文章编号：2096-1472(2022)-04-25-05
DOI:10.19644/j.cnki.issn2096-1472.2022.004.006

移动机器人路径规划算法仿真及实现

郭昭烽[1]，谢 玲[2]，刘红英[2]，邱池健[2]

(1.光大环境科技(中国)有限公司，江苏 南京 211102；
2.南京理工大学紫金学院，江苏 南京 210046)
✉louisguo1983@163.com; 12011833@qq.com; 328392801@qq.com; 865457907@qq.com

摘 要：机器人路径规划是机器人领域的一个重要分支，而路径规划算法是机器人的核心内容。本文重点研究路径规划算法，包括基础路径搜索，静态环境、动态环境的路径规划算法，通过仿真实验确定算法的可行性，并进行优化。最后，通过移动机器人实体验证路径规划算法。采用Python作为开发语言，先采用Pycharm作为开发环境并进行算法的实验仿真和对比；随后采用JupyterLab作为开发平台，使移动机器人能按照算法给出的路径移动。实验结果表明该算法可行。

关键词：移动机器人；路径规划；仿真
中图分类号：TP242 **文献标识码**：A

Simulation and Implementation of Path Planning Algorithm for Mobile Robots

GUO Zhaofeng[1], XIE Ling[2], LIU Hongying[2], QIU Chijian[2]

(1.*Everbright Environmental Technology* (*China*) *Co*., *Ltd*., *Nanjing* 211102, *China*;
2.*Nanjing University of Science and Technology ZiJin College*, *Nanjing* 210046, *China*)
✉louisguo1983@163.com; 12011833@qq.com; 328392801@qq.com; 865457907@qq.com

Abstract: Robot path planning is an important branch in the field of robotics, and path planning algorithm is the core of robots. This paper focuses its study on the path planning algorithm, including the basic path search and the path planning algorithm in both the static and dynamic environments. The algorithm feasibility is determined through simulation experiments, and then the algorithm is optimized. Finally, the path planning algorithm is verified by the real mobile robot. First, development language Python and development environment Pycharm are used to conduct experimental simulation and comparison of algorithms. Then JupyterLab is used as the development platform, so that the mobile robot can move along the path given by the algorithm. Experimental results show that the algorithm is feasible.

Keywords: mobile robot; path planning; simulation

1 引言(Introduction)

路径规划技术是机器人领域研究的核心之一。路径规划是指按照一定的评价标准(如选择最短的规划路径、最短的规划时间或者最小的工作代价等)，在存有障碍物的环境地图中，从起点到目标点之间找到可行地避免碰撞的路径[1−2]。路径规划大致可分为全局路径规划和局部路径规划两大类，又有静态、动态两个分支。全局静态规划算法从经典的BFS、DFS发展到后来的Dijkstra算法和A*算法，当外部环境不断发生变化时，又由A*算法演变到D*算法。除去经典的路径规划算法之外，如今人工智能相关算法也为机器人路径规划开拓了另一个方向，如蚁群算法、模拟退火算法、神经网络、粒子群算法和遗传算法等[3−6]。

在机器人运动的过程中起到决定性作用的是路径规划技术。如何提高机器人路径规划的技术，从实质上看，是如何创造出效率性更高的算法，或是针对机器人实际应用场景，将已有的算法进行有针对性的优化，极大程度地促进机器人技术的发展，这对整个机器人的发展领域具有极其重要的意义[7−8]。

2 路径规划原理(Principles of path planning)

机器人路径规划的过程中，主要分三步进行。

基金项目：江苏省教育厅高等学校自然科学研究面上项目(19KJD520007,19KJB520039)；南京理工大学紫金学院科学研究项目(2018ZRKX0401005).

