دانشگاه صنعتی شریف

دانشکده مهندسی مکانیک

عنوان:

تمرینات سری ششم

Path generaton - Obstacle avoidance

نگارش

محمدسعید صافی زاده

استاد راهنما

دکتر بهزادی پور

آذر ماه ۱۴۰۳

فهرست مطالب

۱ صورت سوالات

در این تمرین از روش میدان پتانسیل برای طراحی مسیر ذره در صفحه و جلوگیری از برخورد آن با موانع استفاده می کنیم.

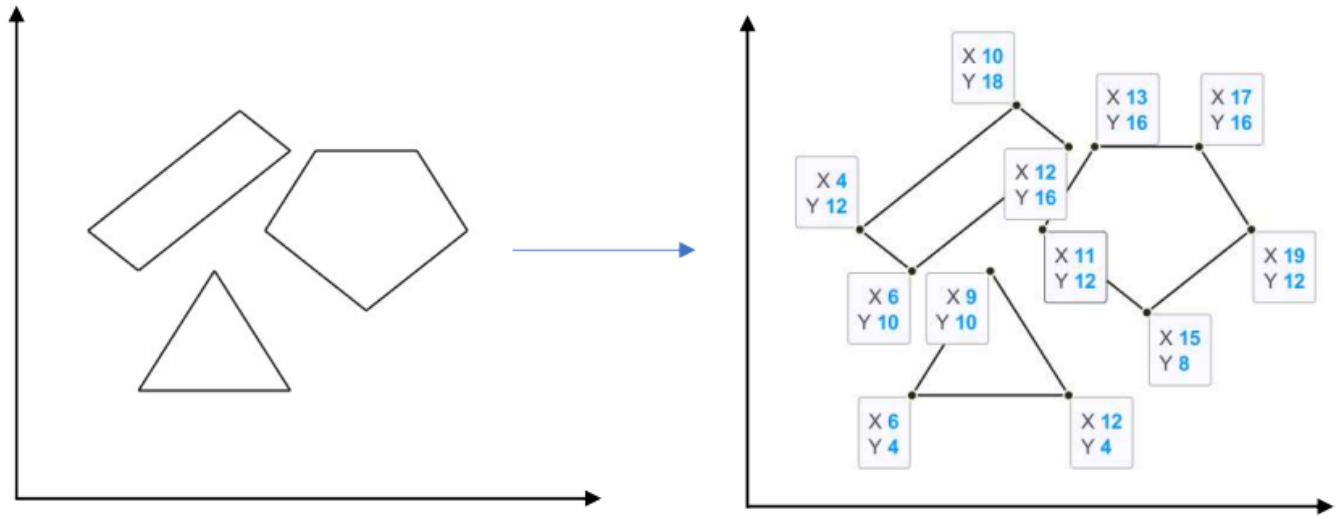
۱.۱ سوال اول

تابع متلبی بنویسید که نقاط مسیر را تولید کند. این تابع ورودی های موقعیت نقطه شروع (X_s)، نقطه پایانی (X_f) ، فاکتور برای جاذبه میدان (η) و ماتریس B که اطلاعات موانع، فاصله اثر و scaling factor (α) را می گیرد. این تابع باید زمانی که به فاصله 0.1 از نقطه نهایی رسیدیم، تولید نقاط را متوقف کند و ماتریس نقاط تولید شده یعنی P را که شامل X_s و X_f است بازگرداند.

$$function\ P = Pathgenerator(Xs, Xf, \eta, B)$$

۲.۱ سوال دوم

در این سوال از ما خواسته شده با اطلاعات $\eta = \alpha = 1$، $\epsilon = 0.1$ و شعاع اثر 2 برای همه موانع تابع نوشته شده در بخش اول را تست کنیم. سپس موانع را به همراه نقاط طی شده برای رسیدن به نقطه نهایی رسم کنیم. (Xs = (1,10), Xf = (22,12))



شکل ۱: تصویری از موانع به همراه مختصات آن ها

۳.۱ سوال سوم

یک خط بین دو نقطه (17.5, 10) و (17.5, 8) به عنوان مانع قرار دهید و دوباره تابع را اجرا کنید. مسیر تا کجا پیش می رود و کجا گیر می کند؟

٤.۱ سوال چهارم(امتیازی)

یک الگوریتم random walk به نحوی که مشکل می نی مم محلی سوال سوم را رفع کند توسعه دهید. این الگوریتم زمانی اجرا می شود که الگوریتم میدان پتانسیل در یک می نی مم محلی گیر افتاده باشد. این الگوریتم یک پرش رندوم با بیشینه طول a برای مثال ۶ در یک زاویه رندوم را به گونه ای که به موانع برخورد نکند تولید می کند. برای این سوال می توانید از یک الگوریتم دیگر برای تعیین زاویه پرش برای افزایش شانس فرار استفاده کنید و زمانی که از این پرش را انجام دادید دوباره از تابع پتانسیل برای ادامه مسیر برای رسیدن به نقطه نهایی استفاده کنید.

