

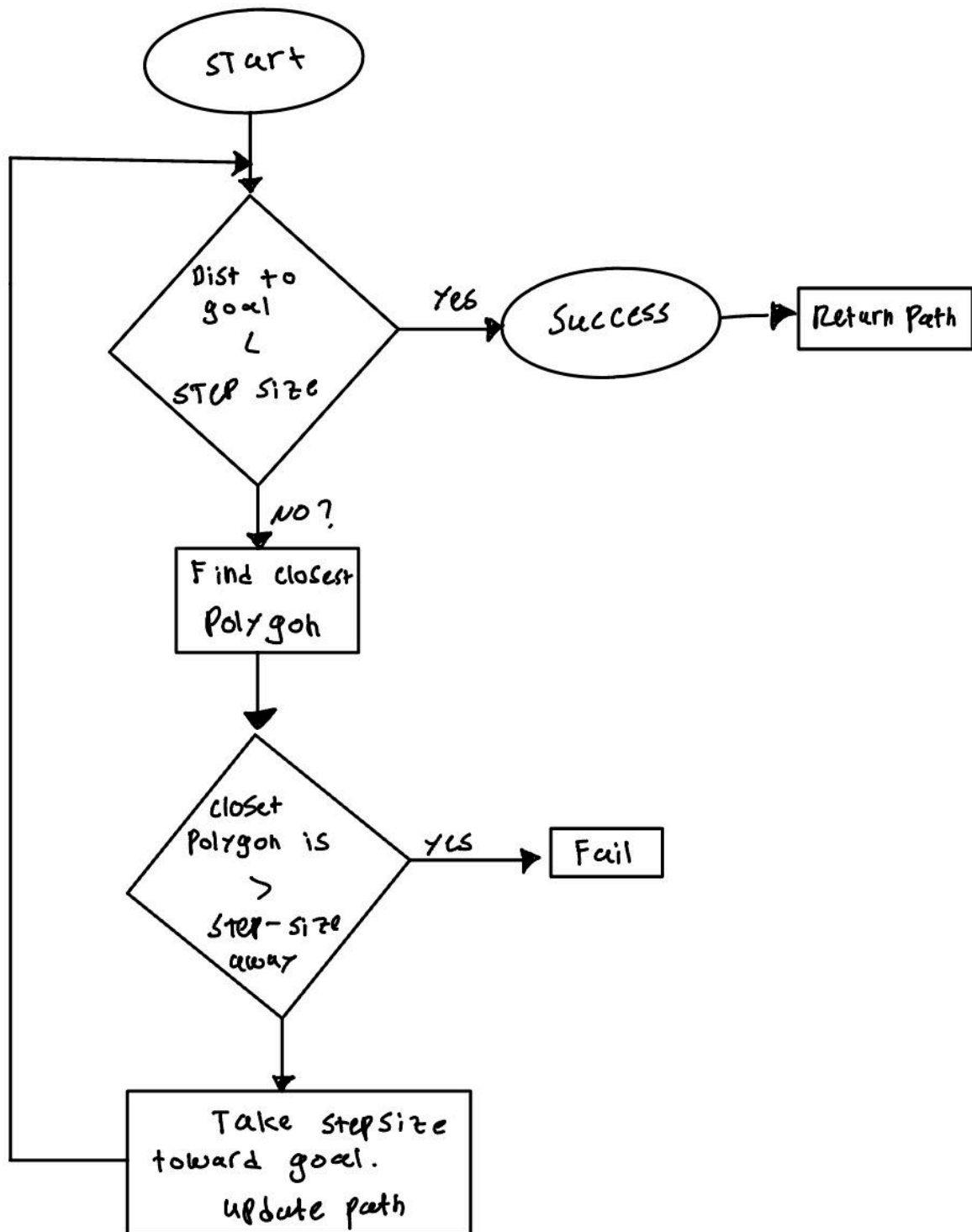
# **ME145 Robotic Planning and Kinematic: Lab 2**

## **Bug 1 Algorithm**

**Elijah Perez**

**Winter 2025, Feb 27**

Sketched Flow chart:



