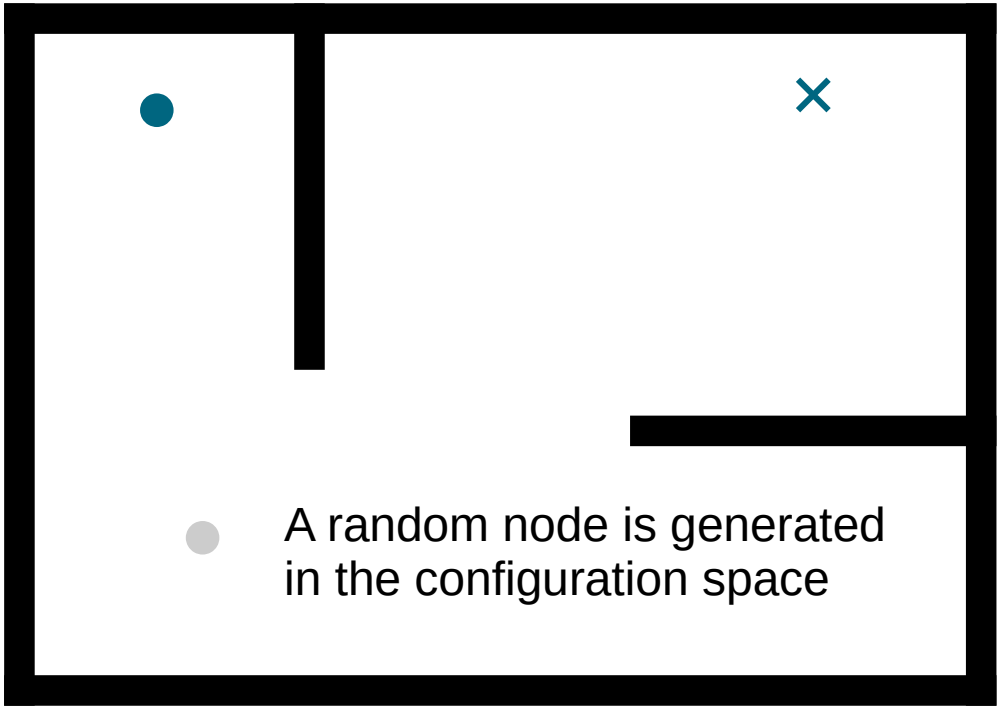
- Create a tree (graph)
- Between each node the path is feasible
- The graph connects the Robot’s initial pose (starting point) to the target (goal point)

