Path Planning in Environments of Different Complexity

Try it in MATLAB

This example demonstrates how to compute an obstacle free path between two locations on a given map using the Probabilistic Roadmap (PRM) path planner. PRM path planner constructs a roadmap in the free space of a given map using randomly sampled nodes in the free space and connecting them with each other. Once the roadmap has been constructed, you can query for a path from a given start location to a given end location on the map.

In this example, the map is represented as an occupancy grid map using imported data. When sampling nodes in the free space of a map, PRM uses this binary occupancy grid representation to deduce free space. Furthermore, PRM does not take into account the robot dimension while computing an obstacle free path on a map. Hence, you should inflate the map by the dimension of the robot, in order to allow computation of an obstacle free path that accounts for the robot's size and ensures collision avoidance for the actual robot. Define start and end locations on the map for the PRM path planner to find an obstacle free path.

Import Example Maps for Planning a Path

```
load exampleMaps.mat
```

The imported maps are : `simpleMap`, `complexMap` and `ternaryMap`.

```
whos *Map*
```

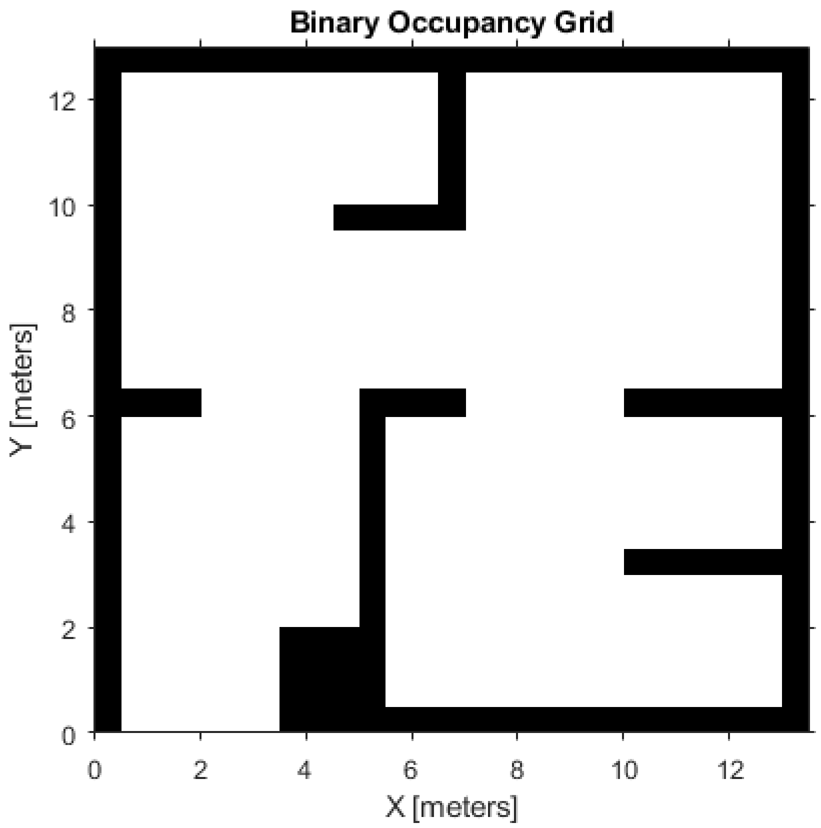
```
  Name                  Size                 Bytes  Class      Attributes

  complexMap           41x52                  2132  logical              
  emptyMap             26x27                   702  logical              
  simpleMap            26x27                   702  logical              
  ternaryMap         501x501               2008008  double               
```

Use the imported `simpleMap` data and construct an occupancy grid representation using the `robotics.BinaryOccupancyGrid` class. Set the resolution to 2 cells per meter for this map.

```
map = robotics.BinaryOccupancyGrid(simpleMap,2);
```

Display the map using the `show` function on the `robotics.BinaryOccupancyGrid` object

```
show(map)
```



Define Robot Dimensions and Inflate the Map

To ensure that the robot does not collide with any obstacles, you should inflate the map by the dimension of the robot before supplying it to the PRM path planner.

Here the dimension of the robot can be assumed to be a circle with radius of 0.2 meters. You can then inflate the map by this dimension using the `inflate` function.

