

به نام نور



تمرین شماره 6 رباتیک

دانشجو: ریحانه نیکوبیان

شماره دانشجویی: 99106747

سال تحصیلی: 1402

(1)

1) Write a MATLAB function that generates the via points of the path. The function receives the coordinates of the starting point X_s , final destination X_f , η which is the scaling factor for the attractive field, and matrix B which stores the information of the obstacles. It has a distance of influence with a scaling factor (α) for the obstacle. The function should stop generating via points when the moving point's distance from X_f is less than 0.1. It should then Return matrix P which has the coordinates of the via points starting from X_s and ending by X_f .

`function P = Path_generator (Xs, Xf, eta, B)`

در این سوال از ما خواسته شده تا مسیر حرکت یک ربات در صفحه (که با یک نقطه شبیه سازی شده است) با حضور موانع را به روش میدان پتانسیل طراحی کنیم طوری که از مبدا مشخص شده به مقصد دلخواه ما برسد.

می دانیم در این روش جهت حرکت ما در جهت بیشترین کاهش پتانسیل برآیند است. که بردار F این جهت را نشان می دهد.

$$\vec{F}(\vec{x}) = -\nabla \left(U_{att}(\vec{x}) + \sum_{i=1}^n U_{rep_i}(\vec{x}) \right)$$

$$\vec{F}_{att}(\vec{x}) = -\nabla U_{att}(\vec{x}) \quad \left(U_{att} = |\vec{x} - \vec{x}_E|^2 \frac{1}{2} \eta \right)$$

$$= \eta (\vec{x}_E - \vec{x})$$

$$\vec{F}_{rep_i}(\vec{x}) = \nabla U_{rep_i}(\vec{x}) = \begin{cases} 0 & \rho_i > \rho_{oi} \\ \alpha \cdot \frac{1}{\rho_i^3(\vec{x})} \left(\frac{1}{\rho_i(\vec{x})} - \frac{1}{\rho_{oi}} \right) (\vec{x} - B_i(\vec{x})) & \rho_i \leq \rho_{oi} \end{cases}$$

$$B_i(\vec{x}) : \text{مختصات نزدیکترین نقطه از مانع } i \text{ به } \vec{x}$$

این روش به این شکل است که از مبدا شروع می کند و بعد از محاسبه F ، یک گام کوچک در جهت F بر میداریم. بعد دوباره F را محاسبه می کنیم و یک گام کوچک به سمت F جدید. تا نهایت به مقصد دلخواه برسیم.

Fatr ناشی از میدان جاذبه مقصد و Fnep ناشی از میدان دافعه هر مانع است. با توجه به مقصد و موانع برای هر کدام به ضریب تعریف میشود که شدت میدان را نشان می دهد. (گاما و الفا)

Xe مقصد را نشان می دهد. پارامترهای دیگر در شکل مشخص اند.

$$(\vec{x}) \text{ هم فاصله } \vec{x} \text{ از منبع } i \text{ (الف)}$$

$$P_{oi} : \text{شعاع اثر مانع } i \text{ (ب)}$$

پایه الگوریتم هم همین است. از مبدا شروع می کند، نیروی برآیند را حساب می کند و یک گام برمیدارد و همینطور تکرار می کند تا جایی که به اندازه کافی به مقصد نزدیک شده باشد.

Fatr که به راحتی محاسبه می شود. اما برای محاسبه Bi (کمترین فاصله در هر لحظه با مانع) کمی دشوارتر می شود. در هر گام، به ازای هر مانع باید این نقطه محاسبه شود. پس یک لوپ به تکرار تعداد موانع باید بنویسیم که به ازای هر مانع Bi، فاصله X تا این نقطه را حساب کند. اگر فاصله کمتر از شعاع اثر مانع باشد، نیرو را با توجه به رابطه بالا حساب کند. و همینطور این مرحله به ازای هر مانع تکرار می شود و نیروها باهم جمع می شوند. در نهایت بردار یکه نیرو حساب می شود و در جهت بردار یکه حرکت می کنیم. داریم:

$$\vec{F_d} = \frac{\vec{F_{att}} + \sum_{i=1}^n \vec{F_{nep_i}}}{|\vec{F_{att}} + \sum_{i=1}^n \vec{F_{nep_i}}|}$$

