PRM & RRT Demonstration in MATLAB

Kristina Collins, kvc2@case.edu; Connor Wolfe, cbw36@case.edu

***Abstract*—This report presents a MATLAB demonstration of PRM and RRT searches in 4D C-space for two 2 DoF robots in a plane. The code generates two plots for each algorithm: a 2D plot showing the robots' start and end positions and obstacles, and a plot showing the algorithm's exploration of the 4D configuration space. Each algorithm is shown to solve the problem of one robot needing to get out of another's way by taking an indirect path. Discussions of theory of operation and implementation summaries are also included.**

I. Introduction

There are two methods for computing trajectories for multiple robots in a single workspace: Centralized and decoupled planning. In decoupled planning, the trajectory of each robot is computed separately, keeping the dimension of the problem relatively low. However, certain maps are largely incompatible with decoupled planning, in cases when one robot has to get out of another's way. In centralized planning, the configuration spaces of the two robots are multiplied together to produce a single configuration space which contains information about both obstacles and collisions between robots. In this paper, PRM and RRT algorithms are each used to solve this problem in 4D space.

II. Probabilistic Roadmapping

A. Theory of Operation

Probabilistic roadmapping is a technique for sampling-based path planning that scales well for higher dimensions. As summarized in (1) and (2), PRM consists of two major phases: learning and query.

1) Learning: In the learning phase, a matrix N is created and populated with possible positions in Cspace. From N, a 2D matrix E is created that expresses the connectivity ("edges") of the nodes. These two matrices together form a "road map" describing the sampled points.

2) Query: In the query phase, a separate algorithm (Dijkstra's algorithm, in our case) consults N and E to determine the "cheapest" path through Cspace. Because, in higher-dimension implementations, this considers time as well as 2D position, the resulting path through Cspace encompasses information about the robots' positions in time, and can be reconstructed to complete and collision-free paths for all robots initially considered.

B. Implementation

This section discusses each section of the code we used to implement PRM.

(1) $N \leftarrow \emptyset$
(2) $E \leftarrow \emptyset$
(3) **loop**
(4) $c \leftarrow$ a randomly chosen free configuration
(5) $N_c \leftarrow$ a set of candidate neighbors of c chosen from N
(6) $N \leftarrow N \cup \{c\}$
(7) **for all** $n \in N_c$, in order of increasing $D(c,n)$ **do**
(8) **if** $\neg$*same_connected_component*(c,n) $\wedge\Delta(c,n)$ **then**
(9) $E \leftarrow E \cup \{(c,n)\}$
(10) update R's connected components

Fig. 1: PRM construction step, the first part of the learning phase, as outlined in (2). A roadmap is constructed of N, a list of all possible points, and E, a matrix describing which among them are connected.

1) Preallocation: A number of parameters are set at the beginning of the code, including the map, the number of robots to consider in the configuration space, and the number of points to sample. This code creates a small sample map labeled "Cuyahoga" which presents the centralized planning problem discussed in the introduction. The map is so named because the problem is similar to rowers ducking into a "safety zone" on the Cuyahoga in order to avoid freighters.

```
%% Create map
Cuyahoga=[0 0 0 0 0; 0 0 0 0 0; 1 1 1 1 1;
    0 0 1 1 0; 0 0 1 1 1; 0 0 0 0 0];
Cuyahoga=ones(size(Cuyahoga))-(Cuyahoga);
mapMatrix=flip(Cuyahoga)';

s = [1, 4, 5, 4]; %start config
g = [5, 4, 1, 4];  %goal config
NumBots=2; %Number of robots in the 2D map
NumNodes=50; %Number of nodes in the PRM
```

2) Configuration Space: The configuration space for a 2D map with an arbitrary number of robots $NumBots$ can be calculated [1] with the following code:

```
Cspace=zeros(size(map, 1), size(map, 2),
```