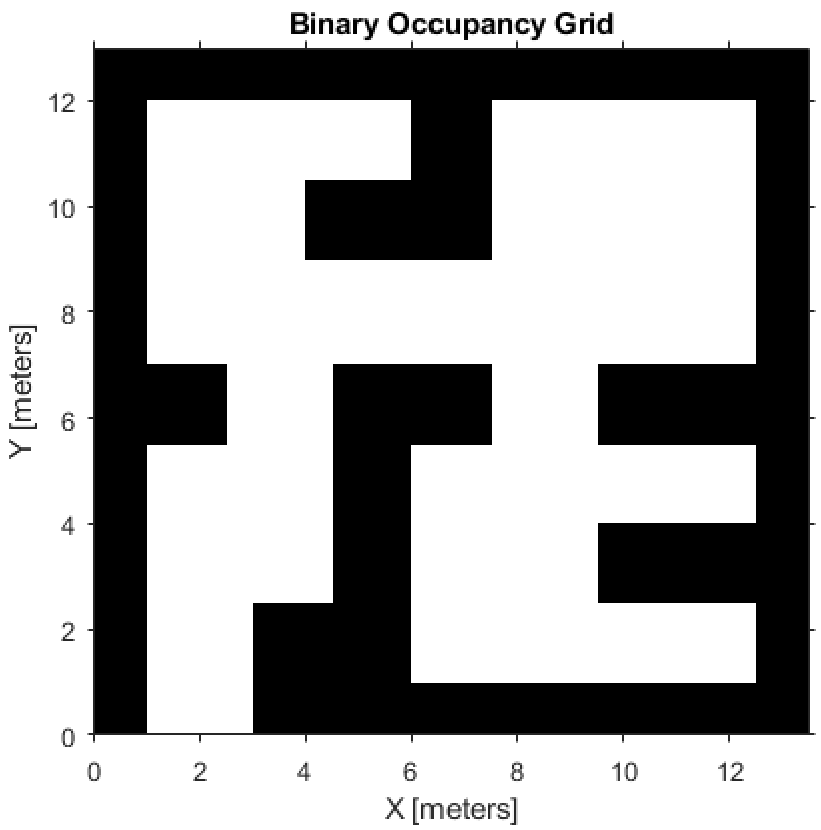
```
robotRadius = 0.2;
```

As mentioned before, PRM does not account for the dimension of the robot, and hence providing an inflated map to the PRM takes into account the robot dimension. Create a copy of the map before using the `inflate` function to preserve the original map.

```
mapInflated = copy(map);
inflate(mapInflated,robotRadius);
```

Display inflated map

```
show(mapInflated)
```



Construct PRM and Set Parameters

Now you need to define a path planner. Create a `robotics.PRM` object and define the associated attributes.

```
prm = robotics.PRM;
```

Assign the inflated map to the PRM object

```
prm.Map = mapInflated;
```

Define the number of PRM nodes to be used during PRM construction. PRM constructs a roadmap using a given number of nodes on the given map. Based on the dimension and the complexity of the input map, this is one of the primary attributes to tune in order to get a solution between two points on the map. A large number of nodes create a dense roadmap and increases the probability of finding a path. However, having more nodes increases the computation time for both creating the roadmap and finding a solution.

```
prm.NumNodes = 50;
```

Define the maximum allowed distance between two connected nodes on the map. PRM connects all nodes separated by this distance (or less) on the map. This is another attribute to tune in the case of larger and/or complicated input maps. A large connection distance increases the connectivity between nodes to find a path easier, but can increase the computation time of roadmap creation.

```
prm.ConnectionDistance = 5;
```

Find a Feasible Path on the Constructed PRM

Define start and end locations on the map for the path planner to use.

```
startLocation = [2 1];
endLocation = [12 10];
```

