Control of Mobile Robots

Planning with Matlab

Prof. Luca Bascetta (luca.bascetta@polimi.it)

Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria

Planning with Matlab

- Matlab supports the use and development of planning algorithms
  - Robotics System Toolbox, includes an implementation of sPRM for mobile robots
  - Navigation Toolbox, includes a general implementation of RRT / RRT* and supports the implementation of custom planning algorithms
- We start looking at PRM considering different values for the planner parameters
- Then we analyze the Motion Planning library that represents an example of a realistic implementation of a planning algorithm

Robotics System Toolbox: PRM

- First, a map of the environment is required

  binaryOccupancyMap creates an occupancy grid map. Each cell has a value representing the occupancy status of that cell. An occupied location is represented as true (1) and a free location is false (0).

- You can load an example matrix representing a map … *ps.mat*, then

  map = binaryOccupancyMap(simpleMap,2);

  show(map)

- And then we create the roadmap

  prmSimple … RM;

  prmSimple.map … map;

  prmSimple.NumNodes = 250;

  show(prmSimple)

Resolution: number of cells per meter

Is one of the maps available in *exampleMaps.mat*

Shows the map

Number of nodes

… ary occupancy grid

Shows the roadmap

Robotics System Toolbox: PRM

- It is now time to query for a path!
- We select a start and a goal location and then we call *findpath*

  startLocation = [2 1];

  endLocation = [12 10];

  path = findpath(prmSimple,startLocation,endLocation);

  show(prmSimple)
- Which parameters can we change? ... the radius used by *Near* function

  prmSimple.ConnectionDistance = 2;

  path = findpath(prmSimple,startLocation,endLocation);

  show(prmSimple)

  The default value is infinity
- You can now play with different environments and test how to adjust N and r

- It is a collection of methods to support path planning with two algorithms (RRT / RRT*) already implemented
- Available state spaces
    - *stateSpaceSE2*, SE(2) state space, composed of state vectors represented by $[x, y, \theta]$, uses Euclidean distance to calculate distance and linear interpolation to calculate translation and rotation of the state
    - *stateSpaceSE3*, SE(3) state space, composed of state vectors represented by $[x, y, z, q_w, q_x, q_y, q_z]$, uses Euclidean distance calculation and linear interpolation for the translation component of the state, uses quaternion distance calculation and spherical linear interpolation for the rotation component of the state

- Available validation functions (collision check)
    - *validatorOccupancyMap*, validates states and discretized motions based on the value in a 2-D occupancy map
    - *validatorOccupancyMap3D*, validates states and discretized motions based on occupancy values in a 3-D occupancy map
    - *validatorVehicleCostmap*, validates states and discretized motions based on the values in a 2-D costmap
- Available planning algorithms
    - *plannerRRT*, creates a rapidly-exploring random tree (RRT) planner for solving geometric planning problems
    - *plannerRRTStar*, creates an asymptotically-optimal RRT planner, RRT*. The RRT* algorithm converges to an optimal solution in terms of the state space distance

Navigation Toolbox: RRT

- You can use an example occupancy … *…aps.mat*
- First, we have to select the state space… we would like to plan for a unicycle robot

  bounds = [map.XWorldLimits; map.YWorldLimits…

  space = stateSpaceSE2(bounds);

- For this state space we define a collision checker

  stateValidator = validatorOccupancyMap(space);

  stateValidator.Map = map;

  stateValidator.ValidationDistance = 0.05;

- We can now define the planner

  planner = plannerRRT(space, stateValidator);

  planner.MaxConnectionDistance = 1.0;

  planner.MaxIterations = 30000;

  planner.MaxNumTreeNodes = 10000;

  planner.GoalBias = 0.05;

  planner.GoalReachedFcn = @isGoalReached;

Bounds on the state variables

x, y, θ

Discretization distance used to check c…

The maximum length of a motion allowed in the tree

Maximum number of iterations *N*. It is different from the maximum number of nodes as we are not using SampleFree

Maximum number of nodes that can be added to the tree

Probability of choosing goal state during state sampling

A user defined function to check if a configuration is in the goal region

Navigation Toolbox: RRT

- An example of function to check if the configuration is in the goal region

```
function isReached = isGoalReached(planner, goalState, newState)
    isReached = false;
    threshold = 0.1;
    if planner.StateSpace.distance(newState, goalState) < threshold
        isReached = true;
    end
end
```

