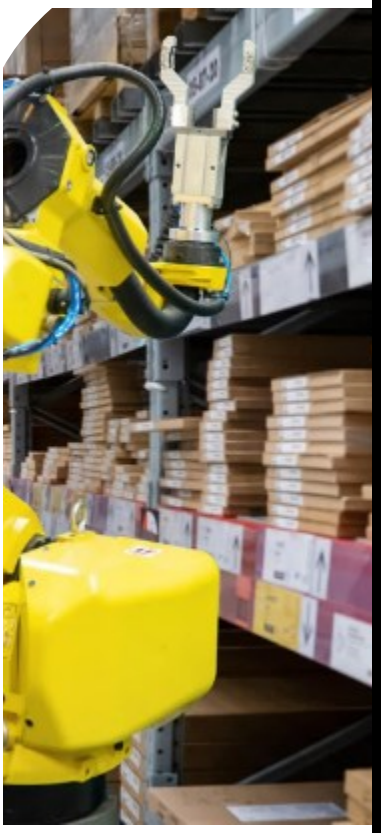
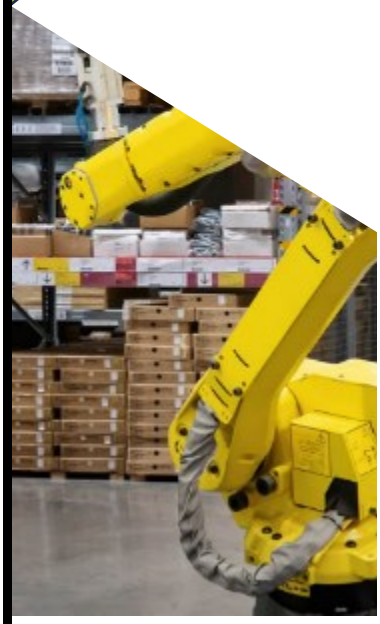
به نام خدا



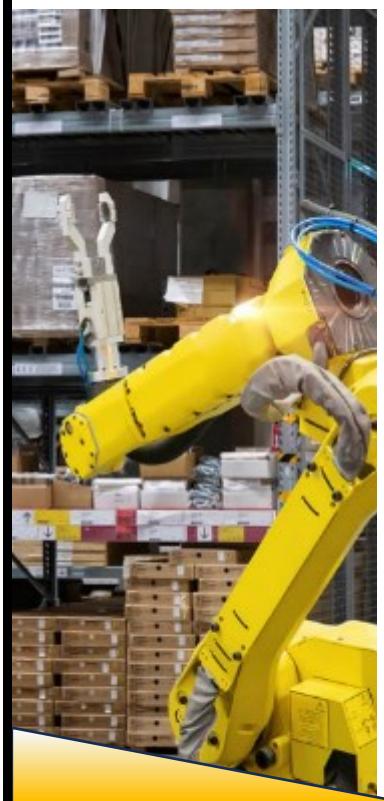


گزارش تمرین شماره 7 رباتیک

نام دانشجو : **عرفان رادفر**

شماره دانشجویی : **99109603**

استاد درس : **دکتر سعید بهزادی پور**



پاییز 1402

Erfan Radfar
99109603

Robotics

Dr. Saeed Behzadipour

فهرست

صورت سوال

طراحی مسیر به روش **Road Mapping**

میخواهیم مسئله تمرین قبل را به روش Road mappingو به کمک الگوریتم Dijkstraحل کنیم. برای این منظور برنامه ای بنویسید که:

الف) تعداد موانع N ، مختصات دو سر خطوط تشکیل دهنده موانع(هر مانع یک پاره خط است)، مختصات نقطه شروع حرکت ومختصات نقطه هدف از یک فایل متن به نام input.txtدر مسیر جاری بخواند. فرمت فایل متن به صورت زیر است:

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

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

سوال امتیازی (۲۵٪)

فرض کنید هندسه ربات به صورت یک mضلعی محدب مدل شده است که دوران نمیکند. مختصات mراس ربات در نقطه اولیه در انتهای فایل input.txtداده میشود:

