MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

TITLE

A* Path planning implementation on 2D Custom Grid, 3D Custom Grid, 3D Custom Grid with Custom Solid Obstacles

This report is briefly divided into the following aspects for each title depicted above

1) Introduction
2) Method
3) Evaluation
4) Analysis
5) Conclusion

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

A* Path Planning Implementation

Introduction:

A-Star algorithm is one of the most efficient path planning algorithms. It finds the least costly path from an initial configuration to a final configuration. It uses a function, **fn** determine which path needs to be taken from the start to the finish. We can call it as an extension to Dijkstra's algorithm.

The main difference between AStar and Dijkstra algorithm is the addition of one more function. Let us first understand the Dijkstra's algorithm and then we derive AStar algorithm from it

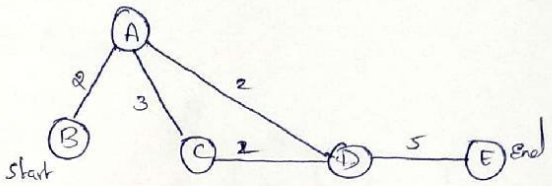
DIJKSTRA's Algorithm

MEAM -520 FINAL PROJECT REPORT

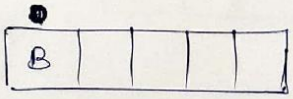
Akarsh.V akarshv@seas.upenn.edu

Dijkstra's Algorithm

Suppose, there are 5 Nodes A, B, C, D, E as shown below

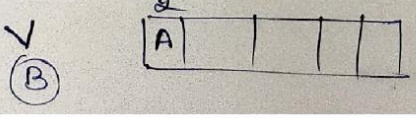


- The lines indicate that, there exists a path between them. Also costs are depicted on those lines. For example, cost of travelling from (B) to (A) or vice versa is 2
- Now suppose we are starting at (B) and we want to reach (D) What will be optimal path or the least cost path?
- This algorithm uses a Queue



We pull out the nodes with minimum cost from the Queue.

- We add (B) to it first and the cost associated from travelling (B) to (B) is 0 → depicted above B in Queue.
- We take out B from Queue and Add its neighbours to the Queue

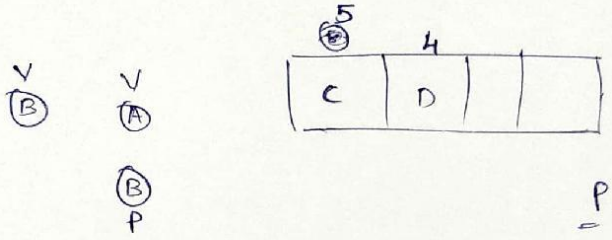


V stands for visited.

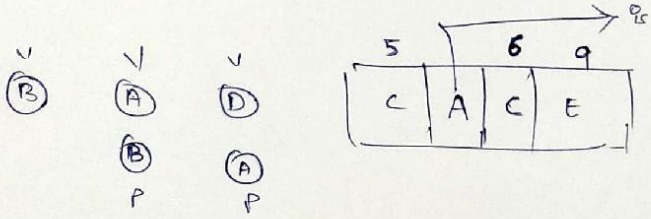
MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

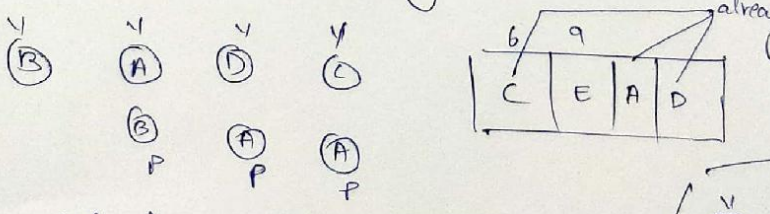
- We shall not add or visit the nodes that we have already visited
- Now, as A is its only neighbour, we take out (A) and say that B is its Parent. As we took out A, we also add its neighbours to the Queue.



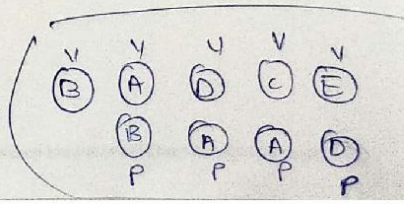
- Now from this the minimum cost node is D as the cost associated is 4 = (B) —2— (A) + (A) —2— (D). ∴ We take out D from the Queue and add its neighbours.