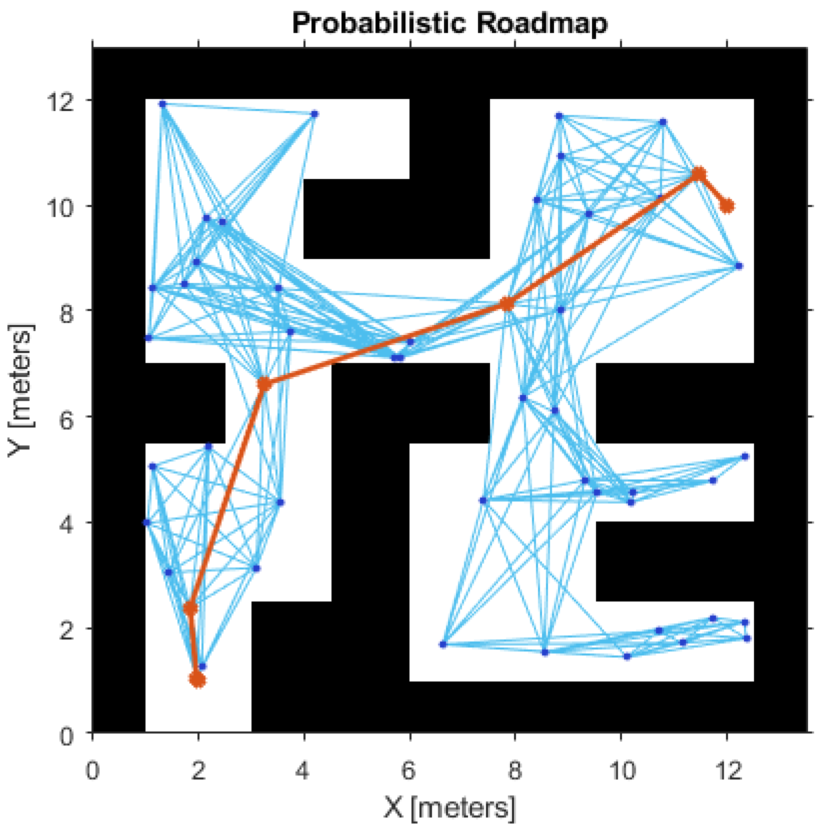
Search for a path between start and end locations using the `findpath` function. The solution is a set of waypoints from start location to the end location. Note that the `path` will be different due to probabilistic nature of the PRM algorithm.

```
path = findpath(prm, startLocation, endLocation)
```

```
path = 7×2

    2.0000    1.0000
    1.9569    1.0546
    1.8369    2.3856
    3.2389    6.6106
    7.8260    8.1330
   11.4632   10.5857
   12.0000   10.0000
```