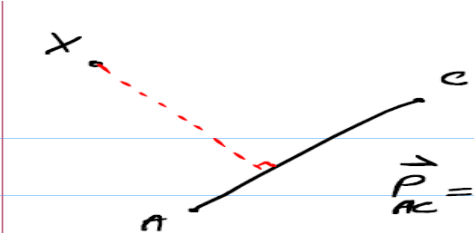
$$\vec{x}_{i+1} \leftarrow \vec{x}_i + \epsilon \vec{F_d}$$

اما محاسبه نقطه Bi:

در حالت کلی نمیشود فهمید Bi روی کدام ضلع مانع است. پس به ازای هر ضلع باید کمترین فاصله X تا آن محاسبه شود. از بین این نقاط، نقطه ای که کمترین فاصله را با ایکس دارد انتخاب می کنیم. پس در لوپ مانع، باید یک لوپ به تعداد تکرار اضلاع قرار بگیرد. روش محاسبه هر نقطه به شرح زیر است:

رابطه خطی که از دو راس می گذرد:

۱. رابطه خطی از A و C می گذرد:



$$\vec{P}_{AC} = \alpha (\vec{A} - \vec{C}) + \vec{C} \quad \alpha \in \mathbb{R}$$

رابطه خطی که عمود بر این خط باشد و از ایکس عبور کند:

$$\vec{q}_\perp = \beta \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} (\vec{A} - \vec{C}) + \vec{X} \quad \beta \in \mathbb{R}$$

برای پیدا کردن محل تلاقی این دو رابطه را باهم برابر می گذاریم و به رابطه ساده شده زیر می رسیم:

$$\begin{aligned} \vec{P}_{AC} = \vec{q}_\perp &= \alpha (\vec{A} - \vec{C}) + \vec{C} = \beta \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} (\vec{A} - \vec{C}) + \vec{X} \\ \Rightarrow \underbrace{\begin{bmatrix} \vec{A} - \vec{C} & -\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} (\vec{A} - \vec{C}) \end{bmatrix}}_D \begin{bmatrix} \alpha \\ \beta \end{bmatrix} &= \vec{X} - \vec{C} \Rightarrow \\ \begin{bmatrix} \alpha \\ \beta \end{bmatrix} &= D^{-1} (\vec{X} - \vec{C}) \end{aligned}$$

از رابطه اول مشخص است اگر α کوچکتر از صفر باشد نقطه عمود قبل از C و اگر بزرگتر از یک باشد بعد از A قرار می گیرد. بنابراین در حالت اول کمترین فاصله تا ضلع نقطه C و در حالت دوم کمترین فاصله X تا ضلع نقطه A می شود. در حالتی که $0 < \alpha < 1$ عمود روی ضلع قرار می گیرد و نقطه برخورد می شود:

$$\vec{H} = \alpha (\vec{A} - \vec{C}) + \vec{C}$$

حال که نقطه B به دست آمد، فاصله هم به راحتی محاسبه می شود. برای هر ضلع حساب می کنیم و از بین این فاصله ها کمترین باید مشخص شود

که می شود کمترین فاصله مانع تا نقطه ایکس. و اگر این فاصله کمتر از شعاع اثر بود نیرو با روابط بالا حساب می شود .

