doi:10.3772/j.issn.1002-0470.2021.05.007

基于 TSA * 算法的双机械臂协同避障规划方法①

胥 芳② 沈旭明 谭大鹏③

(浙江工业大学机械工程学院 杭州 310014)

摘 要 提出了一种基于时序 A *(TSA *)算法的双机械臂路径规划方法。针对 A * 算法在高维环境下计算时间急剧增加的问题,引入期望步数 *Fs* 避开无效节点,减少计算工作量;在高维空间搜索时,因 open 表急剧变大导致维持 open 表最小二叉堆结构的时间大幅增加,通过对 open 表进行拆分,进而降低维持 open 表结构的时间消耗。主臂在静态环境中进行避障路径规划,从臂以主臂为已知动态障碍物进行路径规划。在规划从臂时,通过运动等待策略实现从臂自动做出等待决策。经过仿真实验对比,验证了该算法的有效性和可行性。

关键词 A * 算法;双臂机器人;避障;主从规划;已知动态环境

0 引 言

随着科学技术的进步和现代工业的发展,各种类型的机械臂在各种行业得到越来越多的应用[1]。在很多生产环节中,仅由单机械臂越来越难以满足复杂任务的作业需要。两个机械臂相互协调、相互配合地完成作业的形式能够适应工作任务中不断提高的复杂性、智能性以及系统柔顺性的要求,因此双臂机器人成为研究和应用热点。双臂机器人不同于两个单臂机器人的简单组合,它们在共同的空间内运动时,机械臂与空间障碍物之间、机械臂与机械臂之间可能会发生干涉碰撞,故需要对机械臂和环境之间以及双机械臂之间进行配合协调处理。在双臂机器人运动规划的研究中,如何避免这两类碰撞是关键问题之一[2-4]。

避障路径规划是指在给定的障碍条件以及起始和目标的位姿下,选择一条从起始点到达目标点的路径,使运动物体能安全、无碰撞地通过所有的障碍。在路径规划领域,有 A *、快速搜索随机树(rapidly-exploring random tree, RRT)、Dijkstra、人工势场法(artificial potential field, APF)[5]、遗传算法(genetic algorithm, GA)、蚁群算法(ant colony optimization, ACO)[6]等诸多算法。František 等人[7]比较分析了 A *、Theta * 和 Phi * 等算法的差异性,可以选择适合于具体环境的路径规划方法。Akshay 等人[8]提出了一种基于时间效率的 A * 路径规划方法,只在碰撞阶段之前计算启发函数值,极大地减少了处理时间。Wei 和 Yang[9]考虑了所有几何上合理的路径,并考虑了迂回路径、转弯次数等约束条件,提出了一种完全基于 A * 算法的车辆导航算法。Robert 等人[10]提出了一种基于 A * 算法的稀疏 A * 算法,应用机器人自身运动参数过滤了很多相邻节点。Zhang 等人[11]提出了一种基于网格的矩形拓展算法,可以将高度优化的 A * 算法加速一个数量级以上。目前 A * 算法多应用于平面或者三维环境下的路径规划,其在高维度环境下计算时间较长的特征极大地限制了 A * 算法的运用。

由双机械臂同时进行路径规划存在维度过高、搜索空间过大的情况,本文采用主从机械臂依次规

① 国家自然科学基金(51775501)和浙江省自然科学基金(LR21E050003)资助项目。

② 女,1964 年生,博士,教授;研究方向:计算机智能控制技术,嵌入式系统及其应用;E-mail: fangx@zjut.edu.cn

③ 通信作者,E-mail: tandapeng@zjut.edu.cn

(收稿日期:2020-03-09)

划的方式。将双机械臂依据作业特征分为主、从机械臂。针对静态障碍物对主臂进行避障路径规划,寻找主臂可行路径。将主臂每一时刻的运动位姿视为从臂运动时的动态障碍物,为从臂规划可行运动路径。为了解决 A * 算法在高维度下计算时间过长的问题,本文分析 open 表节点数据特征,识别出其中的无效路径点,并引入新的评估值 *Fs* 以避免对无效路径点的分析,还通过对 open 表结构优化缩短排序时间。本文以双臂机器人为研究对象,提出了基于时序 A * (time sequence A *,TSA *)算法,为A * 算法中路径点附加时间参数,并在规划从臂时允许做出原地等待的决策。该算法能够在已知动态环境下只经过一次规划即可得到全局最优路径,且所得路径的路径长度和路径代价较小。

1　A * 算法模型改进

1.1　A * 算法原理及问题

A * 算法是一种静态路网中求解最短路径的启发式搜索方法,在搜索中加入了与问题有关的启发性信息,指导搜索朝最有希望的方向进行,用于搜索状态空间的最短路径[12]。通过代价评估值 f 确定搜索方向,每次选择 open 表中最小代价节点作为下一个分析节点,重复这一过程直到达到目标节点。A * 算法中对位置的估价是十分重要的,采用不同的估价可以有不同的效果[13]。估价函数计算方式如下式所示。

$$f_{(n)} = g_{(n)} + h_{(n)} \tag{1}$$

其中 $f_{(n)}$ 是节点 n 的估价函数,它表示从起始节点到节点 n,再由节点 n 到目标节点所花费的估计代价。$g_{(n)}$ 是从起始节点到节点 n 的实际代价。$h_{(n)}$ 是从节点 n 到目标节点的估计代价。

A * 算法在二维、三维空间内运用广泛,但是在高维度环境下,例如关节空间内的多自由度机械臂路径规划,A * 算法需要考虑的路径点数呈现指数爆炸特征,open 表急剧变大,实际需要分析的路径点数量急剧增加,同时用于维持 open 表最小二叉堆结构耗时比率大幅增加。这些特性极大地增加 A * 算法的搜索时间。而致力于缩短 A * 算法计算时间的改进大多是通过提取环境信息或者提取运动者本身信息,比如限制或者缩小搜索区域,使用跳点[14]等方式有选择性地将节点放入 open 表,也有学者通过变步长解决搜索数据量大的问题。

在关节空间内规划高自由度机械臂路径时,机械臂的多个关节角相互独立,需要通过正解将关节空间坐标转换成笛卡尔空间坐标才能与环境进行信息交互,缺乏直接的信息对搜索空间内的路径点做出评价和筛选,容易导致需要分析的节点过多,从而消耗过多的计算时间。故而本文从 open 表节点特征及筛选,和 open 表存储结构 2 个方面以期改进 A * 算法,减少计算时间。

