به نام خدا



**گزارش تمرین شماره 7 رباتیک**

نام دانشجو : **عرفان رادفر**

شماره دانشجویی : **99109603**

استاد درس : **دکتر سعید بهزادی پور**



پاییز 1402

فهرست

[صورت سوال 2](#_Toc154770584)

[قسمت الف 3](#_Toc154770585)

[main 3](#_Toc154770586)

[arrange\_obs 4](#_Toc154770587)

[is\_connected 5](#_Toc154770588)

[قسمت ب 6](#_Toc154770589)

[find\_path 7](#_Toc154770590)

[collision 8](#_Toc154770591)

[قسمت امتیازی 11](#_Toc154770592)

[conv\_Hull 13](#_Toc154770593)

# صورت سوال

طراحي مسير به روش Road Mapping

ميخواهيم مسئله تمرين قبل را به روش Road mappingو به كمك الگوريتم Dijkstraحل كنيم. براي اين منظور برنامه اي بنويسيد كه:  
الف) تعداد موانع N ، مختصات دو سر خطوط تشكيل دهنده موانع(هر مانع يك پاره خط است)، مختصات نقطه شروع حركت ومختصات نقطه هدف از يك فايل متن به نام input.txtدر مسير جاري بخواند. فرمت فايل متن به صورت زير است:

N  
Sx1, Sy1, Ex1, Ey1  
. .  
SxN, SyN, ExN, EyN  
X\_start, Y\_start  
X\_end, Y\_end

ب) كوتاه ترين مسير را محاسبه كند و به همراه كليه موانع روي يك شكل رسم كند.  
سوال امتيازي (%٢٥)  
فرض كنيد هندسه ربات به صورت يك mضلعي محدب مدل شده است كه دوران نميكند. مختصات mراس ربات در نقطه  
اوليه در انتهاي فايل input.txtداده ميشود:

m  
X1, Y1  
. .  
Xm, Ym

مجددا كوتاه ترين مسير از نقطه شروع به پايان را با كمك اصلاح موانع و بكارگيري الگوريتم دايكسترا بدست آورده و رسم كنيد.

# 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)

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 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.

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

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

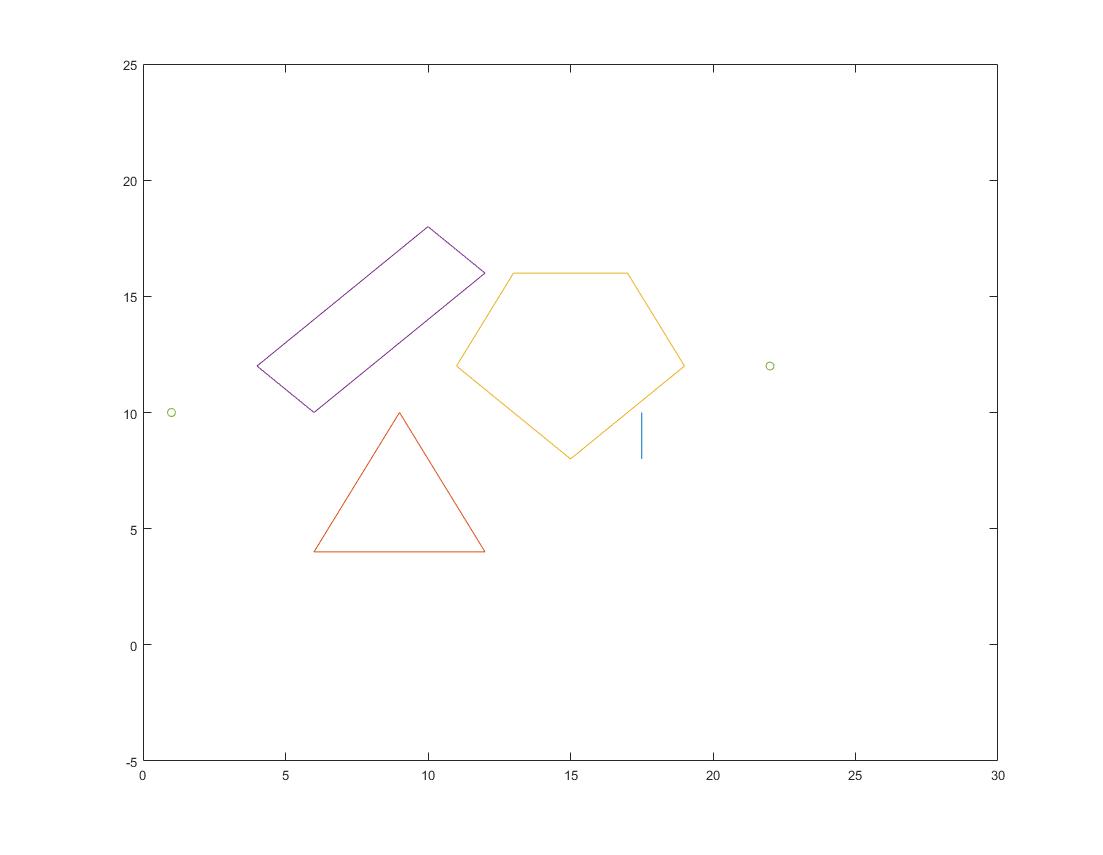
if side(3)==seg(3) && side(4)==seg(4)