RRT – Rapidly-explored Random Tree

The tree is initialized with the “start” node

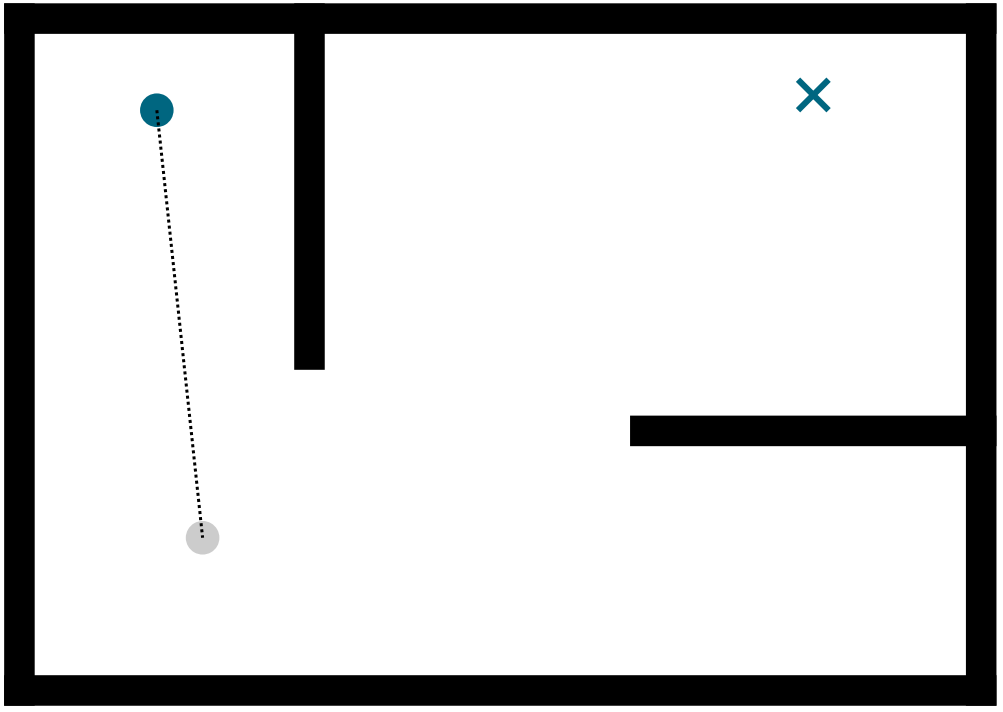


RRT – Rapidly-explored Random Tree



RRT – Rapidly-explored Random Tree

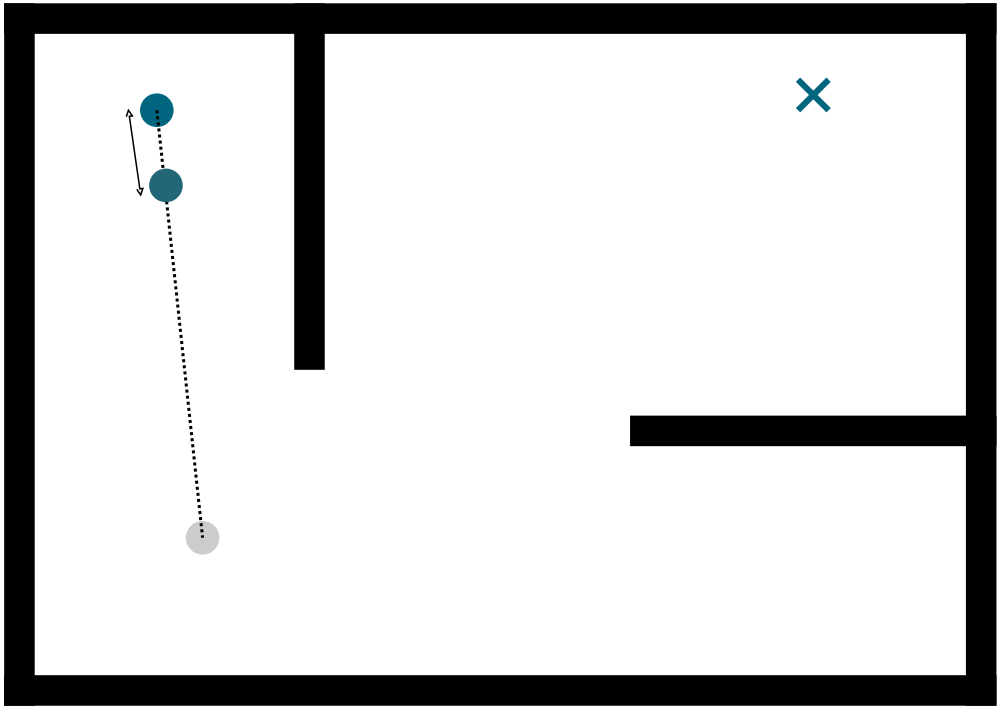
We find the closest node to the random one such that the trajectory between the two nodes stays in the configuration space