1.2　启发函数与 close 表中的无效节点

启发函数 h 的估计方式是决定 A * 算法搜索质量的关键因素,一般需要满足 $h_{(n)} \leqslant hreal_{(n)}$,其中 $hreal_{(n)}$ 为 n 点到目标点 goal 的实际路径代价。当 $h_{(n)} = 0$ 时 A * 算法成为 Dijkstra 算法,通过遍历搜索的方式确保能够找到最优路径,但牺牲了运算时间;当 $h_{(n)} \leqslant hreal_{(n)}$ 时,A * 算法是可以找到最优路径的,但 $h_{(n)}$ 不宜过小,因为 $h_{(n)}$ 越小,A * 需要扩展和分析的节点越多,计算耗时越久;当 $h_{(n)} = hreal_{(n)}$ 时,A * 算法扩展的所有节点都在最优路径上;当 $h_{(n)} > hreal_{(n)}$ 时,A * 算法将无法确保能够找到最优路径。

本文只考虑路径代价由两个节点相对位置决定,而与节点绝对位置无关的环境,此时 g 和 h 常用曼哈顿距离、切比雪夫距离、欧几里得距离、对角线距离等。接下来本文将对 A * 算法 open 表和 close 表节点的 $f_{(n)}$、$freal_{(n)}$ 与目标点 goal 的 $f_{(goal)}$、$freal_{(goal)}$ 进行对比分析。实际总路径代价 $freal_{(n)}$ 计算公式如下:

$$freal_{(n)} = g_{(n)} + freal_{(n)} \tag{2}$$

其中 $hreal_{(n)}$ 由 Dijkstra 算法计算得到。在一个简单环境使用 A * 算法搜索,其节点分布如图 1 所示。

在图 1 中,黑色区域表示障碍物区域,空白小方格表示没有被考虑过,A * 算法的 open 表和 close 表节点依据 f 和 $freal$ 可以简单分为以下 4 个区域。

A 区域节点:$freal_{(a)} > freal_{(goal)}$,$f_{(a)} = f_{(goal)}$;

B 区域节点:$freal_{(b)} > freal_{(goal)}$,$f_{(b)} < f_{(goal)}$;

C 区域节点：$freal_{(c)} = freal_{(goal)}$，$f_{(c)} < f_{(goal)}$；

D 区域节点：$freal_{(d)} = freal_{(goal)}$，$f_{(d)} = f_{(goal)}$。

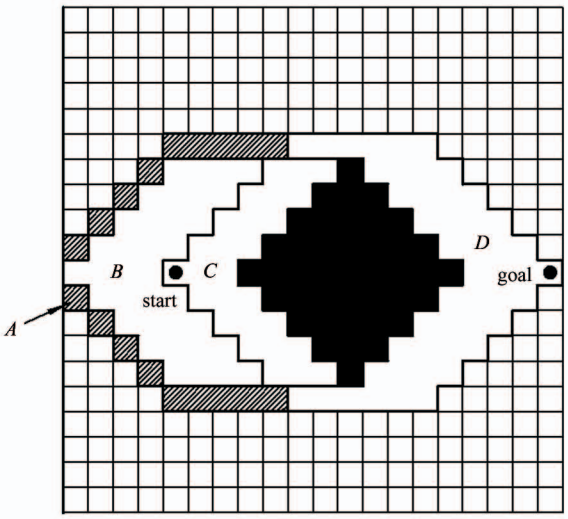


图 1　传统 A * 算法的节点依 f、$freal$ 分组结果

依据 A * 算法特征，求得的最优路径必定是路径代价最小的。因此当 $freal_{(n)} > freal_{(goal)}$ 时，节点 n 必定不会出现在 A * 算法的最优路径上；而满足 $freal_{(n)} = freal_{(goal)}$ 的节点则可能在最优路径上。依据 A * 算法原理，满足 $f_{(n)} < f_{(goal)}$ 的节点必定在发现目标点 goal 之前就从 open 表取出，进行扩展分析并放入 close 表中，而 $f_{(n)} = f_{(goal)}$ 则可能在 open 表，也可能 close 表中。故而 A、B、C、D 4 个区域的节点特征可以用文字描述如表 1 所示。

表 1　传统 A * 算法节点特征

区域	是否在最优路径上	是否被扩展分析
A	一定不在	可能
B	一定不在	一定
C	可能在	一定
D	可能在	可能

像 B 区域节点这类符合“必定被扩展分析，但一定不会出现在最优路径上”的节点可以视为无效节点，如果能够避免对无效节点的分析则可以提高 A * 算法分析效率；而可能在最优路径上的 close 表节点则视为有效节点。

1.3　新的评估函数值 Fs

栅格中的 A * 算法的搜索空间为一个一个的栅格点，其规划出来的路径也是由一个一个的路径点组成。在 A * 算法中，每次从 open 表头取出 f 最小的路径点进行扩展，需要计算相邻点的评估值 f 并在必要时更新路径点之间的父子关系。

最终得到的路径由一系列的路径点组成，而每一个路径点都可以提取出到达所需运动步数 $Gs_{(n)}$ 这一特征参数，其中起点 start 的到达所需步数 $Gs_{(start)} = 0$，父节点 fa 和子节点 son 的到达所需步数关系为

$$Gs_{(son)} = Gs_{(fa)} + 1 \tag{3}$$

为了避免对无效节点的分析，本文引入新的评估函数值：预期总运动步数 Fs。其计算方式与传统 A * 算法的 $f_{(n)} = g_{(n)} + h_{(n)}$ 较为相似，这种基于运动步数的新评价方式可以表达为

$$Fs_{(n)} = Gs_{(n)} + Hs_{(n)} \tag{4}$$

$$Hs_{(n)} = \max(|\theta_{g1} - \theta_{n1}|, |\theta_{g2} - \theta_{n2}| \cdots |\theta_{gk} - \theta_{nk}|) \tag{5}$$

其中，目标点 goal 的坐标为 $[\theta_{g1}, \theta_{g2} \cdots \theta_{gk}]$，当前点 n 的坐标为 $[\theta_{n1}, \theta_{n2} \cdots \theta_{nk}]$。而 $Fs_{(n)}$ 表示从起点经过节点 n 再到达目标的预期总步数；$Gs_{(n)}$ 表示从起始点到节点 n 的实际步数，相邻节点之间的运动只需一步；$Hs_{(n)}$ 表示节点 n 到目标点的估计所需步数，依据式(4)所示的切比雪夫距离计算。

