

# Tree Search for Path Planning

RRT (Rapidly-exploring Random Tree)

**State Space Vehicle Model**

Brian Lesko

Masters of Mechanical Engineering

Ohio State University

April 12, 2023

Dr. Levent Guvenc

Originally a Matlab and Simulink markdown file

By Brian Lesko, Graduate researcher and Teaching associate, Masters of mechanical engineering student

Inspired by Robust Controls of Mechatronic Systems, by Dr. Levent Guvenc

## Contents

1. Contents
2. Introduction
3. Background
- 4.
- 5.
- 6.
- 7.
- 8.
9. Conclusion

# Introduction

Path planning models are critical components in vehicle control systems and mobile robotics. These algorithms enable autonomous systems to find optimal or near-optimal routes from a starting point to a destination, while avoiding obstacles and adhering to certain constraints. Path planning models can have variations with RRT being a popular base choice. RRT is a sampling-based motion planning algorithm that efficiently explores the configuration space of the robot. It builds a tree structure that rapidly expands towards unexplored regions, which makes it particularly suitable for high-dimensional and complex environments. RRT\* is an extension of RRT that improves the optimality of the solution by continuously refining the tree structure. Another variation on the search algorithm is to set constraints. One example is the Dubins Car model, a widely-used method for modeling the constraints of non-holonomic systems, such as cars with limited turning radius. This model simplifies the path planning problem while maintaining crucial constraints, making it computationally efficient. Alternative models include the Reeds-Shepp Car and the Lattice model. A second complication is the implementation of occupancy maps, which are a common method for representing the environment in mobile robotics. They provide a discrete representation of the environment's occupied and free spaces, making it easier to integrate with path planning algorithms. By using occupancy maps, these algorithms can efficiently navigate around obstacles and find feasible paths.

In the following demonstration, the methods discussed will be employed to showcase their effectiveness in path planning and obstacle avoidance for a mobile robotics system. RRT and will be used to plan a path with the Dubins Car model to simulate the constraints of a non-holonomic system all while occupancy maps will be utilized to represent the environment, providing a discrete representation of occupied and free spaces.

## Background

<https://www.mathworks.com/help/nav/ug/plan-mobile-robot-paths-using-rrt.html>

## Setup Process

```
clc; clear; close all;
```

## Import an occupancy map

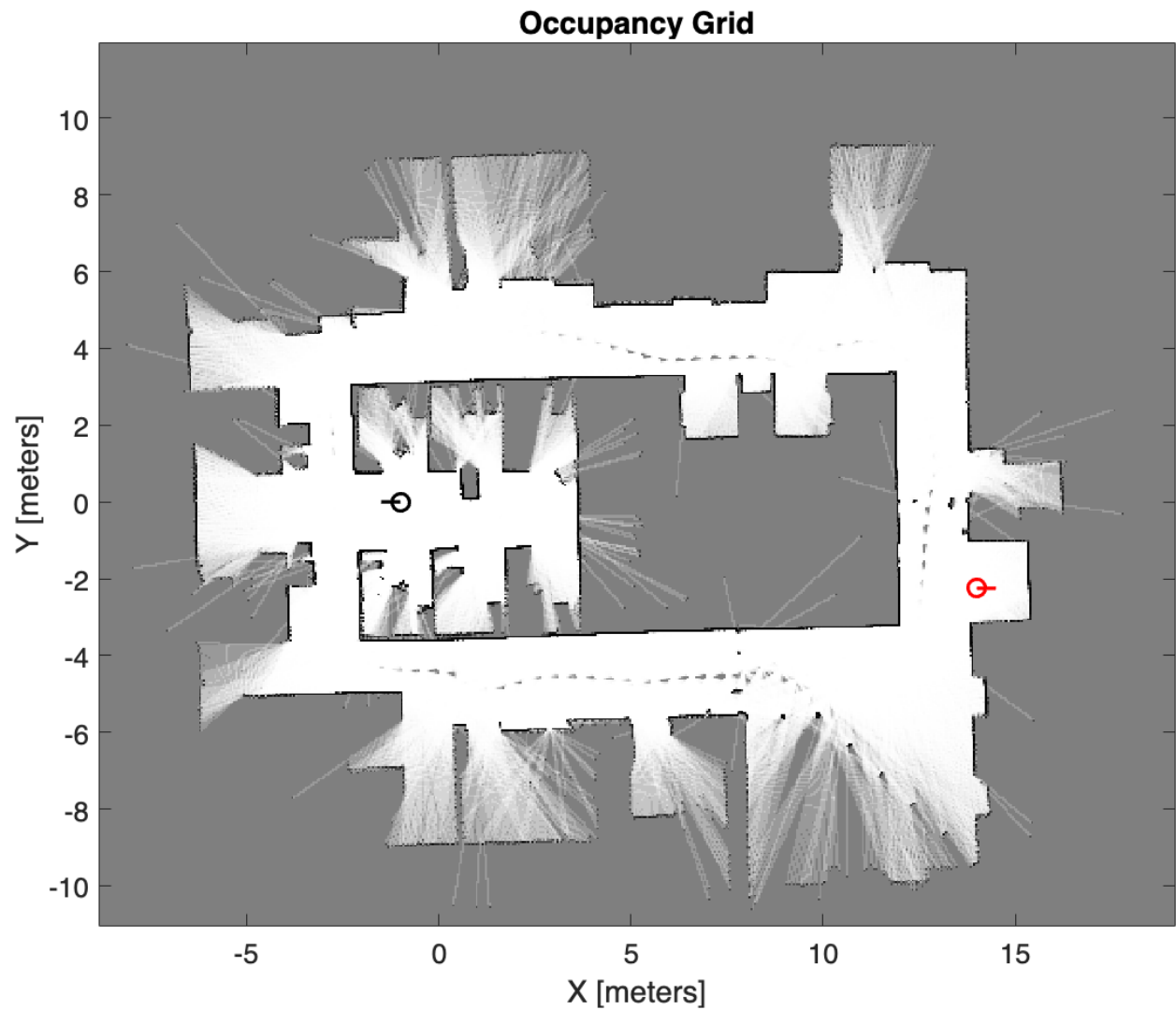
```
load("office_area_gridmap.mat", "occGrid")
```

## Determine Start and Goal Positions

```
start = [-1.0, 0.0, -pi];  
goal = [14, -2.25, 0];
```

## Visualize

```
figure()  
plt = plotMap(occGrid,start,goal)
```



```
plt =  
    1x2 Line array:  
      Line      Line
```

## Define the Vehicle Model constraints

Dubins State Space Model

```
bounds = [occGrid.XWorldLimits; occGrid.YWorldLimits; [-pi pi]];
ss = stateSpaceDubins(bounds);
```

```
ss.MinTurningRadius = .4;
```

## Path Planning Parameters

```
stateValidator = validatorOccupancyMap(ss);  
stateValidator.Map = occGrid;  
stateValidator.ValidationDistance = 0.05;  
planner = plannerRRT(ss, stateValidator);  
planner.MaxConnectionDistance = 2;  
planner.MaxIterations = 30000;  
planner.GoalReachedFcn = @exampleHelperCheckIfGoal;
```

