分类号：TP242.6　　单位代码：10335
密　级：公开　　学　　号：21932069

浙江大学

# 硕士学位论文

**中文论文题目**：引入运动学约束的异构多智能体路径规划方法研究及应用

**英文论文题目**：Research and Application on Heterogeneous Multi-agent Path Finding with Kinematic Constraints

申请人姓名：温力成
指导教师：刘　勇
专业名称：网络空间安全
研究方向：多智能体系统
所在学院：控制科学与工程学院

**论文提交日期** 二〇二二年一月十日

# 引入运动学约束的异构多智能体路径规划方法研究及应用


**论文作者签名：** _____

**指导教师签名：** _____

论文评阅人 1: 隐名评阅

评阅人 2: 隐名评阅

评阅人 3: 隐名评阅

评阅人 4: _____

评阅人 5: _____

答辩委员会主席: 俞俊 教授 杭州电子科技大学

委员 1: 肖俊 教授 浙江大学

委员 2: 朱海洋 教授级高工 物产中大集团

委员 3: _____

委员 4: _____

委员 5: _____

答辩日期: 2022 年 3 月 12 日

Research and Application on Heterogeneous Multi-agent Path Finding with Kinematic Constraints


Author's signature: _____

Supervisor's signature: _____

External Reviewers: Anonymous

Anonymous

Anonymous

Examining Committee Chairperson:

Jun Yu Professor HDU

Examining Committee Members:

Jun Xiao Professor ZJU

Haiyang Zhu Professorate Senior Engineer WZ Group

Date of oral defence: March 12th, 2022

# 浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得浙江大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

# 学位论文版权使用授权书

本学位论文作者完全了解浙江大学有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权浙江大学可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

# 致　　谢

我仍能清楚地记得高考结束的第二天走进紫金港校园时阴雨绵绵，倏忽发觉，时间已经走过了七个年头。剑桥公社的烧烤店开了又关，杭州夏末的梅雨季却是一如既往的潮湿。现在，我即将告别我的学生生涯，在此我想感谢一些陪我走到今天的人们。

首先感谢我的导师刘勇教授，是他的认可给了我进入实验室的机会。刘老师对学生认真负责，提供了良好的硬件环境和自由的科研空间，指引我在科研道路上砥砺前行。刘老师拥有开阔的学术视野，能准确发觉领域中有重要研究和应用价值的课题。在此由衷感谢刘老师在整个硕士生涯对我莫大的帮助和指导。

其次感谢 APRIL 机器人智能感知与学习实验室的所有同学，很荣幸和各位在教十八 101 度过了一个个或快乐或悲伤或惊喜或疲惫的日夜。我会怀念和昕欣、嘉卿一起出差在海边调试的风尘仆仆；我会怀念和老刘、浩浩、雪梦一起在赶完论文死线后约过的高强度娱乐局；我怀念震哥在路径规划领域对我的无私提携、指点迷津；我怀念楚娟、邹豪、小倩、阿林分享他们各自方向的学术前沿、工作体验。愿实验室蒸蒸日上，越来越好。

感谢 Jean-Marc 和赵宣博士对我硕士阶段工作的认可以及在相关研究中提供的帮助。

感谢我的父母和家人。我的父母提供了他们能给予我最好的教育机会和资源，在我身后默默支持着我一路求学。他们用自己的言传身教奠定了我生命中最重要的底色，这是在任何学校都无法学到的。祝他们身体健康，平安顺遂。

最后，感谢我的母校浙江大学。竺老校长问“诸位到浙大来做什么？将来毕业后要做什么样的人？”这是一个到今天刚好能回答一半的问题。求是园七载，我从一个刚成年懵懂未知的少年逐渐成长为了如今离开象牙塔跃入人海的青年。我醉心校园中走过的每一寸土地，也欢喜北街的静谧，堕落街的烟火气，青芝坞的人声鼎沸，西溪路的车水马龙。至于毕业以后，我自认无法如那些光辉的名字一样灿若星辰，但求能牢记求是创新的校训，坚持自己所选择的方向，无曰已是，无曰遂真。

欲买桂花同载酒，终不似，少年游。再见啦，我的学生时代。愿自己在之后的日子里，相信未来，热爱生命。

温力成

2022 年 3 月于求是园

# 摘　要

近年来随着多智能体系统在机器人和人工智能领域的广泛应用，多智能体路径规划问题被越来越多的学者关注和研究。高效地为系统中的智能体规划出尽可能短的避障路径同时保证智能体间不发生碰撞，成为了多智能体协作中的一个重要任务。同时，随着智能体控制系统的智能化水平不断提高，其面临的网络安全威胁也随之增加。如何保障系统安全稳定运行也成为多智能体系统亟待解决的重大挑战。本论文提出了一套可应用到实际工业场景中的全流程多智能体路径规划解决方案. 该方案能够在连续工作空间内为异构多智能体系统规划出符合智能体运动学模型的路径，同时也能鲁棒地处理系统在实际路径执行过程中发生的异常情况。本论文的主要工作和创新点如下：

1. 提出了一种针对阿克曼模型的多智能体路径规划方法，该求解器采用了分层搜索框架：在上层搜索中，本文提出了车身约束树的概念，只需检测规划方案中路径间是否存在车身冲突，而不必考虑具体的运动学约束；而在下层搜索中，本文提出了时空混合 A* 算法来进行单智能体路径规划。该方法可以在连续工作空间中为大量阿克曼运动模型的智能体规划安全避障路径，规划方案质量远超基线算法。
2. 提出了一种适用于异构多智能体系统的在线路径规划方法。该方法在前述方法的基础上，通过对下层单智能体路径规划算法一般化，将方法适用范围进一步扩展到包含不同大小、形状及运动学模型的异构多智能体系统中。同时，本文将滚动时域碰撞解决算法加入基于冲突搜索框架中，通过交替进行路径规划和智能体执行，解决了现实工业应用场景中存在寻路任务流的问题并增加了规划系统的运行效率。
3. 提出了一种基于行动依赖图的多智能体路径规划方案后处理框架。该框架利用前述路径求解器生成的规划结果来分析智能体间移动的优先级关系，仅需智能体和规划器之间进行少量通信，即可主动检测出系统内出现异常行为的智能体。通过该后处理框架，本文保障了多智能体系统在长时间内安全鲁棒地执行规划方案。

**关键词：**多智能体系统；路径规划；移动机器人；工业控制系统安全

# Abstract

With the widespread use of multi-agent systems in the field of robotics and artificial intelligence recently, the multi-agent path finding problem, also known as MAPF, has received an increasing amount of attention and research. Efficiently finding the shortest obstacle-free paths for all agents in the system while ensuring no collisions between them is a fundamental task in multi-agent collaboration. Meanwhile, as the level of intelligence in robot control system increases, so do the cybersecurity threats it faces. Ensuring the safe and stable operation of the system become a major challenge for multi-agent systems. This dissertation introduces a full-process multi-agent path finding scheme that can be applied to real-world industrial scenarios. The method is able to generate MAPF solutions for heterogeneous multi-agent systems in a continuous workspace that is consistent with agents' kinematic models while also robustly resolving the abnormal situation during actual execution. The main contributions can be summarized as follows:

1. A conflict-based search algorithm for the Ackermann kinematic model is proposed. The method adopts a hierarchical framework. In the high-level search, this dissertation proposes the concept of body conflict tree, which only requires checking for body conflicts between paths without considering specific kinematic constraints. While in the low-level search, this dissertation proposes a spatiotemporal hybrid-state A* algorithm to perform the single-agent path planning. This method allows planning kinematic-feasible paths for a large number of Ackermann agents in a continuous workspace and maintains a short search time.
2. An online MAPF method for heterogeneous multi-agent systems is proposed. Based on the previous MAPF solver for the Ackermann model in the continuous workspace, the low-level single-agent path planner is generalized and extended to heterogeneous multi-agent systems containing agents with different sizes, shapes, and kinematic models. This dissertation incorporates the rolling-horizon collision resolution algorithm into the conflict-based search framework, effectively reducing the agents' idle time and efficiently coping with the flow of path-finding tasks in the online MAPF problem.
3. A MAPF post-planning framework based on the action dependency graph is proposed. By utilising the planning results generated by the aforementioned path solver to analyse the

priority relationships of movements between intelligences, the framework can proactively detect intelligences with abnormal behaviour in the system with only a small amount of communication between the intelligences and the planner. With this active detection mechanism, this dissertation ensures that the multi-intelligent system executes the planning solution safely and robustly over a long period of time.