- Now, we can see that 5 C is minimum, cost and its parent is A.
∴ We take out C and add its neighbours



∴ We are left out with E with its parent D →



MEAM -520 FINAL PROJECT REPORT

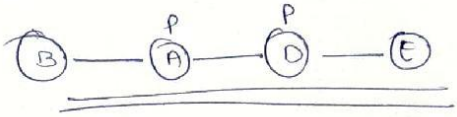
Akarsh.V akarshv@seas.upenn.edu

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

• Now, we have all the cells visited and also the parents attached.

∴ If we want to go from (B) to (E), we have



MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

In Dijkstra's algorithm, we ended up checking the minimum cost function. Let's denote it as **hn**. Now let's denote a function **gn** which gives us the distance between the respective node and the target node. Astar algorithm uses both hn and gn and calls it as fn.

fn =hn+gn

hn = cost of the path

gn = distance from the node to the target

It chooses the traversing cell based on the minimum value of the fn's.

PESUDOCODE:

1) take in the obstacles, start position and the target position
2) Create 2 lists, opens list and closed list where open list finally gives the path by giving the parents and parent's parents and so on. Closed list gives the obstacles and already visited nodes
3) add all the obstacles to the closed list and the start position
4) find all the neighbors of the start position and find the fn values of each node
    a) for k= 1:-1:-1
       for j= 1:-1:-1

       for l=1:-1:-1

       if (k~=j || k~=l|| l~=j|| k~=0) %The node itself is not its successor

       xsuccessor = node_x+k;

       ysuccessor = node_y+j;

       zsuccessor = node_z+l;

    b) check if these successors are within the boundaries and delete them if they are obstacles
    c) if they pass the above check find the fn

```
neighbours(neighbourscount,1) = xsuccessor;
neighbours(neighbourscount,2) = ysuccessor;
neighbours(neighbourscount,3) = zsuccessor;
neighbours(neighbourscount,4) = hn+sqrt((node_x-xsuccessor)^2
             + (node_y-ysuccessor)^2 +(node_z-zsuccessor)^2);
```

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

```
 neighbours(neighbourscount,5) = sqrt((xTarget-xsuccessor)^2 +
(yTarget-ysuccessor)^2 +(zTarget-zsuccessor)^2);%distance between
node and goal
neighbours(neighbourscount,6)=neighbours(neighbourscount,4)+neigh
bours(neighbourscount,5);%fn
neighbourscount=neighbourscount+1;
```

5) Apply a brute force traversing, by adding each neighbor to the open list and if the neighbor is already on the open list, then check the minimum fn value of it from reaching through the previous parent and current node. Decide the parent based on the minimum value of fn and update its parent accordingly

6)
```
if(neighbours(i,1) == openlist(j,2) && neighbours(i,2) ==
openlist(j,3)&&neighbours(i,3) == openlist(j,4) )
         openlist(j,10)=min(openlist(j,10),neighbours(i,6));
         if openlist(j,10)== neighbours(i,6)
             %update parents,gn,hn
             openlist(j,5)=xNode;
             openlist(j,6)=yNode;
             openlist(j,7)=zNode;
             openlist(j,8)=neighbours(i,4);
             openlist(j,9)=neighbours(i,5);
         end
```

7) Find the right successor to traverse from the current cell by finding the minimum fn value on comparing all the successors fn value.
8) After going to that node repeat the above same process until you reach the target node
9) End the while loop if you reach the target node
10) Find the final path by traversing from the target to the start position using the updated parents from the open list
11) Plot the path

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

METHOD:

Three codes are attached to this report. All the programs include custom obstacles, custom GRID size

1) AStar in 2D GRID
2) AStar in 3D GRID
3) AStar in 3D GRID with cubic obstacles

INPUTS to the above codes

1) Astar in 2D asks the user to choose the start point, obstacles points and the target point by an interactive environment. It is a Script
2) AStar in 3D GRID asks the user to input the start, obstacles(array nx3, n=no.of obstacles) and target point by entering them in the command window and run the function with these inputs
3) AStar in 3D GRID asks the user to input the start, center (array nx3, n=no.of obstacles). Center points are taken and a cube is generated around them which gives us the cubic obstacles and target point by entering them in the command window and run the function with these inputs

All the codes are almost similar, so let us take AStar in 3D GRID with cubic obstacles which covers the rest of the two.