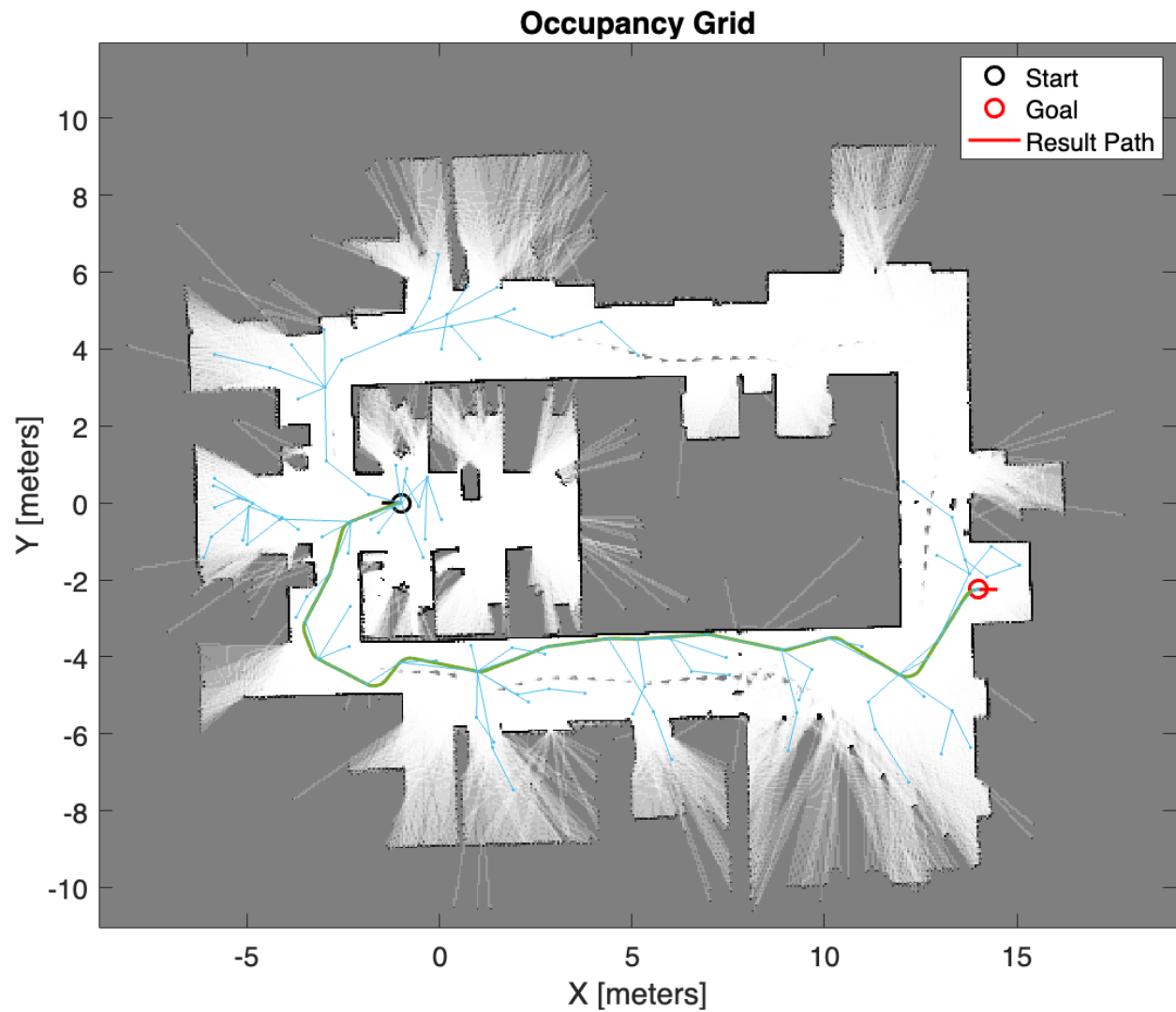
## Run the path generator

```
rng(0, 'twister')  
[pthObj, solnInfo] = plan(planner, start, goal);
```

## Plot the Path

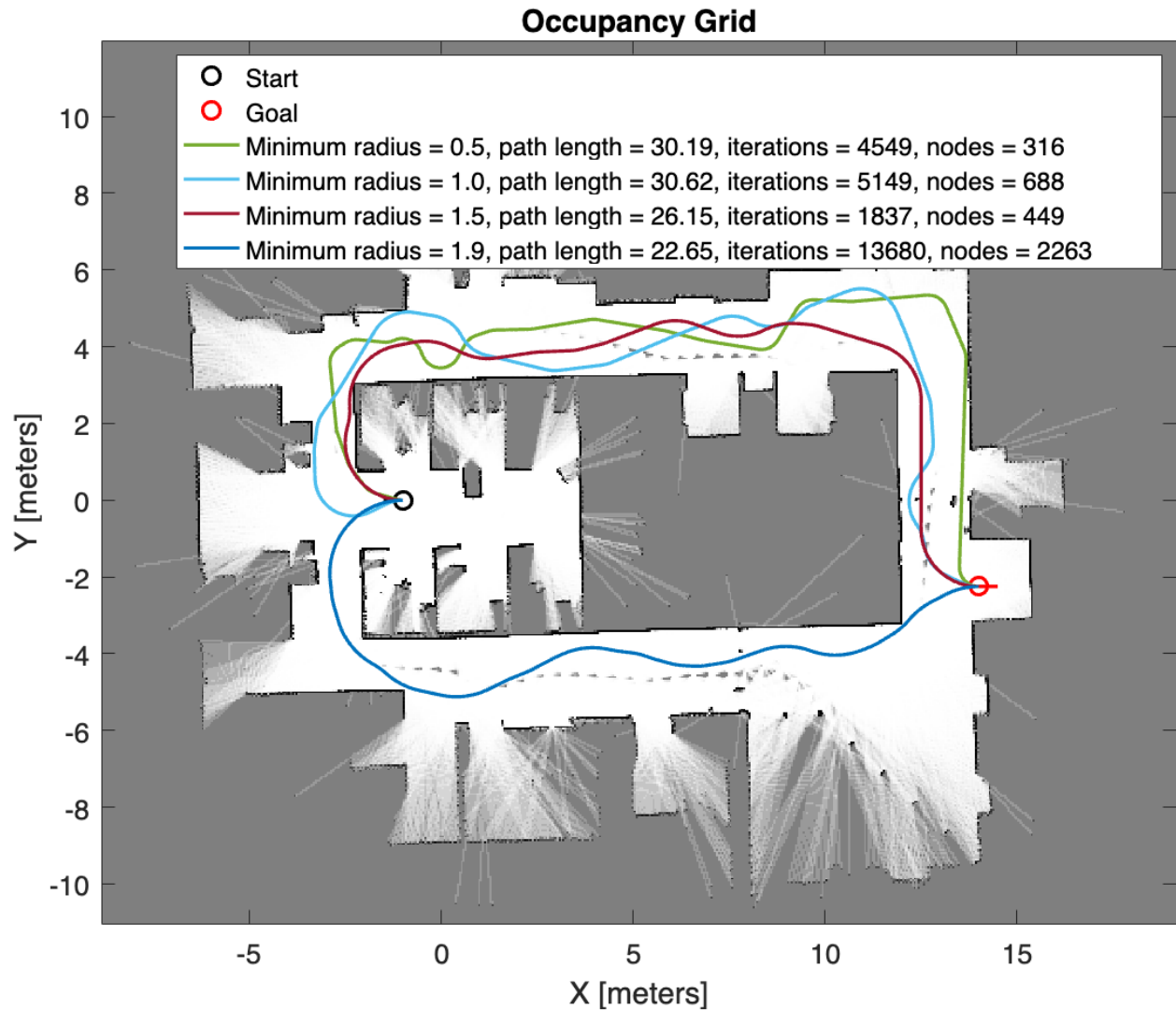
Show the occupancy map. Plot the search tree from the solnInfo. Interpolate and overlay the final path.

```
figure()  
plotMap(occGrid, start, goal);  
plotResult(pthObj, solnInfo);  
plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-');  
legend('Start', 'Goal', '', 'Result Path')
```



## Effect of minimum turning radius on RRT

```
radii = [0.5,1,1.5,1.9];  
plot = 1;  
[lengths, iterations, nodeCounts] =  
varyRadii(radii,plot,occGrid,start,goal);
```



As the turning radius minimum increases, the path length decreased while the iterations and nodes increased. For this occupancy map, a turning radius of  $>1.9$  results in no solvable paths to the goal because of the narrow hallways.