۲ پاسخ سوال اول

۱.۲ تشریح تئوری مسئله و روند کلی حل

برای طراحی مسیر برای یک نقطه متحرک در صفحه با حضور موانع به شکل چندضلعی محدب از روش میدان پتانسیل به صورت زیر عمل می کنیم. بر اساس این روش یک میدان پتانسیل از نوع جاذبه برای نقطه ی نهایی Xf در نظر می گیریم و یک سری میدان های پتانسیل از نوع دافعه برای موانع موجود و بر این اساس در جهت کاهش کلی میدان حرکت می کنیم.
ایتدا به تعریف میدان ها می پردازیم. میدان پتانسیل جاذبه برای نقطه نهایی Xf به صورت زیر در می آید:

$$U_{att}(\vec{X}) = \frac{1}{2}|\vec{X} - \vec{X_f}|^2\eta$$

میدان پتانسیل دافعه در اطراف موانع نیز به شکل زیر در خواهد آمد:

$$U_{rep_i}(\vec{X}) = 0 \;\; if \;\; \rho_i(\vec{x}) >= \rho_{0i}$$

$$U_{rep_i}(\vec{X}) = \frac{1}{2}\alpha_i(\frac{1}{\rho_i(\vec{X})} - \frac{1}{\rho_{0i}})^2 \;\; if \;\; \rho_i(\vec{x}) < \rho_{0i}$$

که در اینجا $\rho_i(\vec{X})$ فاصله $\vec{X}$ از مانع i ام و ρ_{0i} شعاع اثر مانع i ام می باشد. چون باید در جهت منفی گرادیان میدان پتانسیل حرکت کنیم، گرادیان هر دو مورد را به دست می آوریم. گرادیان میدان جاذبه به شکل زیر می باشد:

$$-\nabla U_att(\vec{X}) = \eta(\vec{X_f} - \vec{X})$$

و گرادیان میدان دافعه مولنع به صورت زیر خواهد شد:

$$-\nabla U_{rep_i}(\vec{X}) = 0 \;\; if \;\; \rho_i(\vec{x}) >= \rho_{0i}$$

$$-\nabla U_{rep_i}(\vec{X}) = \alpha_i\frac{1}{\rho_i^3(\vec{X})}(\frac{1}{\rho_i(\vec{X})} - \frac{1}{\rho_{0i}})(\vec{X} - B_i(\vec{X}))$$

که در این جا $B_i(\vec{X})$ نزدیک نقطه مانع i ام به نقطه $\vec{X}$ است. در ادامه به نحوه یافتن $B_i(\vec{X})$ خواهم پرداخت.
در مرحله بعدی بردار جهت حرکت ترجیهی را با با استفاده از روابط زیر بدست می آوریم و به اندازه ϵ حرکت می کنیم.

$$\vec{F_d} = \frac{\vec{F}(\vec{X})}{|\vec{F}(\vec{X})|}$$

$$\vec{X}_{next} = \vec{X} + \epsilon\vec{F_d}$$

۲.۲ توضیح نحوه تئوری یافتن مقدار $B_i(\vec{X})$

برای یافتن $B_i(\vec{X})$ رئوس چندضلعی محدب را دریافت می کنیم و برای هر ضلع نزدیک ترین فاصله به آن ضلع را پیدا می کنیم. نهایتا در بین این مجموعه جواب کمترین فاصله پاسخ ما خواهد بود. برای مثال برای یک ضلع با دو راس P و Q ابتدا معادله خط ضلع و خط عمود گذرنده از آن ضلع و نقطه کنونی را پیدا می کنیم و با مساوی قرار دادن این دو شیب این دو خط را می یابیم:
عبارت پارامتریک مولد خط اول:

$$(\vec{Q} - \vec{P})\alpha + \vec{P}$$

عبارت مولد خط دوم:

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} (\vec{Q} - \vec{P})\beta + \vec{X}$$

