دانشگاه صنعتی شریف

دانشکده مهندسی مکانیک

عنوان:

تمرینات سری هفتم

Road mapping - Dijkstra’s algorithm

نگارش

محمدسعید صافی زاده

استاد راهنما

دکتر بهزادی پور

آذر ماه ۱۴۰۳

فهرست مطالب

۱ صورت سوالات

می خواهیم مسئله تمرین قبل را به روش Road mapping و به کمک الگوریتم Dijkstra حل کنیم. برای این منظور برنامه ای بنویسید که الف) تعداد موانع N، محتصات دو سر خطوط تشکیل دهنده موانع(هر مانع یک پاره خط است)، مختصات شروع حرکت و محتصات نقطه هدف از یک فایل متن به نام input.txt در مسیر جاری بخواند. فرمت فایل متن به صورت زیر است:

```
N
Sx1, Sy1, Ex1, Ey1
.
.
SxN, SyN, ExN, EyN
X_satrt, Y_start
X_end, Y_end
```

ب) کوتاه ترین مسیر را محاسبه و به همراه کلیه موانع روی یک شکل رسم کند.

۲ سوال امتیازی

فرض کنید هندسه ربات به صورت یک m ضلعی محدب داده شده است که دوران نمی کند. محتصات m راس ربات در نقطه اولیه در انتهای فایل input.txt داده می شود:

```
m
X1, Y1
.
.
Xm,Ym
```

مجددا کوتاه ترین مسیر از نقطه شروع به پایان را به کمک اصلاح موانع و به کارگیری الگوریتم دایکسترا به دست آورده و رسم کنید.

۳ پاسخ سوال اول

۱.۳ توضیح تئوری و الگوریتم نوشته شده در کد

ابتدا به توضیح خلاصه روند پیاده سازی و تئوری الگوریتم می پردازیم. برای این الگوریتم ابتدا از نقطه پایان شروع می کنیم. به تمامی نقاط گراف نیز به جز نقطه شروع که صفر است عدد بی نهایت می دهیم. با شروع از نقطه پایان، فاصله اقلیدسی آن را از هر همسایه آن پیدا می کنیم. اگر این فاصله کمتر از عدد آن راس بود (که در ابتدا دو همسایه نقطه پایان بی نهایت هستند) این عدد جایگزین عدد قبلی می شود. با عبور از نقطه پایان این نقطه وارد یک مجموعه به نام ملاقات شده می شود و از مجموعه رئوس ملاقات نشده حذف می شود. به همین شیوه عمل می کنیم تا به نقطه شروع برسیم. هنگامی که مجموعه ملاقات نشده تهی شد به یافتن مسیر می پردازیم. از مبدا شروع می کنیم و روی همه اعداد یک لوپ می زنیم. اگر فاصله نقطه ی کنونی با نقطه ی دیگر برابر با اختلاف اعداد اطلاق شده به آنها شد یعنی راس درستی را انتخاب کردیم و کوتاه تریم مسیر از آن می گذرد. پس به آن راس می رویم و دوباره راس بعدی را مطابق همین روش پیدا می کنیم تا به راس پایانی برسیم. در این حین به چند باگ می خوریم:

۱. **باگ اول** : اگر تمام نقاط را در نظر بگیریم که به هم راه دارند یعنی مسیری بین هر دو نقطه دلخواه وجود دارد نهایتاً کوتاه ترین مسیر، مسیر بین شروع و پایان خواهد بود.

روش حل: برای این موضوع تابعی می نوشتم که مانند تمرین قبل برخورد پاره خط حاصل از دو نقطه گراف با اضلاع را چک می کند.

۲. **باگ دوم** : حالا که برخورد با اضلاع چک می کنیم، مشکل دیگری رخ می دهد. اگر مانند تمرین قبل یک ۵ ضلعی داشته باشیم و به یک راس آن برسیم، کوتاه ترین مسیر به آن سمت ۵ ضلعی از وسط آن می گذرد که با شرط برخورد به اضلاع چک نمی شود، چراکه مسیر مجاز است به رئوس برسد و بین رئوس ضلعی نیست تا آن را چک کنیم!