Display the PRM solution.

```
show(prm)
```



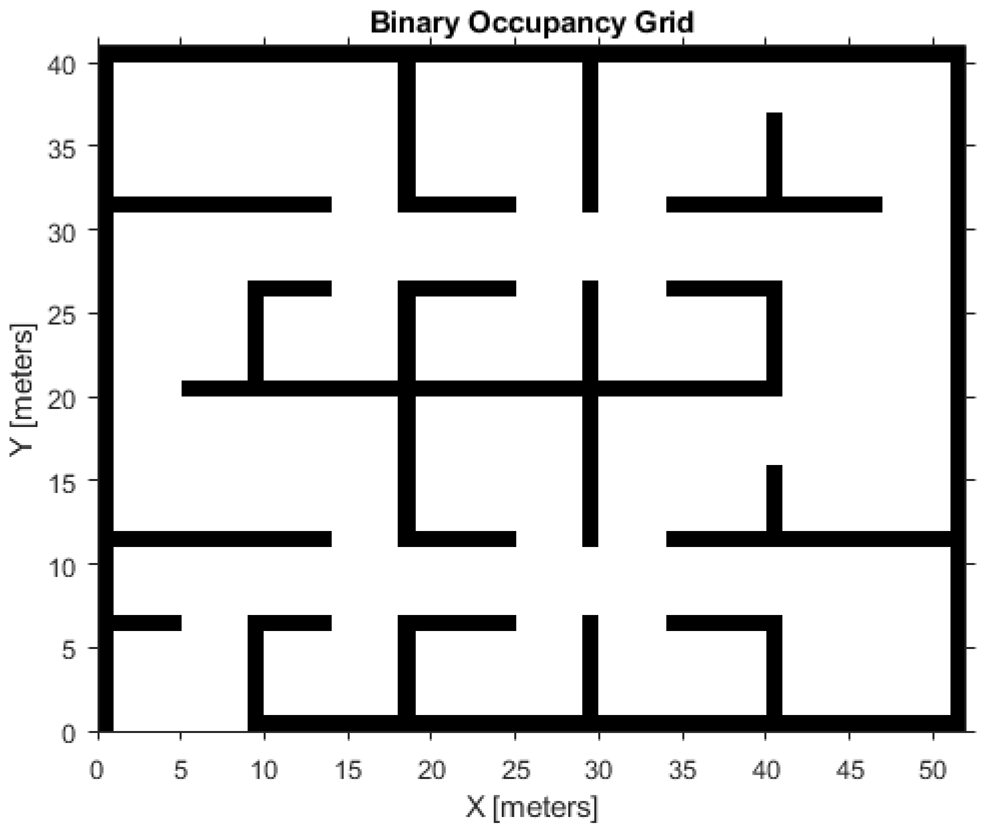
Use PRM for a Large and Complicated Map

Use the imported `complexMap` data, which represents a large and complicated floor plan, and construct a binary occupancy grid representation with a given resolution (1 cell per meter)

```
map = robotics.BinaryOccupancyGrid(complexMap,1);
```

Display the map.

```
show(map)
```



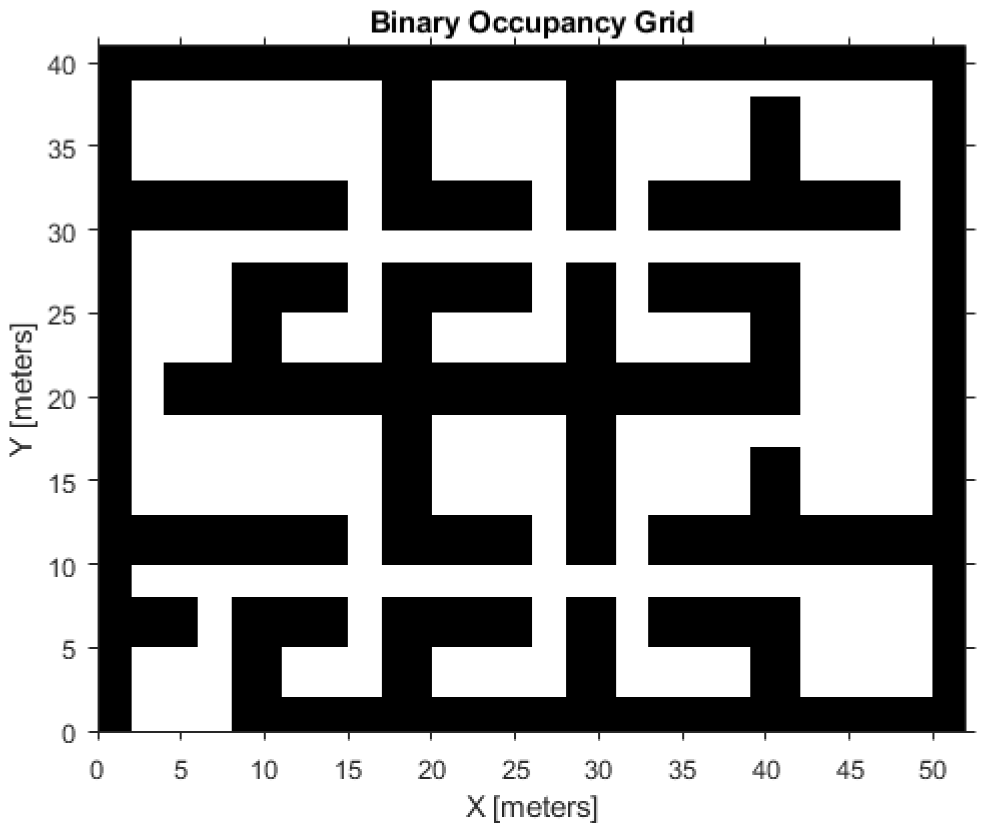
Inflate the Map Based on Robot Dimension

Copy and inflate the map to factor in the robot's size for obstacle avoidance

```
mapInflated = copy(map);
inflate(mapInflated, robotRadius);
```

Display inflated map.

```
show(mapInflated)
```



Associate the Existing PRM Object with the New Map and Set Parameters

Update PRM object with the newly inflated map and define other attributes.