بنابراین با مساوی قرار دادن این دو عبارت و یک سمت بردن آلفا و بتا داریم:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = [\vec{Q} - \vec{P} \quad - \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} (\vec{Q} - \vec{P})]^{-1}(\vec{X} - \vec{P})$$

در نهایت برای تعیین $\vec{B_i}$ از تست زیر استفاده می کنیم:

if $0 <= \alpha <= 1$ $then$ $\vec{B_i} = (\vec{Q} - \vec{P})\alpha + \vec{P}$
else if $\alpha < 0$ $then$ $\vec{B_i} = \vec{P}$
else $\vec{B_i} = \vec{Q}$

۳.۲ کد تابع مدنظر به همراه توضیحات هر بخش

در ابتدا قسمت به قسمت به توضیح عملکرد تابع می پردازم و نهایتاً در انتها کد کلی را قرار خواهم داد.

۱.۳.۲ نحوه ورودی دادن به تابع

تابع ما چهار ورودی کلی، شامل دو نقطه شروع و پایان که باید به صورت یک بردار داده شوند و مقدار η و ماتریس B که حاوی اطلاعات موانع می باشد دارد. در اینجا ذکر این نکته الزامی است که ماتریس B باید به شکل خاصی ورودی بگیرد. به این صورت که در سطر اول، اطلاعات مربوط به مانع اول(بیشترین ضلع) را وارد می کنیم که در ستون اول تعداد اضلاع آن مانع، سپس برای هر راس مقدار ایکس و وای را می نویسیم در انتها نیز به ترتیب مقدار ρ و Scaling factor یعنی α را وارد می کنیم. در سطر های بعدی اطلاعات مربوط به بقیه موانع را وارد می کنیم که طبیعتاً برای اینکه اندازه ماتریس رعایت شود به جای مختصات رئوسی که نداریم مقدار صفر را وارد می کنیم. برای مثال ماتریس B سوال دوم به صورت زیر در می آید:

```
B = [5,13,16,17,16,19,12,15, 8,11,12,2,1;
4, 4,12,10,18,12,16, 6, 10, 0,0,2,1;
3, 9,10,12, 4, 6, 4, 0, 0, 0, 0,2,1;
];
```

که برای نمونه سطر دوم بیانگر یک چهار ضلعی(درایه اول) با ایکس و وای های مشخص شده و نهایتا $\rho = 2$ و $\alpha = 1$ می باشد.

۲.۳.۲ توضیحات بخش به بخش تابع نوشته شده

ابتدا تعداد موانع را در متغیر N می ریزیم و نقطه اولیه را در ماتریس نقاط P قرار می دهیم. در یک While شرط پایان محاسبات را که رسیدن به نزدیک نقطه نهایی است اعمال می کنیم. سپس نیروی میدان جاذبه نقطه نهایی را که به نقطه کنونی وارد می شود پیدا می کنیم(F_{att}) و را در یک متغیر دیگر به نام F می ریزیم.

```
N = height(B_main);
path = [1;10];
while norm(Xf-X) > 0.1
F_att = eta * (Xf - X);
F = F_att;
```

اکنون با استفاده از یک حلقه بر روی موانع پیمایش می کنیم تا برای هر مانع مقدار B_i را پیدا کنیم. در ابتدا این مقدار را یک عدد بزرگ می گذاریم چون بعداً جایگزین می شود مهم نیست. سپس یک متغیر که آن نقطه از مانع که به ازای آن این مقدار B_i کمینه است را در خود نگه می دارد به نام Bi Point تعریف می کنیم. این مقادیر مرتبط با هر مانع را که در هر سطر هستند در هر مرحله در متغیر B می ریزیم و مقادیر α و ρ را در دو متغیر a و rep influence distance قرار می دهیم. و به انتهای B دو مختصات اول را اضافه می کنیم تا تمامی اضلاع تشکیل شوند.

```
 for j=1:N
Bi = 1000;
Bi_point = [];
B = [];
a = B_main(j,end); %scaling factor
rep_influence_distance = B_main(j,end-1); % distance of influence
B = B_main(j,2:2*B_main(j,1)+1);
B = [B, B(1), B(2)];
```