- We are now ready to start the planner

```
start = [-1.0, 0.0, -pi];
goal = [14, -2.25, 0];
[pthObj, solnInfo] = plan(planner, start, goal);
```

The planned path

Information about the solution, including the tree

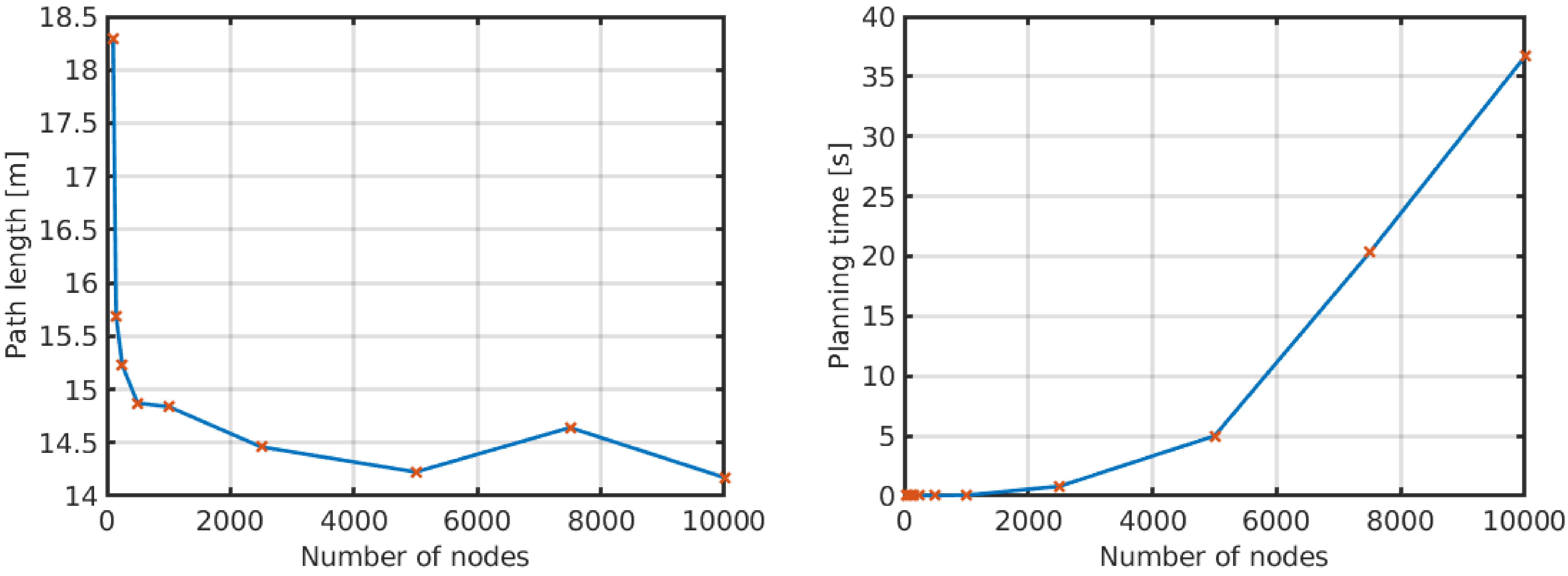
Navigation Toolbox: RRT

- We can now plot the tree and the resulting path

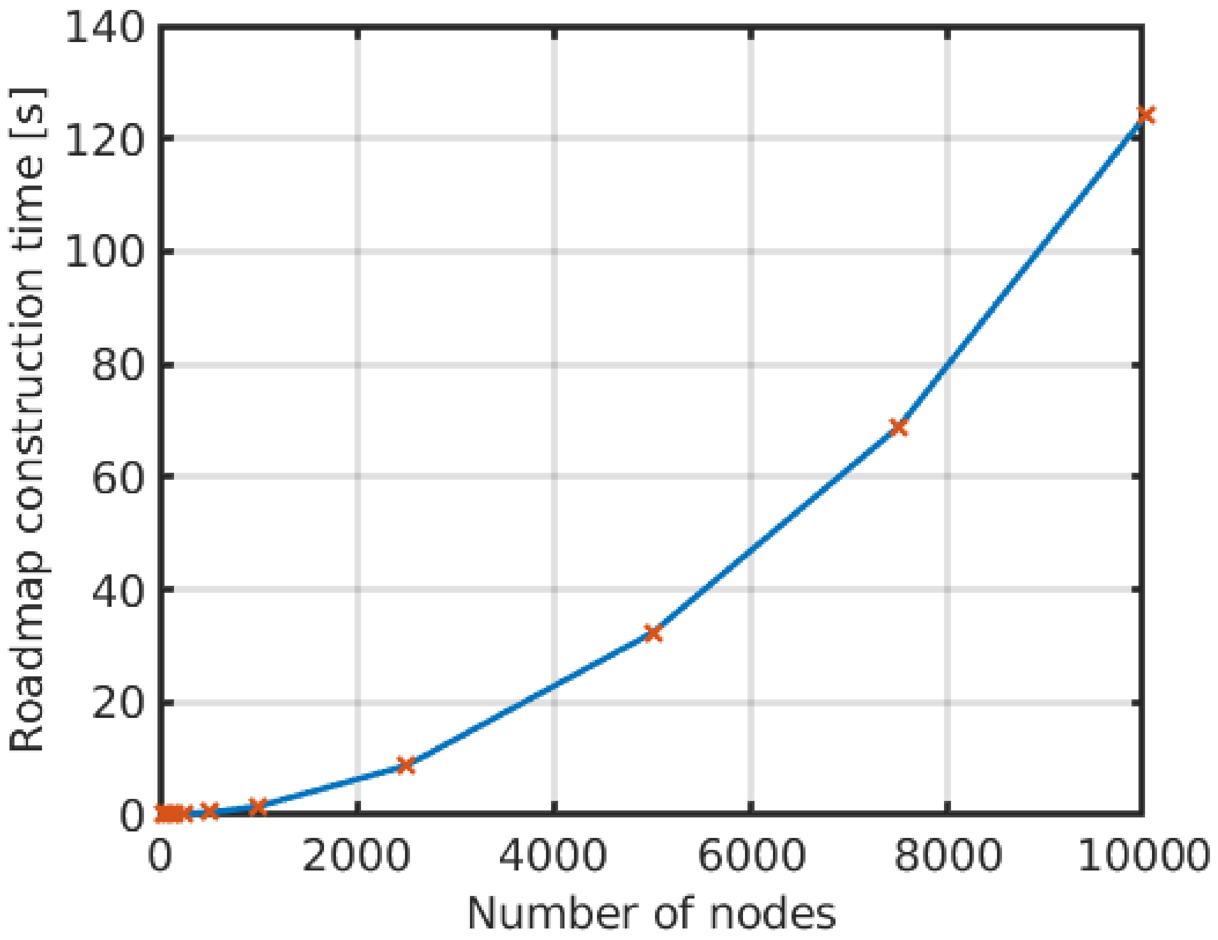
```
show(map),hold on
plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-')

interpolate(pthObj,300);
plot(pthObj.States(:,1), pthObj.States(:,2), 'r-', 'LineWidth', 2),hold off
```

- You can now test changing the *MaxConnectionDistance* to see how the solution changes
- How is this parameter connected to our version of the RRT algorithm?