首先是根据已知的地图信息来建模，障碍点、可行点、加权值等相关的地图数据参数被存储成可被算法识别的数据流。

其次，选择合适的算法进行研究，根据机器人所需要执行的任务进行实际优化，这一点可以使机器人具有应对单一任务的专一性，根据实际情况消除不必要的计算过程，提高机器人的路径规划效率。

最后，根据改进的路径规划算法为机器人寻找一条由机器人当前位置到目标点位置的无碰撞最优路径，机器人根据路径信息及配置的物理参数进行寻路。机器人路径规划原理如图1所示。

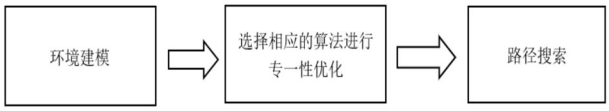


图1 路径规划原理

Fig.1 Principles of path planning

3 路径规划环境建模(Path planning environment modeling)

研究机器人路径规划，离不开对机器人移动环境的数据建模，环境建模是算法执行的关键，算法要依据环境信息给出结果。将机器人移动的实际三维环境空间进行规划划分，提取环境的信息特征，抽象计算机可以存储和操作的数字信息。算法能根据建模的环境信息进行运算和反馈。对于局部路径规划而言，环境建模需对环境出现的新变量进行及时处理，能够高效地、准确地将环境信息进行更新。这样机器人就可以利用建立起来的地图信息进行路径搜索。常见的环境建模方法有栅格模型、几何模型、拓扑模型、三位模型等。

本文采用栅格模型作为仿真实验的建模方法。栅格法环境建模的具体步骤为：首先将规划环境栅格化，处理成统一的单元模型数据，根据障碍物在环境中的分布信息，将对应的单元栅格处理成障碍栅格，其余为自由栅格。其次链接各个栅格处理成图，然后在图中搜寻一条可行路径，将路径的坐标存储反馈给机器人，坐标即是单元栅格的单元格编号。最后机器人根据所得编号对应的实际坐标进行移动完成路径搜索。其优点在于需要构成的二维阵列易于实现，且容易被计算机存储、操作和显示，同时易于修改和扩大地图规模，便于实验数据的获得。

4 路径规划算法研究及仿真(Research and simulation of path planning algorithm)

首先设置一个尺寸为15×15栅格的障碍地图作为每个算法的统一输入源，其次设置不同尺寸的地图，对各个算法进行路径搜索时间的统计。本文对Dijkstra、A*、D*的核心算法进行研究，在Windows 10系统中使用Pycharm开发平台，用Python语言对各个算法进行编程实现，以及设计GUI使算法中的各个步骤进行可视化处理，以此进行仿真实验。

4.1 环境建模

使用栅格法建立一张完全有向图，每一个栅格可以连接上下左右四个点。每个栅格的基础属性为笛卡尔坐标系的坐标(x，y)和一个判断是否为自由栅格的属性，以及一个保存可探索的栅格坐标的数组，其他属性根据不同算法重新构建。栅格模型建立的地图如图2所示。