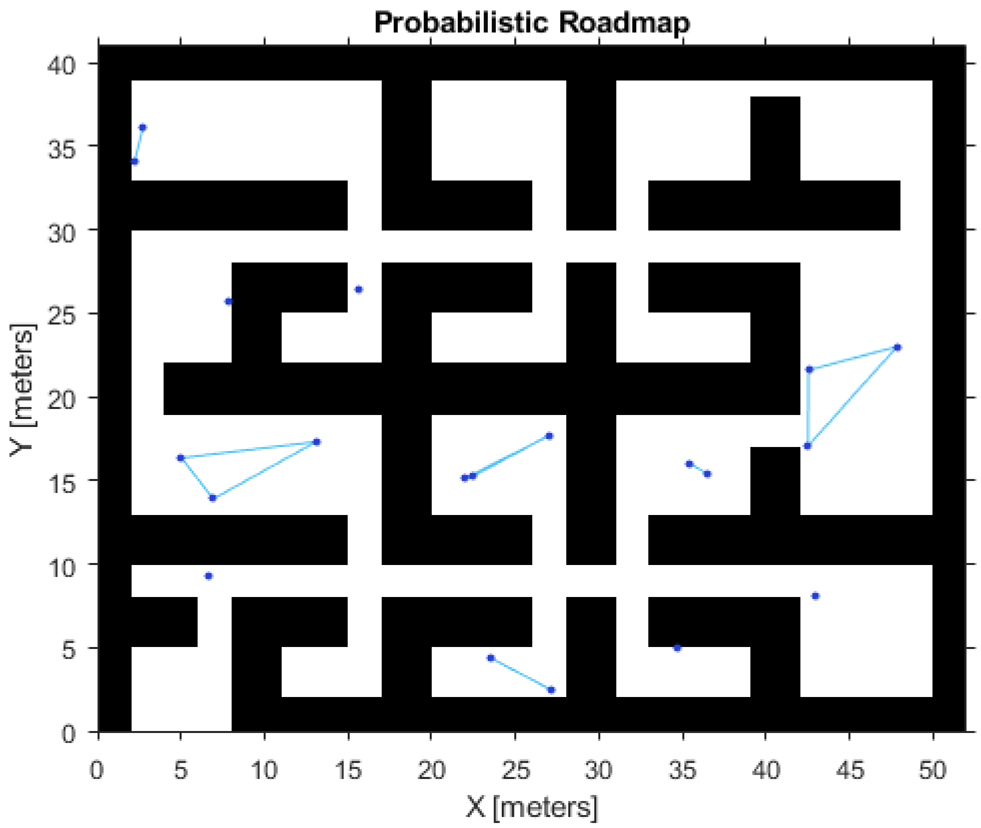
```
prm.Map = mapInflated;
```

Set the `NumNodes` and the `ConnectionDistance` properties.

```
prm.NumNodes = 20;
prm.ConnectionDistance = 15;
```

Display PRM graph.

```
show(prm)
```



Find a Feasible Path on the Constructed PRM

Define start and end location on the map to find an obstacle free path.

```
startLocation = [3 3];
endLocation = [45 35];
```

Search for a solution between start and end location. For complex maps, there may not be a feasible path for a given number of nodes (returns an empty path).

```
path = findpath(prm, startLocation, endLocation);
```

Since you are planning a path on a large and complicated map, larger number of nodes may be required. However, often it is not clear how many nodes will be sufficient. Tune the number of nodes to make sure there is a feasible path between the start and end location.

```
while isempty(path)
    % No feasible path found yet, increase the number of nodes
    prm.NumNodes = prm.NumNodes + 10;

    % Use the |update| function to re-create the PRM roadmap with the changed
    % attribute
    update(prm);

    % Search for a feasible path with the updated PRM
    path = findpath(prm, startLocation, endLocation);
end
```