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

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

قسمت الف

main

برای این قسمت، ابتدا فایل main را باز می‌کنیم. با استفاده از readlines داده ها را استخراج میکنیم و با دستور str2num ، موانع و مختصات ربات را به ترتیب در obs_arr و rob_arr ذخیره می کنیم. از arrange_obs استفاده می‌کنیم تا موانع به ترتیب در یک آرایه cell به نام obs ذخیره شوند. در نهایت نمودار موانع را می‌کشیم.

```
%%% First we determine obstacles and robot dimension
file=readlines('input.txt');
r_N=find(file=='N'); % row number for 'N'
r_m=find(file=='m'); % row number for 'm'
obs_arr=[]; %obstacle array
rob_arr=[]; %robot array
start_end=[]; %start and end point for robot path
for i=r_N:r_m
    num=str2num(file(i,:));
    if ~isempty(num)
        if length(num)==4
            obs_arr=cat(1,obs_arr,num);
        else
            start_end(end+1,:)=str2num(file(i,:));
        end
    end
end

for i=r_m:size(file,1)
    num=str2num(file(i,:));
    if ~isempty(num)
        rob_arr=cat(1,rob_arr,num);
    end
end

obs=arrange_obs(obs_arr);
for i=1:length(obs)
        plot(obs{i}(:,1),obs{i}(:,2));
        hold on
end
scatter(start_end(:,1),start_end(:,2));
```

arrange_obs

در این تابع آرایه پاره خط ها به صورت رندوم دریافت شده و نقاط هر مانع و بسته یا باز بودن آن مشخص می‌شود.

```
function obs=arrange_obs(obs_arr)
%%% Now we rearrange and find each close loop obstacles
k=0; % Number of obstacle groups
obs={}; % Each cell array is all points for on obstacles
while ~isempty(obs_arr)
    k=k+1; % We go after the next obstacle
    side=obs_arr(1,:);
    obs_arr(1,:)=[];
    obs{k}=[side(1:2);side(3:4)];
    signal=1;
    while signal
        signal=0;
        i=1;
        while i<=size(obs_arr,1)
            seg=obs_arr(i,:);
            [state,new_end]=is_connected(side,seg);
            if state
                obs{k}(end+1,:)=new_end;
                side=seg;
                obs_arr(i,:)=[];
                signal=1; % if signal is 1, it means we at least find one
                % connected line segement
            end
            i=i+1;
        end
    end
    if obs{k}(1,1)~=obs{k}(end,1) || obs{k}(1,2)~=obs{k}(end,2)
        temp=obs{k}(1,:); obs{k}(1,:)=obs{k}(2,:);
        obs{k}(2,:)=temp;
    end
end
end
```

این حلقه تا زمانی ادامه پیدا می‌کند که تمام پاره خطها بررسی شده باشد. در هر حلقه، یک پاره خط انتخاب شده وبا is_connected بررسی می‌شود که آیا با پاره خط دیگری، راس مشترک دارد یا خیر. اگر نداشت، یعنی یک مانع "باز" پیدا کرده ایم، در غیر اینصورت، پاره خط متصل را انتخاب می‌کنیم و مراحل تکرار می‌شود.

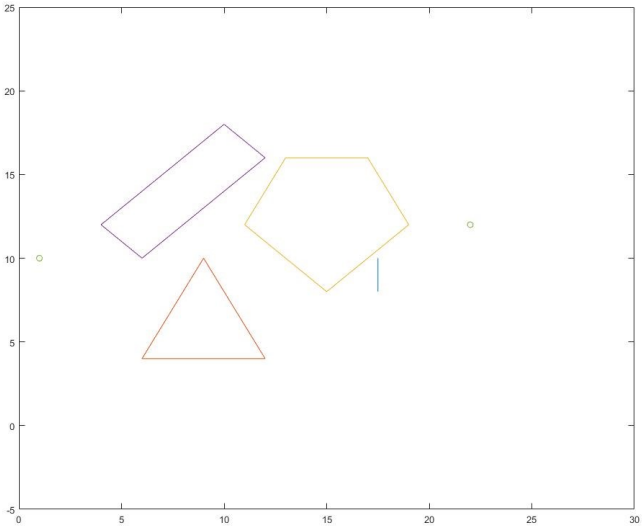
در اینجا نیز، صرفا مانع حلقه بسته را طوری منظم می‌کنیم که دو نقطه ابتدایی و انتهایی یکی باشند.