در حلقه ی بعدی بر روی اضلاع یک مانع پیمایش می کنیم و برای هر ضلع کمترین فاصله تا مکان فعلی را می یابیم و نهایتا در هر مرحله کمترین مقدار را به همراه مختصات آن ها ذخیره می کنیم. در این جا از همان روش توضیح داده شده برای یافتن Bi در بالا استفاده شده است. به این صورت که معادله خط هر ضلع و عمود بر این ضلع که گذرنده از نقطه فعلی است را پیدا می کنیم و بر اساس آن مقدار Bi مرتبط با آن ضلع را می یابیم:

```
for i=1:2:length(B)-2
Q = [B(i+2); B(i+3)];
P = [B(i); B(i+1)];
alphabeta = inv([Q-P -[0 -1 ; 1 0]*(Q-P)])*(X-P);
alpha = alphabeta(1);
beta = alphabeta(2);
if (0 < alpha) && (alpha < 1)
M = ((Q-P)*alpha + P);
Bi_new = norm(M-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = M;
end
elseif alpha <= 0
Bi_new = norm(P-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = P;
end
else
Bi_new = norm(Q-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = Q;
end
end
end
```

در حلقه بیرونی که برای هر مانع می باشد با استفاده از کد زیر مقدار نیرویی که از طرف این مانع به نقطه فعلی وارد می شود را محاسبه می کنیم و به مجموع نیرو ها اضافه می کنیم:

```
if norm(Bi_point - X) <= rep_influence_distance
F_rep = (a / (norm(Bi_point - X)^3)) * (((1/norm(Bi_point - X))
 - (1/rep_influence_distance))) *  (X - Bi_point);
```

```
F = F + F_rep;
end
```

نهایتا قبل از دوباره اجرا شدن while برای نقطه بعدی، باید نقطه جدید را بیابیم که با استفاده از کد زیر این عمل امکان پذیر خواهد شد:

```
f_d = F / norm(F);
epsilon = 0.1; % jump steps
X = X + epsilon * f_d;
path(1:2,end+1) = vpa(X);
```

و نهایتاً بیرون حلقه While مقدار نقطه نهایی را به متغیر P اضافه می کنیم:

```
P = [path,Xf];
```

۳.۳.۲ کد نهایی و کامل تابع

کد نهایی تابع به صورت زیر می باشد که در پوشه codes نیز ضمیمه شده است:

```
function P = Path_generator(X, Xf, eta, B_main)

N = height(B_main);
path = [1;10];
while norm(Xf-X) > 0.1
F_att = eta * (Xf - X);
F = F_att;
for j=1:N
Bi = 1000;
Bi_point = [];
B = [];
a = B_main(j,end); %scaling factor
rep_influence_distance = B_main(j,end-1); % distance of influence
B = B_main(j,2:2*B_main(j,1)+1);
B = [B, B(1), B(2)];

for i=1:2:length(B)-2
Q = [B(i+2); B(i+3)];
P = [B(i); B(i+1)];
alphabeta = inv([Q-P -[0 -1 ; 1 0]*(Q-P)])*(X-P);
alpha = alphabeta(1);
beta = alphabeta(2);
if (0 < alpha) && (alpha < 1)
M = ((Q-P)*alpha + P);
Bi_new = norm(M-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = M;
end
elseif alpha <= 0
```

```
Bi_new = norm(P-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = P;
end
else
Bi_new = norm(Q-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = Q;
end
end


end

if norm(Bi_point - X) <= rep_influence_distance
F_rep = (a / (norm(Bi_point - X)^3)) * (((1/norm(Bi_point - X))
 - (1/rep_influence_distance))) *  (X - Bi_point);
F = F + F_rep;
end
end

f_d = F / norm(F);
epsilon = 0.1; % jump steps
X = X + epsilon * f_d;
path(1:2,end+1) = vpa(X);

end
P = [path,Xf];
```

۳ پاسخ سوال دوم

برای حل این سوال ابتدا مقدار ϵ را در تابع تصحیح و به مقدار ۱.۰ قرار می دهیم. سپس ماتریس B را همانطور که صورت سوال خواسته به صورت زیر تعریف می کنیم:

```
B = [5,13,16,17,16,19,12,15, 8,11,12,2,1;
4, 4,12,10,18,12,16, 6, 10, 0,0,2,1;
3, 9,10,12, 4, 6, 4, 0, 0, 0, 0,2,1;
];
```

سپس تابع قسمت اول را با کد زیر اجرا می کنیم و خروجی آن را در متغیر P می ریزیم:

```
P = Path_generator([1;10],[22;12],1, B);
```

و نهایتاً با دستورهای Polyshape و plot این مقادیر را رسم می کنیم:

```
pgon = polyshape([13 17 19 15 11],[16 16 12 8 12]);
pgon2 = polyshape([4,10,12,6],[12, 18,16,10]);
pgon3 = polyshape([9,12,6],[10,4,4]);
plot(pgon);
hold on
plot(pgon2);
plot(pgon3);
plot(P(1,:),P(2,:));
hold off
```