روش حل : اگر تمام قطر های چندضلعی ها را پیدا کنیم و آن ها را در دسته موانع قرار دهیم با چک کردن برخورد پاره خط حاصل از دو نقطه از گراف و اضلاع و قطر ها این مشکل حل خواهد شد.
۳. **باگ سوم** : برای یافتن قطر ها و رسم موانع نیاز داریم تا تعداد و نوع چندضلعی ها را بدانیم.
روش حل: در ابتدا از دیتا ماتریس موانع را که به اسم obstacle lines می باشد را پیدا می کنیم و چک می کنیم که اگر مختصات دوم هر خط با مختصات اول خط بعدی مشابه بود، این دو ضلع به یکدیگر وصل شده اند و زمانی که این شرط برقرار نباشد بنابراین چندضلعی ما کامل شده است. به این صورت تمام چندضلعی ها را در سلول polygons می ریزیم. اکنون می توانیم برای هر چند ضلعی قطر های آن را پیدا کنیم و هم چنین موانع را به تفکیک رسم کنیم.

۲.۳ توضیح کد

```
%% importing the data and finding all diagonals
data = readmatrix('data.txt', 'NumHeaderLines', 0);
N = data(1,1);
Pi = data(N+2,1:2);
Pf = data(N+3,1:2);
obstacle_lines = data(2:N+1,1:4);
p_obstacle_lines = obstacle_lines;
```

در ابتدا با دستور readmatrix ورودی را از فایل متنی می خوانیم. این دستور خط اول را به عنوان هدینگ شناسایی می کند و آن را وارد ماتریس data نمی کند برای همین از آرگومان NumHeaderLines با مقدار صفر استفاده کردم تا متوجه شود که هدینگ نداریم. سپس تعداد موانع، مختصات شروع و پایان و موانع را به ترتیب در متغیر های N ، Pi ، Pf و obstacle lines ریختم.

۳.۳ توضیح کد

```
lines = obstacle_lines;
points = unique(lines(:, 1:2), 'rows');
for i = 1:size(lines, 1)
points = unique([points; lines(i, 3:4)], 'rows');
end
```

سپس با استفاده از یک حلقه، مختصات های تکرار شده در موانع را حذف کردم و تمامی مختصات ها را در یک ماتریس n در دو ریختم.

٤.۳ توضیح کد

```
polygons = {};
n = size(lines, 1);
visited = false(n, 1);

% Find all polygons in the obstacles list
for i = 1:n
if ~visited(i)
polygon = [];
current_line = i;
while ~visited(current_line)
visited(current_line) = true;
polygon = [polygon; lines(current_line, :)];
next_point = lines(current_line, 3:4);
next_line = find(~visited & (ismember(lines(:, 1:2), next_point, 'rows') ...
 | ismember(lines(:, 3:4), next_point, 'rows')), 1);
```

```
if ~isempty(next_line)
current_line = next_line;
else
break;
end
end
polygons{end + 1} = polygon;
end
end
```

با استفاده از کد بالا ابتدا بر روی تمامی موانع پیمایش می کنیم و الگوریتم توضیح داده شده در مقدمه را پیاده می کنیم به این صورت که با استفاده از حلقه while تا زمانی که مختصات دوم موجود در یک خط با مختصات اول موجود در خط بعدی یکسان باشد، داده های بعدی را نیز عضوی از چندضلعی فعلی در نظر می گیریم و اگر این مورد رخ ندهد، داده بعدی مربوط به یک چندضلعی دیگر خواهد بود. بدیهی است که اگر یک خط با خط بعدی داده مشترکی به گونه ای که شرح داده شد نداشته باشد آن مانع به عنوان یک پاره خط مستقل در نظر گرفته خواهد شد.

در اینجا ذکر این نکته الزامی است که با توجه به منطق کد برای شناسایی چندضلعی ها، حتماً باید مختصات اضلاع به ترتیب و پشت سر هم داده شوند در غیر این صورت در یافتن چندضلعی به مشل می خوریم!

```
% finding diagonals for each polygon
output_lines = lines;
for p = 1:length(polygons)
polygon = polygons{p};
num_points = size(polygon, 1);
if num_points >= 4
unique_points = unique([polygon(:, 1:2); polygon(:, 3:4)], 'rows');
num_points = size(unique_points, 1);
for i = 1:num_points
for j = i+1:num_points
p1 = unique_points(i, :);
p2 = unique_points(j, :);
if ~ismember([p1, p2], polygon, 'rows') && ~ismember([p2, p1], polygon, 'rows')
output_lines = [output_lines; p1, p2];
end
end
end
end
end
```