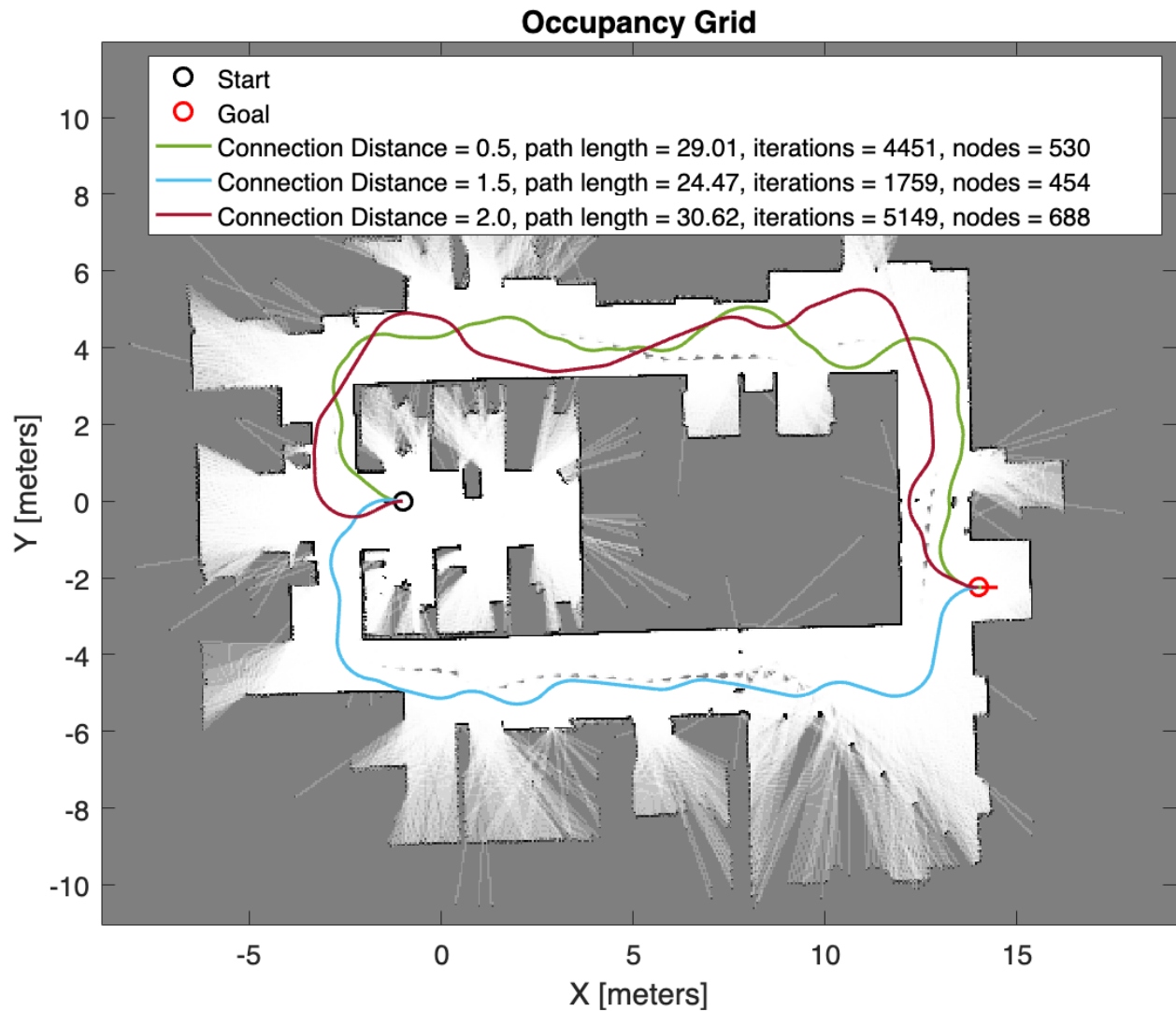
## Effect of Maximum Connection Distance

with radius = 1, maximum connection distances ( 0.5, 1, 1.5, 2)

```

connectionDistances = [.5,1.5,2];
plot = 1;
[lengths, iterations, nodeCounts] =
varyConnectionDistance(connectionDistances,plot,occGrid,start,goal);

```



The Number of nodes and iterations is primarily dependant on if the path is generated on the top or the bottom, but the nodes and interations seem to increase as the connection distance increases



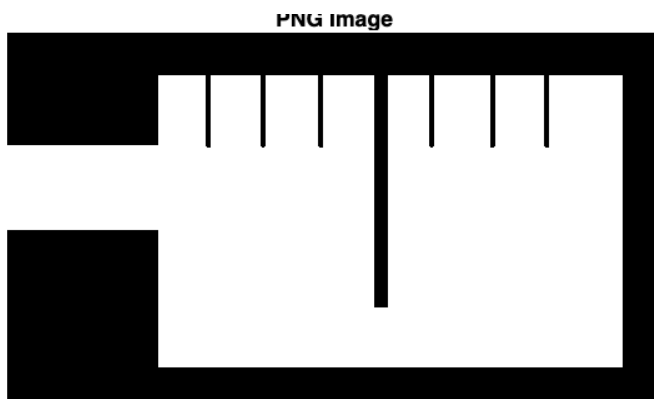
## Create an Occupancy Map from an image

Following the example from: <https://www.mathworks.com/help/nav/ref/binaryoccupancymap.html#buivr49e>.

Create a binary occupancy map

Note to self: `map = binaryOccupancyMap(p)` creates a grid from the values in matrix `p`, possibly useful for the robotics repository

```
close all
image = imread('plot.png');
imshow('plot.png'), title('PNG Image')
```

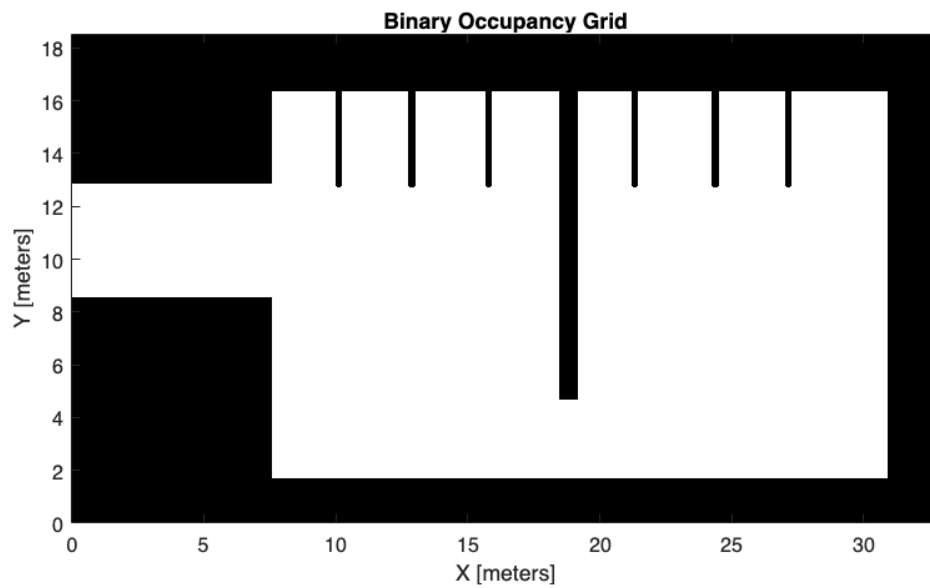


```
% Convert the RGB image to grayscale
gray_image = rgb2gray(image);

% Convert the grayscale image to a binary image
binary_image = imbinarize(gray_image);

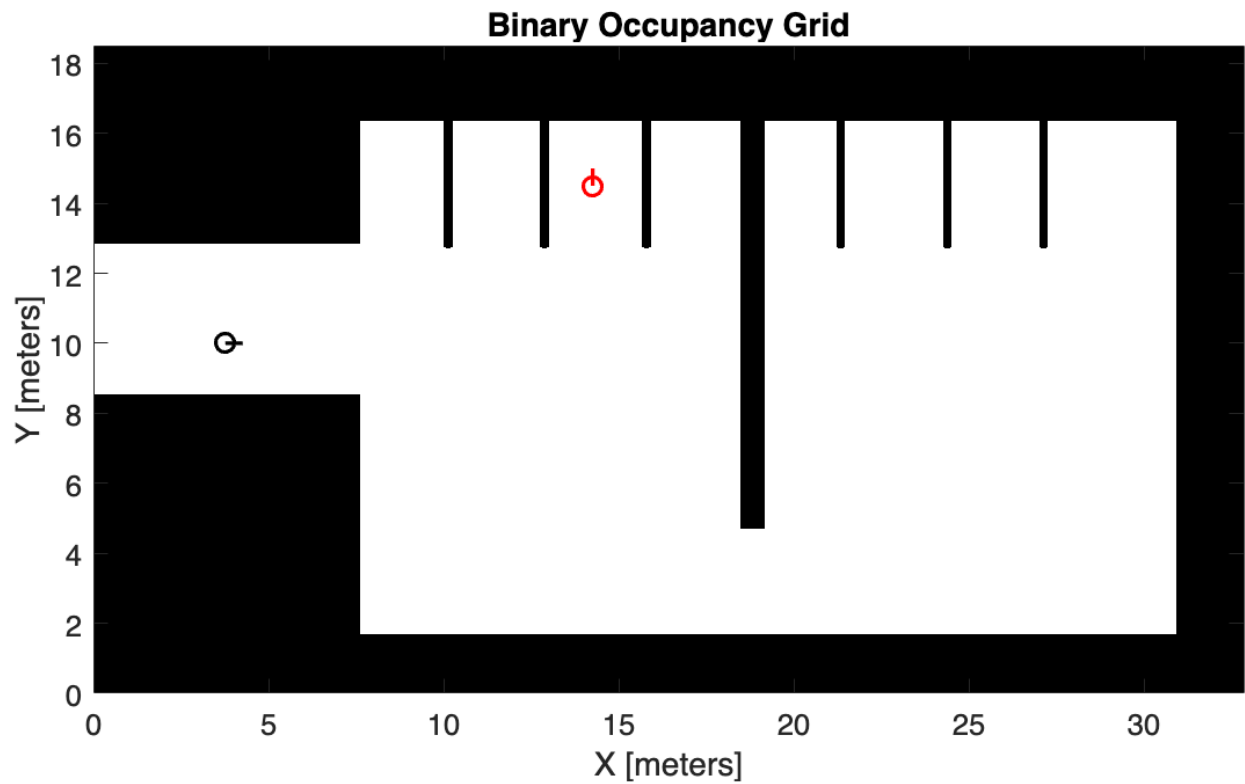
% Invert the binary image
inverted_binary_image = imcomplement(binary_image);

% Create the map
map = binaryOccupancyMap(inverted_binary_image,20); % 20 is the resolution
to scale properly
figure()
show(map)
```