Keywords: Multi-agent system; Path planning; Mobile robots; Industrial control system security

# 目　　次

# 1 绪论

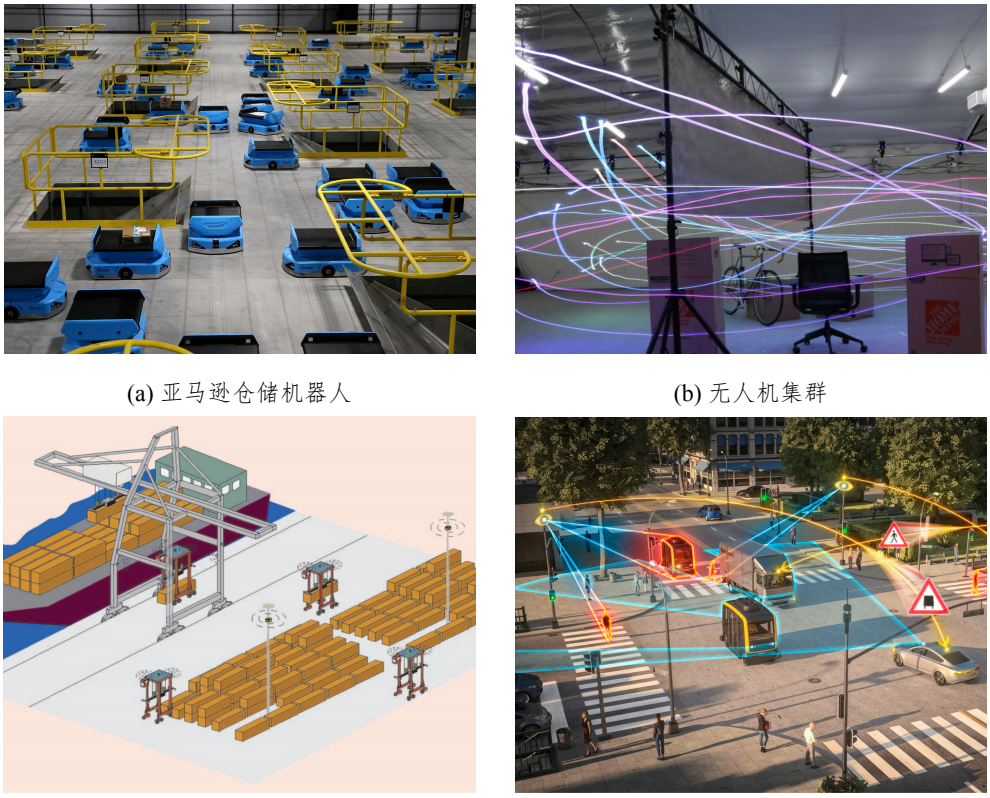
## 1.1 研究背景和意义

随着社会的进步和科技的发展，机器人被广泛运用于农业、工业、医疗和服务等行业中。但人们逐渐发现，单个机器人逐渐无法完成日趋多样化和复杂化的工作任务。自 20 世纪 90 年代初以来，多智能体系统（Multi-Agent System）开始引起广大学者的关注。多智能体系统是由环境中多个智能体组成的集合，该系统可以通过智能体之间以及智能体与环境间的交互、通讯和协作来完成单智能体无法解决的复杂任务[1]。由于多智能体系统相较单个机器人具有工作效率更高、任务领域更广、系统冗余更大及鲁棒性能更好等优势，其已成为当下机器人和人工智能领域的热点话题。

多智能体路径规划问题（Multi-Agent Path Finding，MAPF）是多智能体系统中的一个关键技术，也是多智能体得以协作完成复杂任务的基本保障。具体来说，多智能体路径规划问题是指利用已知的环境信息，为系统中的所有智能体规划出从起始位置到目标位置的路径。这些智能体的路径需要避开环境中的所有障碍物，同时保证与其他智能体不发生碰撞。MAPF 问题在物流仓储[2]、无人机集群[3]、集装箱跨运[4]、机场拖航[5] 及电子游戏[6] 等领域都有着广泛的应用，图 1.1展示了其中一些场景。

过去十年间，关于多智能体路径规划问题的研究一直在迅速发展。多智能体路径规划问题已经被证明是 NP 困难问题[7]，目前主流的 MAPF 问题的求解算法可以分为最优算法和次优算法两大类。最优算法是指当问题存在解时，该算法必定能求解出关于指定目标函数的最优解。最优 MAPF 算法又可以被分为扩展 A* 搜索，代价增长树搜索，基于冲突搜索和基于规约的算法四类。次优 MAPF 算法则牺牲了算法的最优性保证，从而大幅缩短算法的运行时间，为更大规模的多智能体系统进行路径规划。一部分次优算法被称为有界次优算法，它们基于最优 MAPF 算法提出，可以保证返回解的目标函数成本不大于最优解的 $1+\epsilon$ 倍。当 $\epsilon$ 增加时，该类算法的求解时长缩短。因此有界次优算法为使用者在运行时间和求解质量之间提供了一个可控的权衡。

然而，上述大部分方法都对多智能体路径规划问题进行了一些简化假设，这使得它们很难甚至无法应用到实际的工业多机器人系统中。首先，上述方法大都将工作空间离散为栅格地图或者拓扑地图，并将智能体建模为能够全向移动的圆形模型。但是现实中的移动



(a) 亚马逊仓储机器人 (b) 无人机集群

(c) 港口集装箱跨运车 (d) 无人驾驶车路协同

图 1.1 多智能体路径规划问题在不同领域的应用

机器人在一个连续的工作空间中移动，具有独特的外观和运动学约束，通常不能简化为全向运动学模型，很难执行栅格地图中生成的分段线性路径。其次，大部分 MAPF 算法都是离线规划算法，其只能为多智能体系统进行单次路径规划，并假设智能体在到达终点后会永远停在该位置等待。然而，在诸如物流仓储场景等众多工业系统应用中，新的规划任务会不断加入，需要部署在线规划算法引导智能体在完成当前任务后继续执行下一个任务。

与此同时，随着工业控制系统中的智能化程度不断增加，包括多智能体系统在内的工业机器人系统正面临着严重的安全威胁[8]。与传统网络环境不同，工业控制系统同时连接着网络空间和物理空间。在网络层面，智能体与规划器之间的通信大多使用 WiFi、蓝牙等公开通信信道，通信信息极易被监听甚至篡改，导致智能体无法接收到准确可靠的控制指令；在安全层面，大量工业机器人缺乏严格的身份认证机制，致使攻击者可以在未经授权的情况下，通过篡改固件或利用漏洞的方式远程对工业场景中的设备进行恶意控制[9]。前述 MAPF 方法通常只考虑对多智能体系统进行路径规划，而并不考虑实际执行阶段智能体能否按照规划好的路径安全可靠地执行，缺乏对于上述网络安全威胁的有效防御。一旦发生安全威胁事件致使多智能体系统被恶意控制偏离既定规划方案，不仅会给工厂带来巨大的经济损失，甚至还可能对员工的人身安全造成危害。

针对上述问题，本文着重研究并提出了一套可应用到实际工业场景中的全流程多智能体路径规划解决方案。本文首先基于冲突搜索算法，提出了一种满足阿克曼运动学约束的多智能体路径规划器，它能够在连续工作空间内为大量阿克曼机器人规划运动学可行的安全避障路径。随后将该方法推广到异构多智能体系统中，使得算法可以为包含不同形状及运动学模型的多智能体进行路径规划。本文利用滚动时域的碰撞解决算法，将在线多智能体路径规划问题分解为一系列不同时间窗口下的规划问题，使生成的解决方案能够应对持续新增的规划任务，并有效减少智能体的闲置时间。最后，本文基于行动依赖图构建了多智能体路径规划的后处理框架。它只需要智能体之间进行少量的通信，即可识别检测出系统内出现异常行为的智能体，进一步保障了多智能体系统在长时间内安全鲁棒运行。

## 1.2 国内外研究现状

本节将从最优 MAPF 算法、次优 MAPF 算法和对经典 MAPF 问题的扩展三个方面对国内外目前的研究现状进行概述。

1）**最优 MAPF 算法：**最优 MAPF 算法是指当问题存在解时，该算法必定能求解出相对于指定目标函数的最优解。最优算法又可以被分为扩展 A* 搜索，代价增长树搜索，基于冲突搜索和基于规约的算法四类。

A* 搜索算法是一个常见的单智能体路径规划算法[10]，它利用启发式函数极大提升了搜索效率。但是当它扩展到多智能体系统中时，其搜索的状态空间会随着智能体数量的增加而产生指数级别增长，从而无法直接应用到 MAPF 问题中。运算符分解（Operator Decomposition，OD）方法[11] 针对 A* 搜索中状态空间指数级增长的问题，引入了中间状态的概念，每次仅扩展系统中一个智能体的下一步行动，大幅降低了每次扩展的分支数量，并可以及时对启发式函数值过大的状态节点进行剪枝，如图 1.2所示。增强局部扩展 A*（Enhanced Partial Expansion A*，EPA*）[12] 是 A* 的一个变种，其只将一部分节点加入 open 列表中，能够有效缓解状态空间爆炸。文献[13] 证明了 OD 是 EPA* 的一个特殊情况。独立性检测（Independence Detection，ID）方法[11] 则试图将原 MAPF 问题解耦为若干个包含较少智能体的 MAPF 问题。其首先让每个智能体在忽略其他智能体的情况下独自规划一条最优路径，再对这些路径执行独立性检测，将有冲突的智能体合并为一个元组进行重新求解，提高了求解效率。独立性检测是一个非常通用的 MAPF 问题求解框架并被后续很多研究引入。M* 算法[14] 为了解决搜索中状态空间爆炸的问题，动态地改变搜索过程中每一次扩展的分支数量。当一个智能体与其他智能体不存在路径冲突时，其每次只朝着单智能体最优路径扩展一步。M* 算法对发生冲突概率不高的一类 MAPF 问题有着很好的