Analyzing the results: PRM
Exploring a different set of nodes, from 25 to 10000

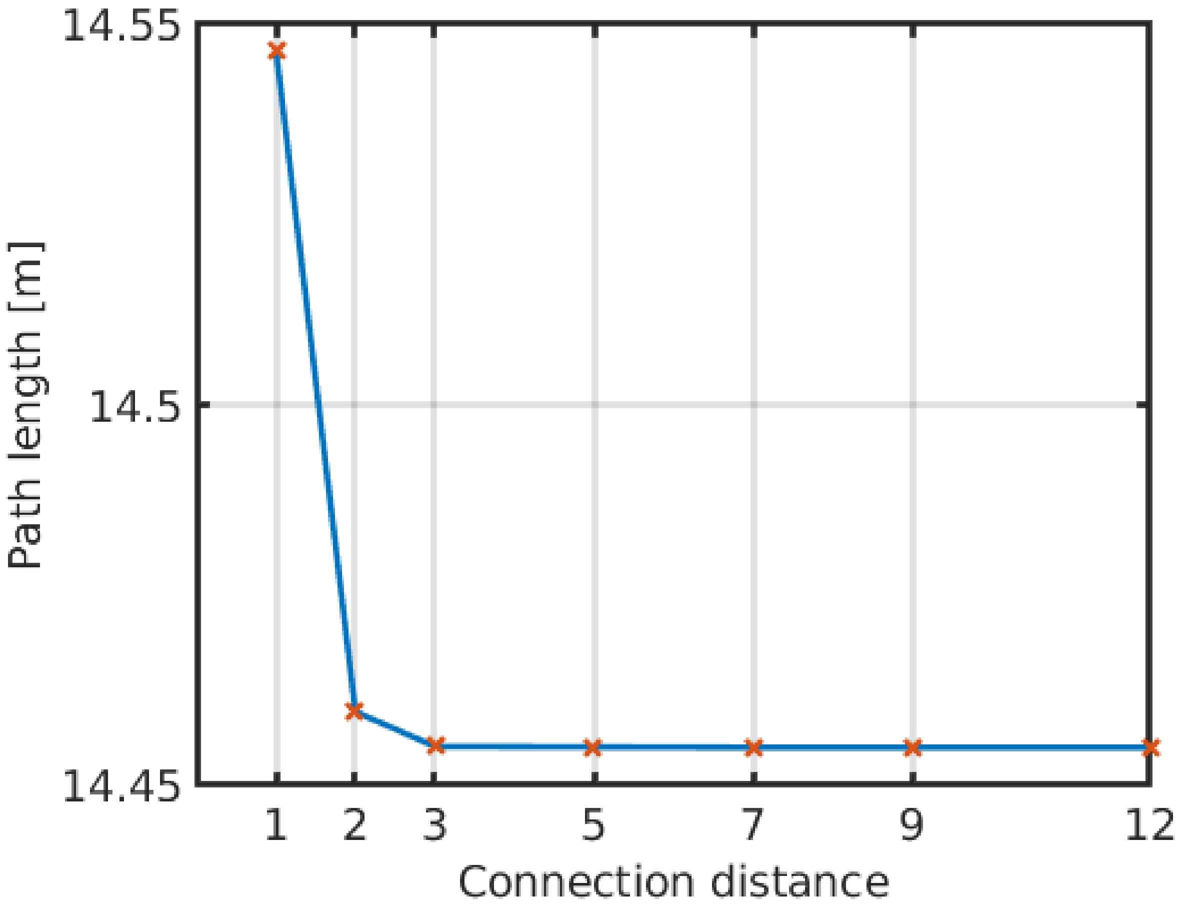