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);

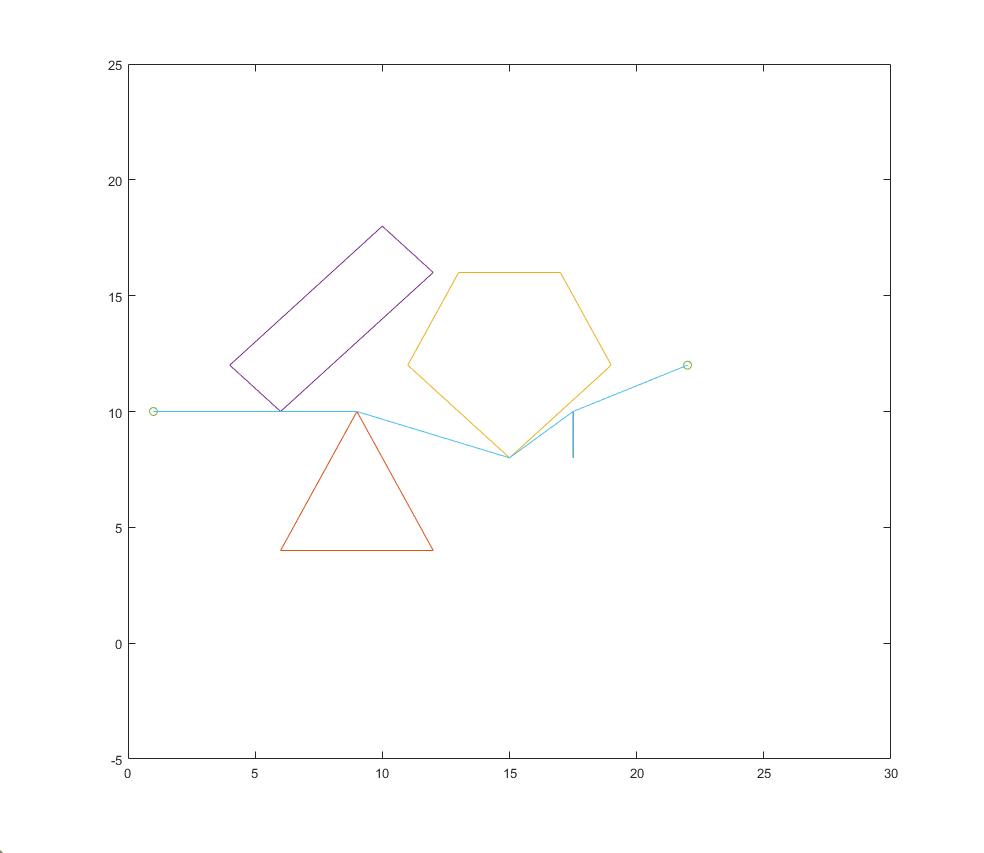
hold on

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=[];

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

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;

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

if points{i}(j,3)<dis\_min

dis\_min=points{i}(j,3);

visp\_tmp=[i,j];

end

end

end

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 indexو 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

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.