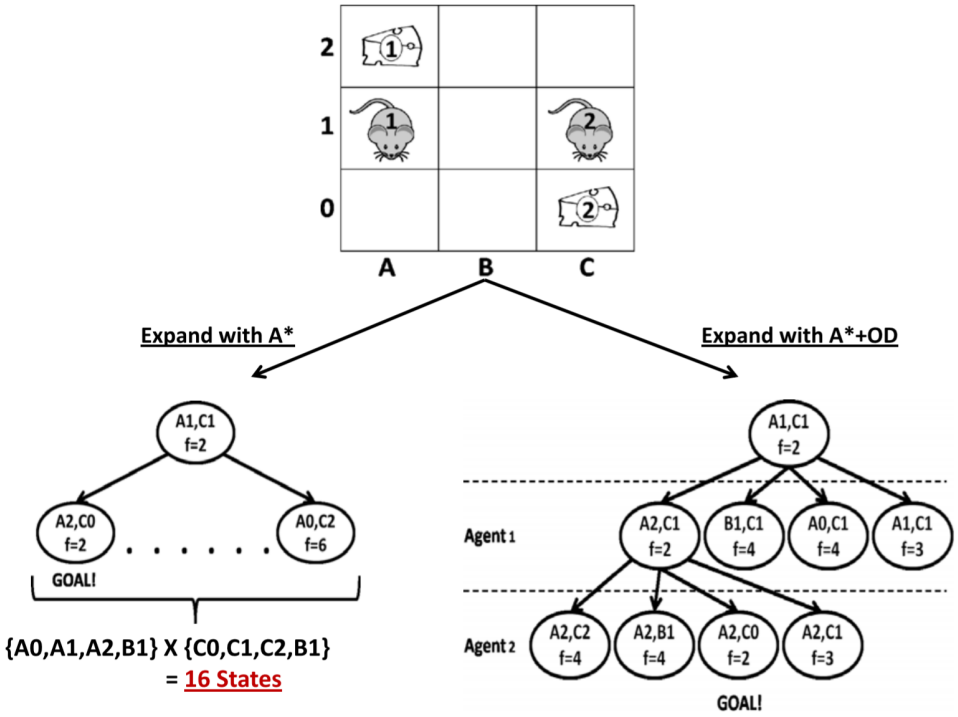



图 1.2 使用传统 A* 进行搜索和使用 OD 方法进行搜索并剪枝的效果对比[11]

求解表现。递归 M*（rM*）[15] 是 M* 的一个改进版本。rM* 试图识别冲突集中的智能体，将其解耦为没有冲突的智能体分组并递归地解决问题。rM* 中的 A* 算法也可以使用 OD 方法替代，进一步增加了可求解的智能体数量，该方法被称为 ODrM*[16]。

不同于基于 A* 搜索的算法，代价增长树搜索（Increasing Cost Tree Search，ICTS）[18] 算法并不直接在多智能体系统的状态空间内搜索，而是将搜索过程分为了上下两层搜索。ICTS 的上层搜索会给系统内每个智能体分配一个路径代价 $C_i$，下层搜索接受上层指定的路径代价集并使用 MDD[17] 验证是否存在一条代价等于 $C_i$ 的路径且不与其他智能体发生冲突，如图 1.3所示。如果当前所有智能体都满足其对应的路径代价 $C_i$，即说明找到了 MAPF 问题的可行且最优的解；如果不满足，则选取当前一个不满足其代价的智能体并指定 $C_i+1$ 为下一个代价增长树中的节点。ICTS 存在的一个问题是无法快速验证给定的

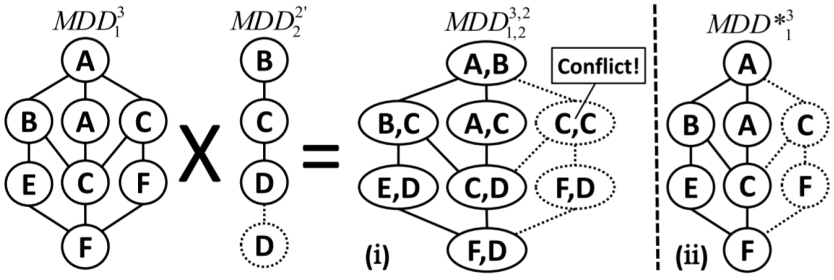



图 1.3 ICTS 算法中下层搜索使用 MDD[17] 验证是否存在不与其他智能体发生冲突的路径[18]

路径代价集是否存在解。一个有效的加速 ICTS 的方法[18] 是选取一对智能体并验证它们是否能在满足当前代价的情况下规划出互不冲突的路径，如果不行就快速地返回无解对 ICTS 进行剪枝。虽然该方法在实践中效果很好，但目前并没有理论指导选取哪些智能体对进行代价检查。

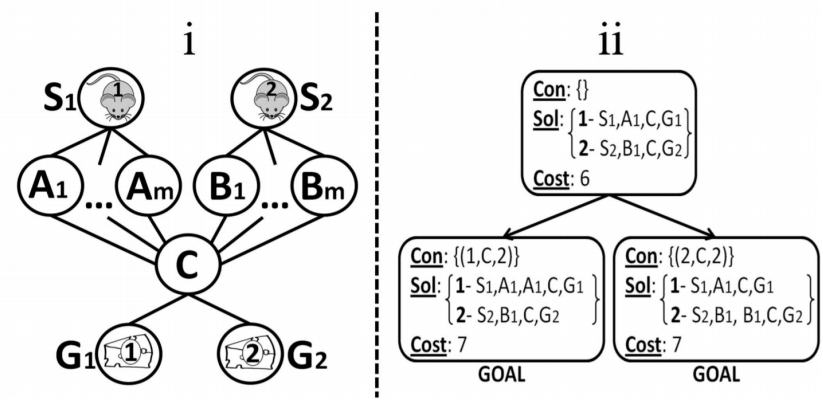



图 1.4 CBS 算法中二叉约束树构建示意图：左图是一个 MAPF 实例，右侧是基于该实例建立的二叉约束树，在经过一次冲突解决后，CBS 算法就找到了最优解。[19]

基于冲突的搜索算法（Conflict-Based Search，CBS）[19] 与 ICTS 一样，也是一个分层搜索的最优 MAPF 算法。CBS 中的核心概念是冲突约束条件 $(a_i, v, t)$，代表智能体 $a_i$ 被禁止在时间 $t$ 处于位置 $v$。CBS 的上层搜索在一棵二叉约束树（Binary Constraint Tree）上进行，树上的每个节点都包含一个约束条件集和与一个满足该约束集合的多智能体路径规划方案。接着算法检查该方案中是否存在一对智能体 $a_i$ 和 $a_j$ 的路径发生冲突。若存在，当前约束树节点就会生成两个子节点，并分别向两个子节点的约束条件集中添加 $(a_i, v, t)$ 和 $(a_j, v, t)$ 约束条件。拥有新约束的智能体会使用下层搜索重新规划一条符合约束的新路径，如图 1.4所示。CBS 的下层搜索则可以接受上层的约束集合并为智能体 $a_i$ 规划出一条满足约束的最优路径，现有的单智能体路径规划算法（如 A*）可以很容易整合为 CBS 的下层搜索。CBS 搜索算法采用最佳优先搜索，该算法优先扩展全局目标函数代价最小的节点，因此保证了第一个可行解就是全局最优解。在智能体路径耦合度不高的情况下，CBS 算法在较大场景中表现出了良好的求解效率。CBS 近年来也产生了很多变体：ICBS[20] 提出了主要冲突、半主要冲突和非主要冲突的概念，如图 1.5所示，通过智能地选择每次迭代中解决哪些冲突来加速约束树的求解速度；HCBS[21] 在上层搜索中也加入了启发式搜索的思想，以便更好的对二叉约束树进行剪枝；文献[22] 提出带权依赖图可以代替冲突图计算出更确切的启发值；文献[23] 为了提高规划成功率将搜索时间分成几部分，在每部分打乱智能体顺序进行重新规划；Li 等人[24] 提出在每次扩展时生成三个而非两个子节点，相比

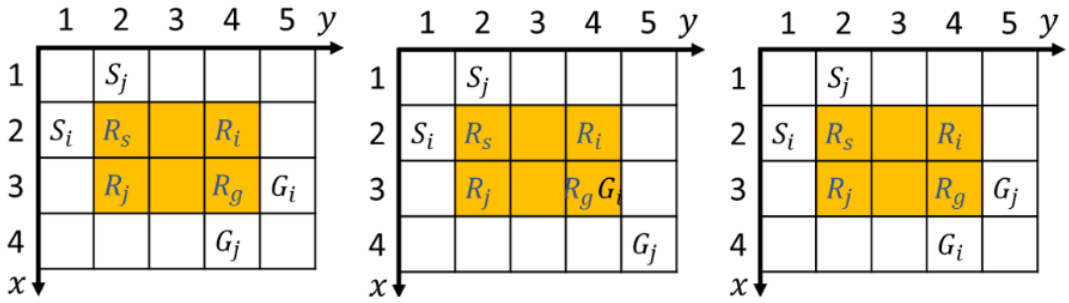



图 1.5 ICBS 算法中的冲突示意图，从左到右依次为：主要冲突、半主要冲突和非主要冲突[20]

原版 CBS 新增了智能体 $a_i$ 和 $a_j$ 都不能在 $t$ 时间处于位置 $v$ 的约束，从而保证了满足三个节点约束的解互不包含。

基于规约的方法是指将 MAPF 编译为一个其他标准问题，并使用为该标准问题设计的算法来进行求解。这类方法有时也被称为基于还原（Reduction-based）的 MAPF 求解器[25]。约束编程（Constraint Programming，CP）是一类基于规约求解 MAPF 问题的方法。这类方法将给定的问题建模为约束满足问题（CSP）或者约束优化问题（COP），然后使用通用的约束解算器来对问题进行求解。CSP 有一个特例称为布尔可满足性（SAT）问题，其可以使用专用的 SAT 求解器。将 MAPF 问题规约为标准问题的好处目前通用约束解算器已经非常高效，如现代 SAT 求解器可以解决超过一百万个变量的 SAT 问题。一个常见的 MAPF 规约方法[26] 是将问题拆分为两个子问题，首先找到最优的目标函数值，其次以该目标函数值设定区间，在给定目标函数区间内找到 MAPF 问题的一个可行解。Surynek 等人[27] 探索了五种不同的用 SAT 对 MAPF 进行建模的方法，研究了不同的建模选择是如