[1] The point of using PRM for a mapping problem such as this one is to diminish the amount of computing time required by avoiding having to compute a higher dimension C-space entirely. It could be argued that explicitly computing the C-space, as is done here, is not in keeping with this. However, calculating the configuration space *a priori* and consulting it for the relevant points, rather than checking each point individually, doesn't affect the workings of the PRM itself, and makes the PRM a bit easier to follow, so it's been left in this form.

```
size(map, 1), size(map, 2));
for i=1:size(map,1)
    for j=1:size(map, 2)
        if map(i, j)==1
        % if an obstacle exists in the map
            itself:
            Cspace(i, j, :,:)=1;
            Cspace(:,:,i, j)=1;
        end
        Cspace(i, j, i, j)=1;
        % Collision checking - our two
           robots can't be in the same
           place.
    end
end

Cspace=logical(Cspace);
%Cast to a logical array.
```

We constructed N using the code below, and added the start and goal points to it. This represents one half of our roadmap: the feasible points sampled from the configuration space.

```
%% PRM Learning: Construction
N = zeros(NumNodes, 2*NumBots); % Set of
    workable configs in Cspace
randConfig = ones(1, 2*NumBots); c=
    randConfig; %preallocation

for i=1:NumNodes %populate N
    search=1; %start looking for a new
        free point
    while search==1
        for j=1:size(randConfig, 2)
            randConfig(j)=randi([1, size(
                Cspace, j)])
        end
        if Cspace(randConfig(1),
            randConfig(2), randConfig(3),
            randConfig(4))==0 %if
            unoccupied
            c = randConfig%randomly chosen
                free config
            search=0; %found a free point
        end
    end % Find an open configuration c
    N(i, :)=c; %Populate N with open
        configurations
end
```

Next, we calculate the second half of our roadmap: the matrix E, which indicates whether points are connected:

```
%% PRM Roadmap: Populating E
E = eye(NumNodes); % Set of edges;
    expresses connectivity. Preallocation.
cellDist=ones(1, 2*NumBots) %
    Preallocation for local planner below.
    N-dimensional.
for i=1:NumNodes % populate E, to use with
    Dijkstra later
    for j=1:NumNodes
        for k=1:2*NumBots
            cellDist(k)=dist(N(i, k), N(j,
                k));
        end
        if max(cellDist)==1
            E(i, j) = 1; E(j, i)=1;
            foo=foo+1;
        end
    end
end
```

The local planner function used here only checks to see if cells are adjacent and valid in Cspace. This is inefficient and incomplete, and doesn't work on larger maps, but does produce good results for our narrow passage problem.

We used a Dijkstra algorithm implementation available on the MATLAB File Exchange (3) in the query phase. It's called with:

```
%% PRM Query
[cost route]=dijkstra(E, 1, 2); %replaced
    1 and 2 with s and g
course=zeros(length(route), size(N, 2));
for i=1:length(route) %vector of positions
    in Cspace
    course(i, :)=N(route(i), :)
end
```

The original algorithm from (3) is listed below:

```
function [e L] = dijkstra(A,s,d)

if s==d
    e=0;
    L=[s];
else

A = setupgraph(A,inf,1);

if d==1
    d=s;
end
A=exchangenode(A,1,s);

lengthA=size(A,1);
W=zeros(lengthA);
for i=2 : lengthA
    W(1,i)=i;
    W(2,i)=A(1,i);
end

for i=1 : lengthA
    D(i,1)=A(1,i);
    D(i,2)=i;
end
```

```
D2=D(2:length(D),:);
L=2;
while L<=(size(W,1)-1)
    L=L+1;
    D2=sortrows(D2,1);
    k=D2(1,2);
    W(L,1)=k;
    D2(1,:)=[];
    for i=1 : size(D2,1)
        if D(D2(i,2),1)>(D(k,1)+A(k,D2(i
            ,2)))
            D(D2(i,2),1) = D(k,1)+A(k,D2(i
                ,2));
            D2(i,1) = D(D2(i,2),1);
        end
    end

    for i=2 : length(A)
        W(L,i)=D(i,1);
    end
end
if d==s
    L=[1];
else
    L=[d];
end
e=W(size(W,1),d);
L = listdijkstra(L,W,s,d);
end
```