که کد کامل این بخش در فایل Q2 در پوشه Codes آورده شده است. و کامل آن به صورت زیر می باشد:

```
B = [5,13,16,17,16,19,12,15, 8,11,12,2,1;
4, 4,12,10,18,12,16, 6, 10, 0,0,2,1;
3, 9,10,12, 4, 6, 4, 0, 0, 0, 0,2,1;
];

P = Path_generator([1;10],[22;12],1, B);

pgon = polyshape([13 17 19 15 11],[16 16 12 8 12]);
pgon2 = polyshape([4,10,12,6],[12, 18,16,10]);
pgon3 = polyshape([9,12,6],[10,4,4]);
plot(pgon);
hold on
plot(pgon2);
plot(pgon3);
plot(P(1,:),P(2,:));
hold off
```

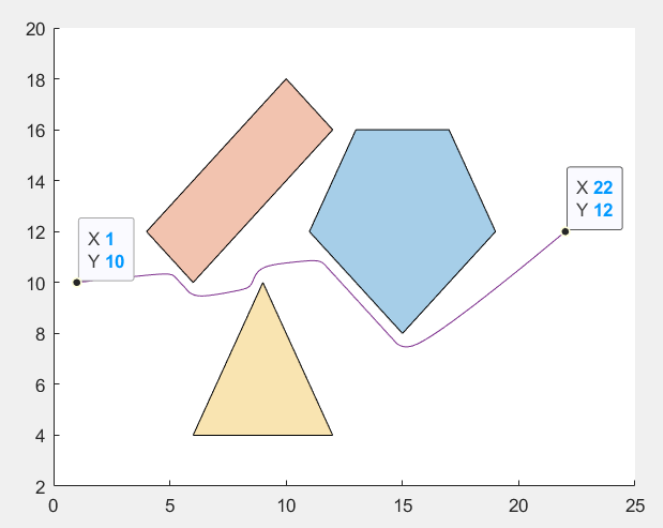
هم چنین در شکل ۲ می توانید خروجی این کد را مشاهده کنید که نقطه مورد نظر به چه صورتی از میان موانع عبور کرده است. هم چنین نقاطی که طی می شوند در متغیر P ذخیره شده اند که در مد متلب می توان آن ها را مشاهده کرد.

٤ پاسخ سوال سوم

برای این سوال ابتدا مانع جدید را به صورت زیر تعریف می کنیم:

```
B = [5,13,16,17,16,19,12,15, 8,11,12,2,1;
4, 4,12,10,18,12,16, 6, 10, 0,0,2,1;
3, 9,10,12, 4, 6, 4, 0, 0, 0, 0,2,1;
2,17.5,10,17.5,8,0,0,0,0,0,0,0,2,1;
];
```

سپس کد مربوط به پاسخ سوال دوم را دوباره اجرا می کنیم. اجرای کد تمام نمی شود بنابراین شرط پایانی ما یعنی رسیدن به نزدیک نقطه پایانی ارضا نشده است، بنابراین مقادیر نقاط طی شده را بررسی می کنیم. برای این کار یک شرط جدید به تابع اضافه می کنیم که اگر در اعداد این مرحله و دو مرحله قبل نزدیک هم شده بودند برنامه را متوقف کند که کد آن به صورت زیر می شود:



شکل ۲: مسیر طی شده برای رسیدن به نقطه نهایی

```
W = width(path);
if W > 3
if norm(X-path(:,end-2)) < 0.01
flag = true;
break
end
end
```

در این جا برای ایکس های بدست آمده بزرگ تر از سه مرحله شرط را چک می کنیم و یک فلگ تعریف کرده ام تا خروجی Pرا متناسب با شرایط تنظیم کند. کد این مورد در پایین قابل ملاحظه است:

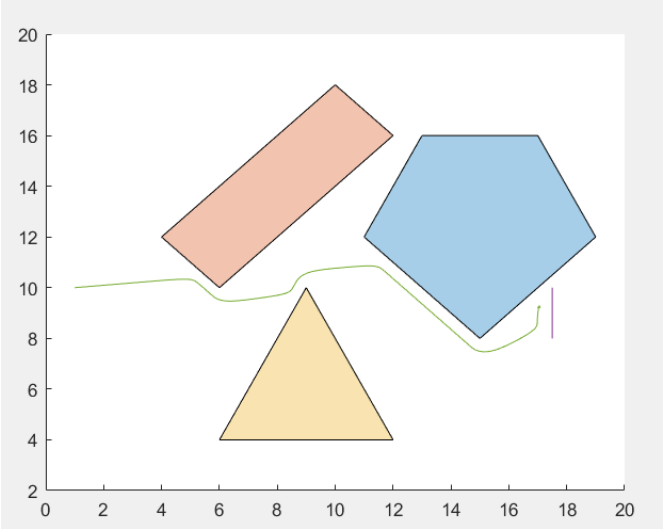
```
if flag
P = path;
else
P = [path,Xf];
end
```

و کد کامل برای این سوال به صورت زیر می باشد:

```
B = [5,13,16,17,16,19,12,15, 8,11,12,2,1;
4, 4,12,10,18,12,16, 6, 10, 0,0,2,1;
3, 9,10,12, 4, 6, 4, 0, 0, 0, 0,2,1;
2,17.5,10,17.5,8,0,0,0,0,0,0,0,2,1;
];

P = Path_generator([1;10],[22;12],1, B);

pgon = polyshape([13 17 19 15 11],[16 16 12 8 12]);
```

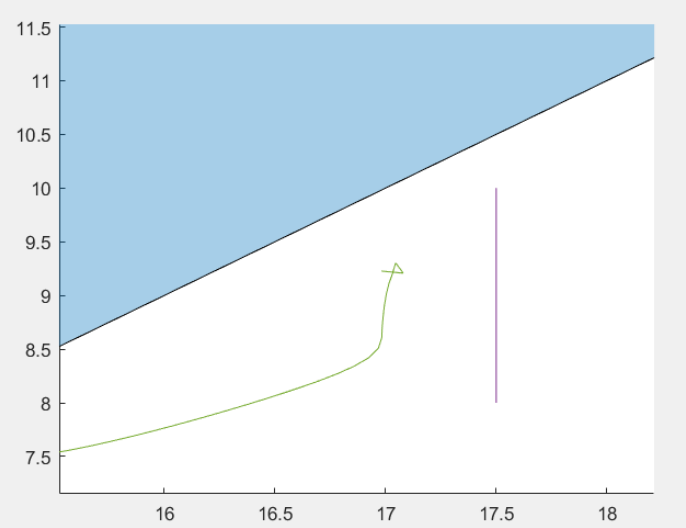
```
pgon2 = polyshape([4,10,12,6],[12, 18,16,10]);
pgon3 = polyshape([9,12,6],[10,4,4]);
plot(pgon);
hold on
plot(pgon2);
plot(pgon3);
plot([17.5,17.5],[8,10]);
plot(P(1,:),P(2,:));
hold off
```

با اجرای این کد به مسیر شکل ۳ می رسیم. با بدست آوردن دو مختصات آخر، به مقادیر (17.079, 9.2059) و (16.9810, 9.2255) می



شکل ۳: مسیر طی شده برای سوال سوم

رسیم و این بدین معناست که بین این دو نقطه یک مینیمم محلی وجود دارد.



شکل ۴: بزرگنمایی شده مسیر طی شده برای سوال سوم

۵ پاسخ سوال چهارم

برای نوشتن تابعی که پرش رندوم را انجام دهد کافی است مقدار X جدید را پیدا کنیم و سپس بررسی کنیم که آیا معادله خط محل پیدا شده با محل قبلی با معادله خط اضلاع برخورد دارد یا نه. این بررسی را به صورت زیر انجام می دهیم:

$$\vec{PQ} = Q - P \; ; \; where \; PQ \; represents \; a \; side$$

$$\alpha\vec{PQ} + \vec{P} \; ; \; is \; a \; side \; equation$$

$$\vec{X'X} = X' - X \; ; \; where \; X'X \; represents \; the \; vector \; of \; former \; and \; new \; point$$

$$\beta\vec{X'X} + X \; ; \; is \; equation \; of \; the \; second \; line$$

با مساوی قرار دادن دو معدله خط اگر دو شرط زیر برقرار باشند نشان می دهد که دو خط تلاقی دارند بنابراین، جهتی که برای پرش به صورت رندوم انتخاب ککرده ایم مناسب نبوده و باید یک جهت دیگر را انتخاب کنیم:

$$if \;\; 0 <= \alpha <= 1 \;\; and \;\; 0 <= \beta <= 1 \; then \; we \; have \; an \; intersection$$