Find the start and goal positions

```
start = [3.75, 10, 0];  
goal = [14.25, 14.5, pi/2];  
plotMap(map,start,goal)
```



```
ans =  
1x2 Line array:  
Line    Line
```

## RRT with the "Garage" occupancy map

```
minTurningRadius = 5;  
maxConnectionDistance = 2;
```

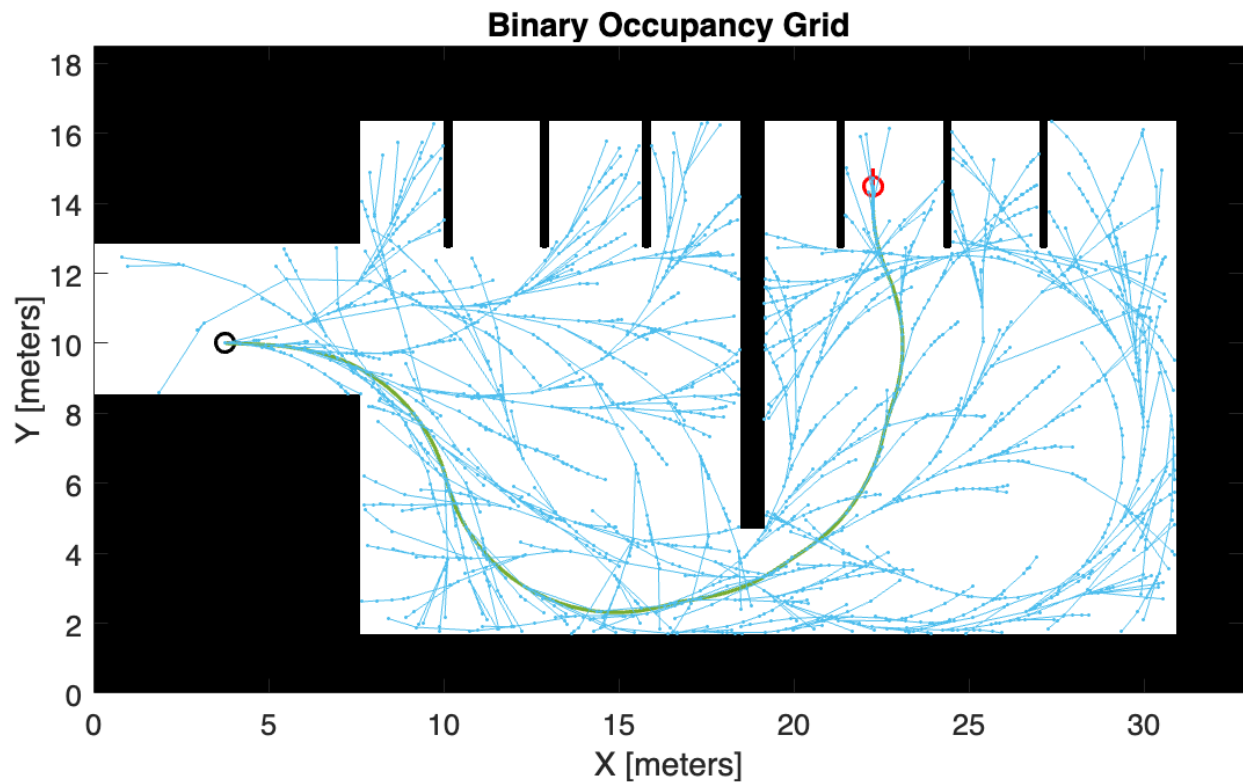
```
goal = [22.25, 14.5, pi/2];  
constrainedRRT(minTurningRadius,maxConnectionDistance,map,start,goal);
```

```
ans =
```

```
Line with properties:
```

```
    Color: [0.4660 0.6740 0.1880]  
    LineStyle: '-'  
    LineWidth: 2  
    Marker: 'none'  
    MarkerSize: 6  
    MarkerFaceColor: 'none'  
    XData: [3.7500 4.0380 4.3260 4.6139 4.9018 5.1857 5.4680 5.7487 6.0294 6.3345 6.6353 6.9306  
    YData: [10 9.9927 9.9819 9.9711 9.9603 9.9415 9.9066 9.8607 9.8144 9.7543 9.6753 9.5778 9.4
```

```
Show all properties
```