对图 1 中的 A * 节点（包括 open 节点和 close 节点）提取特征值 Fs，并按照 Fs 与 $Fs_{(goal)}$ 的对比将所有节点进行分组，结果如下。

图 2 中传统 A * 算法的节点可以依据 Fs 分为 2 部分。

E 区域节点：$Fs_{(e)} > Fs_{(goal)}$；

F 区域节点：$Fs_{(f)} \leqslant Fs_{(goal)}$。

对比图 1 和图 2，可以发现 E 区域等于 A、B 区域之和；F 区域等于 C、D 区域之和。即符合 $Fs_{(n)} > Fs_{(goal)}$ 的同时也符合 $freal_{(n)} > freal_{(goal)}$；符合 $Fs_{(n)} \leqslant Fs_{(goal)}$ 的同时也符合 $freal_{(n)} = freal_{(goal)}$。

此后，经过多次仿真，能够发现在不同维度、不同搜索空间大小、不同障碍物环境下，只要满足以下 2 个条件就能够保证上述结论成立。（1）在不考虑障碍物时，两个路径点之间的路径代价由两个点的相对位置决定，与路径点的绝对位置无关，即搜索空

间内所有节点不受到环境的特殊加权处理;(2) 启发函数 $h_{(n)}$ 使用曼哈顿距离、切比雪夫距离、欧几里得距离、对角线距离等计算方式,同样没有额外的数据处理。

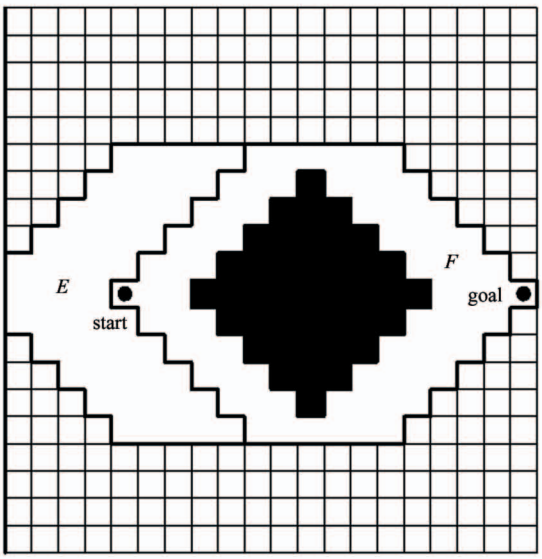


图 2　传统 A * 算法的节点依 *Fs* 分组结果

在图 2 所示的二维空间中,搜索较为简单,close 表中无效节点占 close 表节点的比例不高。而经过多次随机环境下的仿真模拟,能够发现当搜索空间的维度变高时,close 表节点数变多时,无效节点占 close 表总节点数的比例也随着增加。在某一次障碍物和起点目标点随机生成的四维搜索空间中,A * 算法的可抵达的 close 节点随 *Fs* 的分布情况如图 3所示。除去因障碍物而不可抵达的 close 表节点,图 3 中可抵达的 close 表总节点数为 285 019,其中无效节点数量为229 606,占可达close表节点的

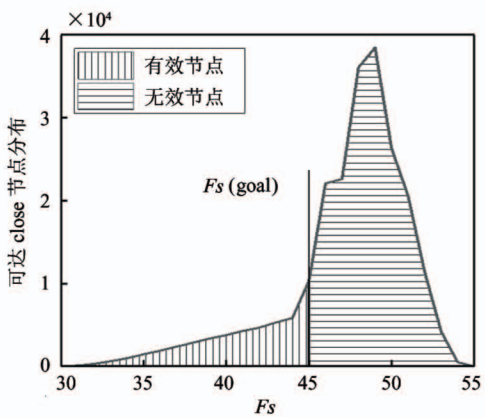


图 3　四维环境中可达 close 节点随 *Fs* 的分布情况

80.56%,而有效节点数为55 413,只占可达 close 表节点的 19.44%。

由于在搜索结束之前无法确定 $Fs_{(goal)}$ 的值,故而应优先分析 *Fs* 较小的 open 表节点(即 *F* 区域优先于 *E* 区域),当 *Fs* 相同时优先处理 *f* 较小的 open 表节点,进而避免对 open 表中无效节点的分析。

添加 *Fs* 后 A * 算法的新评估值 $F_{(n)}$ 计算方式为

$$F_{(n)} = k \times Fs_{(n)} + f_{(n)} \tag{6}$$

其中,*k* 为一个极大的数,以确保 *Fs* 优先考虑。

1.4　open 表结构优化

传统 A * 算法每次拓展时都要从 open 表中寻找 *f* 最小的节点进行拓展并放入 close 表中,但在 open 表中寻找代价最小节点会消耗大量的运算时间,进而极大地降低 A * 算法效率。许多学者针对 open 表的排序问题进行了大量的研究[15]。将最小二叉堆 heap 用于 open 表节点排序,将 open 表中所有节点按照 *f* 的大小排列成最小二叉堆结构,使得 open 表中 *f* 最小的点始终在表头,在 A * 算法搜索期间,只进行 3 种操作:(1)取出表头 node,并将表末的一个 node 移至表头;(2)将新 node 添加进 open 表;(3)将 open 表中的一个 node 的 f_{old} 更新为更小的 f_{new}。上述 3 种操作过后,都要进行数据顺序调整使 open 表重新成为最小二叉堆。通过这种方法,可以用较少的二叉堆维护时间代替原本较大的搜索 open 表中最小 *f* 节点的搜索时间,进而减少了搜索时间[16]。

若 open 表结构无序,则寻找 open 表最小 *f* 的点的时间复杂度为 $O(n)$,而以最小二叉堆结构构建 open 表时,时间复杂度降为 $O(\log n)$。但是随着环境从二维、三维转换到更高维度时,open 表的数据量呈现指数增长趋势,维护 open 表结构的时间大幅增加,其占全部计算用时的比例也随之大幅增加。

如下文图 5(c)的 heap 曲线所示,在路径点扩展的方式和其余信息不变时,仅扩大 open 表数据量,路径点扩展速度会因维持 open 表结构用时的增加而显著降低。