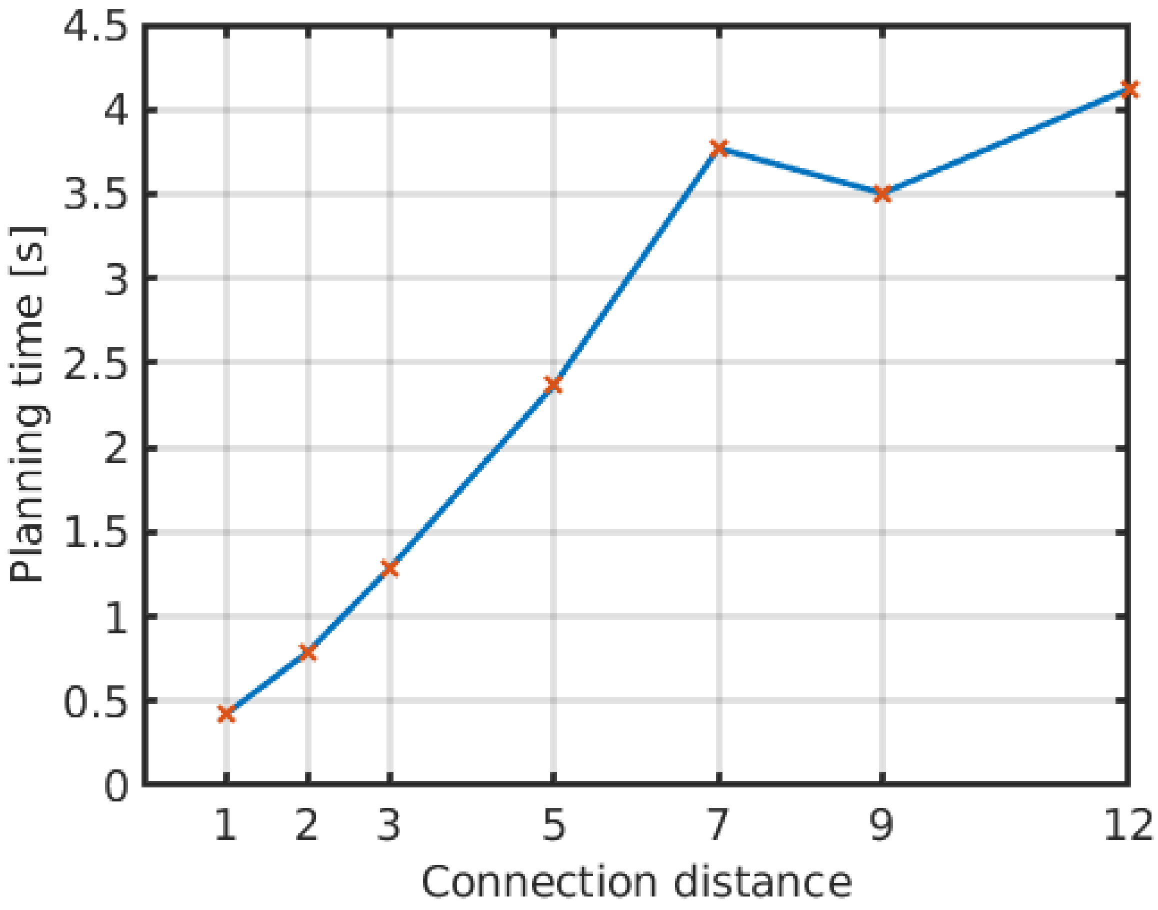
Analyzing the results: PRM
Exploring a different set of nodes, from 25 to 10000