AStar in 3D GRID with cubic obstacles

The functions used in this code are

1) Astarcubic → this is the main function
2) Add_to_openlist
3) find_neighbours
4) min_fn

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

Astarcubic

- First we generate the start, target and obstacles. The below pictures shows how to generate the obstacles

```
bluff=0;
i=1;
while(bluff==0)
    xcoord(i,:) = [(center(i,1)-0.25),center(i,1),(center(i,1)+0.25)];
    ycoord(i,:) = [(center(i,2)-0.25),center(i,2),(center(i,2)+0.25)];
    zcoord(i,:) = [(center(i,3)-0.25),center(i,3),(center(i,3)+0.25)];
    if(i==size_center)
        bluff=1;
    end
    i=i+1;

end

n=1;
for p=1:1:size_center
for k=1:1:3
    for l=1:1:3
        for m=1:1:3

            obstacles(n,1)= xcoord(p,k);
            obstacles(n,2)= ycoord(p,1);
            obstacles(n,3)= zcoord(p,m);
            n=n+1;
        end
    end
end
end
```

I took the center coordinate and generated six points around it and used those combination to find the points that approximates to a cube.

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

- Then we plot the grid.

```
xend=10;
yend=10;
zend =10;
max=10;
j=0;
x = 1;
y = 1;
axis([1 xend+1 1 yend+1 1 zend+1])
%setting the range or the grid cells length to be 1
tickValues = 1:1:xend;
set(gca,'XTick',tickValues)
tickValues = 1:1:yend;
set(gca,'YTick',tickValues)
tickValues = 1:1:zend;
set(gca,'ZTick',tickValues)
grid on;
hold on;
```

We can modify the grid, i.e. change its lengths by simply changing the **xend, yend, zend** values.

- We take in the obstacles and the start position to the closed list

```
k=1;
bluff=0;
while(bluff==0)
    closedlist(k,1) = obstacles(k,1);
    closedlist(k,2) = obstacles(k,2);
    closedlist(k,3) = obstacles(k,3);
    scatter3(closedlist(k,1),closedlist(k,2),closedlist(k,3),'MarkerEdgeColor','k', 'MarkerFace
    if(k==size_obstacles)
        bluff=1;
    end
    k=k+1;
end


closedlistcount=size(closedlist,1);
```

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

- We then find the neighbors of the start node

```
function neighbours=find_neighbours(node_x,node_y,node_z,hn,xTarget,yTarget,zTarget,closedlist,xend,yend,zend)

neighbours=[];
neighbourscount=1;
c2=size(closedlist,1);
for k= 1:-1:-1
    for j= 1:-1:-1
        for l=1:-1:-1
            if (k~=j || k~=l|| l~=j|| k~=0)  %The node itself is not its successor
                xsuccessor = node_x+k;
                ysuccessor = node_y+j;
                zsuccessor = node_z+l;
                if( (xsuccessor >0 && xsuccessor <=xend) && (ysuccessor >0 && ysuccessor <=yend)&& (zsuccessor >0 && zsuccessor <=zend))%node
                    flag=1;
                    for c1=1:c2
                        if(xsuccessor == closedlist(c1,1) && ysuccessor == closedlist(c1,2)&& zsuccessor == closedlist(c1,3))
                            flag=0;
                        end
                    end
                    if (flag == 1)
                        neighbours(neighbourscount,1) = xsuccessor;
                        neighbours(neighbourscount,2) = ysuccessor;
                        neighbours(neighbourscount,3) = zsuccessor;
                        neighbours(neighbourscount,4) = hn+sqrt((node_x-xsuccessor)^2 + (node_y-ysuccessor)^2 +(node_z-zsuccessor)^2);%cost o
                        neighbours(neighbourscount,5) = sqrt((xTarget-xsuccessor)^2 + (yTarget-ysuccessor)^2 +(zTarget-zsuccessor)^2);%distan
                        neighbours(neighbourscount,6) = neighbours(neighbourscount,4)+neighbours(neighbourscount,5);%fn
                        neighbourscount=neighbourscount+1;
                    end
                end
            end
        end
```