## RRT\* with the "Garage" occupancy map

But with RRT\*

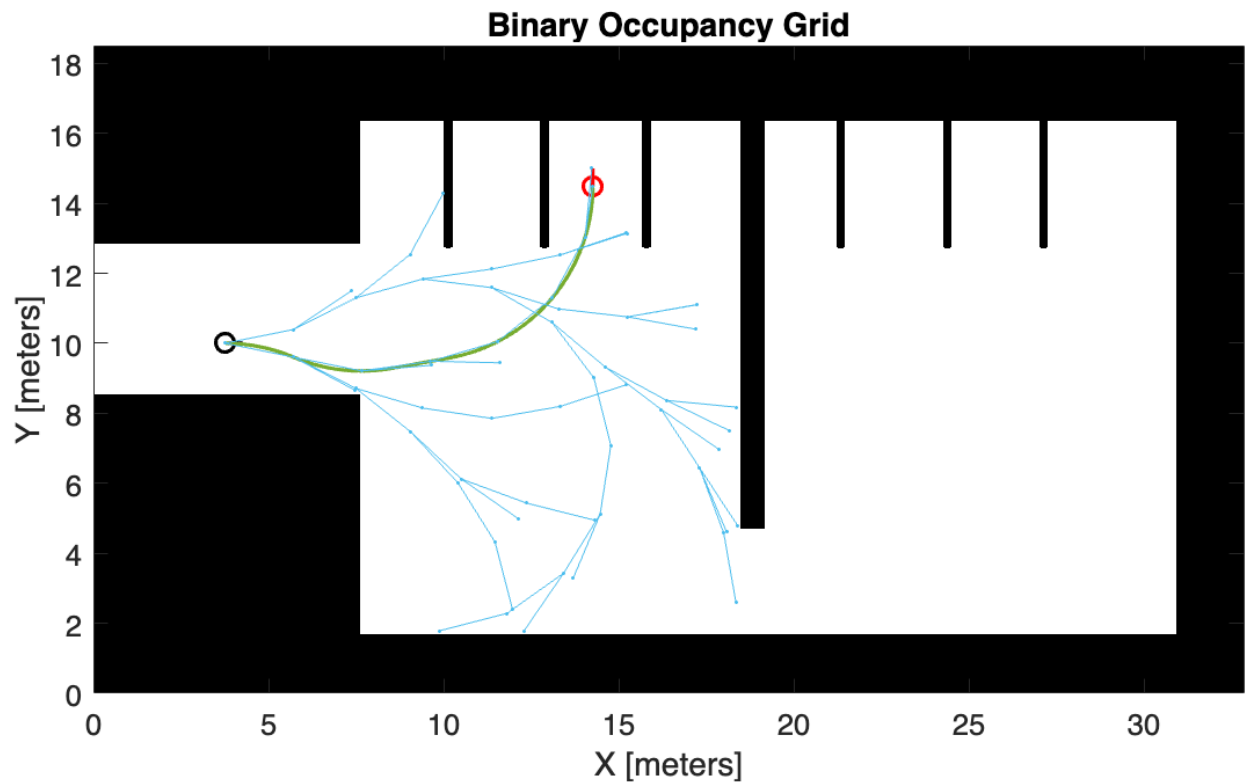
```
% set the mininum turning radius to 5 m. Set the maximum connection
distance to 2m
minTurningRadius = 5;
maxConnectionDistance = 2;
```

```
goal = [14.25, 14.5, pi/2];  
constrainedRRTStar(minTurningRadius,maxConnectionDistance,map,start,goal);
```

```
ans =
```

```
Line with properties:
```

```
    Color: [0.4660 0.6740 0.1880]  
    LineStyle: '-'  
    LineWidth: 2  
    Marker: 'none'  
    MarkerSize: 6  
    MarkerFaceColor: 'none'  
    XData: [3.7500 3.7944 3.8389 3.8833 3.9277 3.9721 4.0165 4.0609 4.1053 4.1496 4.1939 4.2381  
    YData: [10 9.9998 9.9992 9.9982 9.9968 9.9951 9.9929 9.9903 9.9874 9.9840 9.9803 9.9761 9.9721 9.9681 9.9641 9.9601 9.9561 9.9521 9.9481 9.9441 9.9401 9.9361 9.9321 9.9281 9.9241 9.9201 9.9161 9.9121 9.9081 9.9041 9.9001 9.8961 9.8921 9.8881 9.8841 9.8801 9.8761 9.8721 9.8681 9.8641 9.8601 9.8561 9.8521 9.8481 9.8441 9.8401 9.8361 9.8321 9.8281 9.8241 9.8201 9.8161 9.8121 9.8081 9.8041 9.8001 9.7961 9.7921 9.7881 9.7841 9.7801 9.7761 9.7721 9.7681 9.7641 9.7601 9.7561 9.7521 9.7481 9.7441 9.7401 9.7361 9.7321 9.7281 9.7241 9.7201 9.7161 9.7121 9.7081 9.7041 9.7001 9.6961 9.6921 9.6881 9.6841 9.6801 9.6761 9.6721 9.6681 9.6641 9.6601 9.6561 9.6521 9.6481 9.6441 9.6401 9.6361 9.6321 9.6281 9.6241 9.6201 9.6161 9.6121 9.6081 9.6041 9.6001 9.5961 9.5921 9.5881 9.5841 9.5801 9.5761 9.5721 9.5681 9.5641 9.5601 9.5561 9.5521 9.5481 9.5441 9.5401 9.5361 9.5321 9.5281 9.5241 9.5201 9.5161 9.5121 9.5081 9.5041 9.5001 9.4961 9.4921 9.4881 9.4841 9.4801 9.4761 9.4721 9.4681 9.4641 9.4601 9.4561 9.4521 9.4481 9.4441 9.4401 9.4361 9.4321 9.4281 9.4241 9.4201 9.4161 9.4121 9.4081 9.4041 9.4001 9.3961 9.3921 9.3881 9.3841 9.3801 9.3761 9.3721 9.3681 9.3641 9.3601 9.3561 9.3521 9.3481 9.3441 9.3401 9.3361 9.3321 9.3281 9.3241 9.3201 9.3161 9.3121 9.3081 9.3041 9.3001 9.2961 9.2921 9.2881 9.2841 9.2801 9.2761 9.2721 9.2681 9.2641 9.2601 9.2561 9.2521 9.2481 9.2441 9.2401 9.2361 9.2321 9.2281 9.2241 9.2201 9.2161 9.2121 9.2081 9.2041 9.2001 9.1961 9.1921 9.1881 9.1841 9.1801 9.1761 9.1721 9.1681 9.1641 9.1601 9.1561 9.1521 9.1481 9.1441 9.1401 9.1361 9.1321 9.1281 9.1241 9.1201 9.1161 9.1121 9.1081 9.1041 9.1001 9.0961 9.0921 9.0881 9.0841 9.0801 9.0761 9.0721 9.0681 9.0641 9.0601 9.0561 9.0521 9.0481 9.0441 9.0401 9.0361 9.0321 9.0281 9.0241 9.0201 9.0161 9.0121 9.0081 9.0041 9.0001 8.9961 8.9921 8.9881 8.9841 8.9801 8.9761 8.9721 8.9681 8.9641 8.9601 8.9561 8.9521 8.9481 8.9441 8.9401 8.9361 8.9321 8.9281 8.9241 8.9201 8.9161 8.9121 8.9081 8.9041 8.9001 8.8961 8.8921 8.8881 8.8841 8.8801 8.8761 8.8721 8.8681 8.8641 8.8601 8.8561 8.8521 8.8481 8.8441 8.8401 8.8361 8.8321 8.8281 8.8241 8.8201 8.8161 8.8121 8.8081 8.8041 8.8001 8.7961 8.7921 8.7881 8.7841 8.7801 8.7761 8.7721 8.7681 8.7641 8.7601 8.7561 8.7521 8.7481 8.7441 8.7401 8.7361 8.7321 8.7281 8.7241 8.7201 8.7161 8.7121 8.7081 8.7041 8.7001 8.6961 8.6921 8.6881 8.6841 8.6801 8.6761 8.6721 8.6681 8.6641 8.6601 8.6561 8.6521 8.6481 8.6441 8.6401 8.6361 8.6321 8.6281 8.6241 8.6201 8.6161 8.6121 8.6081 8.6041 8.6001 8.5961 8.5921 8.5881 8.5841 8.5801 8.5761 8.5721 8.5681 8.5641 8.5601 8.5561 8.5521 8.5481 8.5441 8.5401 8.5361 8.5321 8.5281 8.5241 8.5201 8.5161 8.5121 8.5081 8.5041 8.5001 8.4961 8.4921 8.4881 8.4841 8.4801 8.4761 8.4721 8.4681 8.4641 8.4601 8.4561 8.4521 8.4481 8.4441 8.4401 8.4361 8.4321 8.4281 8.4241 8.4201 8.4161 8.4121 8.4081 8.4041 8.4001 8.3961 8.3921 8.3881 8.3841 8.3801 8.3761 8.3721 8.3681 8.3641 8.3601 8.3561 8.3521 8.3481 8.3441 8.3401 8.3361 8.3321 8.3281 8.3241 8.3201 8.3161 8.3121 8.3081 8.3041 8.3001 8.2961 8.2921 8.2881 8.2841 8.2801 8.2761 8.2721 8.2681 8.2641 8.2601 8.2561 8.2521 8.2481 8.2441 8.2401 8.2361 8.2321 8.2281 8.2241 8.2201 8.2161 8.2121 8.2081 8.2041 8.2001 8.1961 8.1921 8.1881 8.1841 8.1801 8.1761 8.1721 8.1681 8.1641 8.1601 8.1561 8.1521 8.1481 8.1441 8.1401 8.1361 8.1321 8.1281 8.1241 8.1201 8.1161 8.1121 8.1081 8.1041 8.1001 8.0961 8.0921 8.0881 8.0841 8.0801 8.0761 8.0721 8.0681 8.0641 8.0601 8.0561 8.0521 8.0481 8.0441 8.0401 8.0361 8.0321 8.0281 8.0241 8.0201 8.0161 8.0121 8.0081 8.0041 8.0001 7.9961 7.9921 7.9881 7.9841 7.9801 7.9761 7.9721 7.9681 7.9641 7.9601 7.9561 7.9521 7.9481 7.9441 7.9401 7.9361 7.9321 7.9281 7.9241 7.9201 7.9161 7.9121 7.9081 7.9041 7.9001 7.8961 7.8921 7.8881 7.8841 7.8801 7.8761 7.8721 7.8681 7.8641 7.8601 7.8561 7.8521 7.8481 7.8441 7.8401 7.8361 7.8321 7.8281 7.8241 7.8201 7.8161 7.8121 7.8081 7.8041 7.8001 7.7961 7.7921 7.7881 7.7841 7.7801 7.7761 7.7721 7.7681 7.7641 7.7601 7.7561 7.7521 7.7481 7.7441 7.7401 7.7361 7.7321 7.7281 7.7241 7.7201 7.7161 7.7121 7.7081 7.7041 7.7001 7.6961 7.6921 7.6881 7.6841 7.6801 7.6761 7.6721 7.6681 7.6641 7.6601 7.6561 7.6521 7.6481 7.6441 7.6401 7.6361 7.6321 7.6281 7.6241 7.6201 7.6161 7.6121 7.6081 7.6041 7.6001 7.5961 7.5921 7.5881 7.5841 7.5801 7.5761 7.5721 7.5681 7.5641 7.5601 7.5561 7.5521 7.5481 7.5441 7.5401 7.5361 7.5321 7.5281 7.5241 7.5201 7.5161 7.5121 7.5081 7.5041 7.5001 7.4961 7.4921 7.4881 7.4841 7.4801 7.4761 7.4721 7.4681 7.4641 7.4601 7.4561 7.4521 7.4481 7.4441 7.4401 7.4361 7.4321 7.4281 7.4241 7.4201 7.4161 7.4121 7.4081 7.4041 7.4001 7.3961 7.3921 7.3881 7.3841 7.3801 7.3761 7.3721 7.3681 7.3641 7.3601 7.3561 7.3521 7.3481 7.3441 7.3401 7.3361 7.3321 7.3281 7.3241 7.3201 7.3161 7.3121 7.3081 7.3041 7.3001 7.2961 7.2921 7.2881 7.2841 7.2801 7.2761 7.2721 7.2681 7.2641 7.2601 7.2561 7.2521 7.2481 7.2441 7.2401 7.2361 7.2321 7.2281 7.2241 7.2201 7.2161 7.2121 7.2081 7.2041 7.2001 7.1961 7.1921 7.1881 7.1841 7.1801 7.1761 7.1721 7.1681 7.1641 7.1601 7.1561 7.1521 7.1481 7.1441 7.1401 7.1361 7.1321 7.1281 7.1241 7.1201 7.1161 7.1121 7.1081 7.1041 7.1001 7.0961 7.0921 7.0881 7.0841 7.0801 7.0761 7.0721 7.0681 7.0641 7.0601 7.0561 7.0521 7.0481 7.0441 7.0401 7.0361 7.0321 7.0281 7.0241 7.0201 7.0161 7.0121 7.0081 7.0041 7.0001 6.9961 6.9921 6.9881 6.9841 6.9801 6.9761 6.9721 6.9681 6.9641 6.9601 6.9561 6.9521 6.9481 6.9441 6.9401 6.9361 6.9321 6.9281 6.9241 6.9201 6.9161 6.9121 6.9081 6.9041 6.9001 6.8961 6.8921 6.8881 6.8841 6.8801 6.8761 6.8721 6.8681 6.8641 6.8601 6.8561 6.8521 6.8481 6.8441 6.8401 6.8361 6.8321 6.8281 6.8241 6.8201 6.8161 6.8121 6.8081 6.8041 6.8001 6.7961 6.7921 6.7881 6.7841 6.7801 6.7761 6.7721 6.7681 6.7641 6.7601 6.7561 6.7521 6.7481 6.7441 6.7401 6.7361 6.7321 6.7281 6.7241 6.7201 6.7161 6.7121 6.7081 6.7041 6.7001 6.6961 6.6921 6.6881 6.6841 6.6801 6.6761 6.6721 6.6681 6.6641 6.6601 6.6561 6.6521 6.6481 6.6441 6.6401 6.6361 6.6321 6.6281 6.6241 6.6201 6.6161 6.6121 6.6081 6.6041 6.6001 6.5961 6.5921 6.5881 6.5841 6.5801 6.5761 6.5721 6.5681 6.5641 6.5601 6.5561 6.5521 6.5481 6.5441 6.5401 6.5361 6.5321 6.5281 6.5241 6.5201 6.5161 6.5121 6.5081 6.5041 6.5001 6.4961 6.4921 6.4881 6.4841 6.4801 6.4761 6.4721 6.4681 6.4641 6.4601 6.4561 6.4521 6.4481 6.4441 6.4401 6.4361 6.4321 6.4281 6.4241 6.4201 6.4161 6.4121 6.4081 6.4041 6.4001 6.3961 6.3921 6.3881 6.3841 6.3801 6.3761 6.3721 6.3681 6.3641 6.3601 6.3561 6.3521 6.3481 6.3441 6.3401 6.3361 6.3321 6.3281 6.3241 6.3201 6.3161 6.3121 6.3081 6.3041 6.3001 6.2961 6.2921 6.2881 6.2841 6.2801 6.2761 6.2721 6.2681 6.2641 6.2601 6.2561 6.2521 6.2481 6.2441 6.2401 6.2361 6.2321 6.2281 6.2241 6.2201 6.2161 6.2121 6.2081 6.2041 6.2001 6.1961 6.1921 6.1881 6.1841 6.1801 6.1761 6.1721 6.1681 6.1641 6.1601 6.1561 6.1521 6.1481 6.1441 6.1401 6.1361 6.1321 6.1281 6.1241 6.1201 6.1161 6.1121 6.1081 6.1041 6.1001 6.0961 6.0921 6.0881 6.0841 6.0801 6.0761 6.0721 6.0681 6.0641 6.0601 6.0561 6.0521 6.0481 6.0441 6.0401 6.0361 6.0321 6.0281 6.0241 6.0201 6.0161 6.0121 6.0081 6.0041 6.0001 5.9961 5.9921 5.9881 5.9841 5.9801 5.9761 5.9721 5.9681 5.9641 5.9601 5.9561 5.9521 5.9481 5.9441 5.9401 5.9361 5.9321 5.9281 5.9241 5.9201 5.9161 5.9121 5.9081 5.9041 5.9001 5.8961 5.8921 5.8881 5.8841 5.8801 5.8761 5.8721 5.8681 5.8641 5.8601 5.8561 5.8521 5.8481 5.8441 5.8401 5.8361 5.8321 5.8281 5.8241 5.8201 5.8161 5.8121 5.8081 5.8041 5.8001 5.7961 5.7921 5.7881 5.7841 5.7801 5.7761 5.7721 5.7681 5.7641 5.7601 5.7561 5.7521 5.7481 5.7441 5.7401 5.7361 5.7321 5.7281 5.7241 5.7201 5.7161 5.7121 5.7081 5.7041 5.7001 5.6961 5.6921 5.6881 5.6841 5.6801 5.6761 5.6721 5.6681 5.6641 5.6601 5.6561 5.6521 5.6481 5.6441 5.6401 5.6361 5.6321 5.6281 5.6241 5.6201 5.6161 5.6121 5.6081 5.6041 5.6001 5.5961 5.5921 5.5881 5.5841 5.5801 5.5761 5.5721 5.5681 5.5641 5.5601 5.5561 5.5521 5.5481 5.5441 5.5401 5.5361 5.5321 5.5281 5.5241 5.5201 5.5161 5.5121 5.5081 5.5041 5.5001 5.4961 5.4921 5.4881 5.4841 5.4801 5.4761 5.4721 5.4681 5.4641 5.4601 5.4561 5.4521 5.4481 5.4441 5.4401 5.4361 5.4321 5.4281 5.4241 5.4201 5.4161 5.4121 5.4081 5.4041 5.4001 5.3961 5.3921 5.3881 5.3841 5.3801 5.3761 5.3721 5.3681 5.3641 5.3601 5.3561 5.3521 5.3481 5.3441 5.3401 5.3361 5.3321 5.3281 5.3241 5.3201 5.3161 5.3121 5.3081 5.3041 5.3001 5.2961 5.2921 5.2881 5.2841 5.2801 5.2761 5.2721 5.2681 5.2641 5.2601 5.2561 5.2521 5.2481 5.2441 5.2401 5.2361 5.2321 5.2281 5.2241 5.2201 5.2161 5.2121 5.2081 5.2041 5.2001 5.1961 5.1921 5.1881 5.1841 5.1801 5.1761 5.1721 5.1681 5.1641 5.1601 5.1561 5.1521 5.1481 5.1441 5.1401 5.1361 5.1321 5.1281 5.1241 5.1201 5.1161 5.1121 5.1081 5.1041 5.1001 5.0961 5.0921 5.0881 5.0841 5.0801 5.0761 5.0721 5.0681 5.0641 5.0601 5.0561 5.0521 5.0481 5.0441 5.0401 5.0361 5.0321 5.0281 5.0241 5.0201 5.0161 5.0121 5.0081 5.0041 5.0001 4.9961 4.9921 4.9881 4.9841 4.9801 4.9761 4.9721 4.9681 4.9641 4.9601 4.9561 4.9521 4.9481 4.9441 4.9401 4.9361 4.9321 4.9281 4.9241 4.9201 4.9161 4.9121 4.9081 4.9041 4.9001 4.8961 4.8921 4.8881 4.8841 4.8801 4.8761 4.8721 4.8681 4.8641 4.8601 4.8561 4.8521 4.8481 4.8441 4.8401 4.8361 4.8321 4.8281 4.8241 4.8201 4.8161 4.8121 4.8081 4.8041 4.8001 4.7961 4.7921 4.7881 4.7841 4.7801 4.7761 4.7721 4.7681 4.7641 4.7601 4.7561 4.7521 4.7481 4.7441 4.7401 4.7361 4.7321 4.7281 4.7241 4.7201 4.7161 4.7121 4.7081 4.7041 4.7001 4.6961 4.6921 4.6881 4.6841 4.6801 4.6761 4.6721 4.6681 4.6641 4.6601 4.6561 4.6521 4.6481 4.6441 4.6401 4.6361 4.6321 4.6281 4.6241 4.6201 4.6161 4.6121 4.6081 4.6041 4.6001 4.5961 4.5921 4.5881 4.5841 4.5801 4.5761 4.5721 4.5681 4.5641 4.5601 4.5561 4.5521 4.5481 4.5441 4.5401 4.5361 4.5321 4.5281 4.5241 4.5201 4.5161 4.5121 4.5081 4.5041 4.5001 4.4961 4.4921 4.4881 4.4841 4.4801 4.4761 4.4721 4.4681 4.4641 4.4601 4.4561 4.4521 4.4481 4.4441 4.4401 4.4361 4.4321 4.4281 4.4241 4.4201 4.4161 4.4121 4.4081 4.4041 4.4001 4.3961 4.3921 4.3881 4.3841 4.3801 4.3761 4.3721 4.3681 4.3641 4.3601 4.3561 4.3521 4.3481 4.3441 4.3401 4.3361 4.3321 4.3281 4.3241 4.3201 4.3161 4.3121 4.3081 4.3041 4.3001 4.2961 4.2921 4.2881 4.2841 4.2801 4.2761 4.2721 4.2681 4.2641 4.2601 4.2561 4.2521 4.2481 4.2441 4.2401 4.2361 4.2321 4.2281 4.2241 4.2201 4.2161 4.2121 4.2081 4.2041 4.2001 4.1961 4.1921 4.1881 4.1841 4.1801 4.1761 4.1721 4.1681 4.1641 4.1601 4.1561 4.1521 4.1481 4.1441 4.1401 4.1361 4.1321 4.1281 4.1241 4.1201 4.1161 4.1121 4.1081 4.1041 4.1001 4.0961 4.0921 4.0881 4.0841 4.0801 4.0761 4.0721 4.0681 4.0641 4.0601 4.0561 4.0521 4.0481 4.0441 4.0401 4.0361 4.0321 4.0281 4.0241 4.0201 4.0161 4.0121 4.0081 4.0041 4.0001 3.9961 3.9921 3.9881 3.9841 3.9801 3.9761 3.9721 3.9681 3.9641 3.9601 3.9561 3.9521 3.9481 3.9441 3.9401 3.9361 3.9321 3.9281 3.9241 3.9201 3.9161 3.9121 3.9081 3.9041 3.9001 3.8961 3.8921 3.8881 3.8841 3.8801 3.8761 3.8721 3.8681 3.8641 3.8601 3.8561 3.8521 3.8481 3.8441 3.8401 3.8361 3.8321 3.8281 3.8241 3.8201 3.8161 3.8121 3.8081 3.8041 3.8001 3.7961 3.7921 3.7881 3.7841 3.7801 3.7761 3.7721 3.7681 3.7641 3.7601 3.7561 3.7521 3.7481 3.7441 3.7401 3.7361 3.7321 3.7281 3.7241 3.7201 3.7161 3.7121 3.7081 3.7041 3.7001 3.6961 3.6921 3.6881 3.6841 3.6801 3.6761 3.6721 3.6681 3.6641 3.6601 3.6561 3.6521 3.6481 3.6441 3.6401 3.6361 3.6321 3.6281 3.6241 3.6201 3.6161 3.6121 3.6081 3.6041 3.6001 3.5961 3.5921 3.5881 3.5841 3.5801 3.5761 3.5721 3.5681 3.5641 3.5601 3.5561 3.5521 3.5481 3.5441 3.5401 3.5361 3.5321 3.5281 3.5241 3.5201 3.5161 3.5121 3.5081 3.5041 3.5001 3.4961 3.4921 3.4881 3.4841 3.4801 3.4761 3.4721 3.4681 3.4641 3.4601 3.4561 3.4521 3.4481 3.4441 3.4401 3.4361 3.4321 3.4281 3.4241 3.4201 3.4161 3.4121 3.4081 3.4041 3.4001 3.3961 3.3921 3.3881 3.3841 3.3801 3.3761 3.3721 3.3681 3.3641 3.3601 3.3561 3.3521 3.3481 3.3441 3.3401 3.3361 3.3321 3.3281 3.3241 3.3201 3.3161 3.3121 3.3081 3.3041 3.3001 3.2961 3.2921 3.2881 3.2841 3.2801 3.2761 3.2721 3.2681 3.2641 3.2601 3.2561 3.2521 3.2481 3.2441 3.2401 3.2361 3.2321 3.2281 3.2241 3.2201 3.2161 3.2121 3.2081 3.2041 3.2001 3.1961 3.1921 3.1881 3.1841 3.1801 3.1761 3.1721 3.1681 3.1641 3.1601 3.1561 3.1521 3.1481 3.1441 3.1401 3.1361 3.1321 3
```