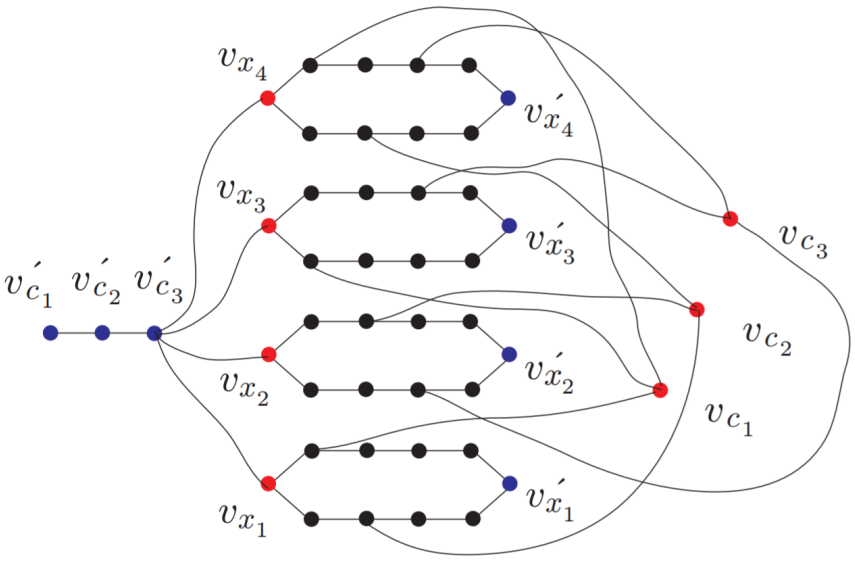



图 1.6 MAPF 问题规约为的整数规划问题示意图[7]

何影响 SAT 求解器的运行时间的。文献[28]使用 Picat[29]这种高级 CP 语言对 MAPF 的几种变体进行建模，用 Picat 语言编写的 CP 可以被自动编译并用通过 SAT 求解器或混合整数线性程序（MILP）求解器来解决[30]。然而，如何为一个给定的 MAPF 问题选择最佳模型和求解器，仍是一个待解决的问题。其他基于规约的方法还包括将 MAPF 规约成答案集编程（ASP）[31]，规约成 SAT 模拟理论（SMT）[32]，规约成旅行商问题（TSP）[33]以及规约成多商品流的整数规划问题[7]，并通过对应问题的求解器进行解决，如图 1.6所示。

2）**次优 MAPF 算法：**虽然关于最优 MAPF 算法的研究已经非常深入且取得了不俗的表现，但由于 MAPF 问题 NP 难的性质，当智能体数量不断增多、环境障碍物越发密集的情况下，最优 MAPF 算法无法在可接受的时间内求出结果。因此次优 MAPF 算法应运而生，这类算法通过牺牲结果的最优性保障来换取求解时间的下降。它们不再追求得到问题的最优解，而关注在有限的时间内给出一个问题的可行解并尽可能缩小可行解和最优解的差距。根据算法是否能保证求解质量不大于最优解的一定倍数，次优 MAPF 算法可以进一步分为有界次优和无界次优算法两种。

一个典型的无界次优 MAPF 算法被称为优先级规划（Prioritized Planning）[34,35]，该方法试图将 MAPF 问题与多智能体系统解耦，它首先给每个智能体分配一个优先级，再根据优先级对智能体依次进行规划，低优先级的智能体需要避让高优先级智能体规划好的路径。尽管优先级规划方法并不具备算法完整性保证，但是该方法实现简单、运行效率高，得到了研究者的广泛采用。文献[36,37]提出了几种方法来合理设计系统中智能体的优先级。

分布式的 MAPF 求解器通常都是无界次优的，由于没有中央服务器统一分析环境并为每一个智能体计算路径，分布式 MAPF 算法天然丧失了最优性保证。一个出色的分布式 MAPF 算法是分层协调 A* 算法（Hierarchy Cooperative A*，HCA*）[34]。该算法依次求解

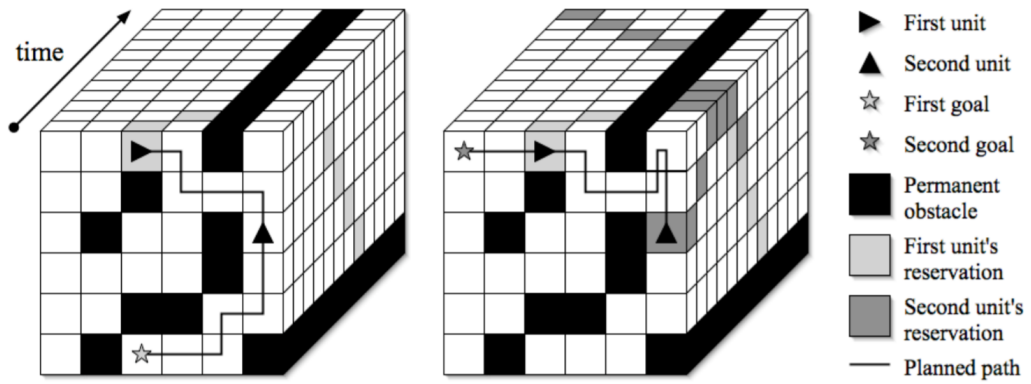



图 1.7 HCA* 算法示意：左图中第一个智能体搜索路径并将其标记到保留表中。右图中第二个智能体在考虑现有保留表的同时搜索自身路径，最后将自身路径也添加到保留表中。[34]

智能体的路径，并将已经求出的路径加入全局保留表（Reservation Table）中。在后续智能体进行规划时，保留表中的路径被标记为在时空上不可通行的，从而解决智能体之间的冲突，如图1.7所示。窗口分层合作 A* 算法（WHCA*）[34] 融合了优先级规划的思想，让智能体在一个优先级规划框架内交替进行规划和执行。在 WHCA* 算法中，智能体只在未来 $x$ 个时间步（被称为一个时间窗口）内避免冲突。另一种分布式求解器则是基于规则的规划算法，这类算法具备完整性保证。Push-and-Swap 算法[38,39] 及其拓展的 Push-and-Rotate 算法[40] 被证明对于所有至少有两个空位置的环境中 MAPF 问题都是完整的。该算法的工作原理是执行一组宏操作器，将一个代理推向其目标位置并交换两个代理的位置。文献[41] 提出了一种基于通讯图的协调规划算法，将通讯协调区域缩小到了 2 步大小的区域内从而减少了通讯量，增加了分布式求解器的应用场景，如图1.8所示。

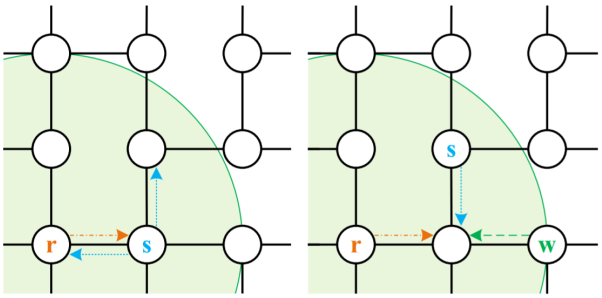



图 1.8 半径为 2 倍边长的通讯协调区域内智能体的协作规划示意[41]

有界次优的 MAPF 算法，有时也被称为近似最优算法，这类算法可以保证返回解的目标函数成本不大于最优解的 $1+\epsilon$ 倍。有界次优算法为使用者在运行时间和求解质量之间提供了一个可控的权衡。加权 A*（Weighted A*）[42] 就是一个基于 A* 的有界次优算法，它使用 $(1+\epsilon)h$ 作为启发式函数在每次迭代中对节点成本进行计算。所有基于 A* 的 MAPF 算法都可以应用这个方法并获得 $1+\epsilon$ 成本的有界次优保证。这个方法同样也可以适用于 M* 算法，被称为膨胀 M*（Inflated M*）算法[15]。CBS 算法同样可以扩展为一系列有界次优 MAPF 算法。增强型 CBS（Enhanced CBS，ECBS）[43] 在上层搜索和下层搜索中都引入了次优性。ECBS 的下层搜索可以使任意近似次优的单智能体路径规划算法，包括前述的 Weighted A*，明确估计搜索（Explicit Estimation Search）[44] 和动态势能搜索（Dynamic Potential Search）[45]。ECBS 的上层搜索使用了一种名为焦点搜索（Focal Search）[46] 的技术来引入次优性。焦点搜索在每次迭代中都从一个 FOCAL 的节点子集中寻找下一个要扩展的节点，FOCAL 集合包含 Open 集合中所有不大于 $(1+\epsilon)$ 倍最优路径成本的节点，而从 FOCAL 中则考虑当前冲突最少的节点进行扩展。通过二级启发式函数，ECBS 算法