If no node is found, we generate a new random node

RRT – Rapidly-explored Random Tree

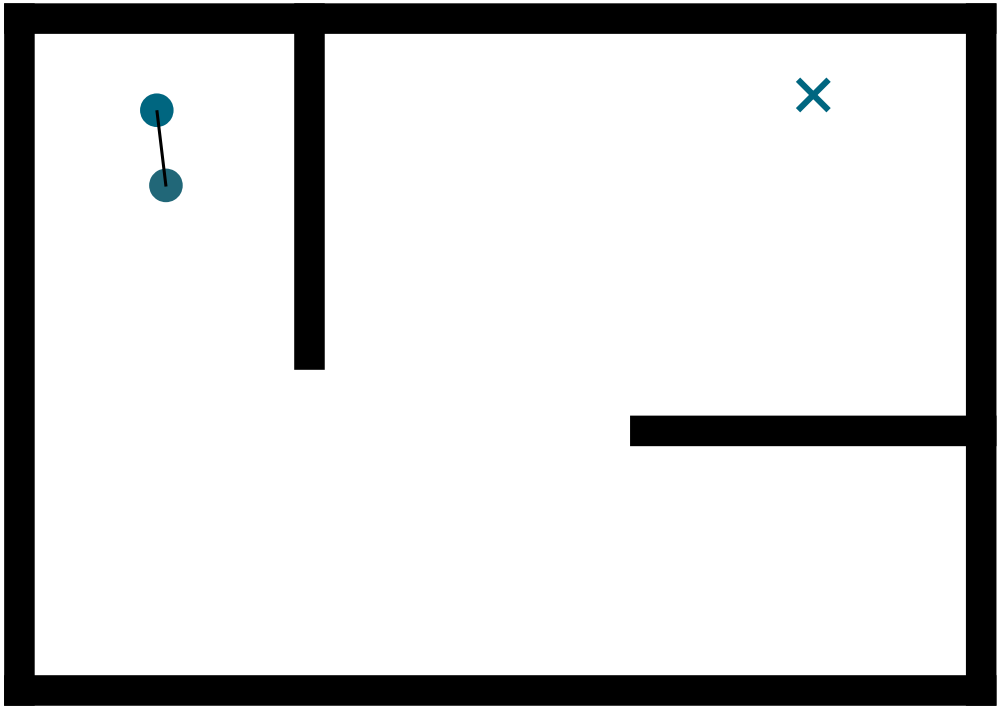
We create a new node in the direction of the random node from the closest one according to a step epsilon