break

end

end

end

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

S1 V1

V0 S2

V1

V0 S2

S1

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

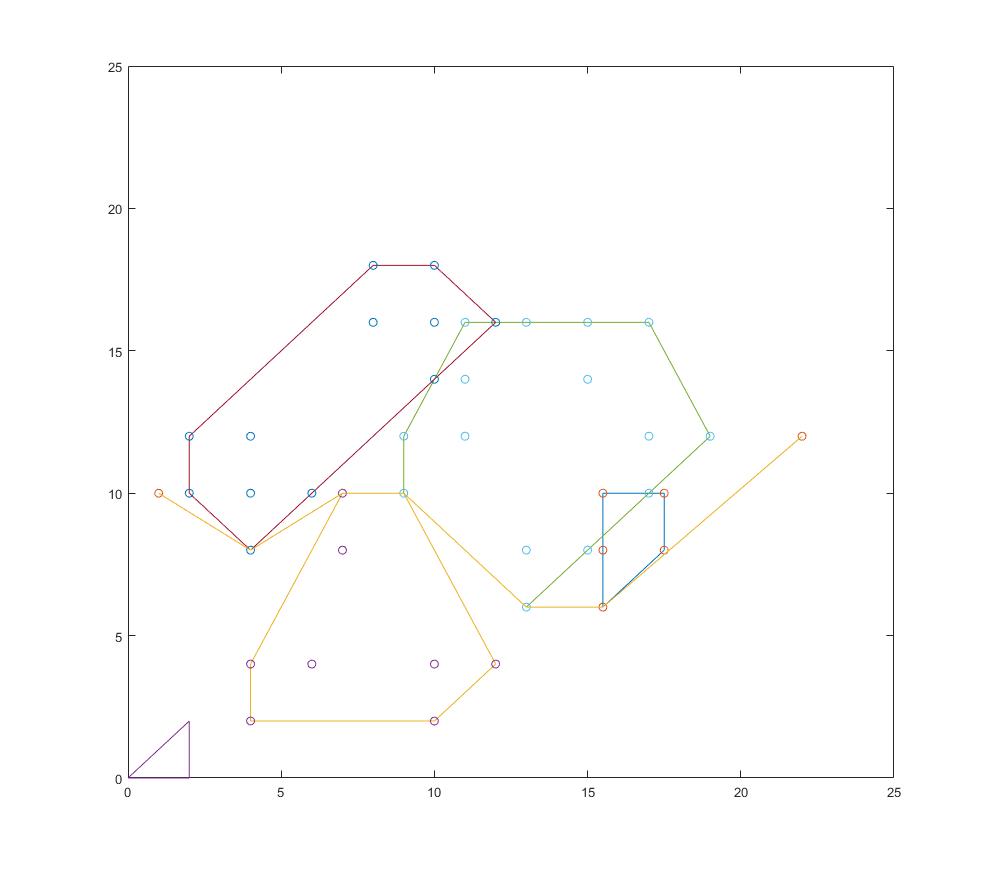
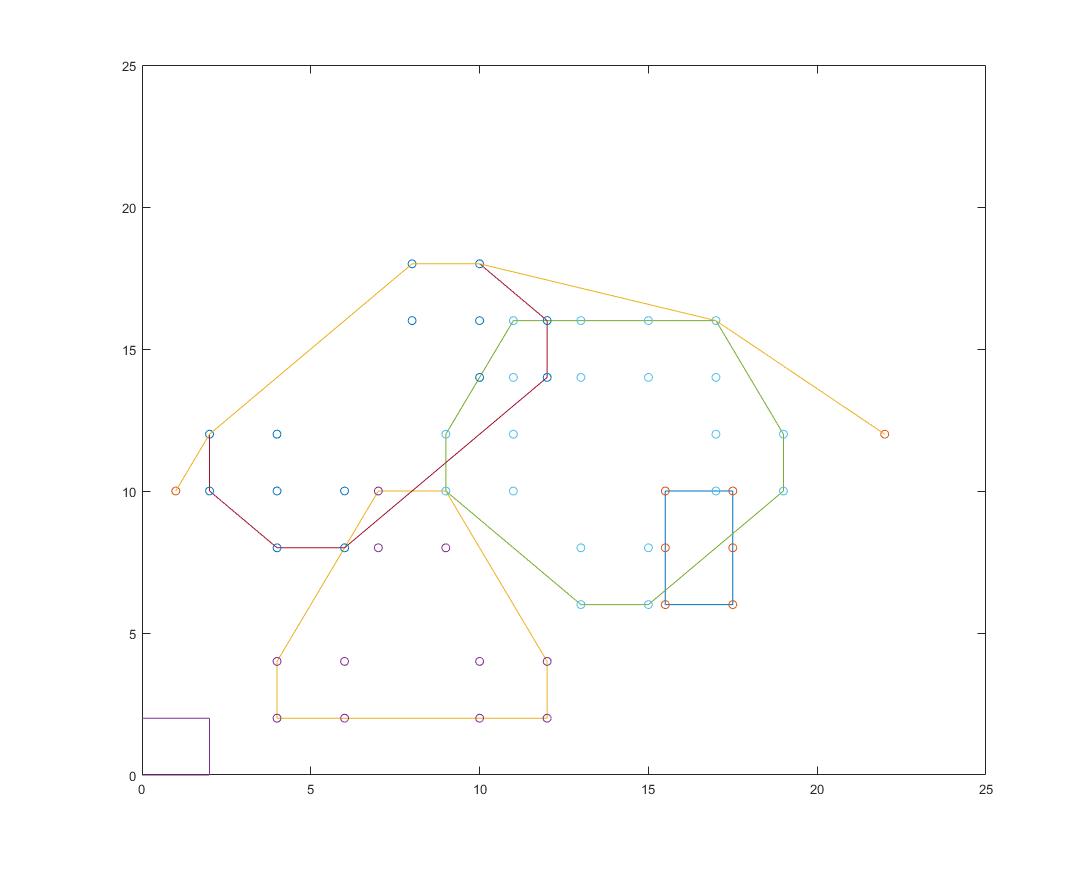
rob=[rob\_arr;rob\_arr(end,:)];