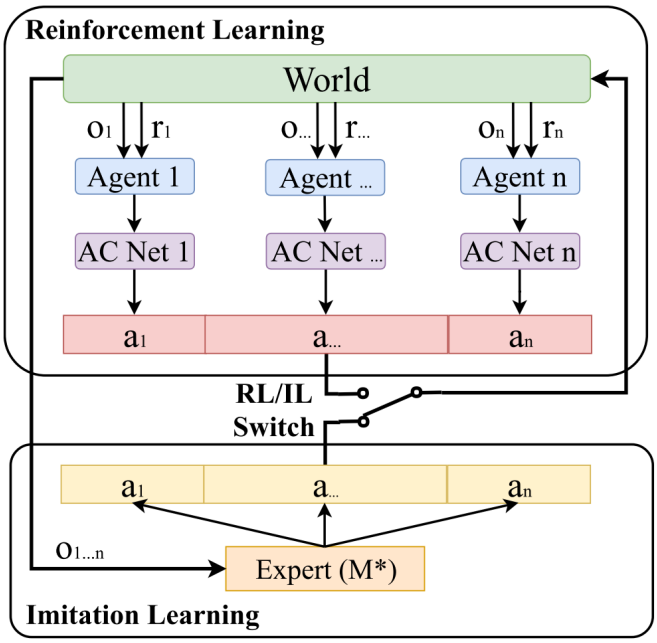



图 1.9 PRIMAL 算法中强化学习和模仿学习的混合网络结构[2]

在保证有界次优性的同时极大提高了搜索效率。文献[47]在 ECBS 的基础上引入了高速路（Highway）的概念，在高速路上的智能体只能单向行驶，进一步缩短了算法运行时间。CBSH 算法[25]通过引入图 1.5中主要冲突的概念进一步改进了 ECBS，是目前先进的次优 MAPF 算法之一。

此外，随着近年来强化学习的兴起，一些国内外的研究者也开始基于强化学习进行多智能体路径规划算法研究。文献[48]采用无模型的在线 Q-learning 的方法，让系统内多个智能体不断与环境交互并得到反馈，把评估结果作为反馈不断优化策略，最终完成未知环境下的 MAPF 任务。文献[49]则先在环境中构建一个人工势场，用每个状态的势能值来表示该状态可以获得的最大累计回报，并将该先验知识融入分层强化学习的框架加快算法的收敛速度。PRIMAL 算法[2]则结合了强化学习和模仿学习进行分布式路径规划的示教，这个框架在训练期间引入了专家 MAPF 规划器的演示，训练后的策略自然地扩展到不同的系统规模和环境上，其网络结构如图 1.9所示。文献[50]将强化学习的方法应用在多智能体编队规划问题上，通过使用分层强化学习框架并引入一种智能体间通讯方式，使得智能体在完成自身规划任务的同时表现出了互相协作的行为。

3）**对经典 MAPF 问题的扩展：**上述大部分方法都是针对经典 MAPF（Classic MAPF）问题研究的，文献[51]中提出了一个公开基准测试集来对各种算法进行统一量化测试，如图 1.10所示。经典 MAPF 问题假设：智能体可以不考虑运动学约束抽象为一个质点、时间是离散的而非连续的以及智能体总在一个无向图或栅格地图中进行规划。但这些假设在

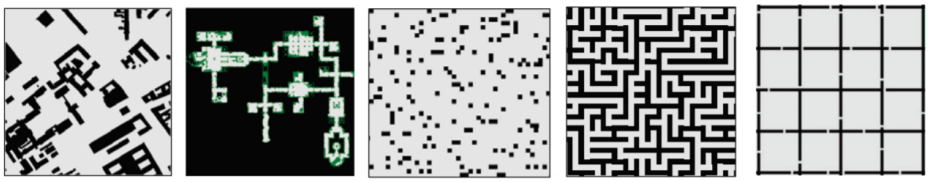



图 1.10 文献中的公开基准测试集由 24 张地图组成，这些地图来自（a）真实城市的地图，（b）视频游戏《龙腾世纪》和《龙腾世纪 2》，（c）有和没有随机障碍物的开放网格，（d）类似迷宫的栅格地图，以及（e）类似房间的栅格地图。[51]

现实世界的多智能体场景中并不一定成立，一些研究人员已经开始关注实际的多机器人系统并在他们的最新研究中取消了上述部分假设。

安全区间路径规划（Safe Interval Path Planning，SIPP）[52] 算法是一个非离散时间的单智能体路径规划算法。SIPP 识别智能体可以占据每个位置的安全时间区间，将位置和安全区间融合为一个大的节点状态并进行 A* 搜索。文献[53] 在 CBS 的框架中应用了 SIPP 算法来解决连续时间的 MAPF 问题。文献[54] 考虑了智能体移动的速度限制并保证了智能体之间必要的安全距离。文献[55] 在扩大搜索节点的同时为一个智能体添加了多个约束，从而扩展了 CBS 框架并能够为占据超过一个栅格的大型智能体提供路径规划。文献[56] 提出了一个针对异构多智能体系统的 MAPF 求解器，它适用于包含不同大小智能体的系统并考虑了机器人实际执行路径时的时间延迟。使用基于优化而非搜索的方法可以在连续空间而非栅格离散空间内为多智能体系统进行路径规划。文献[57] 提出了一种基于优化的异构四旋翼无人机轨迹规划算法，但该方法在二维笛卡尔平面而不是三维物理空间内进行规划，并存在优化时间长的问题。文献[58] 提出了一种更有效的多四旋翼无人机轨迹规划方法。该方法将问题解耦，首先使用 ECBS 算法进行粗搜索，将搜索结果提供给后端轨迹优化模块进行进一步精细规划，如图 1.11所示。该方法优化效率较高，在障碍物密集的环境中也有着很好的表现。文献[59] 将这种方法移植到了具有非完整约束的多移动机器人系统中，也取得了很好的效果。然而，基于优化的方法较为耗时，往往无法在短时间内找到可行的解决方案。

同时，经典 MAPF 问题是一次性的离线规划问题。但在一些应用场景中，存在着一连串的 MAPF 问题等待被顺序解决。这些问题也被称为在线 MAPF 问题。文献[60] 研究仓储场景下多智能体系统中智能体数量不变而寻路任务会不断涌入的问题，文献[61] 则研究了

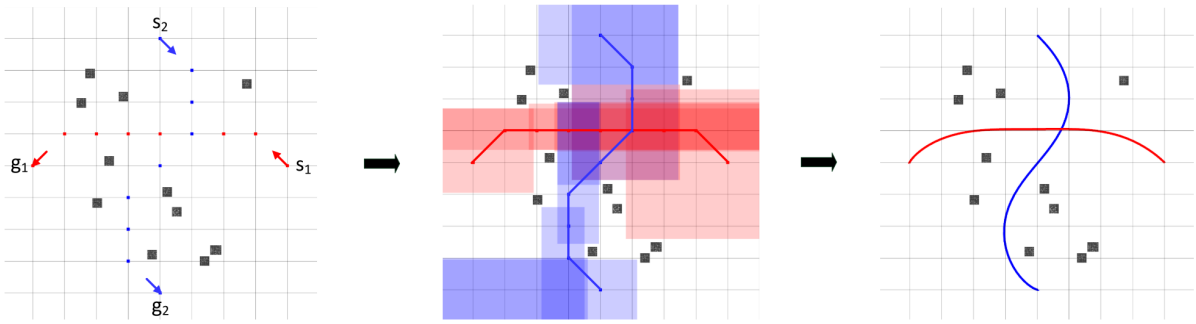



图 1.11 文献中的算法流程：首先使用 CBS 算法进行粗略搜索，然后计算出每个智能体可以通行的安全走廊区域，最后使用基于优化的方法进行轨迹规划。[59]

新智能体会随着时间不断加入系统但是每个智能体都只有一个给定寻路任务的问题。文献[62]提出了滚动时域碰撞解决算法，它将在线规划问题分解为一连串的区间内的 MAPF 实例，求解器只解决一定时间范围内的智能体路径之间的碰撞，而忽略时间范围外的碰撞。此外，还有一类问题被称为多智能体取货和交货（Multi-Agent Pickup-and-Delivery，MAPD）[63]问题。这个问题中，算法不仅需要在没有冲突的情况下为智能体规划路径，还需要分配哪个智能体应该去执行哪个送货任务、前往哪个目的地。SIPPwRT[64]是一个具有代表性的 MAPD 方法，该方法将令牌传递（Token Passing）算法与 SIPP 算法结合起来，可以为不同规模的非完整约束机器人规划路径。MAPF-POST[54]算法是一个鲁棒的多智能体路径执行框架，它接受 MAPF 问题的解决方案作为输入，并对其进行调整以考虑智能体自身的安全约束和运动学约束。但 MAPF-POST 无法对解决方案的执行质量提供任何保证。文献[65]提出了一种基于行动依赖图（Action Dependency Graph，ADG）的多智能体路径执行框架，其能够使用比 MAPF-POST 更少的信息通信量来维持多智能体系统的稳健持久运行。

## 1.3 本文研究内容

前述相关研究中提到的传统 MAPF 求解器在离散空间内求解，只能规划出粗糙的分段线性路径，还需要额外的控制器或路径执行方法来对路径进行跟踪，因此并不能保证在连续工作空间内的最优性。而针对扩展 MAPF 问题的算法较为零散，每个研究关注的具体问题也不尽相同，其在现实工业多机器人系统中具体的应用方式亟待探索。此外，如何抵御潜在的工控网络攻击并识别系统内被恶意控制的智能体，保障多智能体系统安全鲁棒地执行既定路径仍然是一个开放性问题。