故本文通过将 open 表的单个最小二叉堆'heap'依据 open 表节点的 *Fs* 的不同拆分成多个最

小二叉堆，即 $heap_{Fs(start)}$、$heap_{Fs(start)+1}$、$heap_{Fs(start)+2}$…。所有的 open 表节点根据 Fs 放入对应的 $heap_{Fs}$，同一个堆 $heap_{Fs}$ 中的 node 根据 f 的大小按照最小二叉堆结构进行堆内排序。若拆分后的 open 表的各个二叉堆节点数量为 n_1、n_2、n_3、n_4、…、n_k，而拆分前 open 表节点数为 n_{all}，则满足 $n_1 + n_2 + n_3 + \cdots + n_k = n_{all}$，其时间复杂度也从 $O(\log n_{all})$ 降为 $O(\log n_i)$，$1 \leqslant i \leqslant k$。

1.5 改进 A * 算法实现步骤

基于新的启发函数值 Fs 和 open 表结构优化，改进 A * 算法在分析节点时，优先处理 Fs 较小的 heap，再处理 Fs 较大的 heap。在同一个 heap 中，所有节点的 Fs 都相同，优先处理 f 最小的节点。具体 Fs-heaps-A * 实现步骤如下。

步骤 1 初始化环境信息，确定起点 start 和目标 goal，将起点放入 $heap_{Fs(start)}$ 中，当前需要处理的二叉堆 $heap_{Fs\_now}$ 满足 $Fs\_now = Fs(start)$。

步骤 2 从 $heap_{Fs\_now}$ 提取出首节点，放入 close 表。获取并计算相邻节点的 $Fs(new)$ 和 $f(new)$。若相邻节点不在 open 和 close 中则将其放入 $heap_{Fs(new)}$ 中；若位于 open 表且 $Fs(new) < Fs(old)$ 则将相邻节点从 $heap_{Fs(old)}$ 搬移至 $heap_{Fs(new)}$；若 $Fs(new) = Fs(old)$，$f(new) < f(old)$ 则更新节点在 $heap_{Fs(old)}$ 中的位置。

步骤 3 若当前正在处理的二叉堆为空则 $Fs\_now = Fs\_now + 1$。

步骤 4 若目标 goal 进入 open 表则跳转执行步骤 6。

步骤 5 跳转执行步骤 2。

步骤 6 根据父子指针关系获取完整路径。

其中 $heap_{Fs\_now}$ 为当前正在处理的最小二叉堆，也就是 open 表中 Fs 最小的二叉堆。$Fs(new)$，$f(new)$ 为相邻节点在 $heap_{Fs\_now}$ 首节点扩展时计算出的新的评估值。若相邻节点在扩展前就已在 open 表中，则其在 open 表中的评估值记为 $Fs(old)$、$f(old)$，并存放于 $heap_{Fs(old)}$ 中。

1.6 Fs-heaps-A * 仿真验证

为了验证算法的有效性，将传统 A * 算法和所提出的 Fs-heaps-A * 算法以及处于两者中间的算法进行比较。三种算法区别如下。

（1）heap 为传统 A * 算法，open 表结构为一个最小二叉堆 heap。

（2）Fs-heap 为两种算法的中间算法，在传统 A * 中引入 Fs 评价，节点总评价计算方式为 $F_{(n)} = k \times Fs_{(n)} + f_{(n)}$，其中 k 为一个极大的数。而 open 表结构为一个最小二叉堆 heap，没有利用 Fs 分组。

（3）Fs-heaps 为本文提出的算法，在传统 A * 中引入 Fs 评价，同时 open 表依据 Fs 拆分成多个 heap。

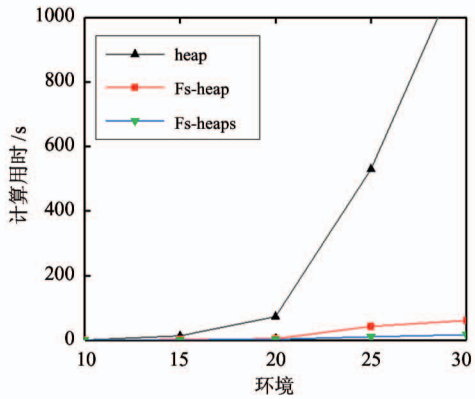
在路径评价方面，本文使用路径长度和路径代价 2 个角度进行比较，其中路径长度为路径上的路径点的数量，表征运动时间；而路径代价则用机械臂关节角累积变化量表示，表征能量消耗。在随机生成的不同环境下，仿真结果如表 2 所示，3 种 A * 算法分析所得的路径在路径长度和路径代价上完全一致。引入 Fs 和 open 表结构调整仅仅影响 open 表节点处理先后顺序和部分节点的父子关系，但不影响整体路径评价和长度。

表 2 3 种 A * 算法路径质量比较

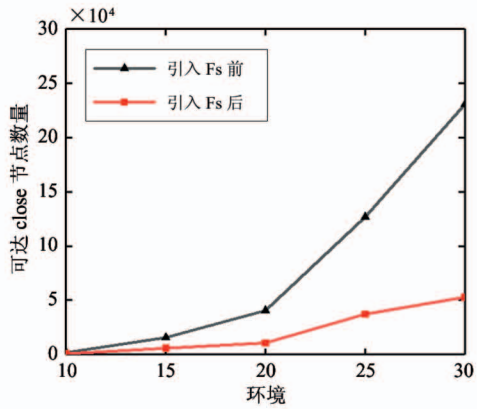
环境	路径长度			路径代价		
	heap	Fs-heap	Fs-heaps	heap	Fs-heap	Fs-heaps
1	8	8	8	16	16	16
2	15	15	15	36	36	36
3	23	23	23	56	56	56
4	30	30	30	76	76	76
5	38	38	38	96	96	96
6	45	45	45	116	116	116

heap、Fs-heap 和 Fs-heaps 这 3 种算法在相同维度不同环境大小下的仿真结果如图 4 所示，在不同维度下的仿真结果如图 5 所示。图 4 的仿真环境是 4 维环境，其中 x 轴表征实际搜索空间的大小，由仿真环境、起始位姿和目标位姿的距离决定。在图 4 和图 5 的子图（b）中，两条折线的差值为无效节点数量，“引入 Fs 后”折线的值为屏蔽无效节点后的有效节点数量。