is_connected

وظیفه این تابع آن است که ببیند بین دو ورودی چهار المانی، آیا نقطه مشترکی وجود دارد یا خیر.

```
% This function checks if two line segments sidere
% connected or not. Returns state 0 if not sidend state 1  if yes
% sidelso, it returns the p
function [state,new_end]=is_connected(side,seg)
state=0;
new_end=0;
if side(1)==seg(1) && side(2)==seg(2)
    state=1;
    new_end=seg(3:4);
end
if side(1)==seg(3) && side(2)==seg(4)
    state=1;
    new_end=seg(1:2);
end
if side(3)==seg(1) && side(4)==seg(2)
    state=1;
    new_end=seg(3:4);
end
if side(3)==seg(3) && side(4)==seg(4)
    state=1;
    new_end=seg(1:2);
end
end
```

در نهایت نمودار موانع و نقاط مبدا و مقصد رسم می‌شود.

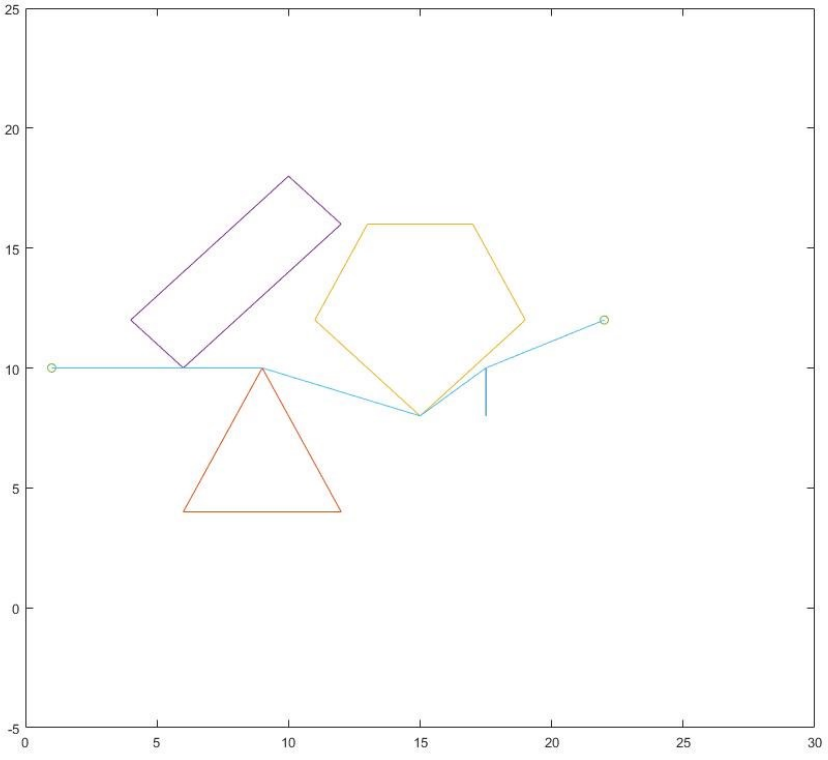


قسمت ب

در تابع main ، با دستور زیر ادامه می‌دهیم.

```
% Now we utilize 'Road Mapping' algoritm
path=find_path(obs,start_end);

hold on
plot(path(:,1),path(:,2));
xlim([0,30]);
ylim([-5,25]);
```

در اینجا با استفاده از تابع، find_path کوتاه ترین مسیر تا مقصد را پیدا می‌کنیم. در نهایت مسیر را رسم می‌کنیم.