کد دقیقاً همین الگوریتم را به زبان برنامه نویسی پیاده می کند:

```
function P=Path_generator(xs,xf,eta,B)
X=xs;
n=size(B,2); %number of obstacle
k=0.1;
P=xs;
gam=1;
while norm(X-xf)>0.1
    gam=gam+1 ;
    Fatt=-eta*(X-xf);
    F=0;
    for i=1:n
        j=size(B{i},2)-2; % number of edge
        min=1000;
        efd=B{i}{1};
        for k=1:j
            if k<j
                C=B{i}{k+2};
                A=B{i}{k+3};

            else
                C=B{i}{k+2};
                A=B{i}{3};
            end
            R=-[0,-1;1,0]*(A-C);
            D=cat(2,(A-C),R);
            T=inv(D)*(X-C);
            alpha=T(1);
            if alpha<0
                H=C;
            elseif alpha>1
                H=A;
            else
                H=alpha*(A-C)+C;
            end
            d=norm(X-H);
            if d<min
                h=H;
                min=d;
            end
        end
        if min<efd
            Fnep=B{i}{2}*(min)^(-3)*((1/min)-(1/efd))*(X-h);
        else
```

```

        Fnep=0;
    end
    F=F+Fnep;
end
Ftot=F+Fatt;
Fd=Ftot/norm(Ftot);
next_step=0.1*Fd+X;
P=cat(2,P,next_step);
e=gam-2;
if e>2
    if next_step==P(:,e)
        break
    end
end
X=next_step;

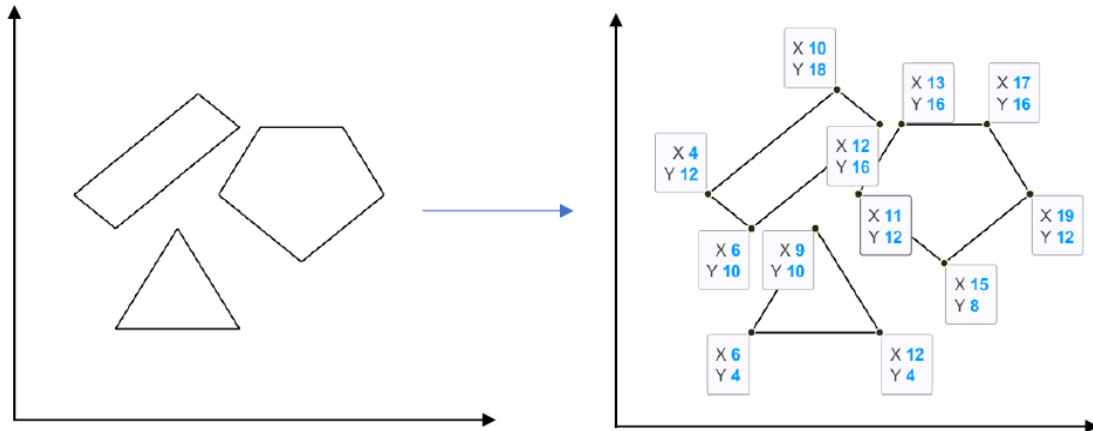
end
end

```

این کد ورودی مبدأ، مقصد، ضریب اثر مقصد و اطلاعات موانع رو به شکل `structure` می گیرد. در `B` به تعداد موانع ارایه وجود دارد. در هر ارایه، درایه اول ضریب اثر مانع، درایه دوم شعاع موثر مانع، و درایه های بعد راس های اضلاع هستند. دقیقاً به شکلی که توضیح دادم نقاط محاسبه می شوند و در `P`، نقاط محاسبه شده ذخیره می شوند.

(2)

2) Test your code for the following example with $\tilde{\eta} = \alpha = 1$, $\varepsilon = 0.1$ and distance of influence equal to 2 for all obstacles. The coordinates of the obstacles' end points are shown in the following figure. Plot the obstacles as well as your generated path in a MATLAB figure and submit a detailed report and your results along with your MATLAB code. $X_s = (1, 10)$, $X_f = (22, 12)$



با توجه به تابعی که در سوال یک نوشتیم ، این سوال حل می شود. فقط کافی است در اسکریپت جدید اطلاعات مانع به فرم مشخص شده در تابع، همچنین مبدا، مقصد و ضریب اثر میدان جاذبه مقصد را مشخص کنیم. مقدار گام نیز باید در خود تابع به صورت دستی وارد می شود. با این ورودی ها تابع سوال قبل فراخوانی کنیم تا مسیر را به شکل ماتریس نقاط P به ما بدهد. با دستور `polyshape` برای هر مانع نقاط راس ها را مشخص می کنیم و با `plot` رسم می کنیم. سپس روی تصویر، مسیر و نقاط ابتدا و انتها را `plot` می کنیم.