- After that we update the parents of the successor nodes, i.e. neighbor nodes by checking the minimum fn values that are readily available in the open list

```
while((xNode ~= xTarget || yNode ~= yTarget || zNode~= zTarget) && NoPath == 1)

    %find the neighbours
 neighbours=find_neighbours(xNode,yNode,zNode,costofpath,xTarget,yTarget,zTarget,closedlist,xend,yend,zend);
 neighbourscount=size(neighbours,1);

%check all the neighbours with the previous parents nwighbours and update
%the parents based on their comparision with fn values
 for i=1:neighbourscount
    flag=0;
    for j=1:openlistcount
        if(neighbours(i,1) == openlist(j,2) && neighbours(i,2) == openlist(j,3)&&neighbours(i,3) == openlist(j,4) )
            openlist(j,10)=min(openlist(j,10),neighbours(i,6));
            if openlist(j,10)== neighbours(i,6)
                %update parents,gn,hn
                openlist(j,5)=xNode;
                openlist(j,6)=yNode;
                openlist(j,7)=zNode;
                openlist(j,8)=neighbours(i,4);
                openlist(j,9)=neighbours(i,5);
            end
            flag=1;
        end

    end
    if flag == 0
        openlistcount = openlistcount+1;
        openlist(openlistcount,:)=add_to_openlist(neighbours(i,1),neighbours(i,2),neighbours(i,3),xNode,yNode,zNode,neighbours(i,4),neig
    end
end
```

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

- We then check the minimum fn among the neighbors and traverse to that node

```
function minimum = min_fn(openlist,openlistcount,xTarget,yTarget,zTarget)

temp=[];
k=1;
flag=0;
goal_arraynumber=0;
for j=1:openlistcount
    if (openlist(j,1)==1)
        temp(k,:)=[openlist(j,:) j];
        if (openlist(j,2)==xTarget && openlist(j,3)==yTarget && openlist(j,4)==zTarget)
            flag=1;

            %Store the array number of the target node
            goal_arraynumber=j;
        end
        k=k+1;
    end
end

%if one of the neighbours is target node
if flag == 1
    minimum=goal_arraynumber;
end
%array number of the smallest node
if size(temp ~= 0)
 [min_fn,temp_min]=min(temp(:,10));
 minimum=temp(temp_min,11);
else
    minimum=-1;%The temp array is empty i.e No more paths are available.
end
```

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

- Finally we generate the final path by traversing the open list parents

```
if ( (xvalue == xTarget) && (yvalue == yTarget) && (zvalue == zTarget))
    inode=0;
   %Traverse openlist and determine the parent nodes
   d=1;
    while(openlist(d,2) ~= xvalue || openlist(d,3) ~= yvalue || openlist(d,4) ~= zvalue)
        d=d+1;
    end

   xparent=openlist(d,5);
   yparent=openlist(d,6);
   zparent=openlist(d,7);

   while( xparent ~= xStart || yparent ~= yStart || zparent ~= zStart )
           finalpath(i,1) = xparent;
           finalpath(i,2) = yparent;
           finalpath(i,3) = zparent;

           %Get the parent's parent node
           e=1;
    while(openlist(e,2) ~= xparent || openlist(e,3) ~= yparent || openlist(e,4) ~= zparent )
        e=e+1;
    end

           xparent=openlist(e,5);
           yparent=openlist(e,6);
           zparent=openlist(e,7);
           i=i+1;
   end
j=size(finalpath,1);
```