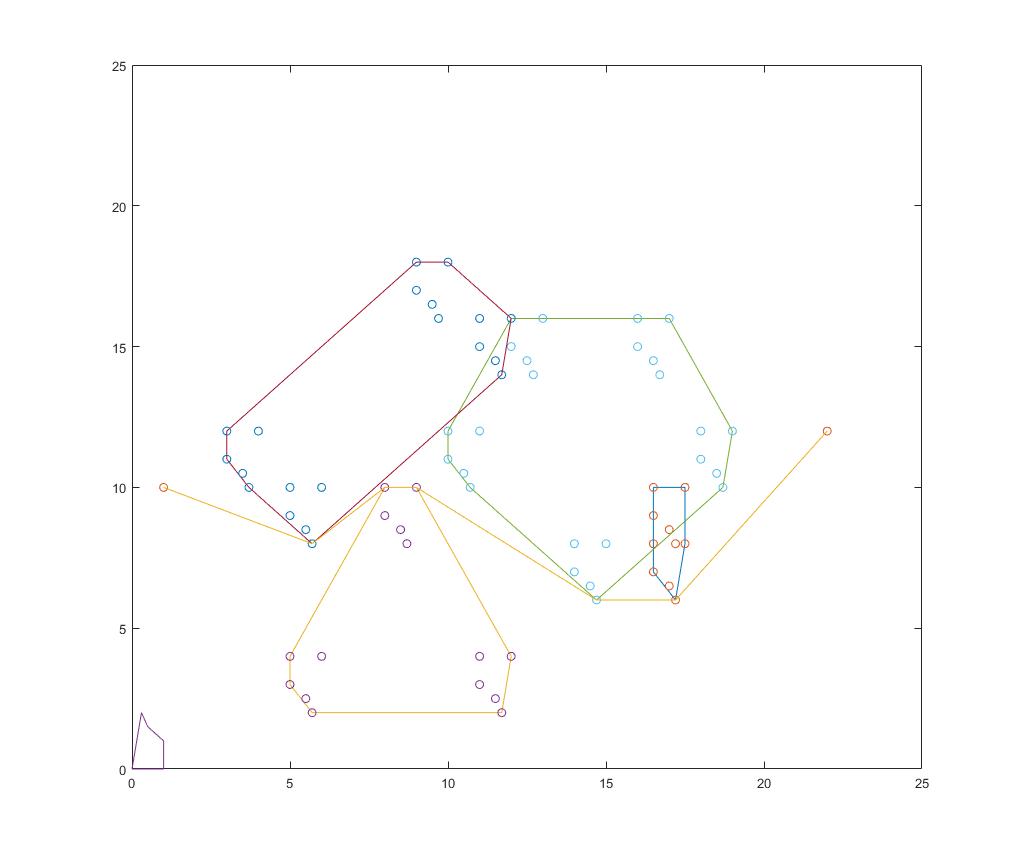
plot(rob(:,1),rob(:,2));

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.





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.