کد این سوال به شکل زیر است:

```
clc
clear
close all

B={2,1,[4;12],[10;18],[12;16],[6;10]},{2,1,[13;16],[17;16],[19;12],[15;8],[11;12]},{2,1,[9;10],[12;4],[6;4]};
xs=[1;10];
xf=[22;12];
eta=1;
[R,P]=Path_generator(xs,xf,eta,B);
obs1=polyshape([4 10 12 6],[12 18 16 10]);
obs2=polyshape([13 17 19 15 11],[16 16 12 8 12]);
obs3=polyshape([9 12 6],[10 4 4]);
plot(obs1);
hold on
plot(obs2);
hold on
```

```

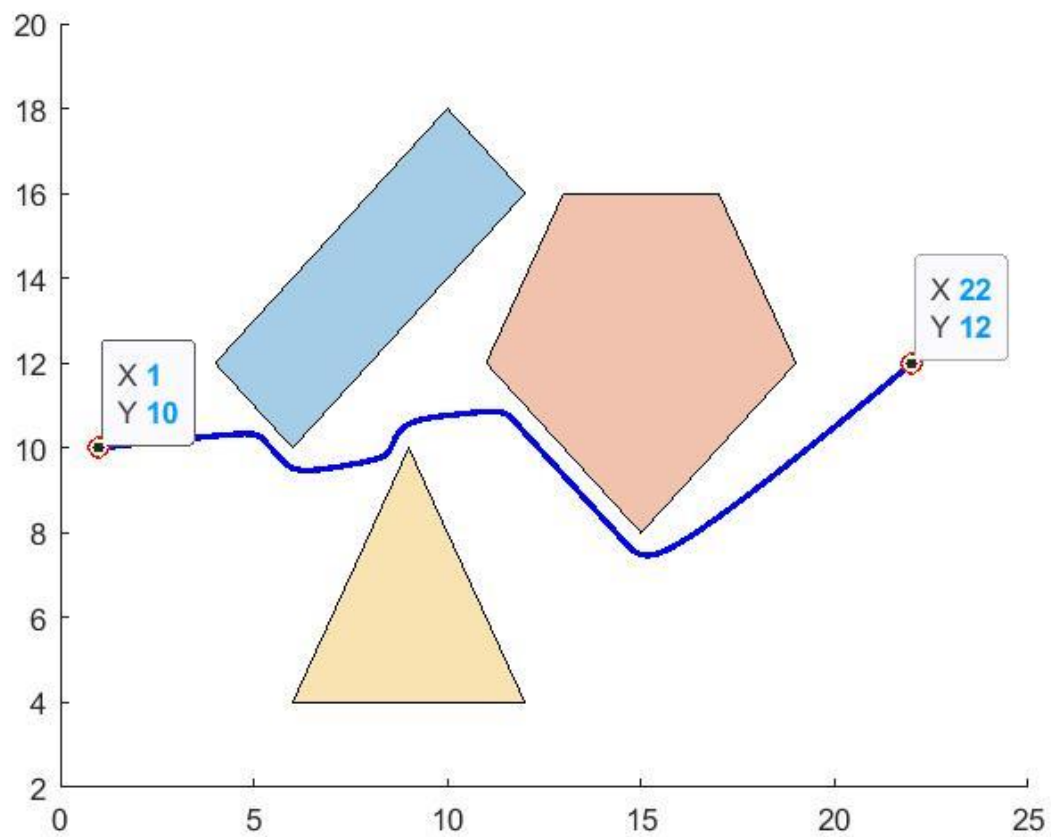
plot(obs3);

hold on
plot(P(1,:),P(2,:), 'blue', 'Linewidth',2)
hold on
plot([1,22],[10,12], 'ro', 'Linewidth',2)

```

خروجی کد:

مسیر مشخص شده به همراه موانع



تابع حالتی که در مینیمم محلی گیر میکند در نظر گرفته شده تا تابع ما در لوپ بی نهایت نیفتد. در همین حالتی وقتی حرکت رو به پیشرفت نیست و در واقع مسیر طی شده را دوباره بر می گردیم، با break از لوپ خارج می شویم و مشخصا مسیر به نقطه قرمز انتها نباید ختم شود. اما اینجا چنین اتفاقی نیفتاده و در مینیمم محلی گیر نکرده ایم.