اکنون برای رفع باگ دوم توضیح داده شده در بالا نیاز داریم تا قطر های چندضلعی ها را به دست آوریم به این منظور از کد بالا استفاده می کنیم. کد بالا ابتدا موانع را می گیرد. برای هر چندضلعی یک سلول جدید تولید می کند و با دو بار استفاده از حلقه for و پیمایش بر روی داده ها و قرار دادن دو محتصات از دو راس غیرمتوالی، تمامی قطر ها را پیدا می کند. شرط num points >=4 نیز برای این استفاده شده است که مثلث و پاره خط قطری ندارند.

```
% Create output list
edited_obstacle_lines = [];
for i = 1:size(output_lines, 1)
edited_obstacle_lines = [edited_obstacle_lines; output_lines(i, :)];
end
```

```
% updating the obstacle lines with diagonals
obstacle_lines = edited_obstacle_lines;
data = [N,0,0,0;obstacle_lines;Pi,0,0;Pf,0,0];
```

سپس مقادیر به دست آمده را(مختصات قطر ها را) به لیست موانع اصلی اضافه می کنیم و با استفاده از دو خط آخر، لیست موانع و دیتا را آپدیت می کنیم تا در هر دو قطر های نیز لحاظ شوند و مسیر از وسط چندضلعی ها عبور نکند.

```
unvisited_points = [data(2:N+1,1:2)];
Total_collection = [];
final_lst = [Pf,0];
flag = true; % is used for the first time
while height(unvisited_points) > 0
if flag
for i=1:height(unvisited_points)
if ~colission_finder(data(i+1,1:2)',Pf',obstacle_lines)
Total_collection(i,:) = [data(i+1,1:2), norm(Pf - data(i+1,1:2))];
else
Total_collection(i,:) = [data(i+1,1:2), inf];
end
end
flag = false;
else
[M,I] = min(Total_collection(:,3));
current_point = Total_collection(I,1:2);
final_lst(end+1,1:3) = Total_collection(I,1:3);
Total_collection(I,:) = [];
[A,B] = ismember(current_point,unvisited_points,'rows');
unvisited_points(B,:) = [];
for j=1:height(unvisited_points)
distance = norm(current_point - unvisited_points(j,1:2));
[C, D] = ismember(unvisited_points(j,1:2),Total_collection(:,1:2),"rows");
if Total_collection(D,3) > distance+M && ...
 ~colission_finder(current_point',unvisited_points(j,1:2)',obstacle_lines)
Total_collection(D,3) = distance+M;
end
end
end

end
```

در ادامه با استفاده از کد بالا در خط اول یک مجموعه ملاقات نشده تشکیل می دهیم. از Total collection برای ریختن مختصات و اعداد مختصات استفاده می کنیم. در ابتدا نیاز است که از نقطه پایانی شروع کنیم و به هر راس، عدد نسبت دهیم. برای این منظور که فقط باید یک بار اتفاق بیفتد از یک flag استفاده می کنیم. سپس با استفاده از حلقه for اگر این فلگ true بود یعنی در اولین لوپ بودیم، با پیمایش بر روی مجموعه ملاقات نشده، فاصله آن ها را تا راس پایانی به دست می آورد و در Total collection می ریزد. در اینجا برای رفع باگ اول بیان شده در بالا دو حالت رخ می دهد. در حالت اول اگر پاره خط حاصل از نقطه پایانی و راس انتهایی اصلاع را قطع نکنند همه چی خوب پیش می رود اما اگر قطع کنند باید مقدار راس را بی نهایت قرار دهیم(در واقع این دو راس به همدیگر متصل نمی شوند) که این کار را با تابعی به نام collision finder انجام داده ام و در ادامه به توضیح نحوه عملکرد آن می پردازم.