Finally, the results are displayed on the map with:

```
%% Animation:
f=figure;
figure(f), subplot(1, 2, 1), title('PRM
    Map');
map=robotics.OccupancyGrid(Cuyahoga, 1);
show(map)
A=animatedline; A.Color='r'; A.LineWidth
    =3; A.Marker='o'; A.MarkerFaceColor='g
    ';
B=animatedline; B.Color='b'; B.LineWidth
    =3; B.Marker='o'; B.MarkerFaceColor='r
    '; B.MarkerSize=4;
x1=course(:, 1)-.5; %.5 shift for
    compatibility with map display
y1= course(:, 2)-.5; x2 = course(:, 3)-.5;
     y2 = course(:, 4)-.5;
for k = 1:length(x1)
    addpoints(A,x1(k),y1(k));
    addpoints(B, x2(k), y2(k));
    pause(1); %slow down display
    drawnow
end

hold on; comet(x1, y1) %an alternative
    display method
```

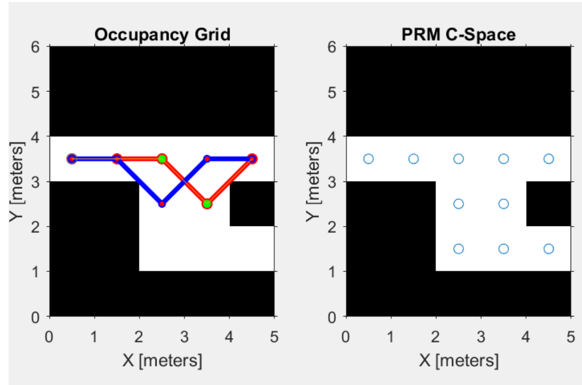


Fig. 2: An example of the output window.

```
subplot(1, 2, 2), show(map)
title('PRM C-Space'); %displays all points
hold on; scatter(N(:, 1)-.5, N(:, 2)-.5);
```

C. Results

An example result is shown in Figure 2. Each stage of the display is shown in the appendix.

III. Rapidly Expanding Random Trees (RRT)

A. Theory of Operation

As outlined in (1), RRT is another probabilistic roadmapping algorithm ideal for the single query problem. RRT differs from PRM in that rather than attempting to compute the entire connectivity of the graph, it will instead only connect a new node to the node which discovered it, and continue to discover new nodes in the direction of its goal state. In this sense, it is ideal for the single query problem, wherein we are given a start and goal configuration and can grow our roadmap from start to goal. It follows then, that in this implementation if another start and goal state were to be added, it is unlikely that the graph would have sufficient connectivity to traverse from the new start to the new goal.

RRT works by maintaining two trees, T_1 rooted at q_{start} and T_2 rooted at q_{goal}. The trees will alternate growing by sampling q_rand from Q_{free}. The method of sampling can greatly affect the efficiency of the algorithm as detailed below. The tree will then find the closest point it can connect to en route to q_{rand} and add this q_{new} to its tree. The second tree will try to connect to q_{new} and if it succeeds, the algorithm terminates. Otherwise, the trees alternate and the algorithm progresses.

B. Implementation

There are several ways to optimize RRT which we implemented. To begin, as RRT is a single query algorithm it makes sense to grow the trees towards each other. To do so, we sampled q_{rand} by selecting a weighted probability between drawing $q_{rand} = q_{goal}, q_{rand}$ in the direction of q_{goal} or q_{rand}

Algorithm 1 RRT

```
procedure RRT(n)
    T1 ← qi
    T2 ← qf
    for i=1:n do do
        qrand ← RANDOM-CONFIG
        qnew1 ← EXTEND(T1, qrand)
        qnew2 ← EXTEND(T2, qnew1)
        if qnew1==qnew2 then
            MERGED
        else
            SWAP(T1,T2)
```

as a random point in in the CSpace. It is important to nc if q_{rand} is always drawn towards q_{goal} RRT could get st a local goal. We tune the probability weighting of these random samples in the results section.

Another means of optimization is the connectivity planner. It is important to notice that there is a tradeoff between a powerful planner which connects the tree as close as possible to q_{rand} and the speed of the algorithm. In a large configuration space, allowing for more rapid samples and less optimal finding of edges is often more efficient. Our planner works in two steps. It first tries to directly connect q_{near} from T to q_{rand}. If this fails, it will move q_{near} on a diagonal route towards q_{rand} until it collides, at which point it returns the last feasible configuration as q_{near}.