(3)

3) Add a line between points (17.5,10) and (17.5,8). Try to generate a path again. Where does the path get stuck?

این سوال، ورودی ها مثل قبل است و فقط یک مانع جدید اضافه می شود که در B جای می دهیم و شکلش را با plot رسم می کنیم. بعد از run شدن، می بینیم که مسیر به مقصد نرسیده و جایی متوقف شده است که همان مینیمم محلی است. آخرین نقطه این مسیر تقریباً محل مینیمم محلی ماست و با دستور plot این نقطه را به شکل ستاره نشان دادیم.

کد تابع:

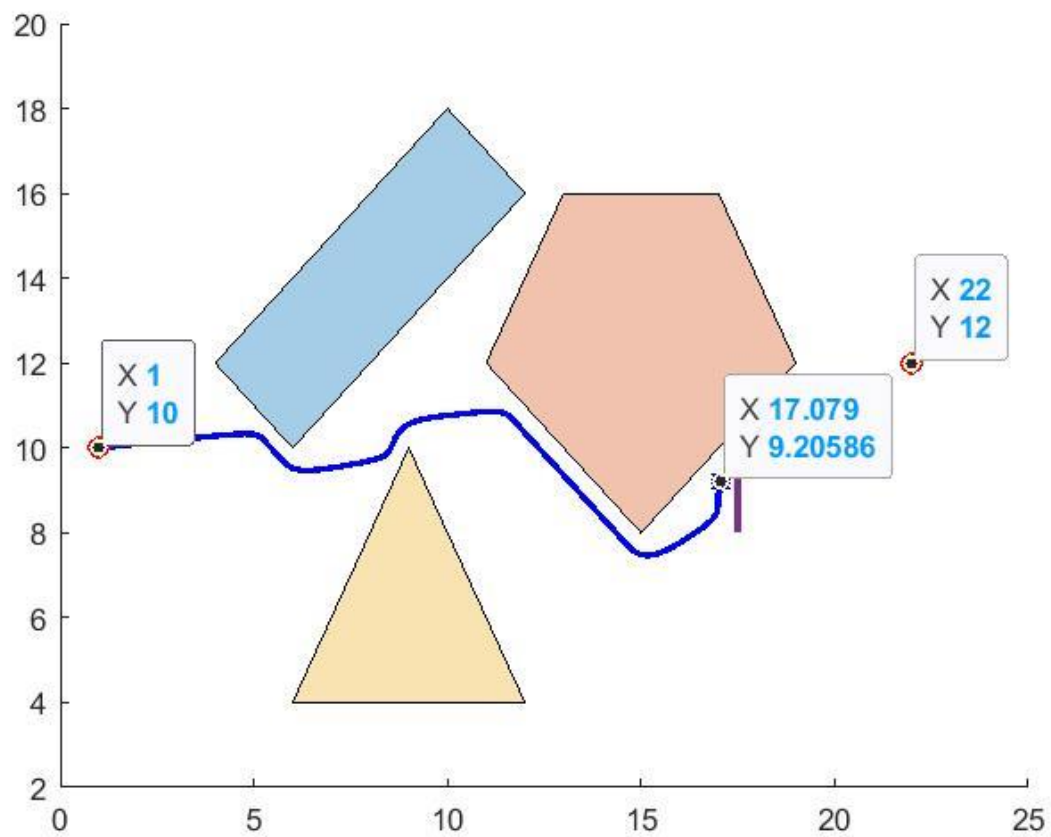
```
B={2,1,[4;12],[10;18],[12;16],[6;10]},{2,1,[13;16],[17;16],[19;12],[15;8],[11;12]},{2,1,[9;10],[12;4],[6;4]},{2,1,[17.5;10],[17.5;8]};
xs=[1;10];
xf=[22;12];
eta=1;
[R,P]=Path_generator(xs,xf,eta,B);
obs1=polyshape([4 10 12 6],[12 18 16 10]);
obs2=polyshape([13 17 19 15 11],[16 16 12 8 12]);
obs3=polyshape([9 12 6],[10 4 4]);

plot(obs1);
hold on
plot(obs2);
hold on
plot(obs3);
hold on
plot([17.5 17.5],[10 8],'Linewidth',2)

hold on
plot(P(1,:),P(2:),'blue','Linewidth',2)
hold on
plot([1,22],[10,12],'ro','Linewidth',2)

stuck=P(:,size(P,2))

hold on
plot(stuck(1),stuck(2),'b*','Linewidth',2)
```