find_path

این تابع با دریافت سلول موانع obs و نقاط شروع و پایان، کوتاه ترین مسیر تا مقصد را پیدا می‌کند.

```
function path=find_path(obs,start_end)
points=obs;
points{end+1}=start_end; % points are all the points including obstacle and
                         % starting and finishing point
% Now add the minimum distance to them (the 3rd column in each cell)
% Also add a signal to each point (the 4th column in each cell) which
% determines whether the point is visited (1) or not (0)
% Furthermore we record the points which should be connected to
% make minimum distance to distination.(5th column=i,6th column=j)
points_num=0; % total number of points
for i=1:size(points,2)
     points{i}=Uniq(points{i}); % To remove repeated points
     points{i}(:,end+1)=1e6;
     points{i}(:,end+1)=0;
     points{i}(:,end+1)=0;
     points{i}(:,end+1)=0;
     points_num=points_num+size(points{i},1);
end
points{end}(2,3)=0; % sets distance of end point to zero
% Now we update distances
visp=[size(points,2),2]; % determines indices of visiting point
visp_pre=[];
visit=0;
while visit<points_num
    if visit, visp=find_smallest_dis(points);end
    if isempty(visp), visit=visit+1; continue, end
    points{visp(1)}(visp(2),4)=1; % sets visited state to one
    visp_tmp=[];
    dis_min=1e6; % minimum differential distance between current and next point
    for i=1:size(points,2)
        for j=1:size(points{i},1)
            if ~points{i}(j,4)
                if ~collision(visp,i,j,obs,points)
                    % distance from end point
                    diff_dis=norm(points{visp(1)}(visp(2),1:2)-
points{i}(j,1:2));
                    end_dis=points{visp(1)}(visp(2),3)+diff_dis;
                    if end_dis < points{i}(j,3)
                        points{i}(j,3)=end_dis;
                        % Identifies which points is next in path
                        % to the destination
                        points{i}(j,5:6)=visp;
                    end
                    if points{i}(j,3)<dis_min
                        dis_min=points{i}(j,3);
                        visp_tmp=[i,j];
                    end
                end
            end
```

تا زمانی که تمام نقاط ملاقات شوند، حلقه را ادامه می‌دهیم.

برای هر نقطه انتخاب شده ، با استفاده از تابع Collision برخورد را بررسی می کنیم. اگر برخوردی با نقطه ملاقات شده مقصد وجود نداشت آنگاه "فاصله" را آپدیت می‌کنیم.

```
        end
    end
    visit=visit+1;
    visp=visp_tmp;
end

path=start_end(1,:);
idx=[size(points,2),1];
while norm(path(end,:)-start_end(2,:))>1e-12
    idx=points{idx(1)}(idx(2),5:6);
    path(end+1,:)=points{idx(1)}(idx(2),1:2);
end
end
```

در این کد، برای هر نقطه به ترتیب در هر ستون مختصات x,y ، کوتاه ترین فاصله از مقصد ، وضعیت ملاقات شده بودن یا نبودن و اندیس نقاط بعدی آن ها که باید برای رسیدن به مقصد طی شود، در آرایه points آورده شده است. در نهایت با استفاده از این اندیس ها، کوتاه ترین مسیر مشخص می‌گردد. همچنین در هر مرحله، از تابع find_smallest_dis برای یافتن کمترین فاصله در بین نقاط ملاقات نشده استفاده می‌کنیم.

collision

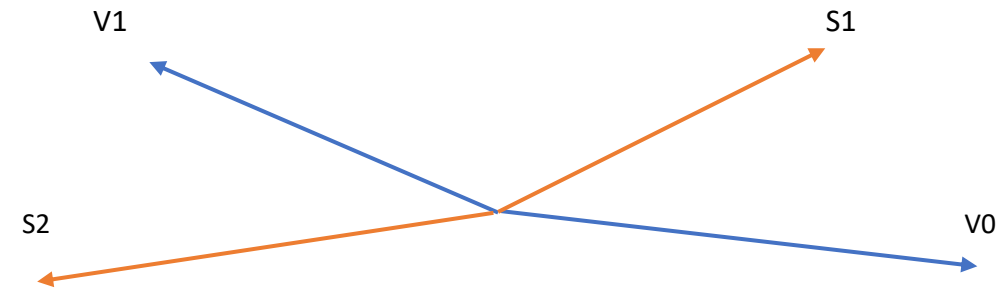
این تابع ، اندیس های نقاط مبدا index و مقصد i_,j_ را دریافت کرده و بررسی می کند که آیا پاره خط داده شده، با مانعی برخورد می‌کند یا خیر. اگر برخورد وجود داشته باشد، یعنی مسیر ممکن نیست و خروجی 1 را بر می‌گرداند. برای بررسی برخورد، ماتریس بردار جهت و بردار آلفا و بتا ساخته می‌شود . در صورتی که بتا بین 0 تا منفی 1 و آلفا بین 0 تا 1 بود، یعنی برخورد وجود دارد. همچنین اگر بردارهای جهت نقاط کنونی-ثانویه و مانع یکسان بود، آنگاه ماتریس A دترمینان صفر دارد و برخوردی نداریم.