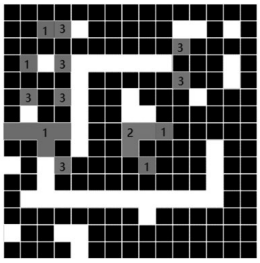


图2 算法仿真的地图

Fig.2 Algorithm simulation map

白色代表障碍栅格，用数字表示的代表自由栅格。黑色栅格代表权值为1，数字3栅格代表权值为2，数字1栅格代表权值为3，数字2栅格代表权值为4。各个栅格的位置如下：

$$Pos(x,y) = x \cdot w + y \cdot h \tag{1}$$

式(1)中，x、y代表该栅格在笛卡尔坐标系下的坐标位置，w、h代表栅格的像素宽度和高度，将在绘制GUI的时候使用。

4.2 Dijkstra算法

Dijkstra算法是从一个顶点到其余各顶点的最短路径算法，解决的是有权图中最短路径问题。其主要特点是从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。本文采用的是对BFS算法的改写，BFS算法中使用的地图，默认权值都为1，即从一个单元栅格到另外的节点单元栅格默认距离都为1。单元栅格的距离默认为1，这种情况表示在平地上运动，而现实的环境情况下，平地只是环境因素的一部分，还有山坡、低洼或者沼泽地等其他不同的环境条件，而在这些有坡度的环境因素下，机器人通过这些地方就要改变其速度，或者以时间为代价通过。也就是说，BFS得出的路径在有权图中并不是最优路径。因此我们通过给每个自由栅格增加一个权的属性，表示当前栅格到其他单元栅格的距离值，在有权图中采用Dijkstra算法求出最短路径。

本文使用一个优先队列存储，将这个要探索的单元格添加到队列中，而单元栅格到单元栅格的距离d为公式(2)所得：

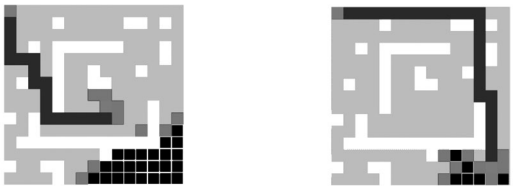
$$d\big((x1,y1),(x2,y2)\big) = dict[x1][y1] + dict[x2][y2] \tag{2}$$

d作为队列的优先级编号，距离越短，优先级越高，即优先队列会将当前单元栅格到附近最短距离的单元栅格出队，判断其当前的状态和后续节点的状态。

在实现Dijkstra算法时，我们引用Python.queue创建一个PriorityQueue，该队列是一个优先搜索队列，不同于普通队列，该队列的出入队是根据队列中对象的优先级而定的。该优先队列有两个参数，即index和object，index为优先队列对象object的优先值，优先队列会根据对象的优先值进行排序，每次得到的队头的对象都是优先值最高的，对应的index为最低数值。优先队列的存储和读取函数为push()和get()，将对象入队：PriorityQueue.push([index,object])；获取队头对

象：PriorityQueue.get()。算法首先根据传入的地图信息初始化distance，将不是终点的节点对向的权值设置为无穷大。算法执行到最后将返回两个字典对象parent和distance，机器人将通过遍历parent字典寻找最短路径。而distance记录点到点的距离，可以通过访问该字典求得最短路径的长度。

分别设置起始点(0,0)、终点(9,9)和起始点(0,0)、终点(14,14)进行仿真，该算法仿真结果如图3所示。



(a)Dijkstra算法路径搜索−1　(b)Dijkstra算法路径搜索−2

图3 Dijkstra算法路径搜索

Fig.3 Dijkstra algorithm path search

根据仿真结果可以看出，第一组参数得出的最短路径为13，第二组参数得出的最短路径为26。通过仿真的路径搜索可视化过程可以看出，Dijkstra算法与BFS算法类似，都是层层逼近终点，但是由于有优先值的加入，使得相对较远的子节点成为优先选择的对象，而不是靠近当前的节点作为下一步搜索的节点。

4.3 A*算法

A*算法引入了启发式搜索这个概念，也就是增加了一个启发函数，该函数能够计算得出一个值，该值代表当前栅格到目的栅格移动的优先级，数值越高优先级越低。可以通过下面的函数来表示节点的优先级：

$$f(n)=g(n)+h(n) \tag{3}$$

其中，$f(n)$是节点n的最终优先级，当算法在选取下轮遍历的节点时，选取当前节点列表中优先级最高的节点作为下次循环的起点；$g(n)$是节点n到目标点的距离；$h(n)$为节点n到目标点的预估代价。$h(n)$启发式函数如下：

