**两点之间的路径优化**

不规则地形条件下的障碍物的避免方法

由于不规则地形下AUV与障碍物不存在确定的空间位置关系，因此用传统的空间几何数学方法是不能判断AUV与障碍物的碰撞关系的。

一种简单而有效的方法为在AUV的任意两个路径点之间做切面，然后取固定数目的等分点来比较等分点的高度与该坐标下的障碍物的高度就可以该直线路段是否与障碍物发生碰撞。

判断路径点i和路径点j与障碍物的碰撞关系

passableJudge(wp[i] wp[j]):

make n=100 equal points pn[100] between wp[i] and wp[j]

for k to 99 do :

compute pn[k]’s ObsFitness[k] with obstacle

if ObsFitness[k] > minObsFitness

minObsFitness <= ObsFitness[k]

end for

if minObsFitness> Limit value

return minObsFitness

else return 0;

End program

QPSO ——两个目标点之间的最短距离与路线计算

**QPSO算法的伪代码说明：**

Global variables：size=20,P=5,dim=3\*P,wp[P+2][3]，

x[size][dim], fitness\_value[size]

pbest[size][dim],pbest\_fitValue[size]

mean\_pbest[dim], gbest[dim]. gbest\_fitValue;

min\_gbest[dim]

Initialize()：

for i=0 to size do

initialize x[i] randly in the search scope

compute fitness\_value[i] by fitnessFuncation()

pbest[i]=x[i]

pbest\_fitValue[i]= fitness\_value[i]

select the min fitness\_value[gbest]

end for

Gbest=x[gbest]

gbest\_fitValue= fitness\_value[gbest]

update():

for i to size do

for j to dim do

update x[i][j] by formula

b=rand()>0.5?-b:b



end for

compute fitness\_value[i] by fitnessFuncation()

if pbest\_fitValue[i]> x[i]

pbest[i] = x[i]

pbest\_fitValue[i]= fitness\_value[i]

end if

select the min fitness\_value[gbest]

end for

Gbest=x[gbest]

gbest\_fitValue= fitness\_value[gbest]

QPSO:

Input:start point M[st],end point M[end]

for k=0 to runTime do

initialize ()

b=0.9

for i=0 to maxgen do

update()

b-=(0.9-0.4)\*i/maxgen

end for

if minGbest<gbest

minGbest=gbest

end if

end for

Output distance[st][end],routeLine[st][end]

End program transCostCompetionRule

fitnessFuncation(x[]):

split x[] to wp[P+2][3] by X,Y,Z

j=cn=0

while j<p+1 do

i= cn=j

while i<=P+1 do

if node i could connect directly with j

i++

else break;

end while

routeLength+=lengthCmp(wp[i],wp[j])

Obsfitness+= passableJudge(wp[cn], wp[j]) //compute obstacle dangerth

j=cn

end while

return routeLength+Obsfitness

TSP

Initial():

for k=0 to psize-1 do

initial particle route pos[k].seq[dim] and velocity[k]

pos[k].fitness=seq[k]’s length

find the pos[k]’s learn

PbestPos[k]= pos[k]

Select the best pos[j] (j=0,1,……k-1) to GbestPos

end for

update():

generate pr (pr(0,1))

for k=0 to psize-1 do

newSeq[k]=

for j=0 to dim-1 do

={e|e}

While newSeq[k] is not full

If

Select edge e from and add to newSeq[k]

Update

Else if and could find a edge from pos[k].seq[dim] to conect

Select a edge from pos[k].seq[dim] and add to newSeq[k]

Update

Else if and couldn’t find a edge from pos[k].seq[dim] to conect

Select a edge from other particle’s velocity and add to newSeq[k]

End if

End while

End for

Update pos[k].fitness by Compute the newSeq[k]’s length

Update PbestPos[k]

copy best newSeq[k] and pos[k].fitness to GBest

End for

For k=0 to maxGen-1 do

Input the distance[nodeNum][ nodeNum]

Initialize()

Update()

Output the best route seq GbestSeq and RouteLength

End for

**不规则地形条件下的多目标点检测**

**TSP路径优化方法**

在不规则地形条件下寻找能访问n个目标检测点之间的最短路线时，有两种做法

1 首先用QPSO优化出任意两点之间的最短距离和最短路线，导入地形条件下的真实距离矩阵，然后再用TSP优化算法优化得到最佳访问序列。

2 利用任意两点之间的直线距离矩阵通过TSP计算出最优路径序列，然后调用QPSO计算出路径序列在地形环境下的真实距离。

**两种QPSO调用次数的比较**

**假设待优化的目标点个数为n**

若采用第一种方法时，寻找两点之间的最优路线时QPSO算法运行20次取最小值，则QPSO算法的调用次数为n（n-1）/2

程序框架如下

For k=0 to 20 do

For m=0 to n-1 do

For j=m+1 to n-1 do

If distance[m][j] couldn’t connect directly

For k=0 to 19 do

run QPSO() to find the best real distance[n][n] and routeLine

end for

end for

end for

end for

For k=0 to 10 do:

Run TSP()

（compute the routeSeq’length through real distance[n][n]）

find the best seq GbestSeq[n+1] and totalRouteLength

End for

当采用第二种方法时，TSP利用直线距离矩阵计算路径序列长度，在得到最优路径序列后，再对序列的路径点之间的长度用QPSO优化；重复运行TSP多次来不断优化路径序列和路径点的之间距离值和路线。若TSP重复运行10次，则QPSO的调用次数为是10n。

程序框架如下 :

For k=0 to 5 do

Compute the line distance[n][n]

Run TSP()

(compute the routeSeq’length through updated distance[n][n])

Then find the best seq GbestSeq

For j=0 to n-1 do

front= GbestSeq[i]

next= GbestSeq[i+1]

If distace[front][next] could’t connect directly

Run QPSO() to optimal distace[front][next]

If distace[front][next] is more beater than ever

Update distace[front][next]

End if

End if

End for

find the best seq GbestSeq[n+1] and totalRouteLength

End for

可以看出，第一种方法的QPSO算法调用次数约为为第二种方法的(n-1)倍

**程序运行结果比较**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **NodeNum** | **MinReusult** | **Runtime(s)** | **RunNum** |
| **1** | **5** | **4486. 27** | **2987** | **5** |
| **2** | **5** | **4486. 49** | **93** | **10** |

**以上实例，如果相同的运行时间下，结果差别如何？是否有独立运行多次，得到最优、最差、平均值、标准差，平均用时？**

**采用相同的地形，对标准的TSP实例进行测试（先用最简单的那个TSP51个城市的实例）。每个实例独立运行10次，得到最优、最差、平均值、标准差，平均用时，列表对比。你还需要记录等间距迭代次数下对应的平均值，用于画出收敛曲线。**

**可以第一种方法的运行时间大约是第二种的32倍左右，结果比第二种方法稍好些，这是因为第一种方法计算最优序列的距离矩阵更精确。**

**文件说明**

makeMonitor.cpp 构造目标侦查点，坐标位置保存至monitorPoint.txt中

QPSO.cpp 寻找任意两点之间的近似最短距离，路线坐标保存至route文件夹,任两点距离值保存至distance.txt中

QPSO-TSP1 第二种方法所在程序文件夹

QPSO\_TSP2 第二种方法所在程序文件夹

每种方法的路线保存在route文件夹 ,运行结果在TSP文件中