بعد از گذر از نقطه پایانی به سراغ کمترین فاصله از نقطه پایانی می رویم و کار را با انتخاب آن نقطه شروع می کنیم(خط ۱۶ کد بالا). چون عدد و مختصات این نقطه فیکس شده است آن را در ماتریس final lst می ریزیم و آن را از دو مجموعه Total collection و unvisited points حذف می کنیم. دلیل یافتن Index این نقطه در خط ۲۰ کد بالا معلوم است چون در مراحل بعدی که رئوس را از ماتریس حذف می کنیم، دیگر در سه مجموعه ای که داریم لزوماً این رئوس Index یکسانی نخواهند داشت.
در این مرحله که نقطه جدید را انتخاب کردیم با استفاده از یک حلقه for در خط ۲۲ بر روی داده های مجموعه unvisited پیمایش می کنیم و علاوه بر چک کردن برخورد این دو نقطه با اضلاع، فاصله را نیز بدست می آوریم، سپس در خط ۲۵ چک می کنیم که اگر از عدد بدست آمده کنونی بزرگتر باشد، با این عدد جایگزین شود. در این جا لوپ تمام می شود و در لوپ جدید داده بعدی بر اساس minimum فاصله انتخاب می شود و دوباره این مراحل تکرار خواهند شد.
محاسبه ی کمترین فاصله از رئوس را برای نقطه شروع جداگانه در کد زیر انجام دادم:

```
% caluclating the starting point
min_starting_point =inf;

for i=1:height(final_lst)
c = norm(Pi-final_lst(i,1:2));
if min_starting_point > c + final_lst(i,3) && ~colission_finder(Pi',final_lst(i,1:2)',obst
min_starting_point = c + final_lst(i,3);
end
end

final_lst(end+1,1:3) = [Pi, min_starting_point];
```

به این صورت که ایتدا کمترین فاصله بی نهایت است، بعد مشابه بالا با تک تک موانع فاصله و برخورد با اضلاع را چک می کنیم و نهایتاً کمترین مقدار را در متغیر min starting point می ریزم.

```
path = final_lst(end,1:3);
current = final_lst(end,1:3);
%checked = [final_lst(end,1:3)];
unchecked = final_lst(1:end-1,:);

while (path(end,3) ~= 0)
for i=1:height(unchecked)
if abs((norm(current(1,1:2)-unchecked(i,1:2)) - abs(current(1,3) - ...
 unchecked(i,3)))) < 0.000001
current = unchecked(i,1:3);
path(end+1,1:3) = current;
%checked(end+1,1:3) = current;
unchecked(i,:) = [];

break
end

end
end
```

در ادامه باید مسیر را پیدا کنیم، بنابراین یک ماتریس مسیر در خط اول تعریف می کنم که شامل نقطه شروع و فاصله آن است و در خط بعدی این نقطه را به عنوان نقطه کنونی در نظر می گیرم. یک لیست نقاط چک نشده تعریف می کنم. سپس با استفاده از حلقه while تا زمانی که نقطه

نهایی که فاصله اش صفر است در ماتریس path قرار نگرفته باشد، مدام حلقه for را ران می کند. در این حلقه for فاصله نقطه ی کنونی را با یکی از رئوس مجموعه چک نشده ها که در حال پیمایش رو آن هستیم بدست می آورد و با اختلاف فاصله ای که خودمان بدست آورده بودیم مقایسه می کند(باید مساوی باشند اما چون در این جا خطای محاسباتی داریم اختلاف کمی دارند). در صورتی که این دو مقدار مساوی باشند بنابراین نقطه جدید ما، پیدا شده و به ماتریس مسیر یعنی path اضافه می شود.

```
plot(Pi(1),Pi(2),'o','color','k','LineWidth',2);
axis equal;
grid on;
hold on
plot(Pf(1),Pf(2),'o','color','k','LineWidth',2);
for i=1:length(polygons)
plot(polygons{i}(:,1), polygons{i}(:,2),'color',[0 0.4470 0.7410],'LineWidth',1)
plot(polygons{i}(:,3), polygons{i}(:,4),'color',[0 0.4470 0.7410],'LineWidth',1)
end

plot(path(:,1),path(:,2),'--ko','LineWidth',1,'MarkerSize',5,'MarkerEdgeColor','k');
hold off
```

در انتها با استفاده از کد بالا به رسم موانع و مسیر می پردازیم.

```
% true if collision occures
function out = colission_finder(point1,point2, obstacle_lines)
for i=1:height(obstacle_lines)
Q = [obstacle_lines(i,3); obstacle_lines(i,4)];
P = [obstacle_lines(i,1); obstacle_lines(i,2)];
A = point1;
B = point2;
%syms alpha beta
%eqn1 = (Q-P)*alpha + P == (A-B)*beta + B;
alphabeta = ([Q-P A-B])\(A-P);
alpha = alphabeta(1);
beta = alphabeta(2);
if (0 < alpha) && (alpha < 1) && (0 < beta) && (beta < 1)
out = true;
return
end
end
out = false;
end
```

همچنین کد تابع collision finder به صورت بالا می باشد. این تابع دو نقطه و ماتریس موانع را دریافت می کند. سپس معادله پاره خط ناشی از دو نقطه را با شیب α و معادله هر ضلع را با شیب β به دست می آورد. با مساوی قرار دادن این دو معادله اگر برخوردی باشد، شرط خط ۱۳ برقرار می شود و این تابع true بر می گرداند که به معنای برخورد می باشد و در غیر این صورت عبارت false را باز می گرداند.