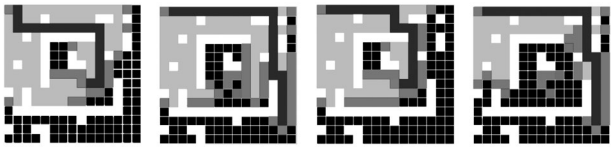
$$d(x,y)=|x_i-y_i|+|x_j-y_j| \tag{4}$$

$$d(x,y)=\sqrt{(x_i-y_i)^2+(x_j-y_j)^2} \tag{5}$$

式(4)是曼哈顿距离的计算，式(5)是欧拉距离的计算，对于不同的地图模型，将选取不同的启发式函数。

首先建立两个列表OPENSET和CLOSESET，列表存储的是节点对象，OPENSET将存放可以被选择移动的节点，CLOSESET存放被选择过或不可选择的节点。根据Point类中的属性进行判断，即Point.obs和Point.close中有一个为True，则这个节点将会被保存在CLOSESET中。算法从起始点开始逐层搜索，将从OPENSET中选取代价最小的节点开始探索，根据当前节点中临近节点列表中的节点的代价，选取最小代价的点，添加到OPENSET中，作为下一次探索的起始点。在退出当前节点时，会将当前节点的now_Point.close设置为True并添加到CLOSESET中。

分别设置起始点(0,0)、终点(9,9)和起始点(0,0)、终点(14,14)，以及分别采用曼哈顿距离和欧拉距离作为启发式，仿真结果如图4所示。



(a)欧拉A*算法−1 (b)欧拉A*算法−2 (c)曼哈顿A*算法−1 (d)曼哈顿A*算法−2

图4 A*算法路径搜索

Fig.4 A* algorithm path search

如图4所示，两个启发式得到的路径结果都相同，图4(a)、图4(c)的距离为22，图4(b)、图4(d)的距离为26。但是我们可以很明显地看出，曼哈顿距离在栅格模型中搜索的范围相对于采用欧拉距离作为启发式的搜索范围更少，也就是说，A*算法在栅格模型中采用曼哈顿距离作为启发式方程，将会缩小时间复杂度，提高算法的效率。

4.4 D*算法

D*算法及其延伸算法广泛应用于移动机器人和自主车辆导航中。D*算法的核心与Dijkstra算法和A*算法类似，需要维护一张OPENSET表。该列表记录每一个单元栅格的五个状态：NEW、OPEN、CLOSED、RAISE、LOWER。NEW代表该节点从未加入OPENSET中。OPEN代表该节点当前处于OPENSET表中。CLOSED代表该节点已被OPENSET列表移除。而RAISE和LOWER则记录移动成本的代价，RAISE代表当前移动成本比未改变地图信息时的成本高；LOWER代表当前移动成本比未改变地图信息时的成本低。有了这五个状态，算法可以在改变地图信息后，对路径进行重新预估和修正。

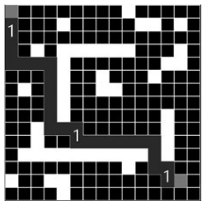
当障碍点的增加数量过多的时候，还会引发另一个问题：死锁的出现，即障碍点的出现，导致自由栅格不能通过临近节点找到一条新的路线。而解决这个问题的方法，则是保存当前路径数据，然后使算法重新规划路径。

首先建立一张维护列表OPENLIST记录节点的状态，D*算法是反向搜索，所以将终点先添加到列表中。每一个单元节点有三个状态：NEW、OPEN和CLOSED，NEW代表当前节点未被加入OPENLIST中；OPEN代表当前节点在OPENLIST中；CLOSED代表当前节点不在OPENLIST中。此时的CLOSED不再是被移除的意思，即该节点曾被列入列表中。算法的搜索是扩张过程，即每次从维护列表中选取一个节点，对其节点的移动代价进行预估，将移动代价最小的节点保存到OPENLIST中，并修改其状态属性，同时将当前节点和选取节点的变化状态传播到临近节点，更新其对应的各个临近节点的权值。D*算法从目标点开始搜索，即每个节点都有一个反向指针，该指针由下一步移动的节点指向当前节点，最终的路径只要按照反向指针指向的节点搜索，就能得到路径。