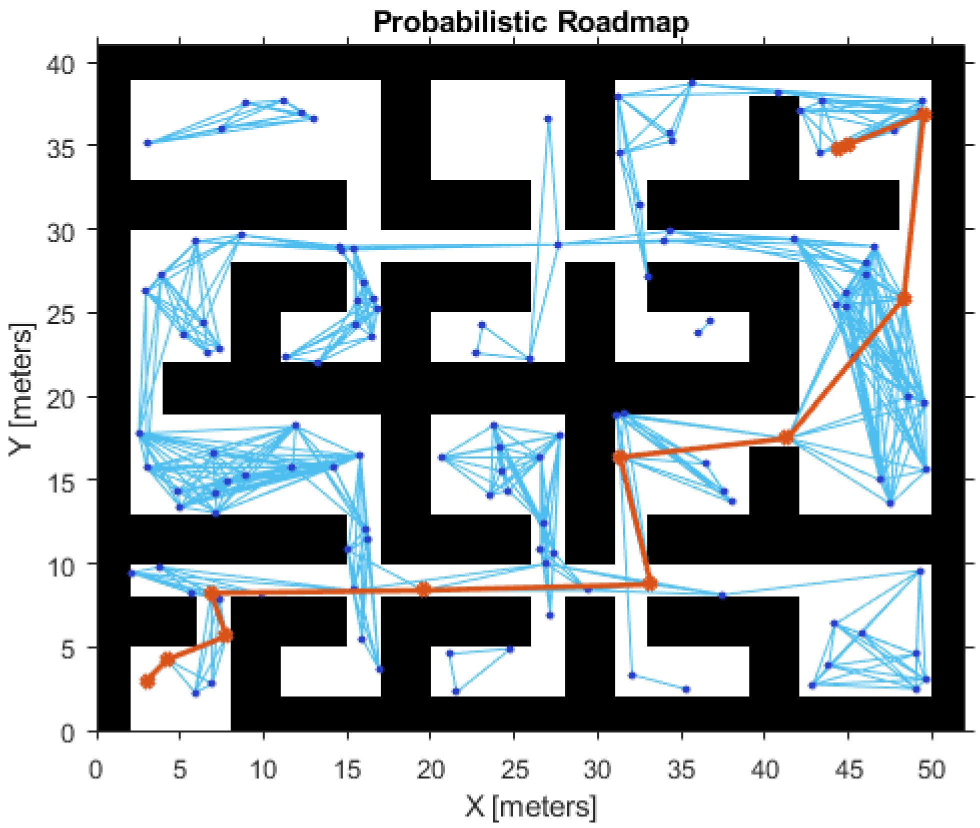
Display path.

```
path
```

```
path = 12×2

     3.0000    3.0000
     4.2287    4.2628
     7.7686    5.6520
     6.8570    8.2389
    19.5613    8.4030
    33.1838    8.7614
    31.3248   16.3874
    41.3317   17.5090
    48.3017   25.8527
    49.4926   36.8804
       ⋮
```

Display PRM solution.

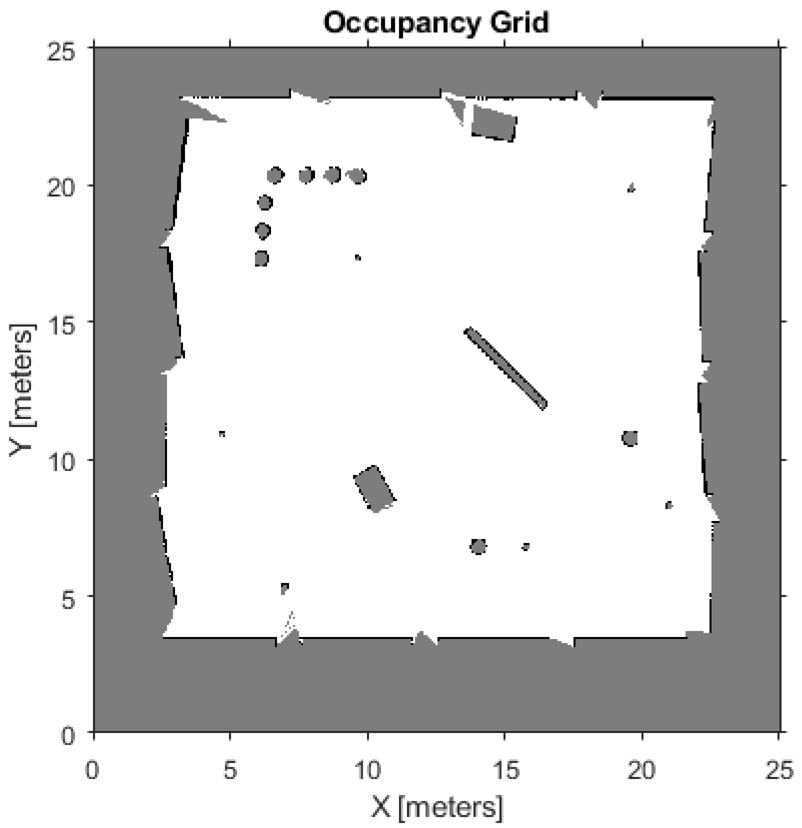
```
show(prm)
```



Use PRM with Probabilistic Occupancy Grid

Construct a `robotics.OccupancyGrid` object using the imported `ternaryMap` data. The `ternaryMap` represents an environment using probabilities, where the probability of occupancy for free space is 0, for occupied space is 1 and for unknown space is 0.5. Here, the resolution of 20 cells per meter is used.

```
map = robotics.OccupancyGrid(ternaryMap,20);
show(map)
```