C. Results

We implement RRT on two challenging maps. The fi demonstrates the algorithm's ability to handle a mu configuration space with obstacle avoidance by forcing the robots into the safe space in the simple map. The larger map presents the challenge of having an obstacle on the direct path between q_{start} and q_{goal}. This forces RRT to rely on random sampling of q_{rand} to traverse the state space. We demonstrate how modifying the choice of sample affects the results in figures (7), (8), (9), and (10).

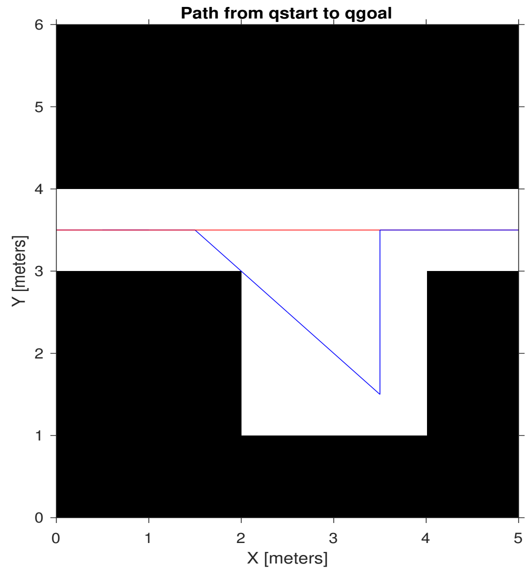


Fig. 3: This plot shows the path of robot 1 (red) and robot 2 (blue) as they alternate their initial poses. Note that in order to move through the graph one must move down into the safe bay while the other crosses above, which RRT successfully finds.

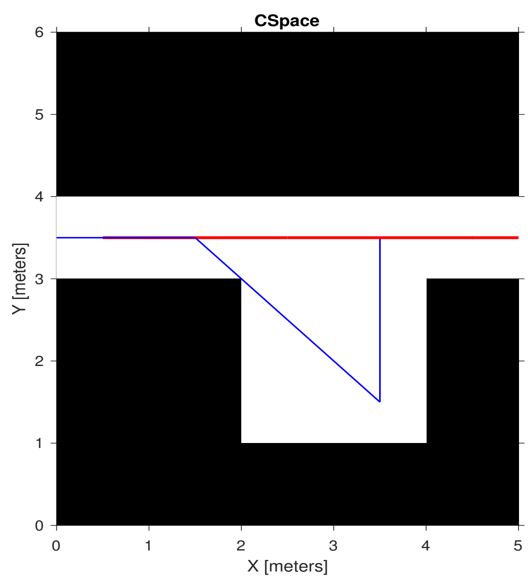


Fig. 4: This plot shows the CSpace of robot 1 (red) and robot 2 (blue) for the samee map as figure (3). Given the small map space, it makes sense that the CSpace is largely full.

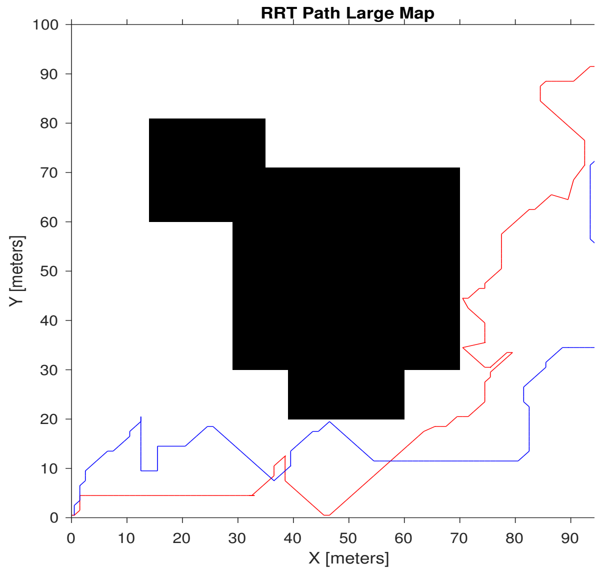


Fig. 5: This plot shows the path of robot 1 (red) and robot 2 (blue) as they alternate their initial poses from the bottom left corner and top right corner respectively. The challenge of this graph is to avoid the obstacle on the direct line to the goal, while still moving towards the goal efficiently.