针对上述问题，本文提出了一个可应用于工业场景的全流程多智能体路径规划解决

方案。本文首先给出了经典 MAPF 问题的数学定义并介绍了基于冲突搜索算法的基本流程。随后基于该算法，提出了一种在连续工作空间内满足阿克曼转向几何的多智能体路径规划算法。接着本文将该方法进一步推广到异构多智能体系统中，并使用滚动时域的碰撞解决算法赋予了其进行在线路径规划的能力。最后本文为了保证多智能体系统能按照规划方案安全鲁棒地完成任务，提出了一个基于行动依赖图的 MAPF 处理框架。本文主要研究内容可以分为以下三部分：

1. 提出了一种针对阿克曼模型的基于冲突搜索算法，该求解器采用了分层搜索框架：在上层搜索中，本文提出了车身约束树的概念，只需检测规划方案中路径间是否存在车身冲突，而不必考虑具体的运动学约束；而在下层搜索中，本文提出了时空混合 A* 算法来进行单智能体路径规划，该方法具备连续工作空间内为大量阿克曼运动模型机器人规划安全避障路径的能力，不仅规划方案质量远超基线算法。本文还提出了一个上述方法的顺序规划版本，其牺牲了部分求解质量进一步减少了算法搜索时间。本文在仿真环境中开展了实验，结果表明提出的方法可以很好地扩展到包含 100 个以上的智能体的系统并保持较高的规划成功率和较短的搜索时间。

2. 提出了一种适用于异构多智能体系统的在线路径规划方法。在前述连续空间 MAPF 求解器的基础上，通过对下层单智能体路径规划算法的一般化，将方法进一步扩展到更接近于实际工业场景的异构多智能体系统中。同时，本文将滚动时域碰撞解决算法加入基于冲突搜索框架中，将需要解决的在线 MAPF 问题拆分为一连串的时间窗口 MAPF 实例，通过交替进行路径规划和路径执行解决了在线路径规划问题中存在寻路任务流的问题并降低了智能体的闲置时间。本文对异构多智能体系统进行了仿真实验并与现有先进方法进行了对比，结果表明本文方法可以将在线 MAPF 规划器的速度提升六倍而几乎不影响规划方案的质量，同时输出的规划方案质量也显著优于对比方法。

3. 提出了一种基于行动依赖图的多智能体路径规划方案后处理框架。该框架利用求解器生成的规划结果来分析智能体间移动的优先级关系，仅需智能体和规划器之间进行少量通信，即可识别系统中被恶意控制的异常智能体。通过该主动检测机制，本文保障了多智能体系统在长时间内安全鲁棒地执行规划方案。本文在 Gazebo 仿真环境中进行了实验，模拟了后处理框架对智能体被恶意攻击并劫持情况的处理。最后本文将该框架与前述规划器整合，形成了一套全流程多智能体路径规划解决方案。在室内场景和室外场景下分别进行了实物实验。在不同的场景和各异的多机器人系统中都验证了该解决方案的实际可行性和执行鲁棒性。

## 1.4 本文结构安排

围绕上述研究内容，本文结构分为如下六章：

第一章介绍了本文的研究背景及意义，阐述了多智能体路径规划问题的概念及其在机器人和工业控制领域的重要研究意义及价值。在此基础上综述了国内外对于该问题的研究现状及存在的问题。最后引出本文的主要研究内容和结构安排。

第二章给出了经典多智能体路径规划问题的数学定义，随后详细介绍了本文相关的两个基础算法，基于冲突搜索搜索算法和混合 A* 搜索算法，为后续章节做准备。

第三章首先在同构多智能体系统中对引入了运动学约束的 MAPF 问题进行研究，并选取阿克曼转向几何作为具体的运动学模型。本章详细介绍了针对阿克曼模型的基于冲突搜索算法及其分层搜索框架：包括上层的车身约束树和下层时空混合 A* 搜索。随后本章介绍了该方法的顺序规划版本。最后，本章提出了一个适用于连续空间内 MAPF 问题的基准测试集，并基于该测试集开展了性能测试和对比试验。

第四章对问题进一步扩展，面向实际工业场景研究异构多智能体系统中的在线 MAPF 问题。本章首先介绍了问题的数学定义，并给出了几个常用的评价指标。随后本章将该问题拆分为异构智能体系统规划和在线多智能体规划两个子问题分别进行解决，并详细阐释了通用模型的单智能体规划器和滚动时域碰撞解决算法。最后，本章进行了异构多智能体系统内的 MAPF 实验以验证方法的可行性并与现有先进方法进行了对比。

第五章研究系统在规划路径后的运行阶段对于潜在网络攻击的抵御能力。本章详细介绍了基于行动依赖图的多智能体路径规划后处理框架，并阐释了该方法如何确保系统安全可靠地执行既定路径。本章在 Gazebo 三维仿真环境中测试了该后处理框架的有效性以及对于异常情况的鲁棒性。最后本章将后处理方法与前两章提出的 MAPF 规划方法进行整合，形成全流程多智能体路径规划解决方案，并在室内场景和室外场景下分别进行了实物实验。

第六章为总结和展望，对本文进行整体性的总结和反思，总结了本文的贡献点并展望了未来改进的方向。

## 1.5 本章小结

本章介绍了本论文的研究背景及意义，阐述了多智能体路径规划问题的概念和研究意义。在此基础上，综述了多智能体路径规划问题在机器人和工业控制领域的国内外研究现状，以及待解决的问题。同时，讨论了不同类型的 MAPF 算法思路，比较了不同算法的特点及优劣。最后阐明了本文的主要研究内容并对本文的章节安排进行了简要的说明。

# 2 相关背景知识介绍

## 2.1 经典 MAPF 问题定义

目前学界对于多智能体路径规划的研究大多是针对经典 MAPF 问题（Classical MAPF）进行的[51]，经典 MAPF 问题有如下假设：智能体可以不考虑运动学约束而抽象为一个质点；时间是离散的而非连续的；以及智能体系统在一个无向图中进行规划。经典 MAPF 问题的数学定义如下:

考虑一个有 $k$ 个智能体的系统，经典 MAPF 问题被定义为一个元组 $\langle G, s, t\rangle$，其中 $G = (V, E)$ 是一个无向图，$s : [1, \ldots, k] \to V$ 将每一个智能体映射到一个起点，$t : [1, \ldots, k] \to V$ 将每一个智能体映射到一个终点。时间被假设为离散的，在每一个时间步中，每个智能体都位于图 $G$ 的一个位置并能进行一个动作（Action）。动作被定义为一个函数 $a : V \to V$，例如 $a(v) = v'$ 意味着如果一个智能体当前时间步位于位置 $v$ 并执行了动作 $a$，那么它在下一个时间步中就会移动到位置 $v'$。在每个时间步中，智能体都有两种类型的动作：等待和移动。等待动作意味着智能体会在当前位置继续等待一个时间步。移动动作则意味着智能体会从其当前的位置 $v$ 移动到图中相邻的位置 $v'$，即 $(v, v') \in E$。对于一个关于智能体 $i$ 的动作序列 $\pi_i = (a_1, \ldots, a_n)$，算法用 $\pi_i[x]$ 来表示智能体从起始位置 $s(i)$ 开始执行前 $x$ 个动作后的位置。正式来说，$\pi_i[x] = a_x(a_{x-1}(\cdots a_1(s(i))))$。当 $\pi_i$ 中的每个动作都恰好在第 $x$ 个时间步被执行，就认为 $\pi_i$ 是智能体 $a_i$ 的一个单智能体路径。一个 MAPF 问题的规划方案（Solution）是一个包含有所有 $k$ 个智能体的单智能体路径集合。图 2.1展示了一个包含 4 个智能体的经典 MAPF 问题实例。在该实例中，无向图 $G$ 被表示为一个栅格地图，每个位置都与其上下左右四个位置相邻，相邻位置之间的边 $e \in E$ 并没有绘制在图上。图中，深灰色栅格为静态障碍物不可通行，相应的，这些位置不被包含在无向图 $G$ 的顶点集合 $V$ 中，与周围位置也不存在连接边 $e$。图中标有蓝色圆形的位置为各个智能体的起始位置 $s_i$，而标有绿色八边形的位置则是各个智能体的终点位置 $t_i$。

MAPF 的一个可行解定义为系统中的每一个智能体按照规划方案执行不会发生碰撞。为了找到可行解，MAPF 求解器在规划过程中引入了冲突的概念，如果一个规划方案中任意两个单智能体路径之间都没有冲突，则该规划方案就是问题的一个可行解。关于经典 MAPF 问题的文献中包含了一些冲突的不同定义，下文列出了几种常见冲突的定义，其中