مسیر در حدود نقطه (9.20586, 17.079) گیر می کند و باید با random walk این مشکل را حل کنیم.

4) (20% Bonus Marks) Implement a random walk algorithm and see if it can solve the local minima problem of Question3. The random walk algorithm starts when the potential field algorithm (PFA) is stuck in a local minima. It makes the robot take a random jump with a maximum length of a (let a be 6 in this example), at a random angle providing that the jump doesn't interfere with any obstacle. You can

propose a different method to determine the angle of jump to increase the chance of escape. Then it runs the PFA from the new position to see if it can reach the target destination. Submit a detailed report for this question.

برای اینکه مشکل مینیم محلی شود ، یک راه حل random walk است. برای اینکه هر وقت در مسیر به مینیم محلی خوردیم مسیر بتواند یک حرکت رندوم انجام دهد path_generator را کمی تغییر می دهیم و به جای اینکه در مینیم محلی stop کند، تابع random walk را فراخوانی می کند.

این تابع به صورت کلی یک قدم رندوم در زاویه رندومی که به مانع برخورد نکند بر میدارد. برای پیاده سازی از نقطه ای که گیر افتادیم، از یک زاویه رندوم شروع می کنیم و یک قدم کوچک در اون جهت برمیداریم و فاصله نقطه جدید رو با موانع چک می کنیم. اگر فاصله از حد مطلوب ما کمتر نبود ، قدم بعدی رو در همون جهت بر میداریم و دوباره فاصله را چک می کنیم. اگر هر جا شرط رعایت نشد، دوباره از نقطه اول با زاویه دیگر همین الگوریتم تکرار می شود تا جایی که یک زاویه پیدا شود که در تمام طول گامی که می خواهیم برداریم به موانع از حدی نزدیک تر نشود. مثلا وقتی $\text{jump}=6$ بود، من طول هر گام را 0.1 انتخاب کردم. زاویه شروع نیز زاویه نقطه گیر کرده تا مقصد است و هربار زاویه جدید یک درجه تغییر می کند.

در نهایت وقتی به طول مورد نظر حرکت کرد، Q به عنوان نقاط این random walk به تابع path_generator باز می گردد و این مسیر به مسیر طی شده اضافه می شود و ادامه مسیر به روش میدان پتانسیل به دست می آید.

چک کردن فاصله با موانع مطابق قبل است و کمترین فاصله نقطه با موانع حساب می شود، اگر این فاصله از حدی کمتر بود (من 0.4 در نظر گرفتم) سراغ زاویه بعد می رویم و در غیر این صورت گام جدید برمیداریم.

طول گام برداشته شده عددی رندوم بین 2 و 6 است.