۵.۳ کد کامل

کد کامل این سوال به شکل زیر می باشد. هم چنین خروجی گرفته شده از نرم افزار متلب کد در یک فایل PDF در پوشه گزارش، ضمیمه شده است.

```
%% importing the data and finding all diagonals
data = readmatrix('data.txt', 'NumHeaderLines', 0);
N = data(1,1);
Pi = data(N+2,1:2);
Pf = data(N+3,1:2);
obstacle_lines = data(2:N+1,1:4);
p_obstacle_lines = obstacle_lines;

lines = obstacle_lines;
points = unique(lines(:, 1:2), 'rows');
for i = 1:size(lines, 1)
points = unique([points; lines(i, 3:4)], 'rows');
end

polygons = {};
n = size(lines, 1);
visited = false(n, 1);

% Find all polygons in the obstacles list
for i = 1:n
if ~visited(i)
polygon = [];
current_line = i;
while ~visited(current_line)
visited(current_line) = true;
polygon = [polygon; lines(current_line, :)];
next_point = lines(current_line, 3:4);
next_line = find(~visited & (ismember(lines(:, 1:2), next_point, 'rows') | ...
 ismember(lines(:, 3:4), next_point, 'rows')), 1);
if ~isempty(next_line)
current_line = next_line;
else
break;
end
end
polygons{end + 1} = polygon;
end

end

% finding diagonals for each polygon
output_lines = lines;
for p = 1:length(polygons)
```

```
polygon = polygons{p};
num_points = size(polygon, 1);
if num_points >= 4
unique_points = unique([polygon(:, 1:2); polygon(:, 3:4)], 'rows');
num_points = size(unique_points, 1);
for i = 1:num_points
for j = i+1:num_points
p1 = unique_points(i, :);
p2 = unique_points(j, :);
if ~ismember([p1, p2], polygon, 'rows') && ~ismember([p2, p1], polygon, 'rows')
output_lines = [output_lines; p1, p2];
end
end
end
end
end

% Create output list
edited_obstacle_lines = [];
for i = 1:size(output_lines, 1)
edited_obstacle_lines = [edited_obstacle_lines; output_lines(i, :)];
end

% updating the obstacle lines with diagonals
obstacle_lines = edited_obstacle_lines;
data = [N,0,0,0;obstacle_lines;Pi,0,0;Pf,0,0];

unvisited_points = [data(2:N+1,1:2)];
Total_collection = [];
final_lst = [Pf,0];
flag = true; % is used for the first time
while height(unvisited_points) > 0
if flag
for i=1:height(unvisited_points)
if ~colission_finder(data(i+1,1:2)',Pf',obstacle_lines)
Total_collection(i,:) = [data(i+1,1:2), norm(Pf - data(i+1,1:2))];
else
Total_collection(i,:) = [data(i+1,1:2), inf];
end
end
flag = false;
else
[M,I] = min(Total_collection(:,3));
current_point = Total_collection(I,1:2);
final_lst(end+1,1:3) = Total_collection(I,1:3);
Total_collection(I,:) = [];
[A,B] = ismember(current_point,unvisited_points,'rows');
```

```
unvisited_points(B,:) = [];
for j=1:height(unvisited_points)
distance = norm(current_point - unvisited_points(j,1:2));
[C, D] = ismember(unvisited_points(j,1:2),Total_collection(:,1:2),"rows");
if Total_collection(D,3) > distance+M && ...
 ~colission_finder(current_point',unvisited_points(j,1:2)',obstacle_lines)
Total_collection(D,3) = distance+M;
end
end
end

end

% caluclating the starting point
min_starting_point =inf;

for i=1:height(final_lst)
c = norm(Pi-final_lst(i,1:2));
if min_starting_point > c + final_lst(i,3) && ...
 ~colission_finder(Pi',final_lst(i,1:2)',obstacle_lines)
min_starting_point = c + final_lst(i,3);
end
end

final_lst(end+1,1:3) = [Pi, min_starting_point];

path = final_lst(end,1:3);
current = final_lst(end,1:3);
%checked = [final_lst(end,1:3)];
unchecked = final_lst(1:end-1,:);

while (path(end,3) ~= 0)
for i=1:height(unchecked)
if abs((norm(current(1,1:2)-unchecked(i,1:2)) - ...
abs(current(1,3) - unchecked(i,3)))) < 0.000001
current = unchecked(i,1:3);
path(end+1,1:3) = current;
%checked(end+1,1:3) = current;
unchecked(i,:) = [];

break
end

end
end

plot(Pi(1),Pi(2),'o','color','k','LineWidth',2);
```

```
axis equal;
grid on;
hold on
plot(Pf(1),Pf(2),'o','color','k','LineWidth',2);
for i=1:length(polygons)
plot(polygons{i}(:,1), polygons{i}(:,2),'color',[0 0.4470 0.7410],'LineWidth',1)
plot(polygons{i}(:,3), polygons{i}(:,4),'color',[0 0.4470 0.7410],'LineWidth',1)
end

plot(path(:,1),path(:,2),'--ko','LineWidth',1,'MarkerSize',5,'MarkerEdgeColor','k');
hold off
```