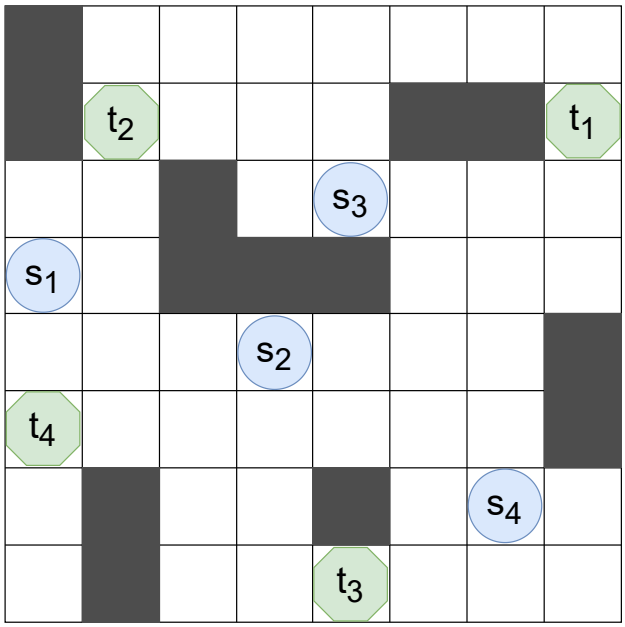


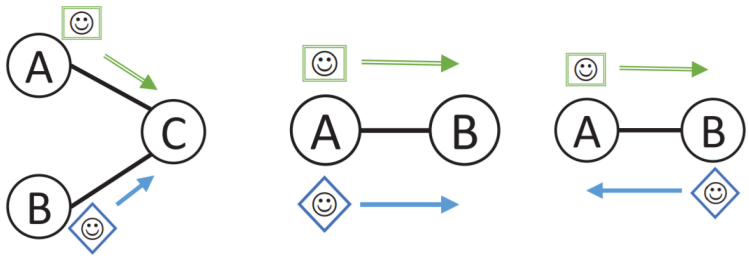

图 2.1 一个经典 MAPF 问题实例

$\pi_i$ 与 $\pi_j$ 是，智能体 $i$ 与智能体 $j$ 的路径:

- 顶点冲突（Vertex conflict）。如果根据路径，智能体 $i$ 与智能体 $j$ 会在同一时间步中占据同一顶点位置，那么 $\pi_i$ 与 $\pi_j$ 之间就会出现顶点冲突。形式上，如果存在一个时间步 $x$，使 $\pi_i[x] = \pi_j[x]$，则称 $\pi_i$ 与 $\pi_j$ 之间存在顶点冲突。
- 边冲突（Edge conflict）。如果根据路径，智能体 $i$ 与智能体 $j$ 会在同一时间步以相同的方向穿越相同的边，那么 $\pi_i$ 与 $\pi_j$ 之间就会出现边冲突。形式上，如果存在一个时间步 $x$，使 $\pi_i[x] = \pi_j[x]$ 且 $\pi_i[x+1] = \pi_j[x+1]$，则称 $\pi_i$ 与 $\pi_j$ 之间存在边冲突。
- 调换冲突（Swapping conflict）。如果智能体 $i$ 与智能体 $j$ 会在一个时间步骤中交换位置，那么 $\pi_i$ 与 $\pi_j$ 之间就会发生调换冲突。形式上，如果存在一个时间步 $x$，使 $\pi_i[x+1] = \pi_j[x]$ 且 $\pi_j[x+1] = \pi_i[x]$，则称 $\pi_i$ 与 $\pi_j$ 之间存在调换换冲突。这种调换在一些 MAPF 文献中也会被归类为边冲突。
- 此外还有跟随冲突（Following conflict）、死锁冲突（Cycle conflict）等。

图 2.2展示了不同类型的冲突。需要注意的是，上述定义的冲突并不能包含 MAPF 问题中所有可能发生的冲突。

在 MAPF 问题的规划方案中，不同的智能体会在不同的时间步到达对应的目标位置。因此，定义一个智能体在到达目标位置后的行为是非常重要的。在不同的研究文献中，对于智能体到达目标点后的行为有两种常见的假设：留在目标处和从目标处消失。留在目标处假设智能体会在其目标处静止等待，直到系统中所有智能体都到达目标。即这个等待的智能体还是会与任何在它到达目标后通过该目标位置的智能体路径产生顶点冲突。在该




(a) 顶点冲突 (b) 边冲突 (c) 调换冲突

图 2.2 常见冲突类型的示意图[51]

假设下，如果对于任意时间步 $t \geq |\pi_i|$，都有 $\pi_j[t] = \pi_i[|\pi_i|]$。而从目标处消失的假设则意味着当智能体到达目标位置时，它会立即消失。因此该智能体在到达目标位置后不会与其他任何智能体发生冲突。从现实意义出发，大部分对经典 MAPF 问题的研究会假设智能体到达目标后会留在目标处。

显然地，在 MAPF 问题中，一些可行解会比另一些可行解表现更好。为了对不同的可行解进行定量分析，经典 MAPF 问题一般会使用两个目标函数来评价可行解：总运行时间（Makespan）和总路径成本（Flowtime）。

- 总运行时间被定义为系统中所有智能体都到达目标位置所需要的时间步。对于一个可行解 $\pi = \{\pi_1, \ldots, \pi_k\}$，总运行时间被定义为 $\max_{1 \leq i \leq k} |\pi_i|$。
- 总路径成本被定义为所有智能体到达其目标位置所需的时间步之和。对于一个可行解 $\pi = \{\pi_1, \ldots, \pi_k\}$，总路径成本被定义为 $\sum_{1 \leq i \leq k} |\pi_i|$。

当然，这两个目标函数并不是经典 MAPF 问题仅有的目标函数，但确实是目前使用范围最广的目标函数。基于编译的 MAPF 求解器已经广泛使用了总运行时间作为评价指标，而大多数基于搜索的 MAPF 算法会使用总路径成本。

## 2.2 相关算法介绍

本节着重介绍两个经典算法，基于冲突搜索算法（Conflict-Based Search，CBS）[19] 和混合 A* 搜索算法（Hybrid-State A*）[66]。前者是一个目前非常流行的基于搜索的最优多智能体路径规划求解器，该算法也衍生出了很多最优和次优 MAPF 研究。后者是一个单智能体路径规划器，它解决了传统 A* 搜索必须要在离散的栅格空间内进行搜索的限制，将智能体的运动学约束融合进了传统 A* 算法中，在继承启发式搜索运行速度快优势的同时也保证了路径满足智能体非完整性约束。

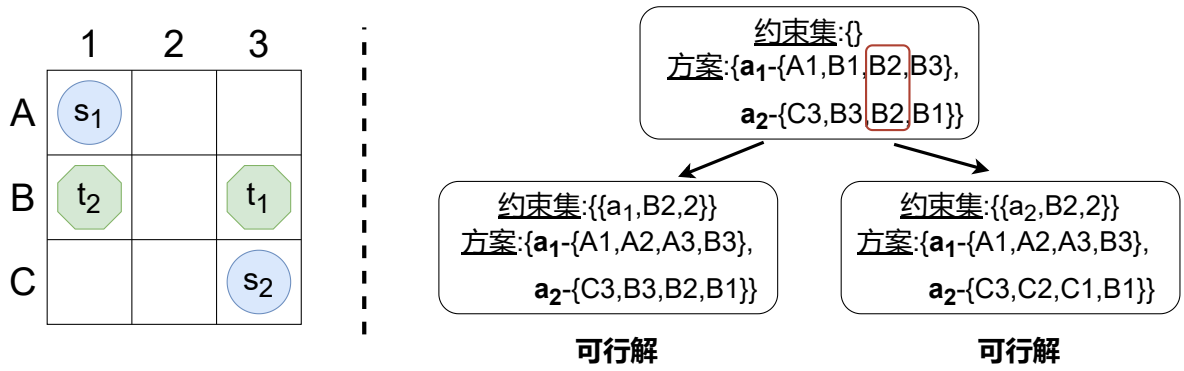



图 2.3 CBS 算法求解两个智能体的 MAPF 问题实例

### 2.2.1 基于冲突搜索算法

在 CBS 算法中有两个核心概念：约束和冲突。一个约束被定义为 $(a_i, v, t)$，意味着智能体 $a_i$ 被禁止在时间步 $t$ 占据无向图 $G = (V, E)$ 中的顶点 $v$，$v \in V$。智能体 $a_i$ 的一条可行路径应当满足所有关于 $a_i$ 的约束。CBS 算法中的冲突被定义为 $(a_i, a_j, v, t)$，这表示智能体 $a_i$ 和 $a_j$ 在时间步 $t$ 同时占据了无向图中顶点 $v$。如果一个规划方案中的所有路径之间都不存在冲突，则这个规划方案就是一个可行解。

CBS 算法的关键思想是通过增加路径约束并找到符合这些约束条件的规划方案，如果这个方案中仍然有路径存在冲突，那么 CBS 就会增加新的约束来解决该冲突。CBS 算法是一个分层搜索算法，上层搜索负责检测规划方案中的冲突并为相应的智能体增加约束条件，而下层搜索负责为单个智能体规划符合约束条件的新路径。下面本文详细介绍该算法的每个部分。

CBS 的上层搜索在一棵约束树（Constraint Tree，CT）上进行，约束树是一颗二叉树。树上的每个节点 $N$ 中包括：

- 一个约束集（$N.constraints$），集合中的每组约束都属于系统中的一个智能体。约束树根节点中的约束集为空集。约束树中一个节点的子节点会继承父节点的所有约束，同时为其中一个智能体添加一个新的约束。
- 一个规划方案（$N.solution$），规划方案是一个包含系统中所有 $k$ 个智能体的路径集。规划方案中任意智能体 $a_i$ 的路径都应该满足约束集中关于 $a_i$ 的所有约束条件。
- 规划方案的成本（$N.cost$），当前规划方案的目标函数值，即规划方案的总运行时间或总路径成本。

当约束树中的节点 $N$ 的规划方案是可行的时，节点 $N$ 就是一个目标节点。CBS 的上层搜索在约束树上执行最佳优先搜索（Best-first search），将当前可扩展节点按其成本排序，这种做法保证了算法找到第一个可行的规划方案就是全局最优解。