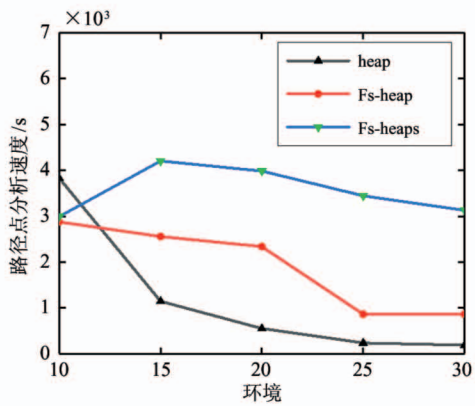
图 4 和图 5 比较了同一环境下 3 种算法在耗时、分析工作量和分析速度方面的差异。当 open 表较小，Fs 函数值计算和 open 表分组需要额外的运算时间，降低了运算效率，但当搜索范围变大时两项改动带来的收益就超过了其额外的时间消耗。在高



(a) 耗时

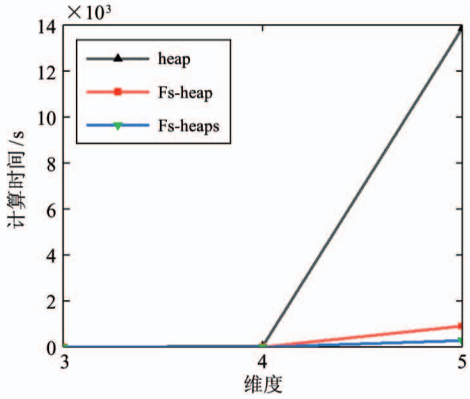


(b) 分析工作量

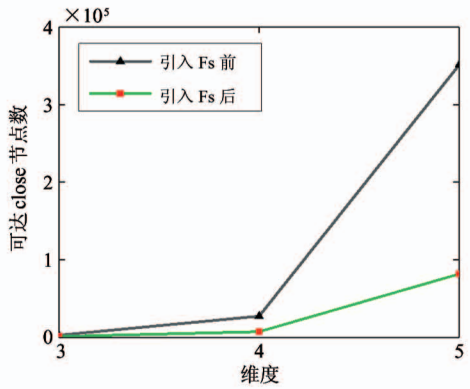


(c) 分析速度

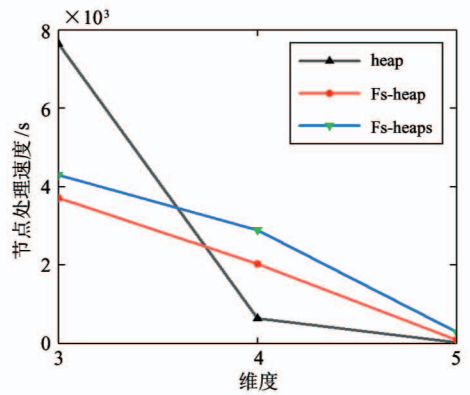
图 4　不同环境大小下 3 种算法仿真比较



(a) 耗时



(b) 分析工作量



(c) 分析速度

图 5　不同维度下 3 种算法仿真比较

维度下优化效果更加明显,用时差距从三维环境时的略差,到四维环境时接近 10 倍,再到五维环境时差距近 50 倍。

在图 4 和图 5 的子图(b)中,当搜索区域变大或者维度增加时,open 表节点数增大,无效节点的增长速度明显超过有效节点的增长速度。

在节点处理速度方面,可以对比图 4 和 5 的子图(c)。由于 3 种 A * 算法在节点扩展时的处理方式相同,引起时间处理速度差异的原因主要来自于维持 open 表结构的用时上的差异。heap 和 Fs-heap 的差异在于前者需要同时扩展有效节点和无效节点,而后则只需要扩展有效节点。在 Fs-heap 引入 *Fs* 后,无效节点同样会进入 open 表,但是因为扩展顺序被延后而不需要扩展(延后至发现目标点之后);Fs-heap 和 Fs-heaps 的差异在于 open 表的拆分,从需要维持整个 open 表简化为只需要维持其中

一部分节点的顺序。仿真结果证明了引入 *Fs* 和 open 表拆分能够有效缓解 A * 算法在高维环境下计算时间过长的问题，为今后 A * 算法在高维环境下的使用提供一定借鉴。

2　时序 A * 算法

2.1　动态已知环境和 *FT*

在双机械臂协调装配过程中，相比于单臂装配，主要差异在于双机械臂之间的碰撞协调处理。以六自由度机械臂为例，如果同时规划双机械臂路径，相邻节点数为有 $3^{12}-1$，而将双臂分主从臂依次规划时的相邻节点数只有 3^6-1，前者的关节位姿空间节点数远远大于后者，使用 A * 算法计算过于复杂和耗时，故本文在规划双臂装配路径规划时采取主从臂依次规划的方法。针对静态障碍物对主臂进行避障规划，寻找主臂可行路径，将主臂每一时刻的运动位姿视为规划从臂运动时的已知动态障碍物，为从臂规划可行运动路径[17]。

动态环境下的路径规划常使用的方法有 2 种。第 1 种是截取瞬时环境信息进行局部路径规划，并不断循环“更新环境信息-后续路径重规划”。第 2 种是先进行全局路径规划，并在局部碰撞区域进行局部路径规划。这两种路径规划方式都是依据瞬时信息进行多次规划，多用于动态未知环境，缺乏对已知动态障碍物（主臂）后续运动信息的充分利用，且该类方式规划得到的路径是基于局部瞬时信息的，从全局角度并非是最优的。

前述 Fs-heaps-A * 算法通过引入 *Fs* 和拆分 open 表的方法，提高了 A * 算法在高维度下的计算效率，却只能用于静态环境。若想要运用于已知动态环境，还需要进一步调整。通过将 *Fs*、*Gs*、*Hs* 这 3 个参数的步数替换为预计总花费时间 *FT*、最快到达所需时间 *GT*、余下路径预计需要时间 *HT*，即可用于已知动态环境。三者的关系如式(7)所示。

$$FT_{(n)}=GT_{(n)}+HT_{(n)} \tag{7}$$

针对路径点不同时间下路径代价不同的特性，计算每一个路径点最快到达所需时间，实现对已知动态障碍物运动信息的充分利用，且只需规划一次即可得到全局路径。通过该方法得到的路径明显优于多次重规划路径。