- We then plot the path by plot3 and scatter3

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

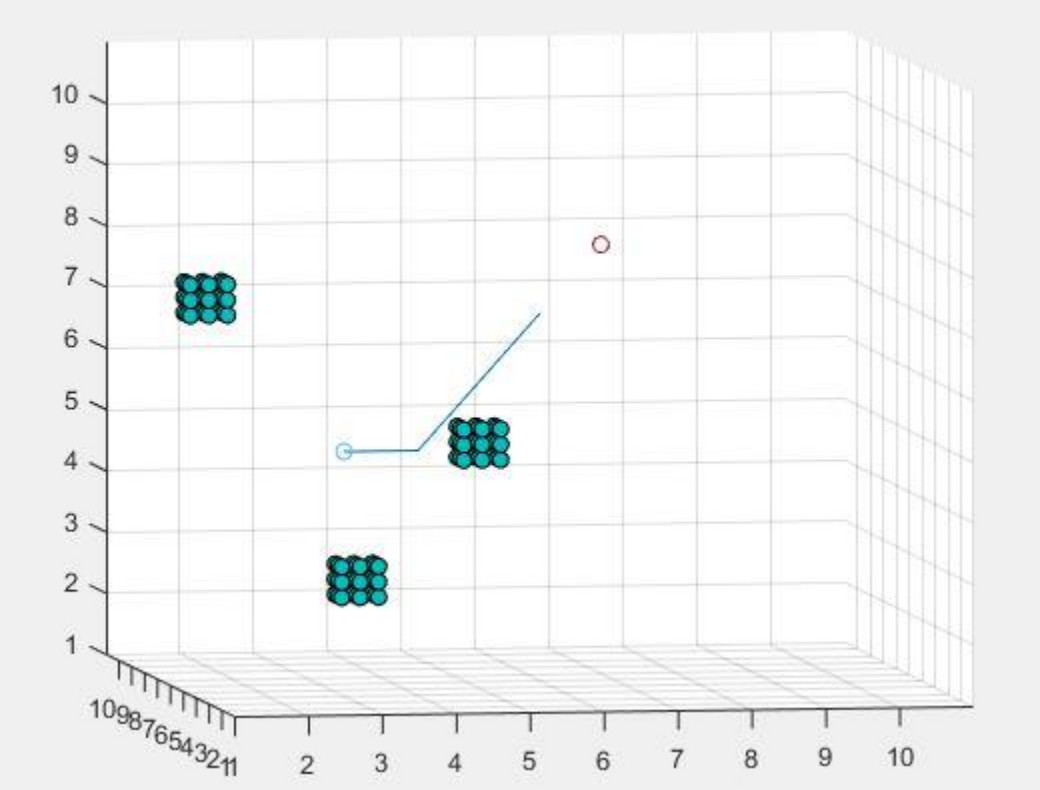
EVALUATION

Test1

start = [7,7,8]

target = [3,4,5]

center = [5,5,5;3,3,3;2,9,7]



Final path = 3 4 5

4 4 5

5 5 6

6 6 7

7 7 8

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

Test2

start = [7 8 8]

target = [3,6,6]