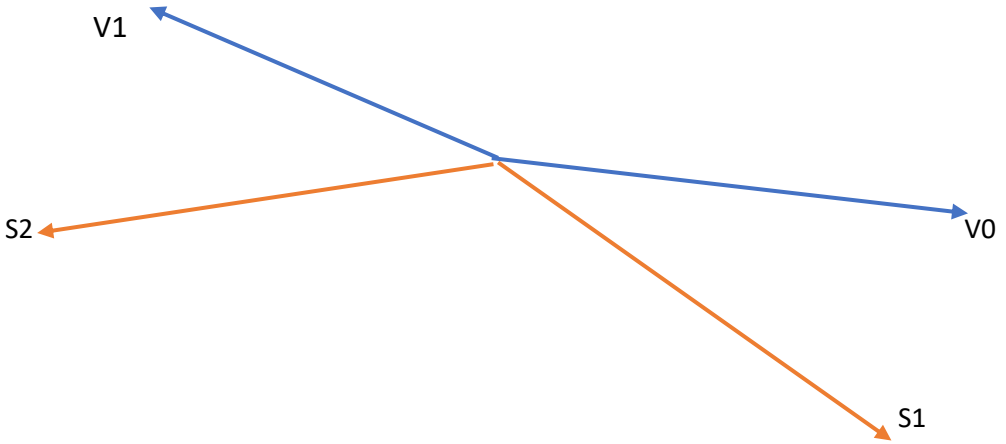
```
function collide=collision(index,i_,j_,obs,points)
% collision either happens when we pass cross through
% a line or when we pass through a close pointstacle
collide=0;
tol=1e-12;
ii=index(1); jj=index(2);
p1=points{ii}(jj,1:2);
v1=points{i_}(j_,1:2)-points{ii}(jj,1:2);
for i=1:size(points,2)-1
    for j=1:size(points{i},1)
        if j==size(points{i},1)
            if size(obs{i},1)==size(points{i},1)
                k=j;
            else
                k=1;
            end
        else
            k=j+1;
        end
        if isequal([ii,jj,i_,j_],[i,j,i,k]) || isequal([ii,jj,i_,j_],[i,k,i,j])
            continue
        end
        p2=points{i}(j,1:2);
        v2=points{i}(k,1:2)-points{i}(j,1:2);
```

```
        A=[v1',v2']; B=(p2-p1)';
        if abs(det(A))>tol
            temp=A\B;
            alpha=temp(1); beta=-temp(2);
            if alpha>tol && alpha<1-tol && beta>-tol && beta<1+tol
                collide=1;
                break
            end
        end

        if ii==size(points,2),continue,end
        if size(obs{ii},1)~=size(points{ii},1)
            if jj==1
                s1=-points{ii}(jj,1:2)+points{ii}(end,1:2);
                s2=points{ii}(jj+1,1:2)-points{ii}(jj,1:2);
            elseif jj==size(points{ii},1)
                s1=-points{ii}(jj,1:2)+points{ii}(jj-1,1:2);
                s2=points{ii}(1,1:2)-points{ii}(jj,1:2);
            else
                s1=-points{ii}(jj,1:2)+points{ii}(jj-1,1:2);
                s2=points{ii}(jj+1,1:2)-points{ii}(jj,1:2);
            end
        elseif jj==1 || jj==size(points{ii},1)
            continue
        else
            s1=-points{ii}(jj,1:2)+points{ii}(jj-1,1:2);
            s2=points{ii}(jj+1,1:2)-points{ii}(jj,1:2);
        end
         ii0=points{ii}(jj,5); jj0=points{ii}(jj,6);
         v0=points{ii0}(jj0,1:2)-points{ii}(jj,1:2);
         tv1=wrapTo2Pi(atan2(v1(2),v1(1))-atan2(s1(2),s1(1)));
         ts2=wrapTo2Pi(atan2(s2(2),s2(1))-atan2(s1(2),s1(1)));
         tv0=wrapTo2Pi(atan2(v0(2),v0(1))-atan2(s1(2),s1(1))+tol);

         if (tv1<ts2 && tv0>ts2) || (tv1>ts2 && tv0<ts2)
            collide=1;
         end

    end
    if collide
        break
    end
end

end
```