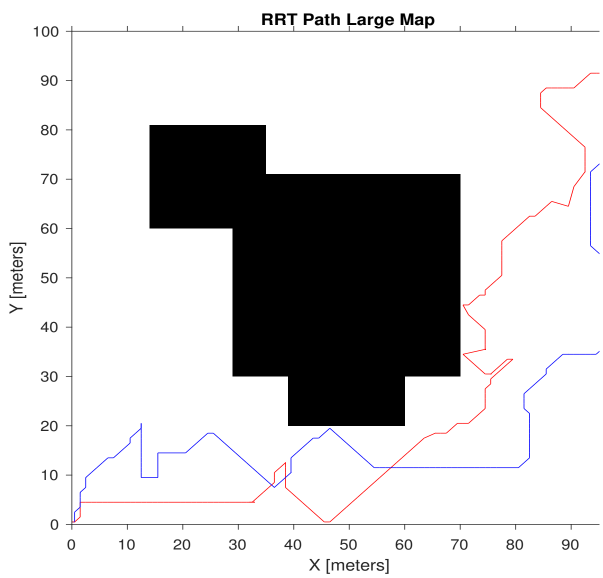


Fig. 6: This plot shows the CSpace of robot 1 (red) and robot 2 (blue) for the same map as figure (5). The configuration space ventured into areas far from the final path which makes sense because in order to find a working path, the robot had to sample many points to avoid the central obstacle.

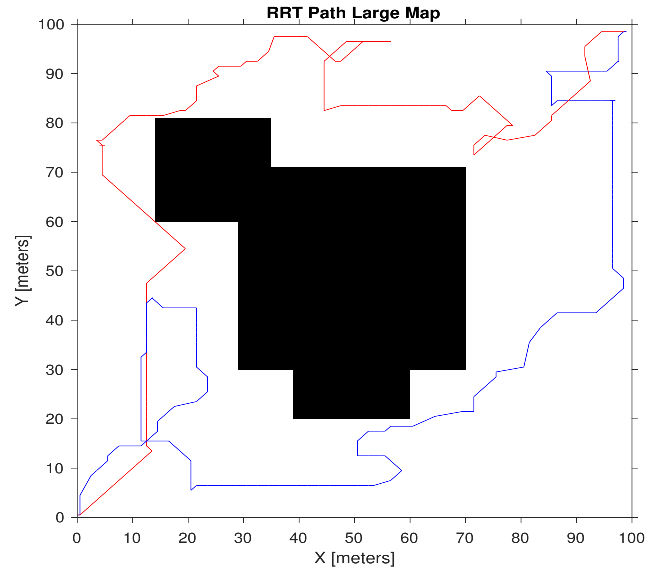


Fig. 7: This plot shows the path of robot 1 (red) and robot 2 (blue) for the same map as figure (5). This iteration drew random samples with 75% probability being along the direction of the goal configuration. The path appears direct as the robots follow the straight line towards it when possible.

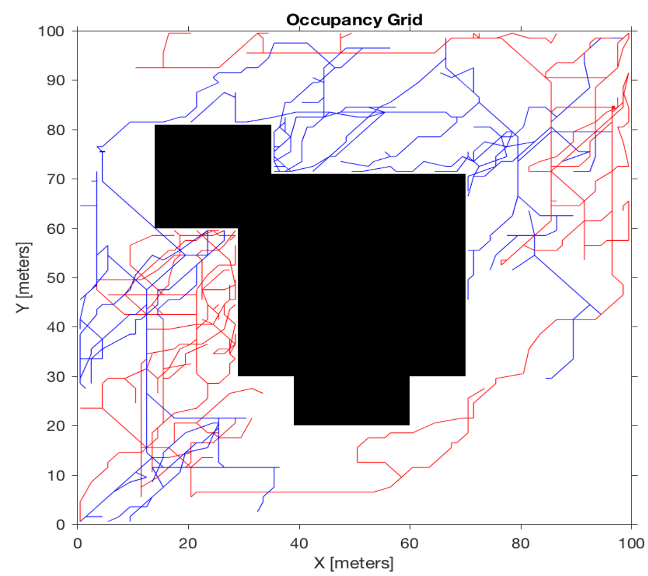


Fig. 8: This plot shows the CSpace of robot 1 (red) and robot 2 (blue) for the same map as figure (5). We see that configurations tend towards the obstacle and get stuck with a high density of samples around that area. The 20% purely random samples allow it to eventually escape and find the efficient path from figure (7).