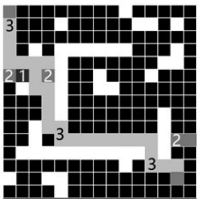
当在指定路径上有障碍物产生时，将检测到障碍物的附近节点放入OPENLIST中，此次将这些节点的预估值进行重新计算，只保留预估值最小的节点，并修改其状态，同时将节点的状态变化传播到临近节点。如果遇到堵塞，则按照Dijkstra算法从节点列表中选取最优的点进行回溯，并更新列表中的节点信息。

设置起点为(0,0)、终点为(14,14)，设置两组不同后续

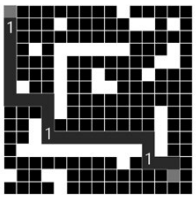
生成的障碍点分别为：第一组：(0,5)，(3,5)，(13,10)，(14,10)；第二组：(5,10)，(5,0)，(5,2)，(12,12)。新生成的障碍点将渲染成数字2栅格，与原有障碍白色栅格形成对比，仿真结果如图5所示。



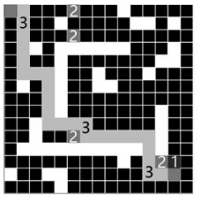
(a)第一组原始路径



(b)第一组重新生成路径



(c)第二组原始路径



(d)第二组重新生成路径

图5 D*算法路径搜索

Fig.5 D* algorithm path search

通过仿真结果可以看出，机器人在地图信息改变之后，在排除锁死情况发生的状况下，大致会沿着原来的路径前进，当遇到障碍物时，会选择绕开障碍物，走临近的可移动节点，直至终点。图5(b)和图5(d)中的数字1栅格表示上一次路径的选择，也就是说障碍生成之后，数字1栅格不是当前路径规划中的最优点。这就是D*算法不同于静态环境下的路径搜索算法的优点，即能在动态环境中实时修改前进路线，保证自身的安全。

4.5 算法仿真数据分析

首先建立尺寸不同的地图，因为可视化算法过程中有大量的图形输出，会影响算法的时间复杂度，所以对于算法运行时间的数据搜集，采取封闭式测试，只保留算法的核心计算代码，仅保留路径输出作为终点，采用python.time.process_time()函数，分别在算法起始点和终点设立start_time、end_time，最后将end_time与start_time相减得出的结果作为算法运行的时间，取各个算法仿真20 次所得的time，去掉最大值、最小值，取平均数作为最后的数据来源。在每轮实验中引入一组随机的障碍点添加到地图中，确保封闭测试得到数据的随机性和准确性。测试20 轮得到数据如表1、表2和表3所示。

表1 路径规划算法的运行时间

Tab.1 Running time of the path planning algorithm

尺寸/栅格	运行时间/s		
	Dijkstra算法	A*算法	D*算法
50×50	0.0156	0.0156	0.0458
100×100	0.0254	0.0312	0.3276
200×200	0.0781	0.1359	2.2932
400×400	0.3437	0.5890	16.7857
800×800	1.5343	2.4827	143.864
1600×1600	6.6562	10.0155	1,240.70

表2 A*算法和D*算法欧拉公式启发式运行时间

Tab.2 Heuristic running time of A* algorithm and D* algorithm Euler formula

尺寸/栅格	运行时间/s	
	A*算法	D*算法
100×100	0.0625	0.3437
200×200	0.1718	2.3391
300×300	0.4687	7.5312
400×400	0.6278	18.9781
500×500	1.0312	32.3213

表3 A*算法和D*算法曼哈顿公式启发式运行时间

Tab.3 Heuristic running time of A* algorithm and D* algorithm Manhattan formula

尺寸/栅格	运行时间/s	
	A*算法	D*算法
100×100	0.0312	0.3281
200×200	0.1406	2.3281
300×300	0.3125	7.5156
400×400	0.6093	18.7565
500×500	0.9687	32.1487