که کد این توضیحات به صورت خط به خط در ادامه آورده شده است: (به همان تابع سوال اول یک بخش اضافه شده است)

```
if W > 3
if norm(X-path(:,end-2)) < 0.001
flag = true;
while flag
direction = randi([0;360]);
disp(direction);
Xprime = [X(1)+6*cosd(direction);X(2)+6*sind(direction)];
rec_A = [];
rec_B = [];
for j=1:N
B = B_main(j,2:2*B_main(j,1)+1);
B = [B, B(1), B(2)];
```

```
for i=1:2:length(B)-2
Q = [B(i+2); B(i+3)];
P = [B(i); B(i+1)];
PQ = Q - P;
XXprime = Xprime - X;

AB = inv([PQ -XXprime])*(X-P);
rec_A = [rec_A AB(1)];
rec_B = [rec_B AB(2)];

end
end
```

در کد بالا ابتدا بررسی می کنیم که نقطه جدید به دست آمده با نقاط قبلی چقدر اختلاف دارد که در صورتی این اختلاف کم باشد به این معنی است که در می نی مم محلی قرار گرفته ایم. سپس به استفاده از تابع رندوم یک زاویه برای پرش بر می گزینیم ومقدار ایکس و وای جدید را می یابیم. نهایتاً با دوحلقه، بر روی موانع و بر روی اضلاع آن ها پیمایش می کنیم، معادله خط آن ها را برابر با معادله خط ایکس جدید و ایکس قدیم قرار می دهیم و ضرایب آلفا و بتا آن ها را در دو متغیر recA و recB می ریزیم.

```
thereisanintersection = false;
for l=1:width(rec_A)
if (0 <= rec_A(l) && rec_A(l) <= 1 ) && (0 <= rec_B(l) && rec_B(l) <= 1)
thereisanintersection = true;
end
end

if ~thereisanintersection
flag = false;
X = Xprime;
end
```

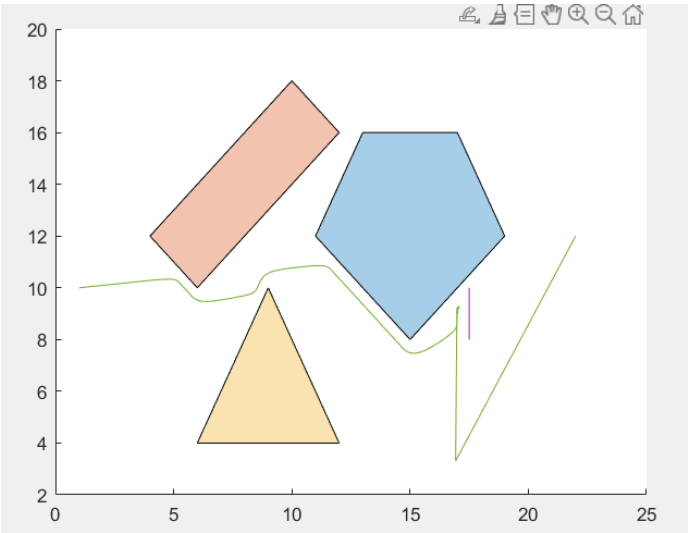
در ادامه همانطور که در کد بالا مشاهده می کنید، بر روی این متغیر های پیمایش می کنیم تا ببینیم آیا برخوردی وجود دارد و این برخورد را با شرط گفته شده در بالا پیدا می کنیم. بنابراین اگر برخوردی نباشد این ایکس انتخابی مورد قبول است و flag را false می کنیم تا دوباره اجرا نشود به دوباره به حلقه اصلی بر می گردیم. کد کامل این تابع در اینجا آورده شده است:

```
function P = Path_generator_with_jump(X, Xf, eta, B_main)


N = height(B_main);
path = [1;10];
while norm(Xf-X) > 0.1
F_att = eta * (Xf - X);
F = F_att;
for j=1:N
Bi = 1000;
Bi_point = [];
B = [];
```