برای حالتی که آلفا صفر یا یک باشد، در مقادیری که برخورد با رئوس باشد، کد نمی تواند به درستی برخورد را پیشبینی کند. برای همین از الگوریتمی که در زیر کد آورده شده استفاده می‌کنیم.

دو حالت زیر برای برخورد با مانع (با فرض آنکه از خارج مانع محدب وارد شویم داریم)





در اینجا v0 معکوس قدم ماقبل و v1 قدم کنونی است. همچنین S1 و S2 نیز دو ضلع مانع محدب هستند. در صورتی که دو ضلع محدب در یک طرف باشند (شکل پایینی) برخورد نداریم اما در غیر اینصورت (شکل بالایی) برخورد داریم.

قسمت امتیازی

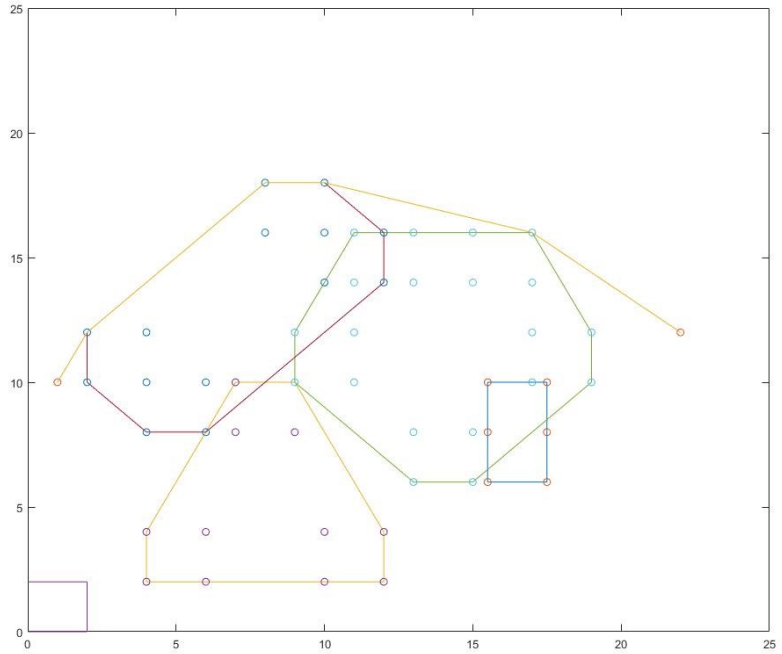
در این قسمت ربات را نقطه ای در نظر نمی‌گیریم بلکه آن را شکل محدب فرض می‌کنیم.