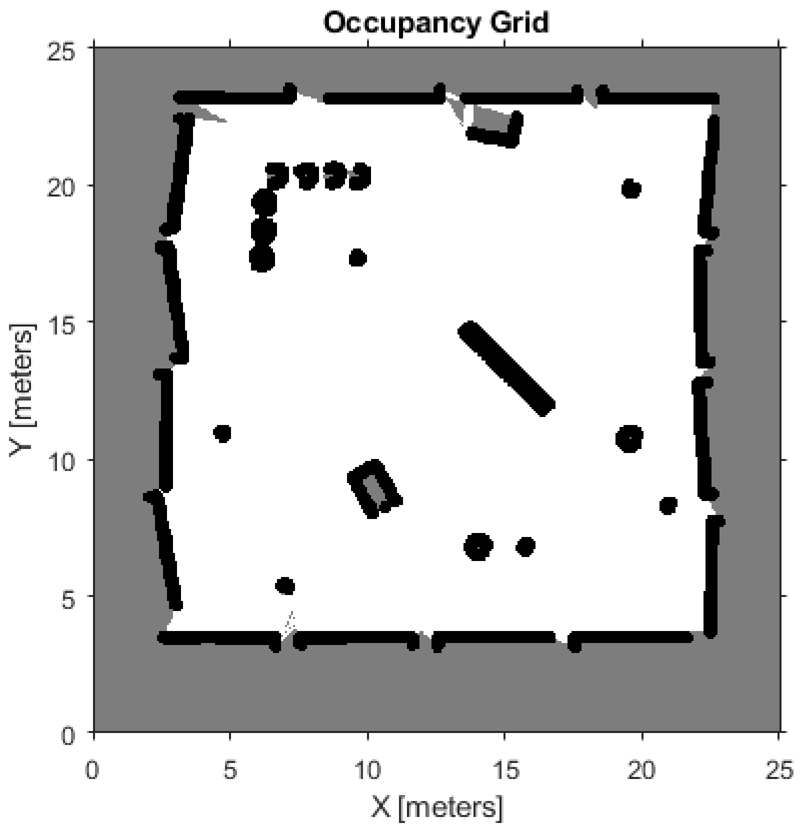
Inflate the Map Based on Robot Dimension

Copy and inflate the map to factor in the robot's size for obstacle avoidance

```
mapInflated = copy(map);
inflate(mapInflated, robotRadius);
```

Display inflated map.

```
show(mapInflated)
```



Associate the Existing PRM Object with the New Map and Set Parameters

Update PRM object with the newly inflated map and define other attributes. PRM uses `FreeThreshold` on the OccupancyGrid object to determine the obstacle free space and computes a path within this obstacle free space. The value of unknown cells in the `ternaryMap` is 0.5, while the default `FreeThreshold` on OccupancyGrid object `mapInflated` is 0.2. As a result the PRM will not plan a path in the unknown region.