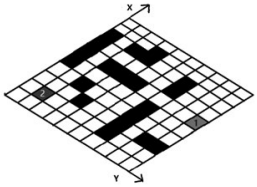
A*算法在将欧拉距离和曼哈顿距离分别作为不同的启发式的时候，采用曼哈顿距离的时间消耗少于欧拉距离，因此在现实环境中，机器人使用A*算法路径规划时，要根据现实环境来选择采用这两者中的哪一个作为启发式函数。在A*算法仿真实验中，我们提出了两个不同的启发式。由于采用栅格模型建模，栅格模型类似于城市网格，可以类比，在设计公交路线系统、搜索城市中点对点的最小路径时，便可以采用启发式为曼哈顿距离的A*算法。而本文所实现的A*算法，其中OPENLIST和CLOSELIST都是采用List的方式存储数据，Python中的列表是基于数组的类型，所以可以改用二叉堆这种数据结构，使其节点的添加和删除的操作速度更快，降低耗时。

D*算法弥补了A*算法不适合在动态环境中规划路径的缺陷。D*算法在未出现新的障碍源时，路径规划算法基本与A*算法一致。在出现障碍源时，D*算法会重新对存储的地图信息进行删除和插入操作，需要更多的计算，所以消耗的时间较多。

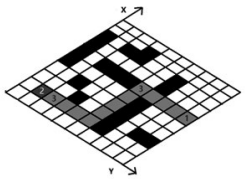
5 移动机器人路径规划实现(Implementation of mobile robot path planning)

路径规划实现如图6所示。

(a)JetbotNano开发主板



(b)实验地图模型图



(c)机器人完成路径移动

图6 路径规划实现

Fig.6 Path planning realization

5.1 开发环境

使用的机器人小车采用JetbotNano作为开发主板，如图6(a)所示，CPU为ARM的A57，操作系统为Unbuntu 18.10，开发平台为JupyterLab，开发语言为Python 3.0。

通过WiFi模板进行PC和机器人之间的数据传输，以及搭载四个直流电机控制车轮的移动。通过对GPIO口的编程，控制直流电机的运动方向和前进速度。

5.2 具体实现

通过分析仿真实验数据报告可以看到，由于我们在静态环境下实现该小车的路径规划功能，因此选取了有代表性的A*算法来为机器人规划行动路径。

首先需要导入地图数据，将二维空间地图抽象成算法可以处理的数据；再根据算法得到的结果，编程控制小车前进。地图的构建方式也是采用栅格模型，将实际地图抽象成二维平面，将每个单元栅格的信息采用一组二维数组来保存。此次实验地图如图6(b)所示，数字1单元格为机器人起点位置，数字2单元格为终点，黑色单元格为地图上的障碍。

在得到A*算法得出的路径列表之后，通过调用集成的开发库，使用主板上的i2c总线对四个直流驱动电机进行控制。调用Jetbot.Robot库创建一个机器人控制对象。Robot库的forward()、backward()、left()、right()、stop()分别控制机器人前进、倒退、左转、右转、停止。根据地图的参数以及实际地面情况，设置对应的前进速度和时间。

设地图*X*轴坐标对应为正东，反方向为正西；*Y*轴坐标对应为正南，反方向为正北。机器人起始点车头对应方向为正南。根据路径列表，机器人首先调整车头方向，调整的方向为即将前进的下一个单元格的方向。设置一个变量保存当前机器人车头正对的方向信息，随后机器人前进一个单元格。方向参数设正南为(0,1)，正西为(−1,0)，正北为(0，−1)，正东为(1,0)，标记数值按照正南方顺时针方向设置为1、2、3、4，并保存在一个判断列表中。车头方向调整伪代码如下：

```
choise=[(0,1)，(−1,0)，(0，−1)，(1,0)]
now_position=(0,1)
next_position=下一个坐标的(x，y)减当前坐标的(x，y)
for  position in choise：
    if  next_position==position：
        now_postion=方向对应标记的数值
```