## BugBase Implementation:

```
1 function Path = computeBug(Pstart,Pgoal,Obstacle,stepsize)
2     %% initializing
3     Pcurrent = Pstart;
4     Path = Pstart;
5     CurrentObstacle = 0;
6
7     while abs(Pcurrent(1,1) - Pgoal(1,1)) > stepsize*1.5 && abs(Pcurrent(1,2) - Pgoal(1,2)) > stepsize*1.5
8         if length(Path(:,1))>=2
9             fprintf("Fail: An obstacle as been hit")
10            break
11        else
12            end
13
14        %% main line intersection to line segment
15
16        for i = 1:length(Obstacle(:,1,1))+1
17            Counter(i) = i ;
18        end
19        Counter(end) = 1;
20
21        for j = 1:length(Obstacle(1,1,:)) %checking each obstacles distance
22            if j == CurrentObstacle
23                if j == length(Obstacle(1,1,:))
24                    j= j+1;
25                end
26            else
27                for i = 1:length(Obstacle(:,1,1)) % Iterating through each obstacles segment
28
29                    x1 = Pcurrent(1);
30                    y1 = Pcurrent(2);
31                    x2 = Pgoal(1);
32                    y2 = Pgoal(2);
33                    P1 = Obstacle(Counter(i),:,j);
34                    P2 = Obstacle(Counter(i+1),:,j);
35
36                    x3 = P1(1);
37                    y3 = P1(2);
38                    x4 = P2(1);
39                    y4 = P2(2);
40
41                    denominator = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
42
43                    % Checking for parallel lines
44                    if denominator == 0
45                        intersection(i,,:) = [NaN NaN];
46                        d(i,j) = 0;
47                    end
48
49                    t = ((x1 - x3) * (y3 - y4) - (y1 - y3) * (x3 - x4)) / denominator;
```

```

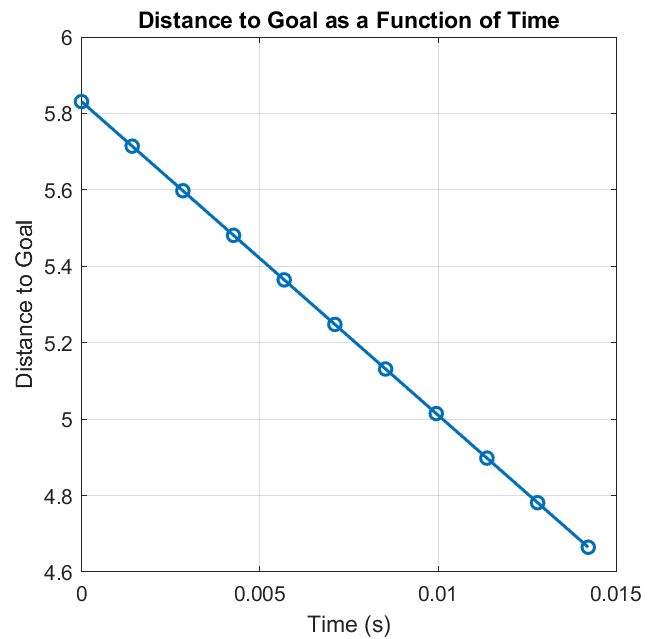
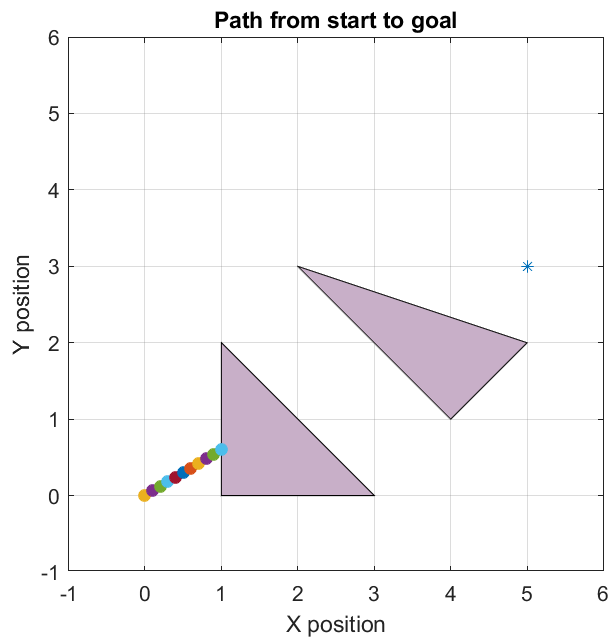
100     for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
101         Path(i,1) = x(n);
102         Path(i,2) = Pcurrent_to_PNext(x(n));
103         n = n+1;
104     end
105
106     break
107 else
108     end
109
110     %% Going from point A to point B
111     % Pcurrent = Path(end,:);
112
113     if PNext == Pcurrent
114         PNext = [x2 y2];
115         return
116     end
117
118     [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
119     x1 = Pcurrent(1);
120     y1 = Pcurrent(2);
121     x2 = PNext(1);
122     y2 = PNext(2);
123     PositionX = [x1,x2];
124
125     Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
126     x = linspace(min(PositionX),max(PositionX),Delta);
127     Pcurrent_to_PNext = @(x) (a/b)*x+c; % Direct line from start to goal
128
129     n = 1;
130     for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
131         Path(i,1) = x(n);
132         Path(i,2) = Pcurrent_to_PNext(x(n));
133         n = n+1;
134     end
135
136     Pcurrent = Path(end,:);
137
138 end
end