If the random node is closer than the epsilon value, it is kept

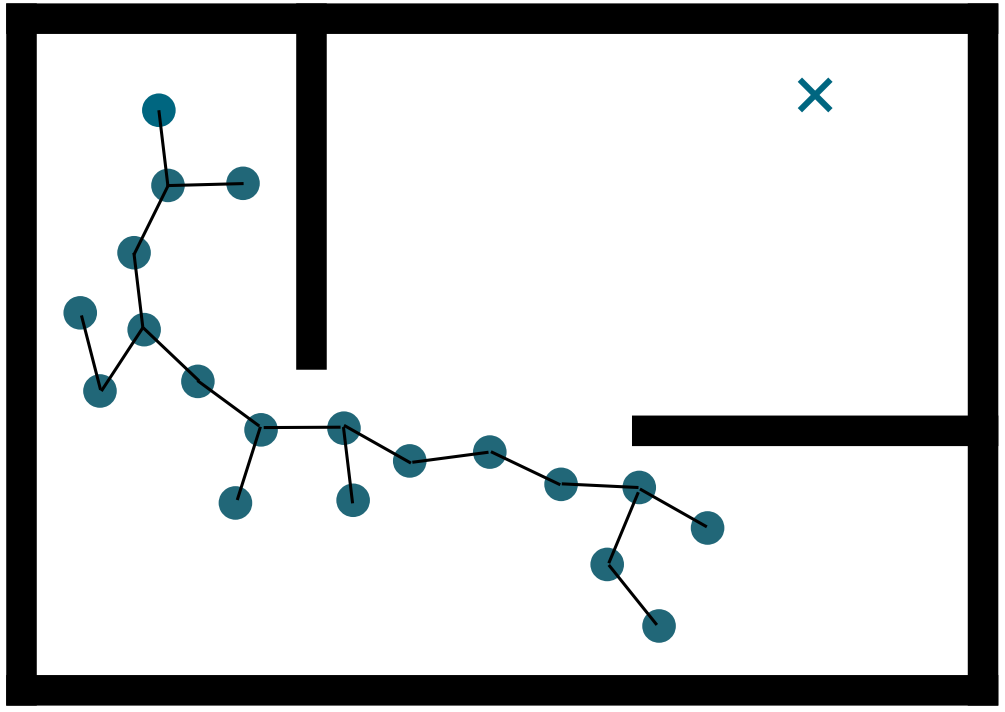
RRT – Rapidly-explored Random Tree

This new node is added to the RRT

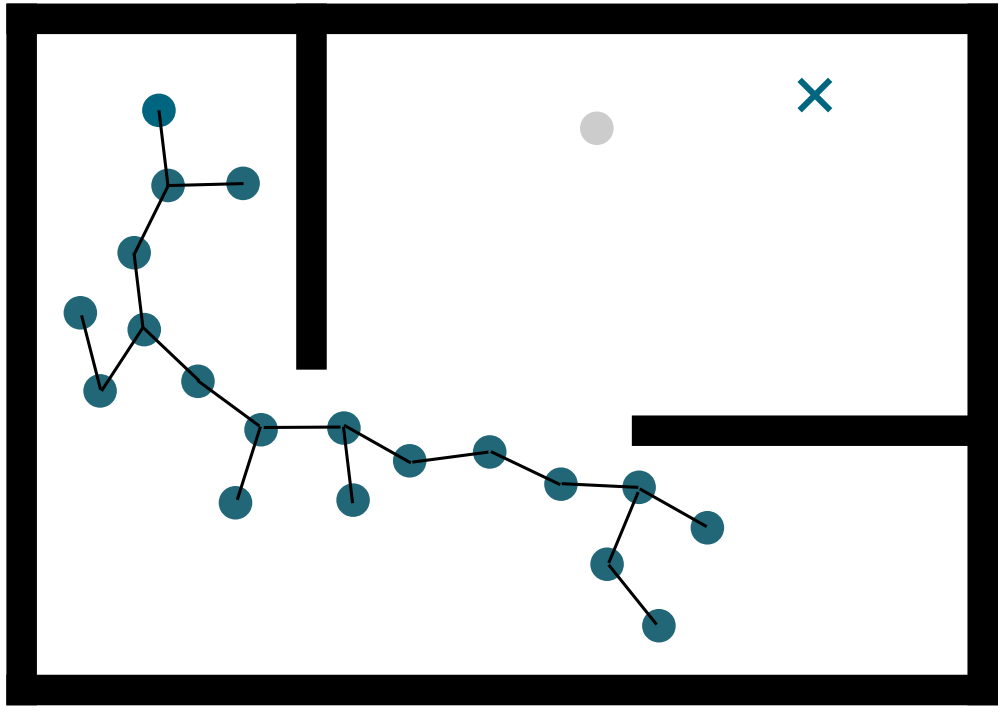


This is done until the new node is close enough to the goal

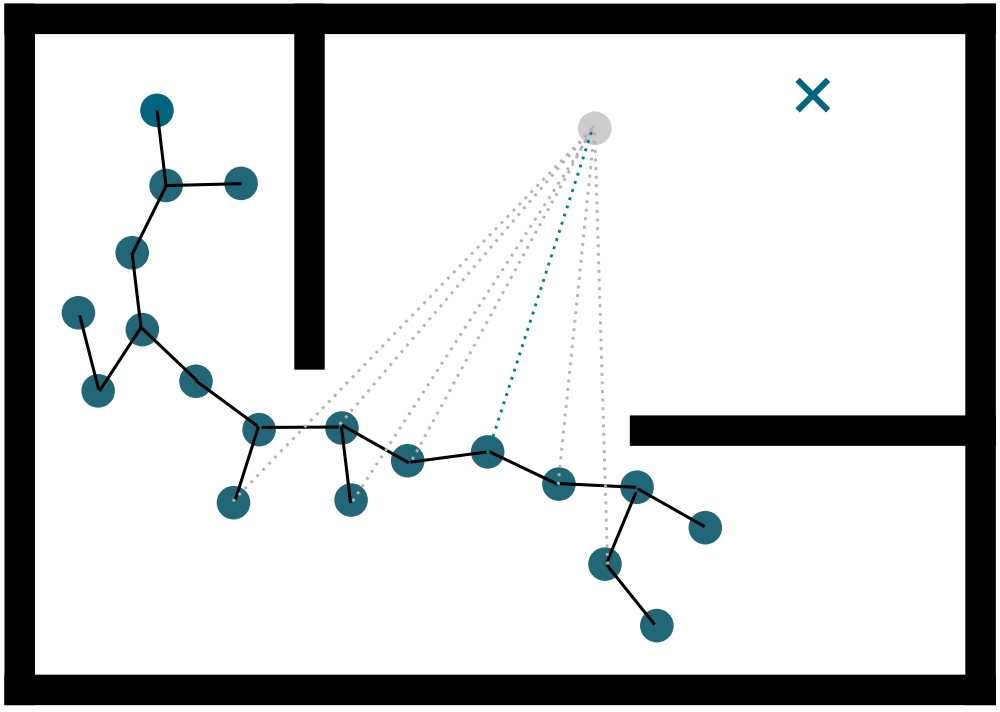
RRT – Rapidly-explored Random Tree



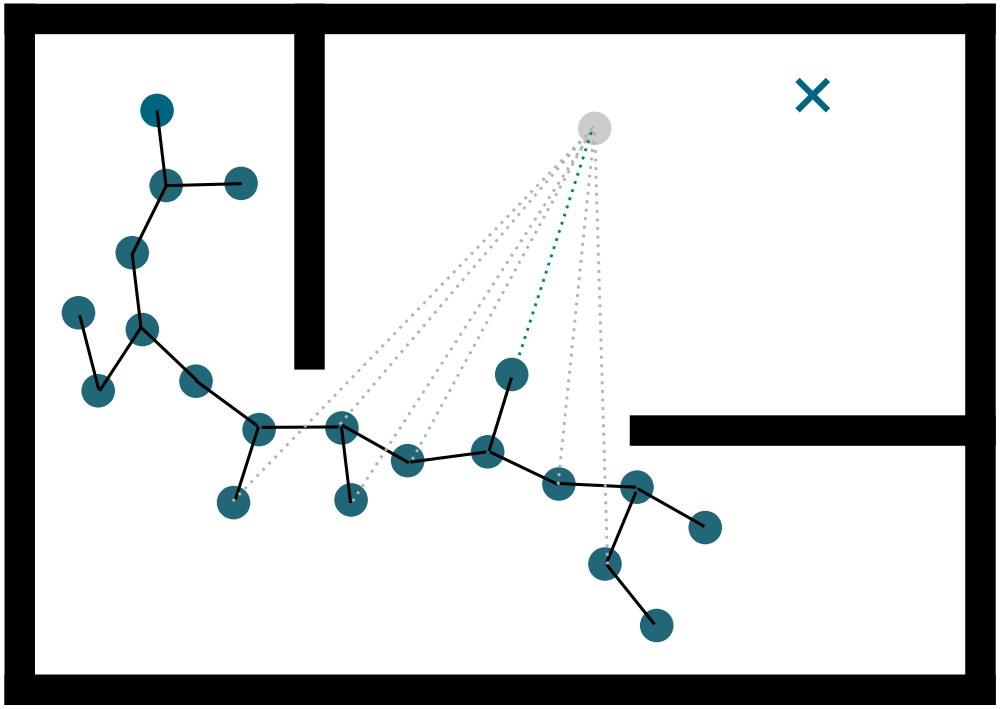
RRT – Rapidly-explored Random Tree



RRT – Rapidly-explored Random Tree



RRT – Rapidly-explored Random Tree



Work to do

- Files to upload in Moodle
  - tp_rrt/rrt_star.py
  - tp_rrt/utils.py
- Warnings
  - Test your code functions after functions
  - Do not modify the other files
  - Do not add any library (numpy for instance…)
- You should use run_rrt_tests to run unit tests

1.1 get_rand_node()

- Returns a random node inside the map

1.2 trajectory_free()

- Returns True if the trajectory from the node “start” to the node “goal” does not intersect an obstacle in the map, otherwise it returns False
- You can use Segment2D static methods...