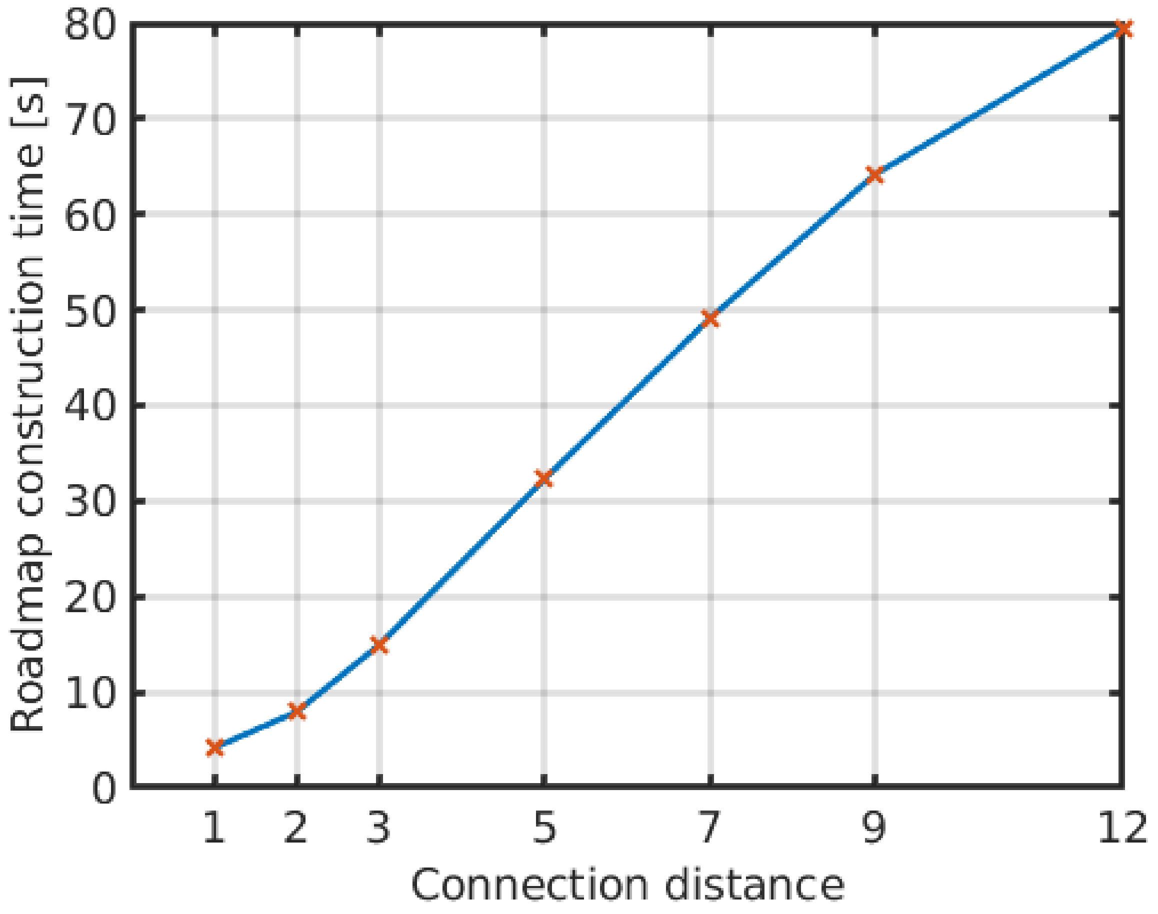
Analyzing the results: PRM

Exploring a different connection distance, from 1 to 12, with 2500 nodes

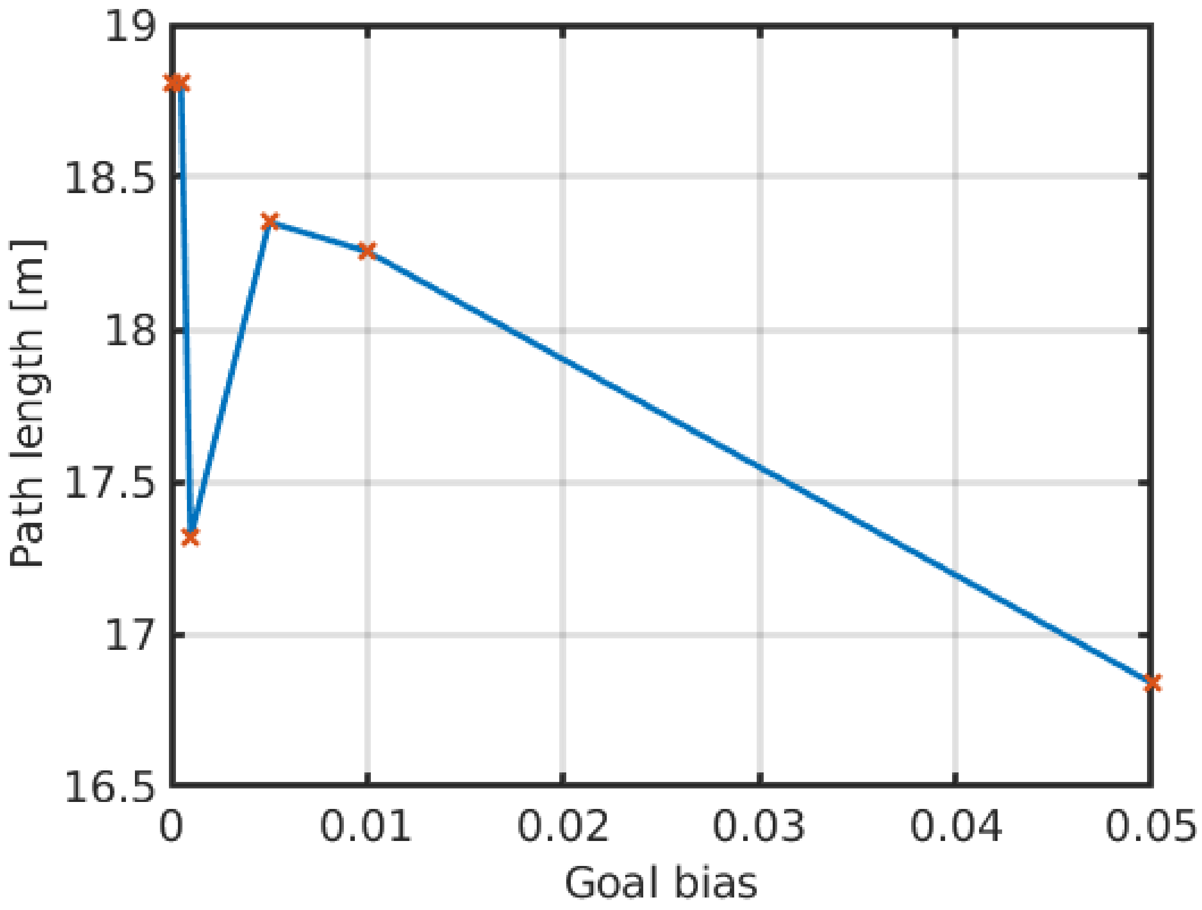


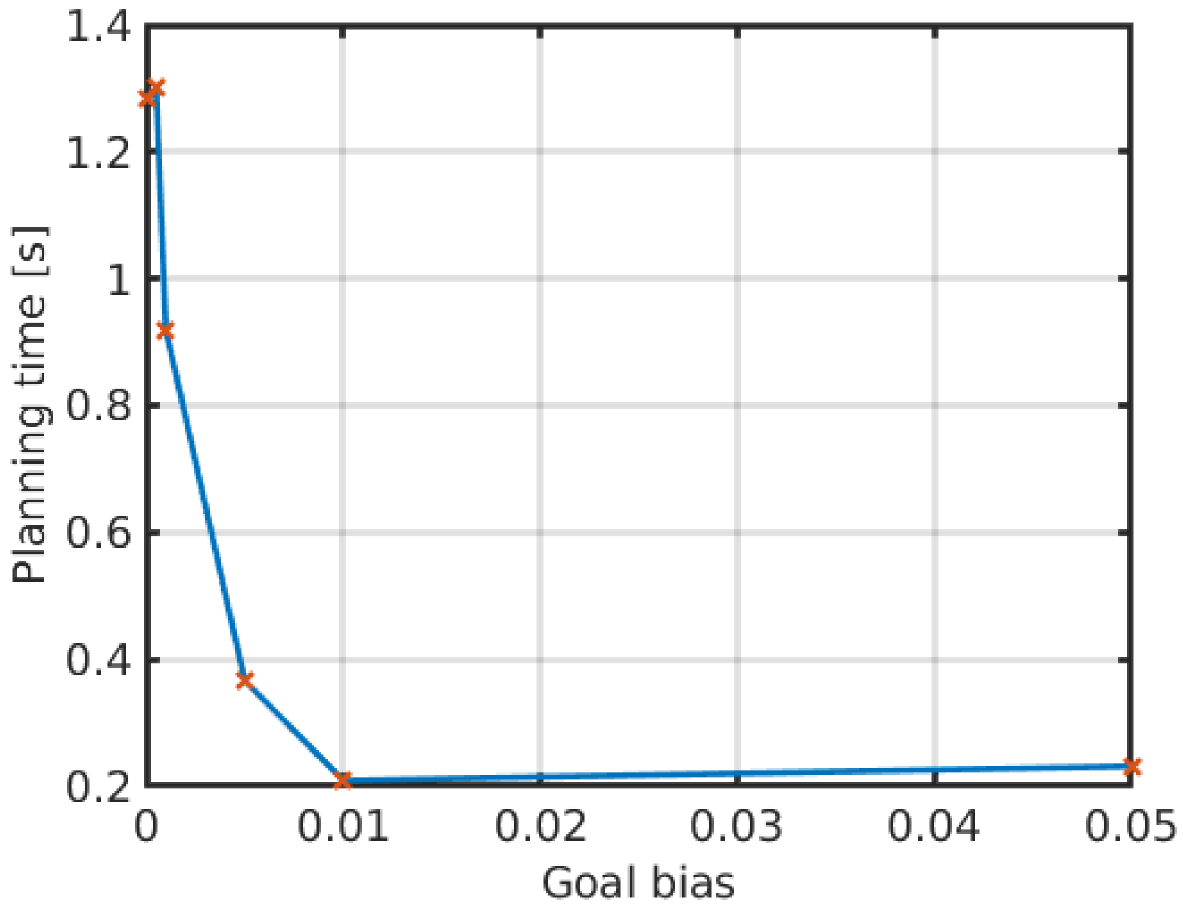


Analyzing the results: PRM
Exploring a different connection distance, from 1 to 12, with 2500 nodes



Analyzing the results: RRT
Exploring a different goal bias, from 0.05 to 0.0001





Analyzing the results: RRT
Exploring a different goal bias, from 0.05 to 0.0001