```

```

50     t = ((x1 - x3) * (y3 - y4) - (y1 - y3) * (x3 - x4)) / denominator;
51     u = -((x1 - x2) * (y1 - y3) - (y1 - y2) * (x1 - x3)) / denominator;
52
53     if u >= 0 && u <= 1
54         intersect_x = x1 + t * (x2 - x1);
55         intersect_y = y1 + t * (y2 - y1);
56         intersection(i,j) = [intersect_x, intersect_y];
57         d(i,j) = norm([x1 y1]-[intersect_x intersect_y]);
58     else
59         intersection(i,j) = [NaN NaN]; % Void lol
60         d(i,j) = NaN;
61     end
62 end
63
64 if all(isnan(d(:,j)), 'all')
65     Obstacle_Hit = [NaN NaN];
66 else % If obstacle has been hit
67     finder = find(min(d(:,j)) == d(:,j));
68     Obstacle_Hit = intersection(finder,:); % I identify where hit is at
69     Obstacle_Hit = Obstacle_Hit(1,:);
70     PNext = Obstacle_Hit;
71     CurrentObstacle = j;
72     CurrentSegmentNumber = [Counter(finder) Counter(finder+1)] ;
73
74     if sqrt((Pgoal(1)-Pcurrent(1))^2+(Pgoal(2)-Pcurrent(2))^2) <= sqrt((PNext(1)-Pcurrent(1))^2+(PNext(2)-Pcurrent(2))^2)
75         Obstacle_Hit = [NaN NaN];
76     else
77     end
78
79     break
80 end
81 end
82 end
83
84 if isnan(Obstacle_Hit)
85     PNext = Pgoal;
86     [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
87
88     x1 = Pcurrent(1);
89     y1 = Pcurrent(2);
90     x2 = PNext(1);
91     y2 = PNext(2);
92     PositionX = [x1,x2];
93
94     Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
95     x = linspace(min(PositionX),max(PositionX),Delta);
96
97     Pcurrent_to_PNext = @(x) (a/b)*x+c; % Direct line from start to goal
98
99     n = 1;
100     for i = length(Path(:,1)): length(Path(:,1))+length(x)-1

```



Input for bug base:

```
clc
close all
clear

Pstart = [0 0];
Pgoal = [5 3];
P(:, :, 1) = [1 0; 1 2; 3 0];
P(:, :, 2) = [2 3; 4 1; 5 2];
stepsize = 0.1;

tic;
Path = computeBug(Pstart, Pgoal, P, stepsize);
```

Output: Plus above graphs

P	3x2x2 double
Path	11x2 double
Pgoal	[5,3]
Pstart	[0,0]
stepsize	0.1000

```
Fail: An obstacle as been hitTotal Path Length: 1.17 units
Computation Time: 0.0121 seconds
>>
```

ii) The BugBase algorithm detects obstacles in its path but fails when one is encountered. In contrast, the Bug 1 algorithm identifies an obstacle in its path and, upon reaching it, circumnavigates the obstacle to find the shortest route to the goal. To achieve this, it leverages previously implemented functions, such as `computeTangentVectorToPolygon`. This function calculates key parameters, including the minimum segment distance, minimum vertex distance, and the U vector. The U vector is crucial for navigating around obstacles, as it runs parallel to segments and turns at vertices. Once the algorithm has fully circled the obstacle, it returns to the point on the obstacle that provides the shortest path to the goal. From there, it continues toward the goal. If another obstacle is detected, the process repeats.