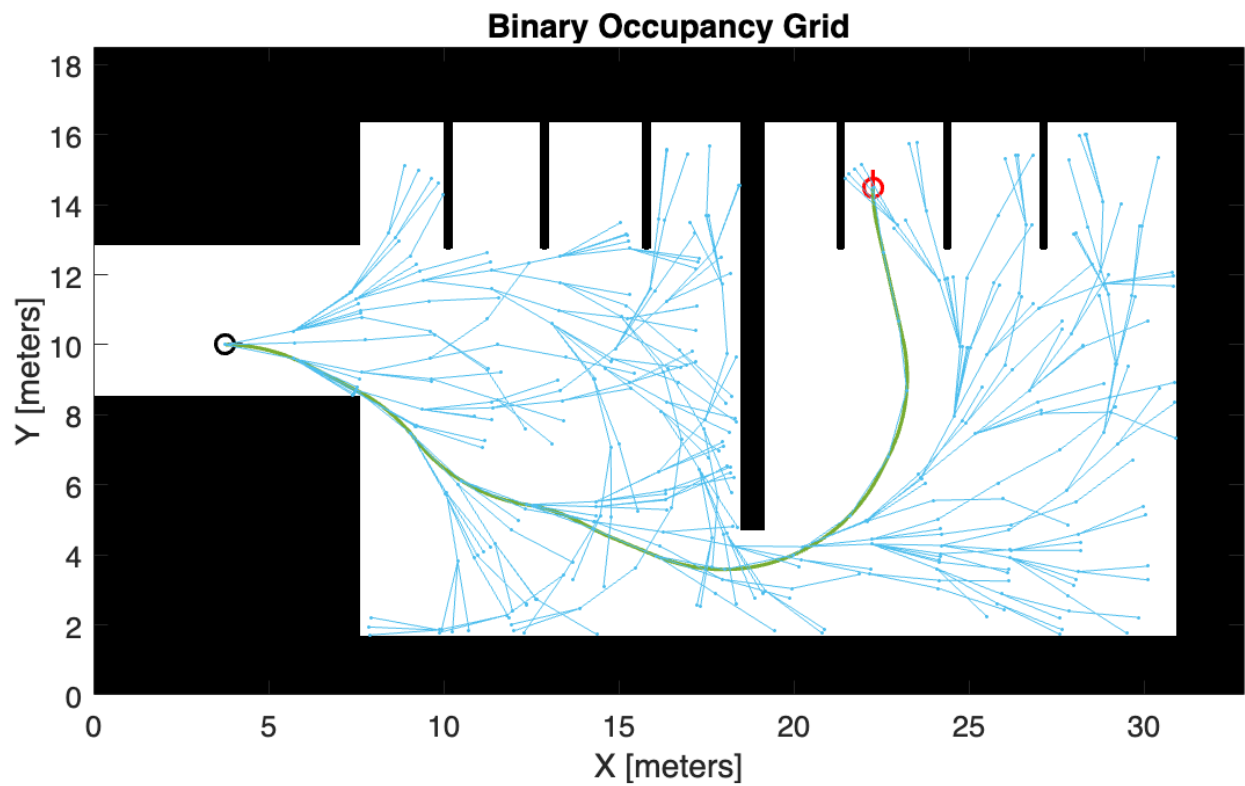


```
goal = [22.25, 14.5, pi/2];
constrainedRRTStar(minTurningRadius,maxConnectionDistance,map,start,goal);
```

```
ans =
  Line with properties:
      Color: [0.4660 0.6740 0.1880]
  LineStyle: '-'
  LineWidth: 2
  Marker: 'none'
```

```
MarkerSize: 6
MarkerFaceColor: 'none'
XData: [3.7500 3.8500 3.9499 4.0498 4.1496 4.2492 4.3486 4.4477 4.5466 4.6451 4.7433 4.8411
YData: [10 9.9990 9.9960 9.9910 9.9840 9.9750 9.9640 9.9511 9.9361 9.9192 9.9003 9.8795 9.8587
```