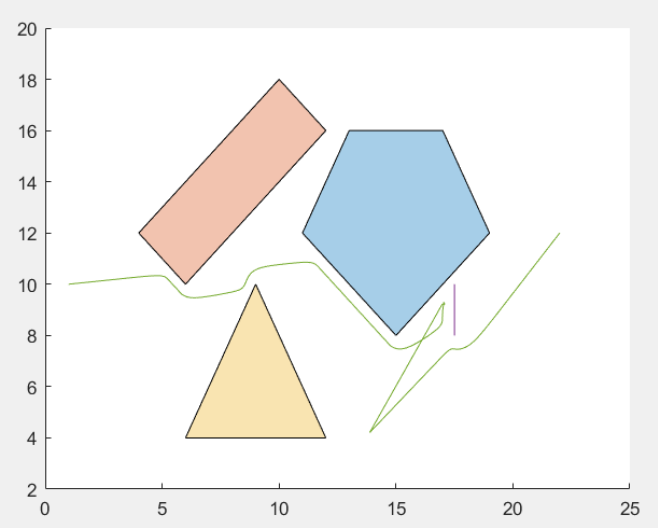
```
a = B_main(j,end); %scaling factor
rep_influence_distance = B_main(j,end-1); % distance of influence
B = B_main(j,2:2*B_main(j,1)+1);
B = [B, B(1), B(2)];

for i=1:2:length(B)-2
Q = [B(i+2); B(i+3)];
P = [B(i); B(i+1)];
alphabeta = inv([Q-P -[0 -1 ; 1 0]*(Q-P)])*(X-P);
alpha = alphabeta(1);
beta = alphabeta(2);
if (0 < alpha) && (alpha < 1)
M = ((Q-P)*alpha + P);
Bi_new = norm(M-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = M;
end
elseif alpha <= 0
Bi_new = norm(P-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = P;
end
else
Bi_new = norm(Q-X);
if Bi_new < Bi
Bi = Bi_new;
Bi_point = Q;
end
end


end

if norm(Bi_point - X) <= rep_influence_distance
F_rep = (a / (norm(Bi_point - X)^3)) * (((1/norm(Bi_point - X))
 - (1/rep_influence_distance))) *  (X - Bi_point);
F = F + F_rep;
end
end

f_d = F / norm(F);
epsilon = 0.1; % jump steps

X = X + epsilon * f_d;
path(1:2,end+1) = vpa(X);
```

```
W = width(path);

if W > 3
if norm(X-path(:,end-2)) < 0.001
flag = true;
while flag
direction = randi([0;360]);
disp(direction);
Xprime = [X(1)+6*cosd(direction);X(2)+6*sind(direction)];
rec_A = [];
rec_B = [];
for j=1:N
B = B_main(j,2:2*B_main(j,1)+1);
B = [B, B(1), B(2)];
for i=1:2:length(B)-2
Q = [B(i+2); B(i+3)];
P = [B(i); B(i+1)];
PQ = Q - P;
XXprime = Xprime - X;

AB = inv([PQ -XXprime])*(X-P);
rec_A = [rec_A AB(1)];
rec_B = [rec_B AB(2)];

end
end

thereisanintersection = false;
for l=1:width(rec_A)
if (0 <= rec_A(l) && rec_A(l) <= 1 ) && (0 <= rec_B(l) && rec_B(l) <= 1)
thereisanintersection = true;
end
end

if ~thereisanintersection
flag = false;
X = Xprime;
end
end
end
end
```

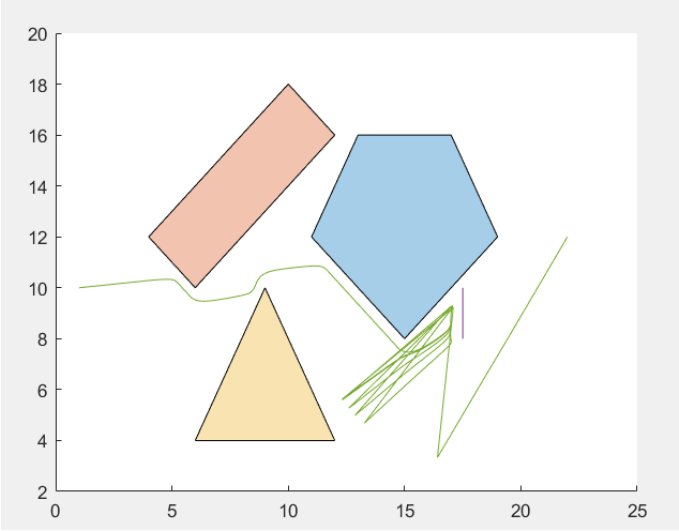
همینطور چند نمونه از این پرش ها را می توانید در شکل های ۵، ۶ و ۷ مشاهده کنید.



شکل ۵: نمونه اول از پرش برای حرکت ربات



شکل ۶: نمونه دوم از پرش برای حرکت ربات



شکل ۷: نمونه سوم از پرش برای حرکت ربات