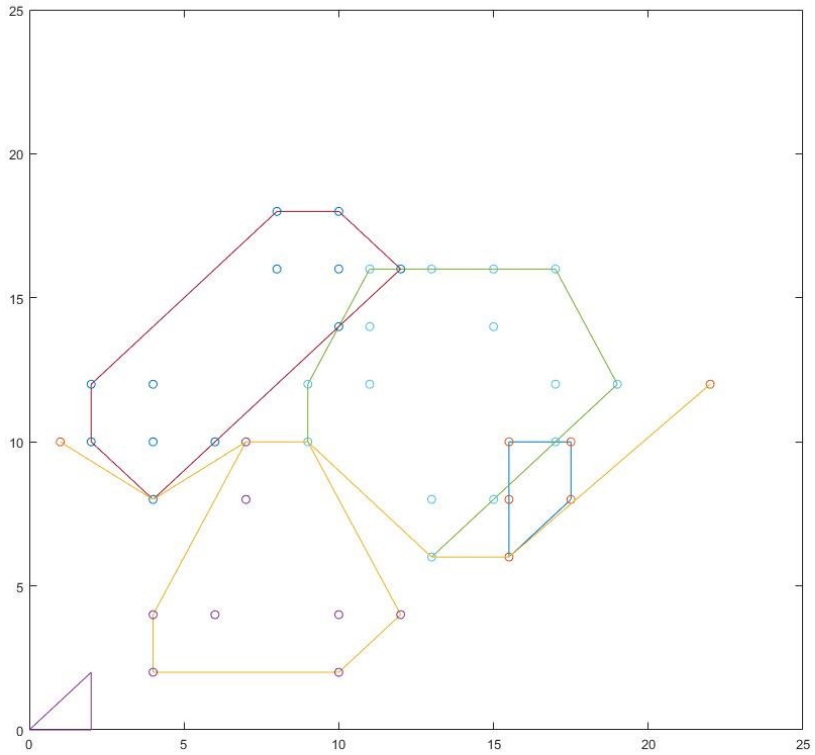
```
%% Part 2
[conv_obs,conv_points]=conv_Hull(obs,rob_arr);
% Now we utilize 'Road Mapping' algoritm
figure
for i=1:length(conv_obs)
        plot(conv_obs{i}(:,1),conv_obs{i}(:,2));
        hold on
        scatter(conv_points{i}(:,1),conv_points{i}(:,2));
        hold on
end
scatter(start_end(:,1),start_end(:,2));

% Now we utilize 'Road Mapping' algoritm
path=find_path(conv_obs,start_end);

hold on
plot(path(:,1),path(:,2));
xlim([0,25]);
ylim([0,25]);

hold on
rob=[rob_arr;rob_arr(end,:)];
plot(rob(:,1),rob(:,2));
```

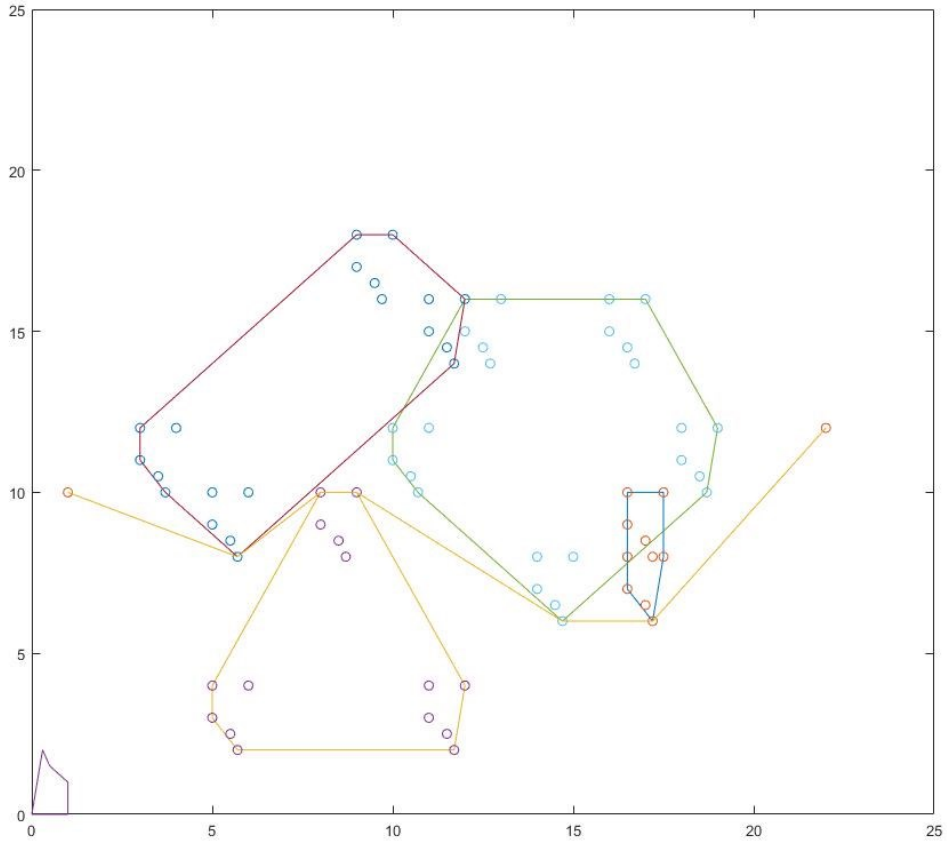
ابتدا با استفاده از تابع conv_Hull ، نقاط مرزی موانع جدید را محاسبه می‌کنیم. سپس دوباره از تابع find_path استفاده کرده تا کوتاه ترین مسیر ممکن برای ربات بین مبدا و مقصد را پیدا کنیم. تابع تشخیص برخورد collision، طوری تعریف شده است که در صورت over-lap موانع، مشکلی در الگوریتم دایکسترا به وجود نیاید. در نهایت شکل نهایی مسیر به فرمت زیر است.