Show all properties



The limitations include not accounting for the width of the vehicle, the RRT\* algorithm creates a smoother path



# Functions

Plot Result Path and tree

```
function h = plotResult(pthObj, solnInfo)
    hold on

    % Search tree
    % plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-');

    % Interpolate and plot path
    interpolate(pthObj,300)
    h = plot(pthObj.States(:,1), pthObj.States(:,2), '-', 'LineWidth', 2);
end
```

## Plot occupancy map

```
function h = plotMap(occGrid,start,goal)
    % occupancy grid
    show(occGrid)

    % Show the start and goal positions of the robot
    hold on
    h(1) = plot(start(1), start(2), 'ko','LineWidth',2,MarkerSize=10);
    h(2) = plot(goal(1), goal(2), 'ro','LineWidth',2,MarkerSize=10);

    % Show the start and goal headings
    r = 0.5;
    plot([start(1), start(1) + r*cos(start(3))], [start(2), start(2) +
r*sin(start(3))], 'k-', 'LineWidth',2)
    plot([goal(1), goal(1) + r*cos(goal(3))], [goal(2), goal(2) +
r*sin(goal(3))], 'r-', 'LineWidth',2)
    set(gcf, 'Position', [0 0 2500 2500])
    set(gca, 'FontSize',16)
end
```