1.3 step_from_to()

- Returns a node at an epsilon distance of the node n1 in the direction of the node n2
- Note that if the distance between the two nodes is lower that epsilon, the node n2 is directly returned

2.1 build_rrt()

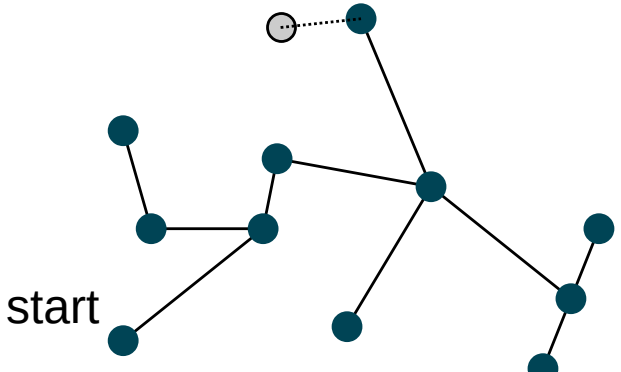
- Remarks:
  - **self.epsilon** (0.5)
  - Ends if the goal is found (distance lower that **self.max_distance**) or if the tree has **max_nodes** nodes
    - Update the succeed variable (True if the goal is reached, False otherwise)

To improve the tree construction: each **self.rand_nodes_before_target** generated nodes, the goal is considered as the random node

- The identifier of a node corresponds to the current size of the tree
- Once the goal has been found, it has to be added to the tree, updating its identifier and its parent

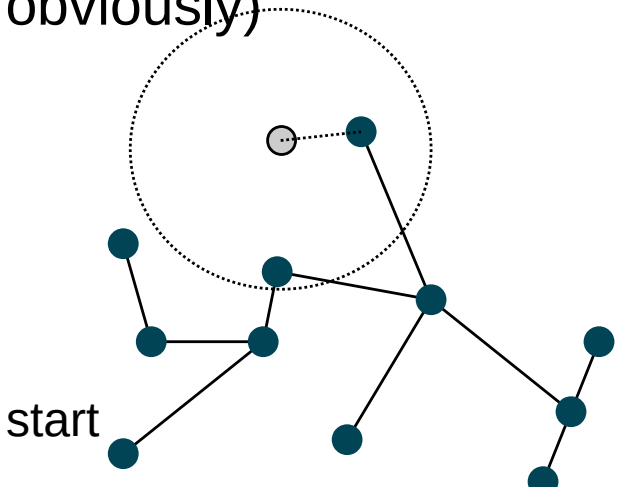
2.2 chose_parent()

- Remarks
  - Chose from the RRT a better parent for the node “node”
  - Best parent: node inside a radius of the node “node” that minimizes the distance (number of nodes) from “node” to “start” (without crossing an obstacle obviously)
  - radius = 0.3
  - Update the cost of the node “node”
  - Returns the updated node “node”



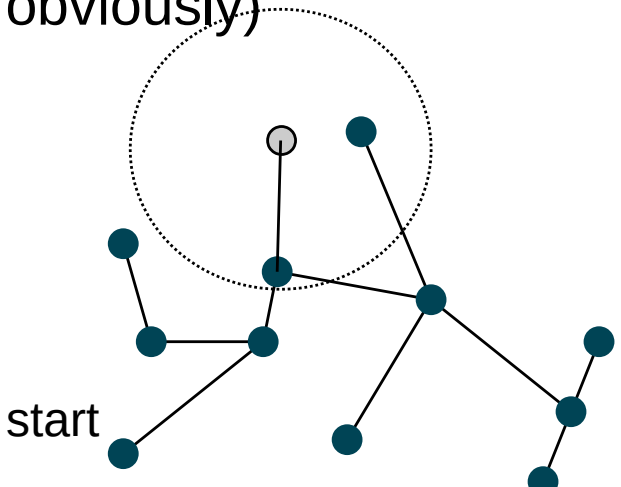
2.2 chose_parent()