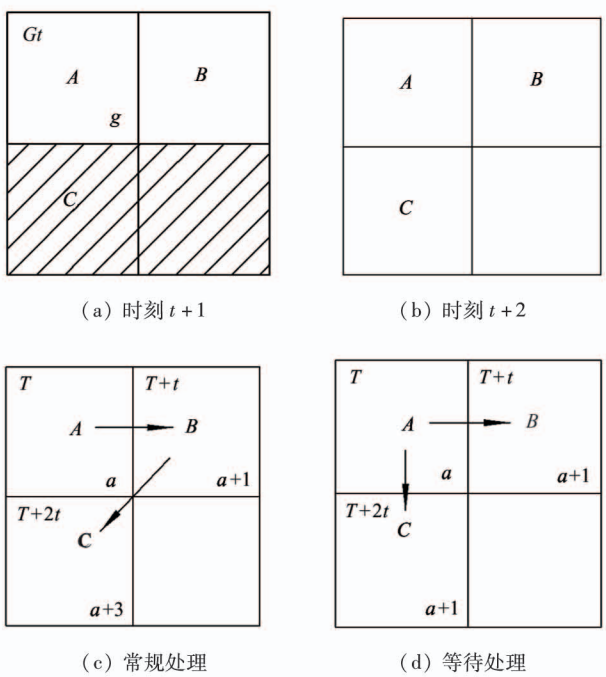
2.2　主从运动时间一致化

本文为主从臂设定了一个公共的单步运动时间间隔 t。通过机械臂 n 个关节角的角速度 $[\omega_1,\ \omega_2,\ \cdots,\omega_n]$ 以及单步时间间隔，可以确定路径点 $[\theta_1,\ \theta_2,\ \cdots,\theta_n]$ 的相邻关节位姿为 $[\theta_1\pm\omega_1 t,\ \theta_2\pm\omega_2 t,\ \cdots,\theta_n\pm\omega_n,\ t]$。这样处理后得到的主从机械臂路径上的相邻路径点的时间间隔均为 t，这就实现了主从机械臂路径上的路径点在时间上能够一一对应，这样做可以避免双机械臂独立运行时可能存在的速度波动带来的不良后果。

2.3　已知动态环境下的等待策略

为了使路径规划更加合理，在从臂路径规划时允许从臂进行“等待 1 ~ n 个单位时间 t”，即从臂关节位姿空间内的相邻路径点的 *Gt* 的差异可以大于单位时间 t。换而言之，节点的 *Gt* 应表示为从起点出发，经过时间 *Gt* 后运动到该节点，且实际处于运动状态的时间可能小于 *Gt*。一个点在扩展时，若其相邻点无法立刻到达，可以选择等待一定时间后再运动到相邻点。下面结合图 6 的 *A*、*B*、*C* 3 点进行说明。

如图 6 所示，图 6(a) 和图 6(b) 为表示环境信息，节点的左上方记录到达所需时间 *Gs*，右下方记



(a) 时刻 $t+1$　(b) 时刻 $t+2$

(c) 常规处理　(d) 等待处理

图 6　等待策略实现方式

录到达所需路径代价 g,阴影表示受障碍物影响而不可抵达状态;图 6(c)和图 6(d)表示 2 种不同处理方式得到的结果,其中箭头表示父子节点关系,箭头指向子节点,箭头尾为父节点。在图 6(a)中,时刻 $T+t$ 时阴影区域被障碍物覆盖而不可抵达;在图 6(b)中,时刻 $T+2t$ 时障碍物离开,下方 2 个位置重新变成可到达状态。

在图 6(c)中进行常规处理,首先进行点 A 的扩展,A 的下一时刻为 $T+t$,此时 B 点处于可抵达状态,而 C 处于不可抵达状态,故只有 B 成为 A 的子节点。随后进行 B 的扩展,B 点的下一时刻为 $T+2t$,此时 C 因障碍物离开而可抵达,故 C 成为 B 的子节点,$Gs_{(C)}=T+2t$, $g_{(C)}=a+3$。最终 A、B、C 3 点的关系如图 6(c)中箭头所示。

在图 6(d)中使用等待策略,首先进行点 A 的扩展,在时刻 B 处于可抵达状态,故 B 成为 A 的子节点,且 $Gs_{(B)}=T+t$;点 C 在时刻 $T+t$ 不可抵达,但不能就此判断 C 不能成为 A 的子节点,而是要判断经过等待后 C 能否成为 A 的子节点。在图 4 的环境中,点 C 在时刻 $T+2t$ 时能够抵达,故 C 成为 A 的子节点,$Gs_{(C)}=T+2t$, $g_{(C)}=a+1$。随后在拓展点 B 时,假设点 C 以 B 为父节点可以计算得到 $Gs_{(C)}=T+2t$, $g_{(C)}=a+3$。相较于 A 为父节点时更差,故 C 的父节点依旧为 A。最终 A、B、C 3 点的关系如图 6(d)中箭头所示。

图 6(d)中点 A 到点 C 的父子关系不是传统 A * 算法中的单步运动,而是先等待单步时间 t,待动态障碍物离开后再从 A 运动到 C。

同理,以主臂为已知动态障碍物进行从臂规划时,每次从 open 表头取出 f 最小的点进行扩展。在扩展时,若相邻点可以直接抵达则直接计算获得其 Gt 和 g;若相邻点经过等待若干时间后因动态障碍物离开后才变成可抵达状态,则计算获得其 Gt 和 g;若相邻点一直等待直到动态障碍物的最后一个位姿依旧无法抵达,则放弃此次扩展,不做任何处理。

2.4　TSA * 算法实现

基于 Fs-heaps-A * 算法的基础,经过时间一致化及引入运动等待策略,本文提出了基于时序的 TSA * 算法。TSA * 算法的具体流程如图 7 和图 8 所