## Vary the turning radius minimum

```
function [lengths, iterations, nodeCounts] =
varyRadii(radii,plot,occGrid,start,goal)
    loop = 1;
    % Plotting only
    if plot == 1
        handles = [];
        % Initialize a cell array to store legend entries
        legendEntries = cell(1, length(radii)+2);
        % Add 'Start' and 'Goal' to legendEntries
        legendEntries{1} = 'Start';
        legendEntries{2} = 'Goal';
```

```

end

close all; figure()
handles = plotMap(occGrid,start,goal);
for radius = radii
    pthObj = 0;
    solnInfo = 0;

    % input occGrid, radius
    % Setup Model
    bounds = [occGrid.XWorldLimits; occGrid.YWorldLimits; [-pi pi]];
    ss = stateSpaceDubins(bounds);
    ss.MinTurningRadius = radius; % what is changing every round

    % Setup RRT
    stateValidator = validatorOccupancyMap(ss);
    stateValidator.Map = occGrid;
    stateValidator.ValidationDistance = 0.05;
    planner = plannerRRT(ss, stateValidator);
    planner.MaxConnectionDistance = 2;
    planner.MaxIterations = 30000;
    planner.GoalReachedFcn = @exampleHelperCheckIfGoal;

    % Solve RRT
    rng(0,'twister')
    [pthObj, solnInfo] = plan(planner, start, goal);

    % Save Results
    lengths(loop) = pthObj.pathLength; iterations(loop) =
solnInfo.NumIterations; nodeCounts(loop) = solnInfo.NumNodes;

    % Plotting only
    if plot == 1
        % Create a legend entry
        legendEntries{loop+2} = sprintf('Minimum radius = %.1f, path
length = %.2f, iterations = %d, nodes = %d',radius, lengths(loop),
iterations(loop), nodeCounts(loop));
        % Plot Results
        hold on;
        handles(loop + 2) = plotResult(pthObj, solnInfo); % Store the
handle
    end

    loop = loop + 1;
end
if plot == 1
    legend(handles,legendEntries); end
end

```

## Vary Connection Distance

```
function [lengths, iterations, nodeCounts] =  
varyConnectionDistance(connectionDistances, plot, occGrid, start, goal)  
    loop = 1;  
    % Plotting only  
    if plot == 1  
        handles = [];  
        % Initialize a cell array to store legend entries  
        legendEntries = cell(1, length(connectionDistances)+2);  
        % Add 'Start' and 'Goal' to legendEntries  
        legendEntries{1} = 'Start';  
        legendEntries{2} = 'Goal';  
    end  
  
    close all; figure()  
    handles = plotMap(occGrid, start, goal);  
    for connectionDistance = connectionDistances  
        pthObj = 0;  
        solnInfo = 0;  
  
        % input occGrid, radius  
        % Setup Model  
        bounds = [occGrid.XWorldLimits; occGrid.YWorldLimits; [-pi pi]];  
        ss = stateSpaceDubins(bounds);  
        ss.MinTurningRadius = 1; % what is changing every round  
  
        % Setup RRT  
        stateValidator = validatorOccupancyMap(ss);  
        stateValidator.Map = occGrid;  
        stateValidator.ValidationDistance = 0.05;  
        planner = plannerRRT(ss, stateValidator);  
        planner.MaxConnectionDistance = connectionDistance;  
        planner.MaxIterations = 30000;  
        planner.GoalReachedFcn = @exampleHelperCheckIfGoal;  
  
        % Solve RRT  
        rng(0, 'twister')  
        [pthObj, solnInfo] = plan(planner, start, goal);  
  
        % Save Results  
        lengths(loop) = pthObj.pathLength; iterations(loop) =  
solnInfo.NumIterations; nodeCounts(loop) = solnInfo.NumNodes;  
  
        % Plotting only  
        if plot == 1  
            % Create a legend entry
```

```

        legendEntries{loop+2} = sprintf('Connection Distance = %.1f,
path length = %.2f, iterations = %d, nodes = %d',connectionDistance,
lengths(loop), iterations(loop), nodeCounts(loop));
        % Plot Results
        hold on;
        handles(loop + 2) = plotResult(pthObj, solnInfo); % Store the
handle
    end

    loop = loop + 1;
end
if plot == 1
    legend(handles,legendEntries); end
end

```

## RRT Path planner with constraints

```

function [lengths, iterations, nodeCounts] =
constrainedRRT(minTurningRadius,maxConnectionDistance,occGrid,start,goal)
    pthObj = 0;
    solnInfo = 0;

    % input occGrid, radius
    % Setup Model
    bounds = [occGrid.XWorldLimits; occGrid.YWorldLimits; [-pi pi]];
    ss = stateSpaceDubins(bounds);
    ss.MinTurningRadius = minTurningRadius; % what is changing every
round

    % Setup RRT
    stateValidator = validatorOccupancyMap(ss);
    stateValidator.Map = occGrid;
    stateValidator.ValidationDistance = 0.05;
    planner = plannerRRT(ss, stateValidator);
    planner.MaxConnectionDistance = maxConnectionDistance;
    planner.MaxIterations = 35000;
    planner.GoalReachedFcn = @exampleHelperCheckIfGoal;

    % Error catching
    if ~stateValidator.isStateValid(start)
        fprintf('X World Limits: [%f, %f]\n', occGrid.XWorldLimits(1),
occGrid.XWorldLimits(2));
        fprintf('Y World Limits: [%f, %f]\n', occGrid.YWorldLimits(1),
occGrid.YWorldLimits(2));
        error('Start state is not valid.');
```

```

    end

    if ~stateValidator.isStateValid(goal)
        fprintf('X World Limits: [%f, %f]\n', occGrid.XWorldLimits(1),
occGrid.XWorldLimits(2));

```

```

        fprintf('Y World Limits: [%f, %f]\n', occGrid.YWorldLimits(1),
occGrid.YWorldLimits(2));
        error('Goal state is not valid.');
```

end

```

% Solve RRT
%rng(0,'twister')
[pthObj, solnInfo] = plan(planner, start, goal);

% Check if a path was found

% Save Results
lengths = pthObj.pathLength; iterations = solnInfo.NumIterations;
nodeCounts = solnInfo.NumNodes;

figure()
plotMap(occGrid,start,goal);
plotResult(pthObj, solnInfo)
plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-');
```

end

## RRTStar Path planner with constraints

```

function [lengths, iterations, nodeCounts] =
constrainedRRTStar(minTurningRadius,maxConnectionDistance,occGrid,start,goal
)
    pthObj = 0;
    solnInfo = 0;

    % input occGrid, radius
    % Setup Model
    bounds = [occGrid.XWorldLimits; occGrid.YWorldLimits; [-pi pi]];
    ss = stateSpaceDubins(bounds);
    ss.MinTurningRadius = minTurningRadius; % what is changing every
round

    % Setup RRT
    stateValidator = validatorOccupancyMap(ss);
    stateValidator.Map = occGrid;
    stateValidator.ValidationDistance = 0.05;
    %planner = plannerRRT(ss, stateValidator);
    planner = plannerRRTStar(ss, stateValidator);
    planner.MaxConnectionDistance = maxConnectionDistance;
    planner.MaxIterations = 35000;
    planner.GoalReachedFcn = @exampleHelperCheckIfGoal;

    % Error catching
    if ~stateValidator.isStateValid(start)
```

```

        fprintf('X World Limits: [%f, %f]\n', occGrid.XWorldLimits(1),
occGrid.XWorldLimits(2));
        fprintf('Y World Limits: [%f, %f]\n', occGrid.YWorldLimits(1),
occGrid.YWorldLimits(2));
        error('Start state is not valid.');
```

end

```

    if ~stateValidator.isStateValid(goal)
        fprintf('X World Limits: [%f, %f]\n', occGrid.XWorldLimits(1),
occGrid.XWorldLimits(2));
        fprintf('Y World Limits: [%f, %f]\n', occGrid.YWorldLimits(1),
occGrid.YWorldLimits(2));
        error('Goal state is not valid.');
```

end

```

% Solve RRT
rng(0, 'twister')
[pthObj, solnInfo] = plan(planner, start, goal);

% Check if a path was found

% Save Results
lengths = pthObj.pathLength; iterations = solnInfo.NumIterations;
nodeCounts = solnInfo.NumNodes;

figure()
plotMap(occGrid, start, goal);
plotResult(pthObj, solnInfo)
plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-');
```

end