- Remarks
  - Chose from the RRT a better parent for the node “node”
  - Best parent: node inside a radius of the node “node” that minimizes the distance (number of nodes) from “node” to “start” (without crossing an obstacle obviously)
  - radius = 0.3
  - Update the cost of the node “node”
  - Returns the updated node “node”



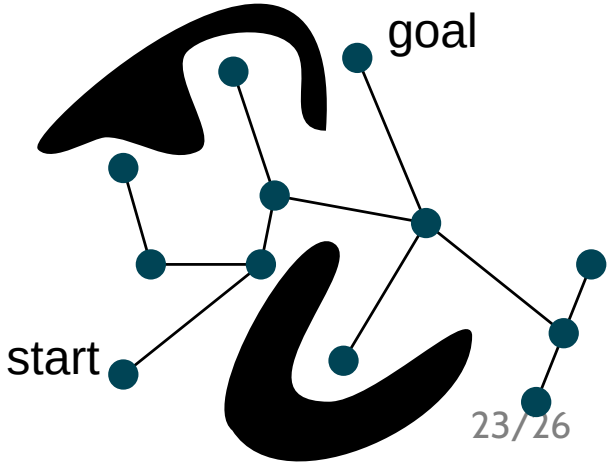
2.2 chose_parent()

- Remarks
  - Chose from the RRT a better parent for the node “node”
  - Best parent: node inside a radius of the node “node” that minimizes the distance (number of nodes) from “node” to “start” (without crossing an obstacle obviously)
  - radius = 0.3
  - Update the cost of the node “node”
  - Returns the updated node “node”



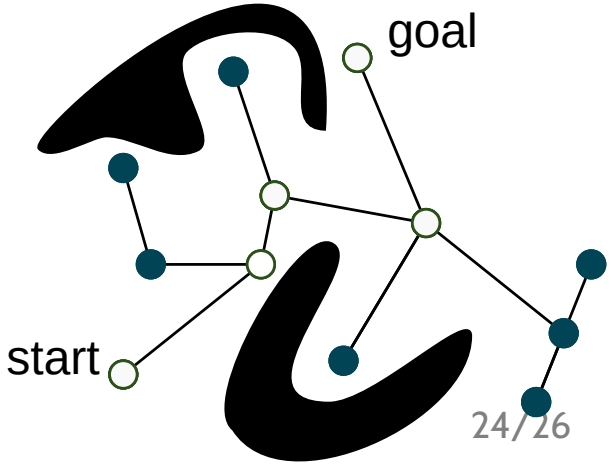
3.1 compute_path()

- Compute a path in the tree to go from the goal node to the starting node



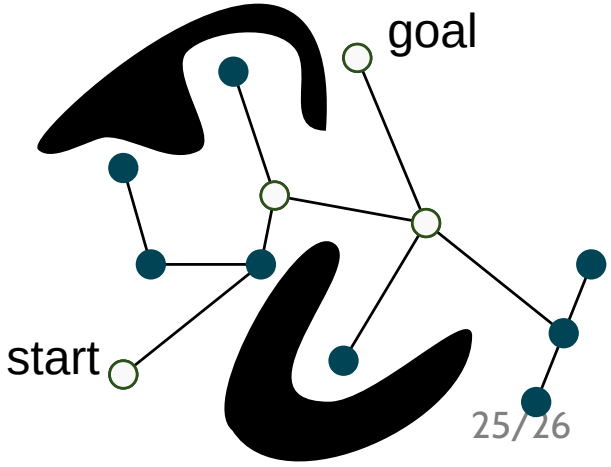
3.1 compute_path()

- Compute a path in the tree to go from the goal node to the starting node



3.2 improve_path()

- Remove the nodes that are not mandatory



4. follow_path

- Compute ul and ur according to the path to follow
  - Turing left, turning right or moving forward
- The robot's pose is known (robot.x, robot.z and robot.theta)
- Remove the reached nodes of the path
  - Min distance to consider a node as reached : 0.05
- Return the command U and the updated path
- The robot should stop when reaching the goal