### Bug1 implementation:

Inputs:

```
1      clc
2      close all
3      clear
4
5      Pstart = [0 0];
6      Pgoal  = [5 3];
7      P(:,1) = [1 0; 1 2; 3 0];
8      P(:,2) = [2 3; 4 1; 5 2];
9      stepsize = 0.1;
10
11     tic;
12     Path = computeBug1(Pstart,Pgoal,P,stepsize);
```

Outputs:

computation_time	0.3820
P	3x2 double
Path	245x2 double
Pgoal	[5,3]
Pstart	[0,0]
stepsize	0.1000

Graphs are below code.

```

1 function Path = computeBug1(Pstart,Pgoal,Obstacle,stepsize)
2     %% initializing
3     Pcurrent = Pstart;
4     Path = Pstart;
5     CurrentObstacle = 0;
6
7 while abs(Pcurrent(1,1) - Pgoal(1,1)) > stepsize*1.5 && abs(Pcurrent(1,2) - Pgoal(1,2)) > stepsize*1.5
8     %% main line intersection to line segment
9
10    for i = 1:length(Obstacle(:,1,1))+1
11        Counter(i) = i ;
12    end
13    Counter(end) = 1;
14
15    for j = 1:length(Obstacle(1,1,:))
16        if j == CurrentObstacle
17            if j == length(Obstacle(1,1,:))
18                j = j+1;
19            end
20        else
21            for i = 1:length(Obstacle(:,1,1))
22                P1 = Obstacle(Counter(i),:,j);
23                P2 = Obstacle(Counter(i+1),:,j);
24
25                x1 = Pcurrent(1);
26                y1 = Pcurrent(2);
27                x2 = Pgoal(1);
28                y2 = Pgoal(2);
29                x3 = P1(1);
30                y3 = P1(2);
31                x4 = P2(1);
32                y4 = P2(2);
33
34                denominator = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
35
36                % Checking for parallel lines
37                if denominator == 0
38                    intersection(i, :, j) = [NaN NaN];
39                    d(i, j) = 0;
40                end
41
42                t = ((x1 - x3) * (y3 - y4) - (y1 - y3) * (x3 - x4)) / denominator;
43                u = -((x1 - x2) * (y1 - y3) - (y1 - y2) * (x1 - x3)) / denominator;
44                if u >= 0 && u <= 1
45                    intersect_x = x1 + t * (x2 - x1);
46                    intersect_y = y1 + t * (y2 - y1);
47                    intersection(i, :, j) = [intersect_x, intersect_y];
48                    d(i, j) = norm([x1 y1]-[intersect_x intersect_y]);
49                else
50                    intersection(i, :, j) = [NaN NaN]; % Void lol
51                    d(i, j) = NaN;
52                end
53            end
54        end
55
56        if all(isnan(d(:, j)), 'all')
57            Obstacle_Hit = [NaN NaN];
58        else
59            finder = find(min(d(:, j)) == d(:, j));
60            Obstacle_Hit = intersection(finder, :, j); % I identify where hit is at
61            Obstacle_Hit = Obstacle_Hit(1, :);
62            PNext = Obstacle_Hit;
63            CurrentObstacle = j;
64            CurrentSegmentNumber = [Counter(finder) Counter(finder+1)] ;
65
66            if sqrt((Pgoal(1)-Pcurrent(1))^2+(Pgoal(2)-Pcurrent(2))^2) <= sqrt((PNext(1)-Pcurrent(1))^2+(PNext(2)-Pcurrent(2))^2)
67                Obstacle_Hit = [NaN NaN];
68            else
69                end
70
71            break
72        end
73    end
74 end
75
76 if isnan(Obstacle_Hit)
77     PNext = Pgoal;
78     [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
79
80     x1 = Pcurrent(1);
81     y1 = Pcurrent(2);
82     x2 = PNext(1);
83     y2 = PNext(2);
84     PositionX = [x1,x2];
85
86     Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
87     x = linspace(min(PositionX),max(PositionX),Delta);
88
89     Pcurrent_to_PNext = @(x) (a/b)*x+c; % Direct line from start to goal
90