控制小车移动的伪代码如下：

```
while 路径列表不为空：
    value=获取当前车头方向信息
    if  value==0：    //如果车头方向与当前一致
        forward()   //小车前进
    if  value in [−2,2]：//如果当前车头与选择方向相反
        overturn()   //小车掉头
    if  value in [−1,3]：//如果前进方向在小车右边
        right()      //小车右转
    if  value in [−3,1]：//如果前进方向在小车左边
        left()       //小车左转
    now_value=next_value        //获取当前小车车头方向
    next_value=save_path.pop()//获取下一个节点的小车状态
```

通过对路径列表的循环遍历，初始化小车车头方向，然后前行，直至到达终点。实验结果如图6(c)所示。数字3单元格代表机器人前进的路线。实验结果是机器人成功实现掉头、转弯、前行等功能，按照A*算法所得的路径列表移动，准确在终点停止。

本次实物演示的地图为一张根据机器人车身定制的2 m × 2 m的塑料布材质的地图，内部大小为20 cm × 20 cm的正方形栅格。地图在空地上展开，在保持平整的情况下，完成路径规划的各项参数为：前进电机速度：0.45(驱动电路参数，需要根据具体环境进行测试和调节)；左转电机参数：0.585；右转电机参数：5.585；延时：1 s。利用Jetbot小车自身所带的摄像头，使小车在每次前进、掉头时，摄像头根据比对地图的数据，保证机器人小车能够时刻居中，路线不偏移。

6 结论(Conclusion)

基于机器人路径搜索算法的研究，本文对Dijkstra算法、A*算法、D*算法进行研究，仿真实现了各个算法在路径搜索上的演示，通过对比各算法的时间复杂度和搜索出的最优路径以及对应的数据，确定在移动机器人实体上采用A*算法。而通过实验可知，在栅格网络中，A*算法采用曼哈顿距离启发式，仿真可以得到更加优异的结果。同时也提出了如何优化A*算法的思路，即使用二叉树堆代替数组来降低算法的时间复杂度。最后在移动机器人上实现路径规划算法的演示，证明了该算法的可行性。

参考文献(References)

[1] 刘杰.基于环境地图的机器人全局路径规划的研究[D].武汉:武汉理工大学,2013.

[2] 黄兴华.基于改进人工势场法的移动机器人路径规划研究[D].重庆:重庆大学,2010.

[3] 张艳,张明路,蒋志宏,等.动态环境下移动机器人路径规划的研究[J].合肥工业大学学报(自然科学版),2020,43(10):1297−1306.

[4] 张红阳,肖宇峰,刘冉.低带宽条件下机器人探索地图的传输方法[J].信息技术与网络安全,2018,37(02):58−62.

[5] 钟灿灿,陈万米.移动机器人的混合式路径规划算法研究[J].自动化仪表,2021,42(09):61−66.

[6] 沈显庆,马志鹏,孙启智,等.改进A*算法的移动机器人的路径规划[J].黑龙江科技大学学报,2021,31(04):494−499.

[7] ARAUJO R. Prune−able fuzzy ART neural architecture for robot map learning and navigation in dynamic environments[J]. IEEE Trans on Neural Network, 2006, 17(05):1235−1249.

[8] 朱诚.基于鲸优化算法的自导航机器人路径规划研究[J].软件工程,2020,23(12):7−11.

作者简介：

郭昭烽(1983−)，男，硕士，工程师.研究领域：自动控制，优化算法.

谢　玲(1984−)，女，硕士，副教授.研究领域：自动控制，人工智能.

刘红英(1987−)，女，硕士，讲师.研究领域：网络安全，人工智能.

邱池健(1998−)，男，本科生.研究领域：自动控制.