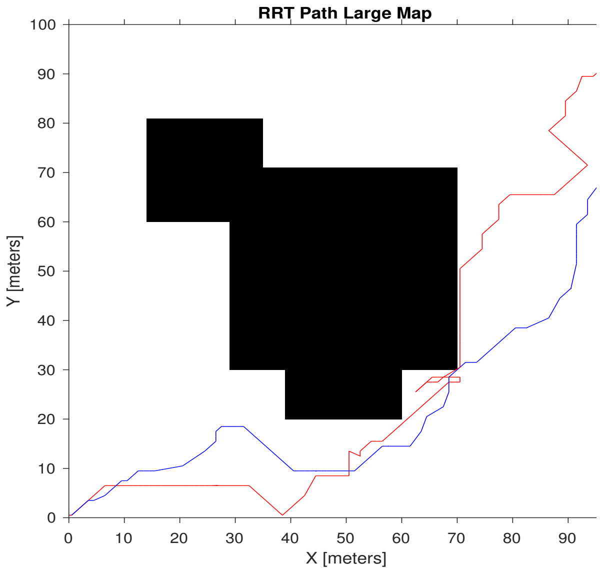


Fig. 9: This plot shows the CSpace of robot 1 (red) and robot 2 (blue) for the same map as figure (5). In this iteration, random samples are drawn with 20% probability of being the goal point, and 50% of samples are purely random. This allows the robot to hug the direct line to the goal while exploring space away from obstacles as well

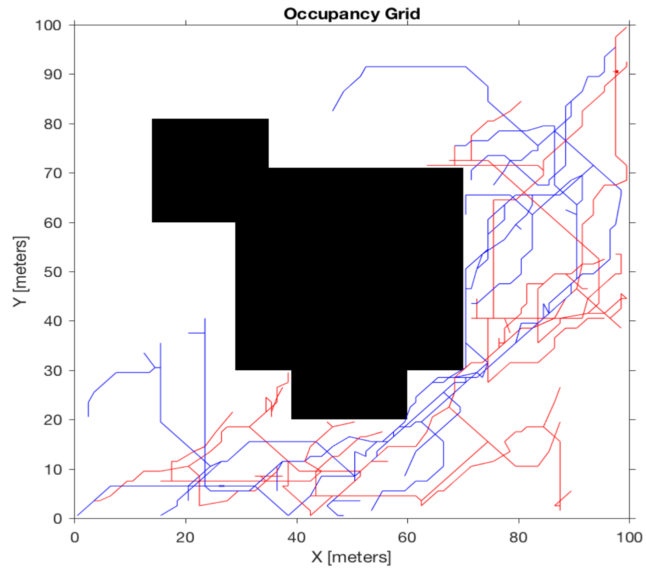


Fig. 10: This plot shows the CSpace of robot 1 (red) and robot 2 (blue) for the same map as figure (9). We see a similar result to figure(9) where the robots tend towards the obstacle, but the higher 50% random samples help the robot find the goal state quicker. This balance of following the direct path to the goal and sampling randomly to avoid getting stuck at obstacles seems optimal

IV. Conclusion

Both algorithms successfully navigated the "safety zone" map. Future work on these would include improving the local planner for the PRM, further tuning the RRT parameters, and refactoring the code for use with continuous maps (as opposed to just occupancy grids) and robust use with problems of arbitrary dimension, i.e., more than 3 robots in 2D space.

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents series)*. A Bradford Book, 2005.

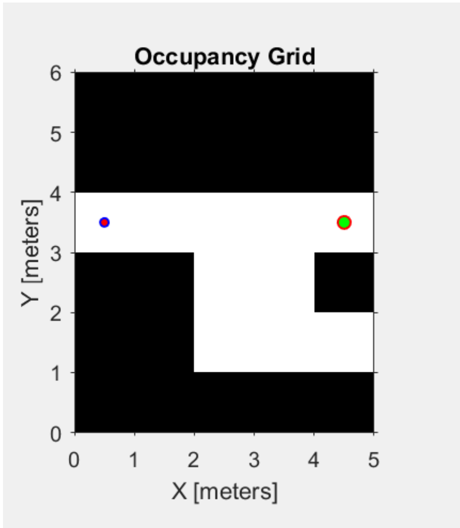
[2] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.