Path_generator2:

```
function P=Path_generator2(xs,xf,eta,B)
    X=xs;
    n=size(B,2); %number of obstacle
    k=0.1;
    P=xs;
    gam=1;
    while norm(X-xf)>0.1
        gam=gam+1 ;
        Fatt=-eta*(X-xf);
        F=0;
        for i=1:n
            j=size(B{i},2)-2; % number of edge
            min=1000;
            efd=B{i}{1};
            for k=1:j
                if k<j
                    C=B{i}{k+2};
                    A=B{i}{k+3};

                else
                    C=B{i}{k+2};
                    A=B{i}{3};
                end
                R=-[0,-1;1,0]*(A-C);
                D=cat(2,(A-C),R);
                T=inv(D)*(X-C);
                alpha=T(1);
                if alpha<0
                    H=C;
                elseif alpha>1
                    H=A;
                else
                    H=alpha*(A-C)+C;
                end
                d=norm(X-H);
                if d<min
                    h=H;
                    min=d;
                end
            end
            if min<efd
                Fnep=B{i}{2}*(min)^(-3)*((1/min)-(1/efd))*(X-h);
            else
                Fnep=0;
            end
            F=F+Fnep;
        end
        Ftot=F+Fatt;
    end
```

```

    Fd=Ftot/norm(Ftot);
    next_step=0.1*Fd+X;
    P=cat(2,P,next_step);
    e=gam-2;
    if e>2
        if next_step==P(:,e)
            Q=Random_walk(xs,xf,eta,B,P);
            P=cat(2,P,Q);
            X=P(:,size(P,2));
            continue
        end
    end
    X=next_step;

end
end

```

Random_walk:

```

function Q=Random_walk(xs,xf,eta,B,P)
    y=size(P,2);
    X=P(:,y);
    n=size(B,2);
    x=xf(1)-X(1);
    z=xf(2)-X(2);
    theta=atan2d(z,x);
    Q=X;
    s=randi([2,6],1);
    for o = theta: -1 :theta-360

        X=P(:,y);
        Q=X;
        for q=1:(s*10)
            Next=X+0.1*[cosd(o);sind(o)];

            min=1000;
            for i=1:n
                j=size(B{i},2)-2; % number of edge
                for k=1:j
                    if k<j
                        C=B{i}{k+2};
                        A=B{i}{k+3};

                    else
                        C=B{i}{k+2};
                        A=B{i}{3};
                    end
                    R=-[0,-1;1,0]*(A-C);
                    D=cat(2,(A-C),R);
                    T=inv(D)*(Next-C);

```

```

        alpha=T(1);
        if alpha<0
            H=C;
        elseif alpha>1
            H=A;
        else
            H=alpha*(A-C)+C;
        end
        d=norm(Next-H);
        if d<min
            h=H;
            min=d;
        end
    end
end
if min<0.4
    break
else
    Q=cat(2,Q,Next);
    X=Next;
    q=q+1;
end
end
if q==(s*10+1)
    break
end
end
end