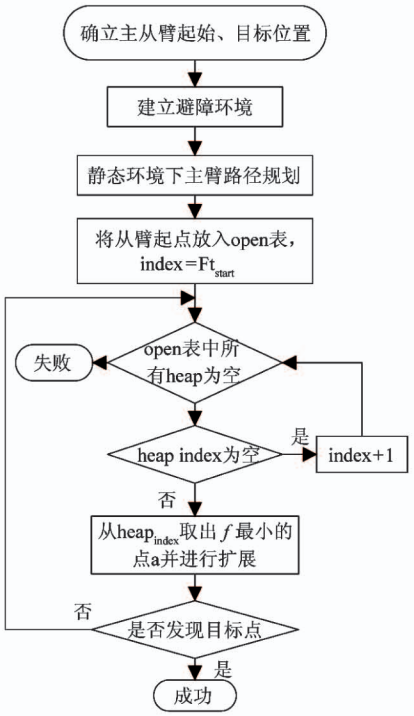


图 7　TSA * 算法总体流程图

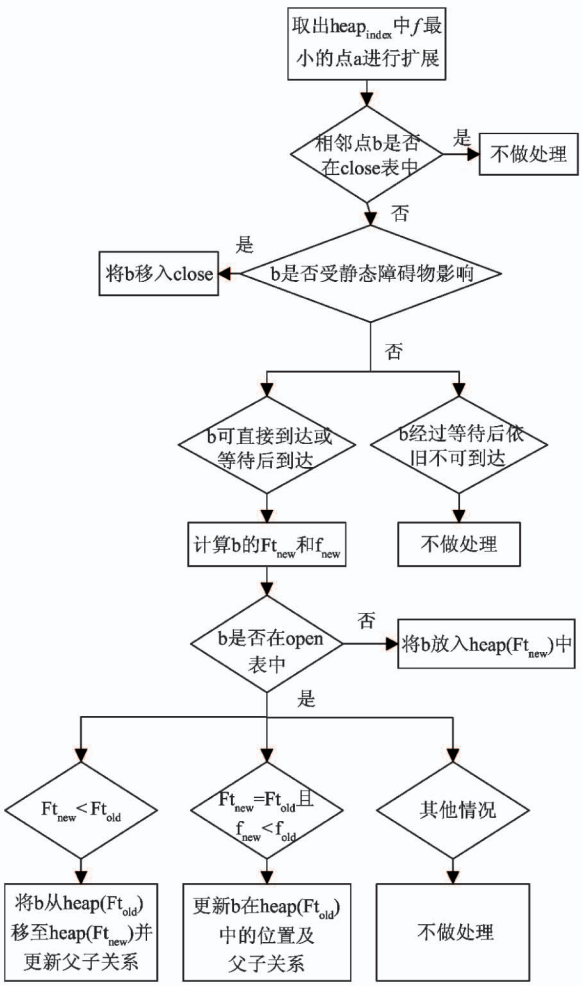


图 8　节点扩展流程

示。由于机械臂预期运动速度与实际运动速度之间可能存在微小误差，所以在得到两条 n 自由的机械臂路径后，需要将其合并成一条 $2n$ 自由度的双臂路径。

3　空间机械臂模型

为了验证本文算法的实用性，以三菱 RV-2F 双机械臂为例进行分析。首先根据实际工作需求确定路径规划的起始和目标位置，将工作空间内的起始和目标坐标通过机械臂逆解转换成关节空间坐标，然后在关节空间内进行避障路径规划。

RV-2F 机器人主机械臂结构及各关节坐标系如图 9 所示，D-H 参数如表 3 所示。其中，d_1 =295 mm，d_4 =270 mm，d_6 = 70 mm，a_2 = 230 mm，a_3 = 50 mm。而从机械臂的关节示意图与主机械臂的关节示意图镜像对称。主机械臂的 D-H 参数 α、θ 与从机械臂的 D-H 参数互为相反数。

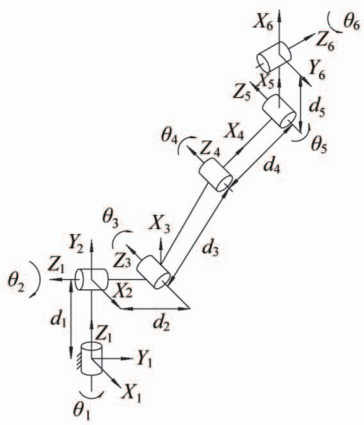


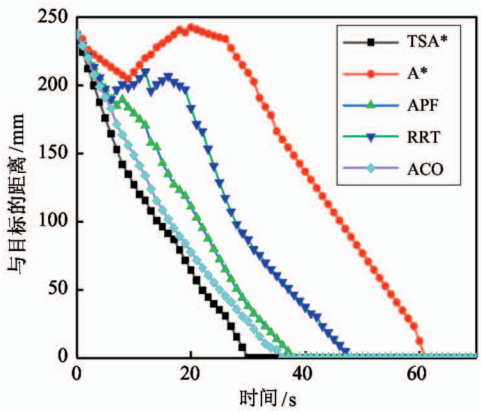
图 9　机械臂 HD 模型结构

表 3　RV-2F 机械臂 DH 参数

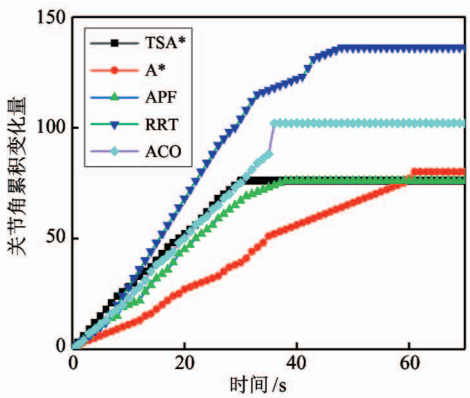
连杆	θ_i /rad	α_{i-1} /rad	a_{i-1} /mm	d_i /mm	θ_i 范围 /rad
1	θ_1	0	0	d_1	−240 ~ +240
2	θ_2	+90	0	0	−120 ~ +120
3	θ_3	0	a_2	0	0 ~ +160
4	θ_4	−90	a_3	d_4	−200 ~ +200
5	θ_5	+90	0	0	−120 ~ +120
6	θ_6	−90	0	d_6	−360 ~ +360

4　TSA * 算法仿真验证

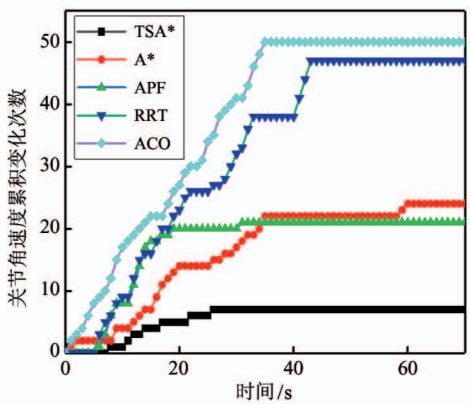
为了验证本文提出的 TSA * 算法，以三菱 RV-2F 机械臂为例进行仿真验证。将 TSA * 分别与 APF、RRT、传统 A * 算法、ACO 进行了比较仿真，结果如图 10 所示。这几种算法中，APF 和 TSA * 算法能够直接用于动态环境，而其他4种算法由于路径