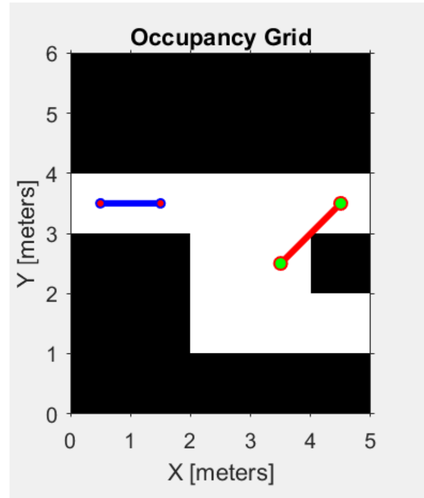
[3] D. Aryo, *MATLAB File Exchange: Dijkstra Algorithm*. MathWorks, 2012. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/36140-dijkstra-algorithm

a

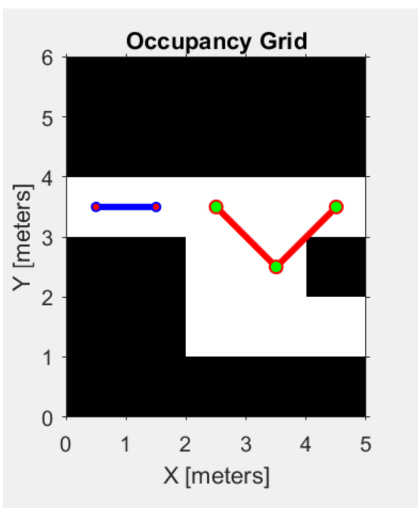
b

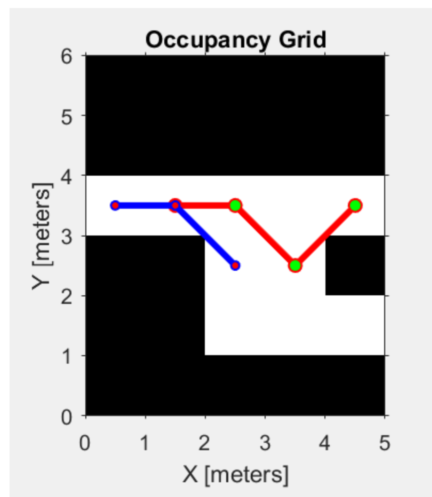
1


Occupancy Grid
Y [meters]
X [meters]

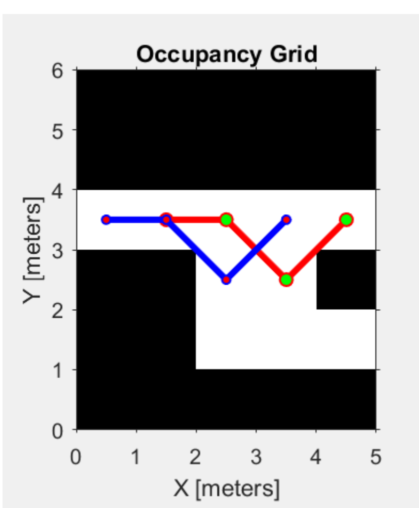

Occupancy Grid
Y [meters]
X [meters]

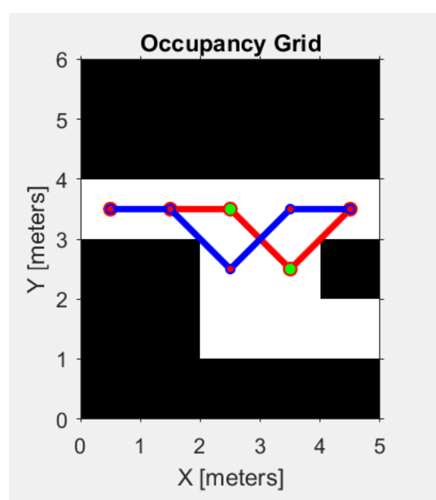
2


Occupancy Grid
Y [meters]
X [meters]


Occupancy Grid
Y [meters]
X [meters]

3


Occupancy Grid
Y [meters]
X [meters]


Occupancy Grid
Y [meters]
X [meters]