همچنین شکل ربات در گوشه پایینی سمت چپ نمودار نمایش داده شده است.

برای چند هندسه ربات متفاوت، اشکال ترسیم شده اند.


0
5
10
15
20
25


0
5
10
15
20
25



باید دقت کرد که اندازه ربات از حدی بیشتر نباشد که حتی در نقطه شروع، با یکی از موانع برخورد داشته باشیم. در این حالت کد خطا می‌دهد که به این معناست که همچین حالتی ممکن نیست.

conv_Hull

وظیفه این تابع آن است که با دریافت هندسه ربات و هندسه موانع، موانع جدید را با الگوریتم convex Hull طراحی کرده و نقاط حول موانع (conv_points)و کوچک ترین چند ضلعی که تمام نقاط را در بر بگیرد (conv_obs) خروجی می‌دهد.

```
function [conv_obs,points]=conv_Hull(obs,rob)
tol=1e-6;
points={};
conv_obs={};
for i=1:size(obs,2)
    if isequal(obs{i}(1,1:2),obs{i}(end,1:2))
        k=size(obs{i},1)-1;
    else
        k=size(obs{i},1);
    end
    z=0;
    for j=1:k
```

```
        for m=1:size(rob,1)
            z=z+1;
            points{i}(z,:)=rob(1,:)-rob(m,:) + obs{i}(j,:);
        end
    end
end
% Now we have the points, we find the convex around each obstacle
for i=1:size(points,2)
    % First, find the furthest left point
    max_x=-inf;
    max_j=1;
    for j=1:size(points{i},1)
        if points{i}(j,1)>max_x
            max_j=j;
            max_x=points{i}(j,1);
        end
    end
    conv_obs{i}(1,:)=points{i}(max_j,:);
    k=1; % counter for 'conv_obs{i}'
    while norm(conv_obs{i}(1,:) - conv_obs{i}(k,:))>tol || k==1
        min_angle=2*pi+tol;
        if k==1
            vector1=[1,0];
        else
            vector1=conv_obs{i}(k-1,:) - conv_obs{i}(k,:);
        end
        for j=1:size(points{i},1)
            vector2=points{i}(j,:) - conv_obs{i}(k,:);
            angle=wrapTo2Pi(atan2(vector2(2),vector2(1))-
atan2(vector1(2),vector1(1)));
            if angle<tol || angle>2*pi-tol || norm(vector2)<tol ||
norm(vector2-vector1)<tol
                continue
            end
            if angle<min_angle
                min_angle=angle;
                conv_obs{i}(k+1,:)=points{i}(j,:);
            end
        end
        k=k+1;
    end
end

end
```

در اینجا زاویه تمام نقاط مرتبط به یک مانع محاسبه شده و کمترین آن ها به عنوان نقطه بعدی انتخاب می‌شود.