۶.۳ رسم کوتاه ترین مسیر طی شده

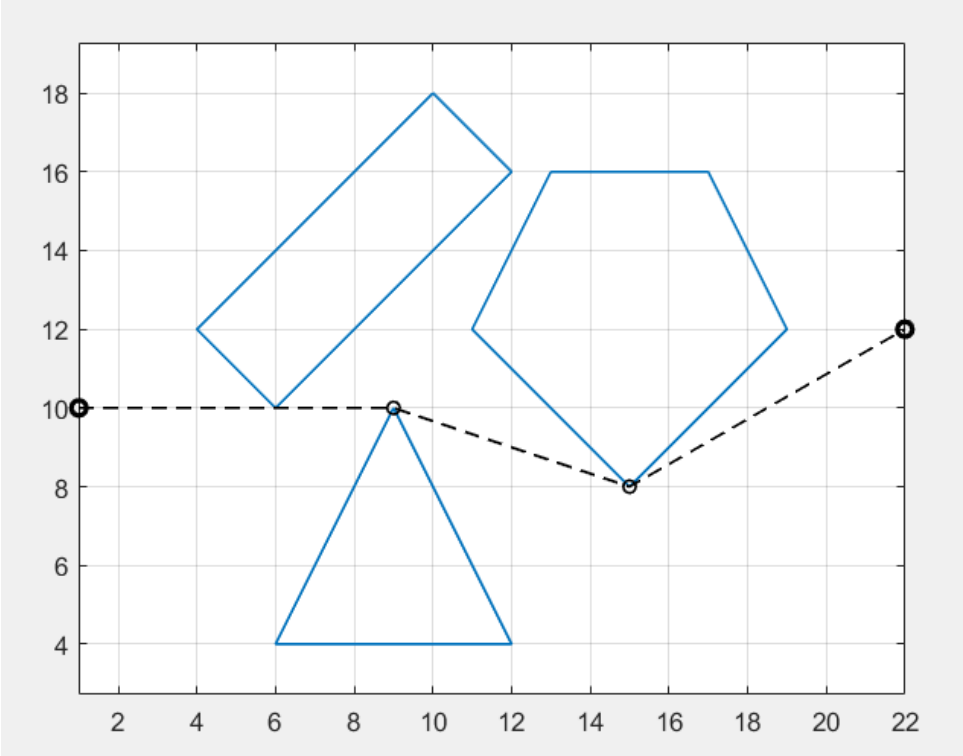
کوتاه ترین مسیری که باید طی شود در ماتریس path ذخیره شده است که برای مثال تمرین قبل به صورت شکل ۱ می باشد. ستون سوم بیانگر فاصله تا نقطه پایان می باشد.

```
path =

    1.0000   10.0000   22.3868
    9.0000   10.0000   14.3868
   15.0000    8.0000    8.0623
   22.0000   12.0000         0
```

شکل ۱: نقاطی که باید برای کوتاه ترین فاصله تا مبدا طی شوند

هم چنین مسیر طی شده و شکل رسم شده برای حرکت ربات بین موانع به صورت شکل ۲ می باشد.



شکل ۲: مسیر حرکت ربات