center =

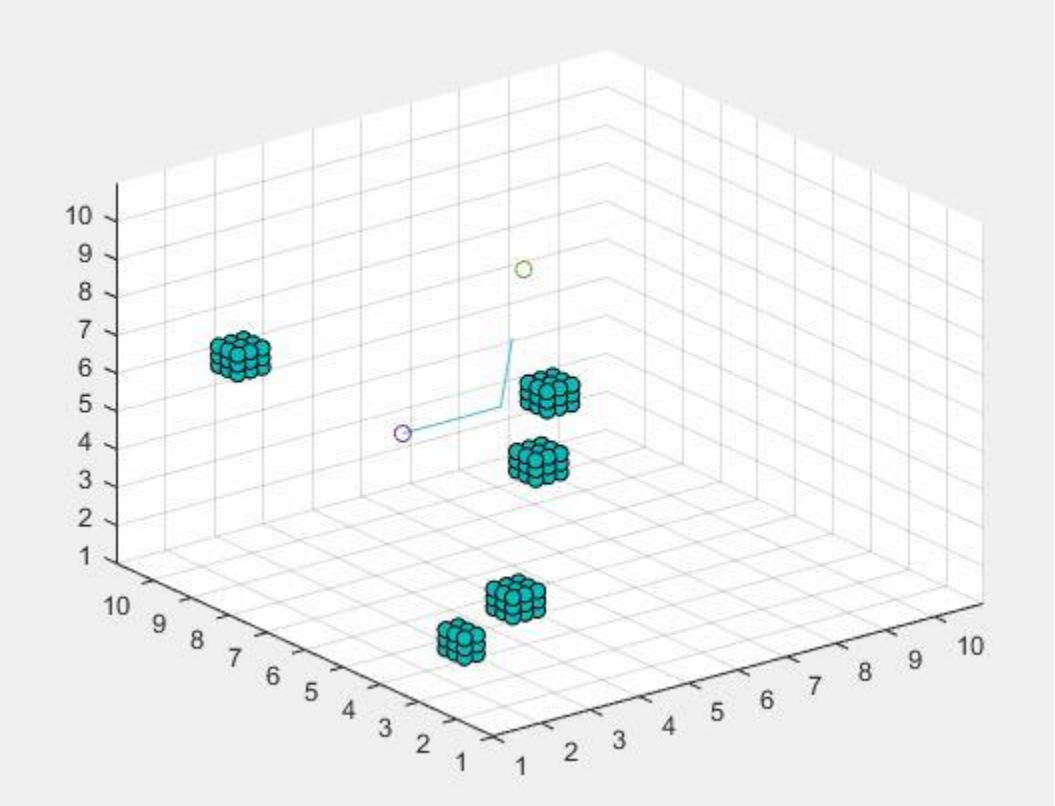
5 5 5

3 3 3

2 9 7

6 6 6

1 2 3



Final path = 3 6 6

4 6 6

5 6 6

6 7 7

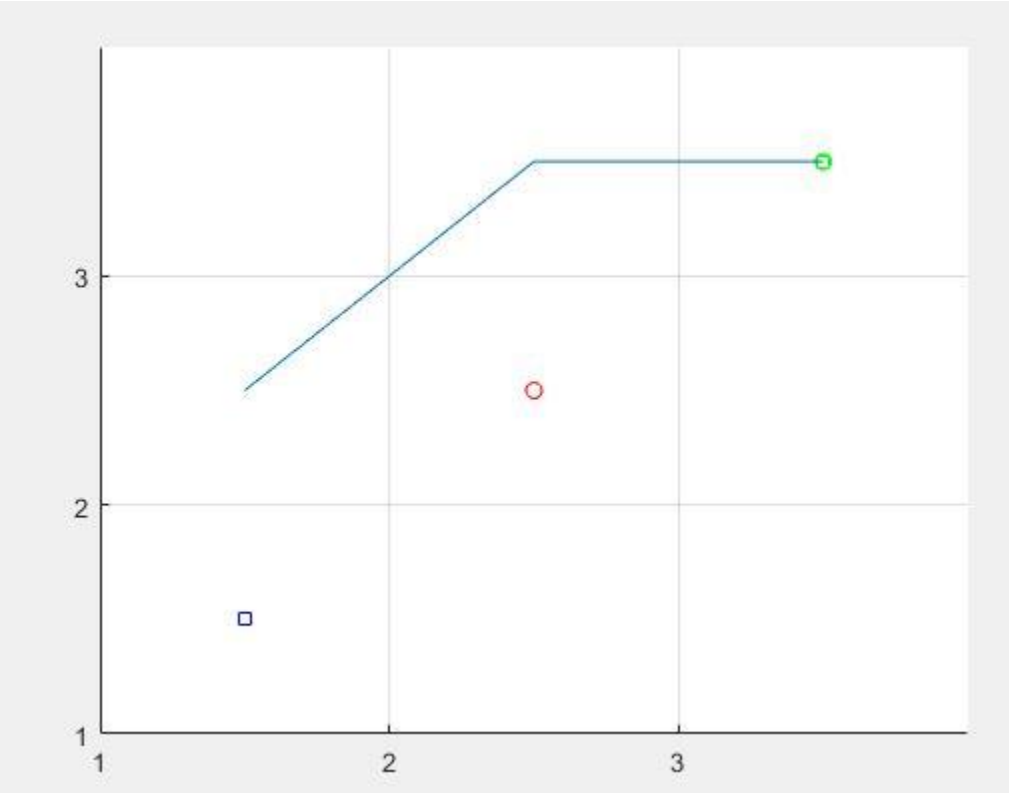
7 8 8

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

Test cases for 2D Astar

Test1:



ANALYSIS:

- As we can see from the test case1 and test case2, the solid object are nothing but a set of points.
- If we want to increase the resolution of the search algorithm i.e be more precise, then this code would not account to that as the optimal path can be through the obstacles and we may never judge that the path is not crossing the obstacles.
- This algorithm can be simply attached to our Lynx by playing with the forward and inverse kinematics.

MEAM -520 FINAL PROJECT REPORT

Akarsh.V akarshv@seas.upenn.edu

CONCLUSION:

- A* is faster than using Dijkstra and uses best-first-search to speed things up.
- Its optimality depends on the heuristic function(fn) used, so, it can return a non-optimal result because of this and at the same time better the heuristic for specific layout, and better will be the results
- It is meant to be faster than Dijkstra even if it requires more memory and more operations per node since it explores a lot less nodes and the gain is good in any case.

- **The code is uploaded as a Zip file, it has two folders**
  1) **2DAstar**
  2) **3DAstar(contains Astarcubic→cubic obstacles and Astar3Dnoobstacles→point obstacles)**

Sources and references:

1) https://en.wikipedia.org/wiki/A*_search_algorithm
2) **PLANNING ALGORITHMS by Steven M. LaValle**
3) https://www.youtube.com/watch?v=KNXfSOx4eEE
4) http://web.mit.edu/eranki/www/tutorials/search/
5) https://pdfs.semanticscholar.org/831f/f239ba77b2a8eaed473ffbfa22d61b7f5d19.pdf