（a）耗时



（b）关节角变化量



（c）角速度变化频率

图 10　TSA * 、A * 、APF、RRT、ACO 的仿真比较

点没有时间参数,只能利用瞬时环境信息进行规划,在已知动态环境中需要进行多次重规划或局部规划才能获得有效的完整路径,难以做到动态环境下的全局最优。

在图 10(a)中,对于相同的已知动态环境,TSA * 求得的路径所需的运动时间最少。机械臂能量消耗是评价避障路径质量的重要因素,主要表现在机械臂的关节角累积变化量和速度变化时的能量消耗,依据图 10(b)和 10(c)中的对比数据可以发现所提出的基于时序 A *(TSA *)算法具有最小的关节角度累计变化量和关节角速度累积变化量,在能量角度和机械臂寿命方面明显优于其他算法。此外,由于 TSA * 算法在规划从臂时允许等待,在面对已知动态障碍物时能够依据附近路径点信息合理地做出等待障碍物离开或者避让的决策,故而得到的路径也更加合理。

5　结　论

通过对 A * 算法节点的数值特征分析对比,本文发现了在路径代价不受环境信息加权影响时 close 表中存在无效节点,进而在 A * 算法中引入新的评估函数 *Fs* 以避免无意义的数据分析和时间消耗。同时,通过对 open 表的结构调整,缩短了 open 表处理时间。以上两点缩短了 A * 算法在 open 表较大时的路径搜索的分析时间,为高维度下 A * 算法的使用提供一定借鉴意义。

本文又在上述改进的基础上提出了一种用于双臂避障路径规划的 TSA * 算法。主臂在静态环境下规划,从臂以主臂为已知动态障碍物进行路径规划。在规划从臂时引入运动等待策略,依据已知动态主臂信息和从臂搜索时的相关数据,判断出是否要进行等待以及等待的时长,进而使得规划出的路径更加合理化。为验证 TSA * 算法的有效性,将 TSA * 算法与常见路径规划算法进行仿真对比。实验结果表明,本文提出的 TSA * 算法求得的路径在路径长度、路径代价和路径平滑性方面均具有较好表现。且在动态已知环境下,该算法具有较高的适应性,充分利用环境信息,通过单次规划即可求得全局最优路径,不需要进行多次重规划。后续将进一步分析主臂从臂的关系,寻找一种合理的方式来确定双机械臂中主、从臂的问题。

参考文献

[1] Ji S M, Huang X H. Review of development and application of industrial robot technology[J]. *Journal of Mechanical and Electrical Engineering*, 2015,32(1): 1-13

[2] 杨冬,李继强, 董跃巍, 等. 冗余双臂机器人实时协调避碰方法研究[J]. 机械设计与制造, 2019(3): 36-40

[3] 韩冬, 崔艳, 杨培林, 等. 空间双臂机器人机械臂最优轨迹规划仿真[J]. 计算机仿真, 2018,35(10): 333-339

[4] 李宪华, 孙青, 范凯杰, 等. 双臂 6R 服务机器人的协作空间分析与仿真[J]. 机械传动, 2018,42(9): 130-134

[5] 范海洲, 陈伟海, 刘敬猛, 等. 用于移动机器人路径规划的曲线轨迹算法[J]. 高技术通讯, 2011,21(5): 516-522

[6] 代亚兰, 熊禾根, 陶永, 等. HAPF-ACO:广义障碍环境下的移动机器人路径规划算法[J]. 高技术通讯, 2018,28(1): 67-77

[7] František D, Andrej B, Martin K, et al. Path planning with modified a star algorithm for a mobile robot[J], *Procedia Engineering*, 2014, 96: 59-69

[8] Akshay K G, Himansh A, Parsediya D K. Time-efficient A * algorithm for robot path planning[J], *Procedia Technology*, 2016, 23: 144-149

[9] Wei Y, Yang X G. A totally astar-based multi-path algorithm for the recognition of reasonable route sets in vehicle navigation systems[J], *Procedia-Social and Behavioral Sciences*, 2013, 96: 1069-1078

[10] Robert J S, Peggy G I, Clickstein S, et al. Robust algorithm for algorithm for real-time route planning[J]. *IEEE Transactions on Aerospace and Electronic System*, 2000, 36(3): 869-878

[11] Zhang A, Li C, Bi W H. Rectangle expansion A * path-finding for grid maps[J]. *Chinese Journal of Aeronautics*, 2016, 29: 1385-1396

[12] 贾庆轩, 陈钢, 孙汉旭, 等. 基于 A * 算法的空间机械臂避障路径规划[J]. 机械工程学报, 2010, 46(13): 109-115

[13] 赵真明,孟正大. 基于加权 A * 算法的服务型机器人

路径规划[J]. 华中科技大学学报, 2008, 36(1): 196-198

[14] 赵晓, 王铮, 黄程侃, 等. 基于改进 A * 算法的移动机器人路径规划[J]. 机器人, 2018, 40(6): 903-910

[15] Ahuja R K, Mehlhorn K, Orlin J B, et al. Faster algorithms for the shortest path problem[J]. *Journal of the Association for Computing Machinery*, 1990, 37(2):213-223

[16] 李志建, 郑新奇, 王淑晴, 等. 改进 A * 算法及其在 GIS 路径搜索中的应用[J]. 系统仿真学报, 2009, 21(10): 3116-3119

[17] 陈波芝, 陆亮, 雷新宇, 等. 基于改进快速扩展随机树算法的双机械臂协同避障规划方法[J]. 中国机械工程, 2018, 29(10): 1220-1226

Looperative obstacle avoidance planning method of two manipulacors based on TSA * algorithm

Xu Fang, Shen Xuming, Tan Dapeng

(School of Mechanical Engineering, Zhejiang University of Technology, Hangzhou 310014)

Abstract

A path planning method of double manipulator is proposed based on time sequence A * (TSA *) algorithm. In the static environment, motion planning for the main arm is obtained. Then the main arm is regarded as dynamic obstacles to obtain a feasible motion trajectory of the slave arm. Aiming at the problem that the computing time of A * algorithm increases sharply in high-dimensional environment, the expected steps *Fs* are introduced to avoid invalid nodes and reduce the computing workload. In high-dimensional space search, the time to maintain the minimum binary heap structure of the open table increases greatly due to the sharp increase of the open table, which can be reduced by splitting the open table. When planning the slave arm, the slave arm can automatically make the waiting decision through the motion waiting strategy. The simulation results show that the proposed algorithm is effective and feasible.

Key words: A * algorithm, dual arm robot, obstacle avoidance, master-slave planning, known dynamic environment