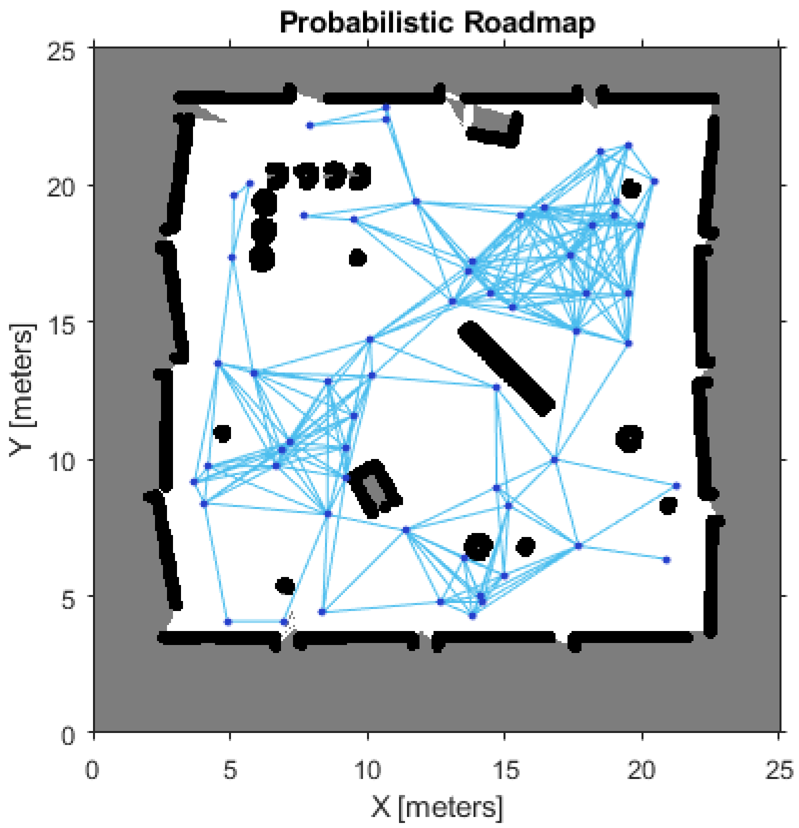
```
prm.Map = mapInflated;
```

Set the `NumNodes` and the `ConnectionDistance` properties

```
prm.NumNodes = 60;
prm.ConnectionDistance = 5;
```

Display PRM graph.

```
show(prm)
```



Find a Feasible Path on the Constructed PRM

Define start and end location on the map to find an obstacle free path.

```
startLocation = [7 22];
endLocation = [15 5];
```

Search for a solution between start and end location. Continue to add nodes until a path is found.

```
path = findpath(prm, startLocation, endLocation);
while isempty(path)
    prm.NumNodes = prm.NumNodes + 10;
    update(prm);
    path = findpath(prm, startLocation, endLocation);
end
```

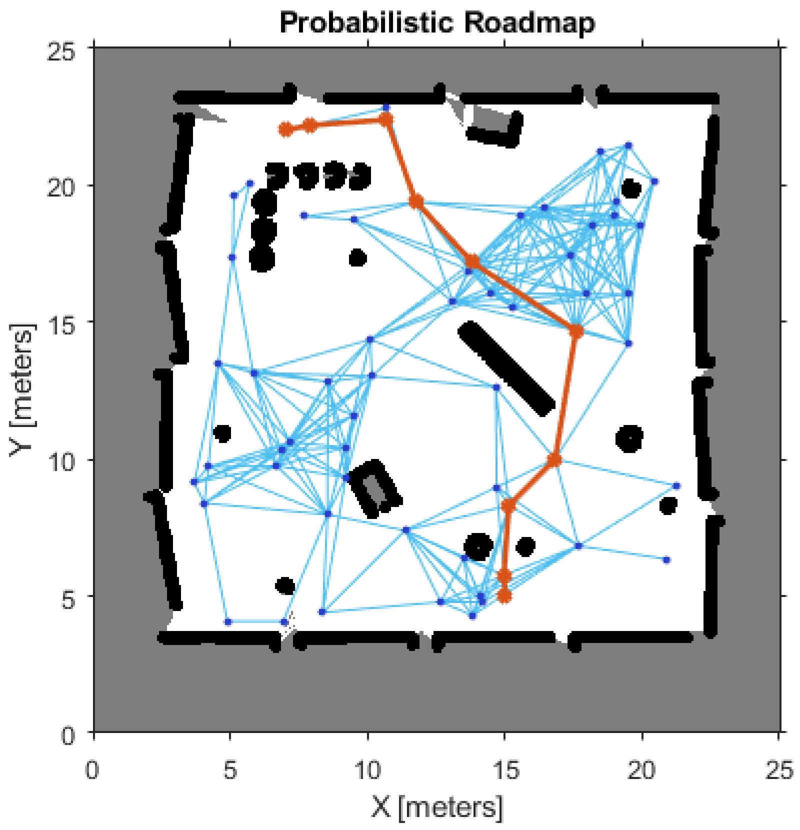
Display path.

```
path
```

```
path = 10×2

    7.0000   22.0000
    7.9059   22.1722
   10.6881   22.3734
   11.7508   19.3716
   13.7982   17.1659
   17.5826   14.6769
   16.8227    9.9964
   15.1329    8.3100
   14.9489    5.7648
   15.0000    5.0000
```

Display PRM solution.

```
show(prm)
```



See Also

- Path Following for a Differential Drive Robot
- Mapping With Known Poses