```



```

90
91     n = 1;
92     for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
93         Path(i,1) = x(n);
94         Path(i,2) = Pcurrent_to_PNext(x(n));
95         n = n+1;
96     end
97
98     break
99     else
100     end
101
102     %% Going from point A to point B
103     % Pcurrent = Path(end,:);
104
105     if PNext == Pcurrent
106         PNext = [x2 y2];
107     end
108
109     [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
110     x1 = Pcurrent(1);
111     y1 = Pcurrent(2);
112     x2 = PNext(1);
113     y2 = PNext(2);
114     PositionX = [x1,x2];
115
116     Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
117     x = linspace(min(PositionX),max(PositionX),Delta);
118     Pcurrent_to_PNext = @(x) (a/b)*x+c; % Direct line from start to goal
119
120     n = 1;
121     for i = length(Path(:,1)): length(Path(:,1))+length(x)-1
122         Path(i,1) = x(n);
123         Path(i,2) = Pcurrent_to_PNext(x(n));
124         n = n+1;
125     end
126
127     Pcurrent = Path(end,:);
128
129     if abs(Pcurrent(1,1) - Pgoal(1,1)) < stepsize*1.5 && abs(Pcurrent(1,2) - Pgoal(1,2)) < stepsize*1.5
130         break
131     else
132     end
133
134     %% Going around object
135     c=0;
136     n=0;
137     for m = CurrentSegmentNumber(1):length(Obstacle(:,1,1))+CurrentSegmentNumber(1)
138         n = n+1;
139         if m >= length(Counter)
140             c=c+1;
141             Order(n) = c;
142         else
143             Order(n) = m;
144         end
145     end
146
147     for i = 1:length(Obstacle(:,1,1))
148
149         x1 = Obstacle(Order(i),1,CurrentObstacle); % point 1
150         y1 = Obstacle(Order(i),2,CurrentObstacle);
151         x2 = Obstacle(Order(i+1),1,CurrentObstacle); % point 2
152         y2 = Obstacle(Order(i+1),2,CurrentObstacle);
153         PositionY = [y1,y2];
154
155         q = Obstacle_Hit;
156         S = [x2,y2] - [x1,y1];
157         W = q-[x1,y1];
158         Z = (dot(W,S))/(dot(S,S)); % scalar 0-1 if with in segment
159
160         k=0;
161         if x1 ==x2 % verticle
162             Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
163             y = (max(PositionY)-((max(PositionY)-min(PositionY))*(1-Z)):stepsize:max(PositionY);
164
165             for n = length(Path(:,1)): length(Path(:,1))+length(y)-1
166                 k = k+1;
167                 Path(n,1) = x1;
168                 Path(n,2) =y(k);
169             end
170         else % None verticle
171             Pcurrent = [Path(end,1) Path(end,2)];
172             PNext = [x2 y2];
173
174             if PNext == Pcurrent
175                 PNext = [x1 y1];
176             end
177
178             [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
179
180             x1 = Pcurrent(1);

```

```

180         x1 = Pcurrent(1);
181         y1 = Pcurrent(2);
182         x2 = PNext(1);
183         y2 = PNext(2);
184
185         Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
186         x = linspace(x1,x2,Delta);
187         Pcurrent_to_PNext = @(x) (a/b)*x+c; % Direct line from start to goal
188
189         k =0;
190         for N = length(Path(:,1)): length(Path(:,1))+length(x)-1
191             k = k+1;
192             Path(N,1) = x(k);
193             Path(N,2) = Pcurrent_to_PNext(x(k));
194         end
195     end
196 end
197 %% Going to Hit spot to complete 1 loop around object
198
199 Pcurrent = [Path(end,1) Path(end,2)];
200 PNext = Obstacle_Hit;
201 [a,b,c] = computeLineThroughTwoPoints(Pcurrent,PNext);
202
203 x1 = Pcurrent(1);
204 y1 = Pcurrent(2);
205 x2 = PNext(1);
206 y2 = PNext(2);
207 PositionY = [y1,y2];
208
209 if x1==x2
210     Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
211     y = linspace(min(PositionY),max(PositionY),Delta);
212     k =0;
213     for n = length(Path(:,1)): length(Path(:,1))+length(y)-1
214         k = k+1;
215         Path(n,1) = x1;
216         Path(n,2) =y(k);
217     end
218 else
219     Delta = sqrt((x1-x2)^2 + (y1 -y2)^2)/stepsize;
220     x = linspace(x1,x2,Delta);
221     Pcurrent_to_PNext = @(x) (a/b)*x+c; % Direct line from start to goal
222
223     k =0;
224     for N = length(Path(:,1)): length(Path(:,1))+length(x)-1
225         k = k+1;
226         Path(N,1) = x(k);
227         Path(N,2) = Pcurrent_to_PNext(x(k));
228     end
229 end
230
231 %% Going to closest point from object to goal
232 P = Obstacle(:, :,CurrentObstacle);
233 [~,~,JumpPoint] = computeDistancePointToPolygon(P,Pgoal);
234
235 X = 0;
236 Y = 0;
237 A = length(Path(:,1));
238 B = length(Path(:,1));
239
240 threshold = stepsize*1;
241 step = Pgoal;
242 JumpPoint = JumpPoint(1,:);
243 while abs(step(1,1) - JumpPoint(1,1)) > threshold || abs(step(1,2) - JumpPoint(1,2)) > threshold
244     A = A+1;
245     Path(A,1) = Path(B-X,1) ;
246     Path(A,2) = Path(B-X,2) ;
247     X = X+1;
248     Y = Y+1;
249     step = Path(A,:);
250 end
251
252 PNext = Pgoal;
253 Pcurrent = JumpPoint;
254 end
255 end

```