```

q4:

```

close all
clc
clear
B={2,1,[4;12],[10;18],[12;16],[6;10]},{2,1,[13;16],[17;16],[19;12],[15;8],[11;12]},{2,1,[9;10],[
12;4],[6;4]},{2,1,[17.5;10],[17.5;8]}};
xs=[1;10];
xf=[22;12];
eta=1;
P=Path_generator2(xs,xf,eta,B);
obs1=polyshape([4 10 12 6],[12 18 16 10]);
obs2=polyshape([13 17 19 15 11],[16 16 12 8 12]);
obs3=polyshape([9 12 6],[10 4 4]);

plot(obs1);
hold on
plot(obs2);
hold on
plot(obs3);
hold on
plot([17.5 17.5],[10 8],'Linewidth',2)

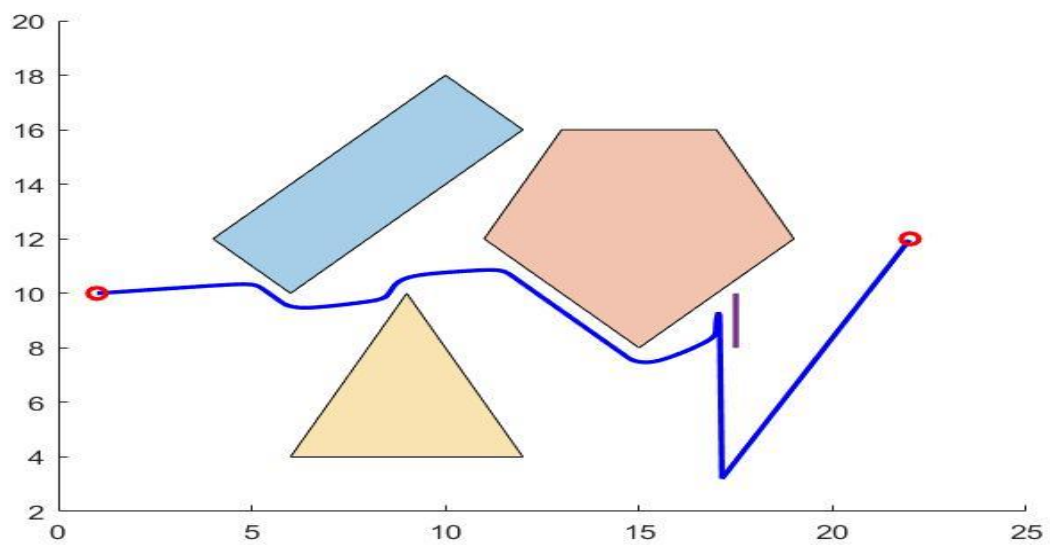
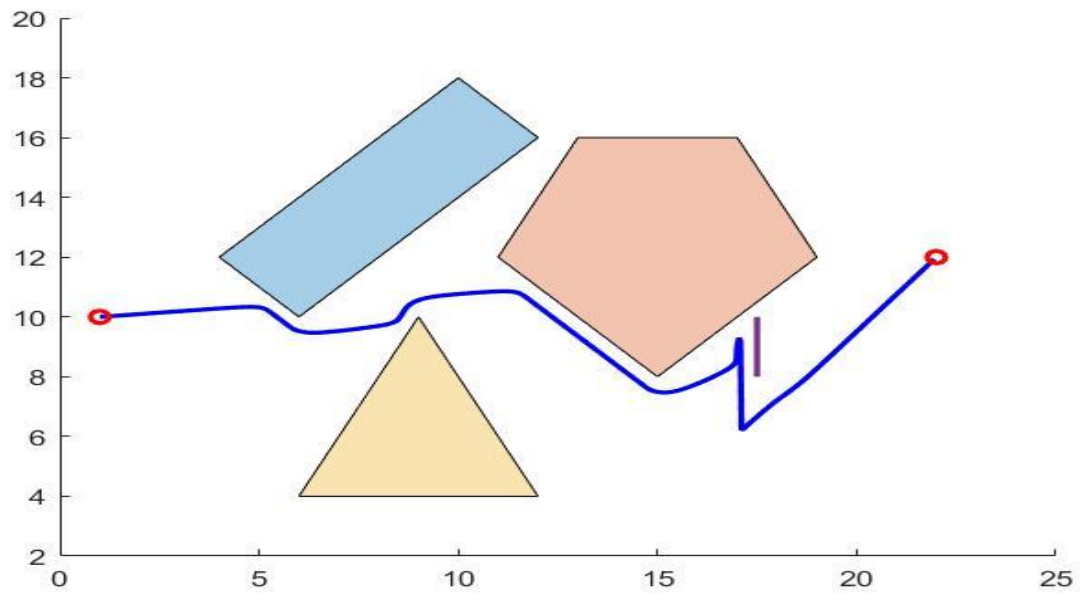
hold on

```

```

plot(P(1,:),P(2,:), 'blue', 'Linewidth',2)
hold on
plot([1,22],[10,12], 'ro', 'Linewidth',2)

```



چون گام ها رندوم است در run های متفاوت طول گام برداشته شده متفاوت است. همانطور که مشاهده می شود، بعد از گیر کردن یک گام به سمت پایین برمیدارد و بعد نقاط جدید به الگوریتم میدان پتانسیل می رود و ادامه مسیر تا مقصد را پیدا می کند.