对于约束树中的一个节点 $N$，首先 CBS 算法会调用下层搜索为每个智能体 $a_i$ 返回一条满足所有 $N.constraints$ 中关于 $a_i$ 约束的最短路径。当为每个智能体都找到了一条可行路径后，算法会检测智能体的路径之间是否存在冲突。具体来说，CBS 算法会迭代所有时间步来验证每个时间步中所有智能体的位置互不冲突。如果规划方案中不存在某两个智能体的路径间存在顶冲突，该约束树节点 $N$ 就是目标节点，算法会返回节点 $N$ 的规划方案 $N.solution$ 作为问题的最优解。反之，如果在检测中发现两个或更多的智能体之间存在冲突 $C = (a_i, a_j, v, t)$，检测过程就会终止，同时该节点被认定为非目标节点。

对于一个非目标节点 $N$，可以知道其规划方案 $N.solution$ 中至少存在一个冲突。当规划方案中冲突数大于一个时，算法会选取时间 $t$ 最小的冲突作为当前节点需要解决的冲突 $C_n = (a_i, a_j, v, t)$，而将其余冲突留给冲突树更深层次进行解决。显然，在一个可行解中，对于冲突 $C_n$ 最多只允许 $a_i$ 和 $a_j$ 中的一个智能体在 $t$ 时间步占据位置 $v$。因此，约束 $(a_i, v, t)$ 或 $(a_j, v, t)$ 中至少有一个需要被添加到节点的约束集 $N.constraints$ 中。为了保证最优性，这两种可能性都被算法考虑。节点 $N$ 被将扩展为两个子节点，其中左子节点通过添加 $(a_i, v, t)$ 约束条件从而处理冲突，而右子节点则添加条件 $(a_j, v, t)$ 进行处理。此外，在实际算法实现中，约束树节点 $N$ 不必保存其所有的约束条件，它可以只保存其新增的约束，并通过遍历从 $N$ 到根节点的路径来提取剩余约束。同样的，除了根节点外，下层搜索只需对与新增约束相关的智能体 $a_i$（或 $a_j$）进行，其他智能体的路径应当保持不变，因为它们并没有被添加新的约束。

CBS 算法的下层搜索则需要为给定的智能体 $a_i$ 在无向图中规划一条满足 $a_i$ 所有相关约束的最优路径。由于上层搜索已经通过冲突和约束将智能体路径互相解耦了，所以 CBS 的下层搜索可以完全忽略除 $a_i$ 以外的其他智能体。由于约束条件 $(a_i, v, t)$ 具有时间和空间两个维度，CBS 算法的下层搜索采用时空 A* 搜索[34] 来实现。时空 A* 搜索使用复合状态 $(v, t)$ 来表示一个搜索状态，$v \in V$ 为当前位置，$t$ 为该状态的时刻。如果当前节点存在约束 $(a_i, v, t)$，则说明当前状态 $(v, t)$ 是一个非法状态，下层搜索将不会继续扩展该状态。同时 CBS 算法在下层使用了重复状态检测和修剪（Duplicate states detection and pruning）技术来加快下层搜索速度。

一个简单的包含两个智能体的 MAPF 问题实例如图2.3所示，智能体 $a_1$ 需要从左上角的位置 A1 移动到右边的 B3 位置，而智能体 $a_2$ 则需要从右下角的位置 C3 移动到左侧位置 B1。算法初始时创建一个包含一个空约束集的根节点 *Root*，结点初始化完成后被放入到开放列表（OPEN）中。然后算法调用下层搜索为两个智能体分别规划一条不考虑任何约束的最短路径。假设下层搜索为 $a_1$ 返回的路径为 {A1,B1,B2,B3}，为 $a_2$ 返回路径

**算法** 1: 基于冲突搜索

Input: A MAPF instance

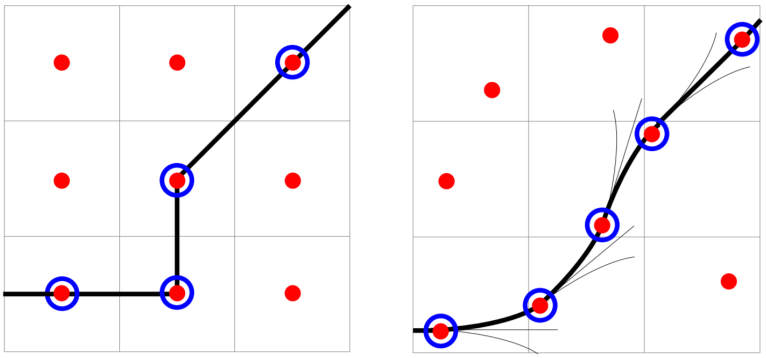
$Root.constraints \leftarrow \emptyset$;
$Root.path \leftarrow$ find path by invoking low-level search, for each $a_i$;
Insert $Root$ to OPEN while OPEN $\neq \emptyset$ do
    $N \leftarrow$ Best node $\in$ OPEN;
    OPEN pop out $N$;
    $C \leftarrow$ search for first conflict $(a_i, a_j, v, t)$ in $N.solution$;
    if $C$ does not exist then
        return $N.solution$;
    end
    foreach $a_i$ appears in $C$ do
        $N' \leftarrow$ New node;
        $N'.constraints \leftarrow N.constraints \cup (a_i, v, t)$;
        $N'.solution \leftarrow N.solution$;
        $N'.solution[i] \leftarrow$ invoke low-level search for $a_i$;
        $N'.cost \leftarrow cost(N'.solution)$;
        Insert $N'$ to OPEN;
    end
end
return $\emptyset$;

{C3,B3,B2,B1}。随后算法检测智能体的路径之间是否存在冲突，发现 $a_1$ 与 $a_2$ 在 $t = 2$ 时刻在 B2 位置发生冲突。于是算法构建两个约束 $(a_1, B2, 2)$ 和 $(a_2, B2, 2)$，并将其分别放入根节点 *Root* 的两个子节点中，并加入开放列表中。对于包含约束 $(a_1, B2, 2)$ 的子节点，下层算法重新规划并返回满足该约束的最短路径 {A1,A2,A3,B3}；对于包含约束 $(a_2, B2, 2)$ 的子节点，下层算法重新规划并返回满足该约束的最短路径 {C3,C2,C1,B1}。这两个规划方案都是满足问题的可行解。CBS 算法的伪代码详见算法 1。

### 2.2.2 混合 A* 搜索算法

混合 A* 搜索与普通 A* 搜索的最大区别在于普通 A* 规划的路径只适用于完整约束的机器人，而混合 A* 算法则对 A* 进行了改进，在离散栅格中也可以保存连续工作空间

的数据，并使得规划路径适用于非完整约束的机器人。混合 A* 算法主要对原始 A* 算法进行了子节点扩展、启发式函数以及分析性扩展三方面的改进，下面本文逐一进行介绍。



(a) A* 算法扩展时只访问对应于栅格中心的状态　　(b) 混合 A* 算法基于动力学约束进行连续空间上的扩展

图 2.4　A* 搜索和混合 A* 算法子节点扩展对比[66]

混合 A* 算法使用连续空间内的子节点扩展方式，智能体的状态表示为 $(x, y, \theta)$，其中 $(x, y)$ 为智能体位置，$\theta$ 为智能体偏航角。子节点扩展方式如图2.4所示，智能体每次向前扩展的方向有三个，分别为直行、左转及右转。其中子节点具体扩展位置会根据智能体最小转弯半径以及节点扩展长度共同决定。通过该扩展方式，能够使得每段规划的路径都满足智能体的动力学约束。

混合 A* 算法由两个启发式函数共同指导来进行搜索。第一个启发式函数忽略环境中的障碍物而考虑智能体的运动学约束。假设当前状态为 $(x, y, \theta)$，目标状态为 $(x_g, y_g, \theta_g)$，则该启发式函数是从当前状态在满足动力学约束的情况下前往目标状态的最短路径长度。混合 A* 算法使用阿克曼转向几何作为运动学约束，因此该启发式函数可以通过 Reeds-Shepp 曲线[67] 进行计算。实验结果显示，相比于直接使用二维欧氏距离作为启发式函数，使用考虑运动学约束的 Reeds-Shepp 曲线来进行计算可以减少一个数量级的扩展节点数量。同时，算法可以事先离线计算到达不同目标状态的 Reeds-Shepp 曲线长度，在运行时通过简单的旋转和平移曲线来匹配当前目标状态，从而大幅提高算法运行效率。第二个启发式函数与第一个正好相反，它忽略智能体的运动学约束但使用障碍物地图来计算到目标的最短路径。该启发式函数可以在二维平面上发现死胡同和类 U 型障碍物，并引导三维空间内的搜索远离该区域。这个启发式函数可以使用动态规划的方法计算得到。这两种启发式方法在数学上都是可接纳的（Admissible），所以混合 A* 算法使用两个启发式函数的最大值来作为状态的最终启发值。

前述子节点扩展部分使用了一个离散的动作空间来进行路径搜索，因此搜索将永远

**算法 2**: 混合 A* 算法

Input: $V_s$,$V_g$

$S_{start} \leftarrow (V_s, 0)$;
OPEN $\leftarrow S_{start}$;
CLOSE $= \emptyset$ while OPEN $\neq \emptyset$ do
    $N \leftarrow$ node with minimum $f$ value in OPEN ;
    OPEN $\leftarrow$ OPEN $\setminus\{N\}$ ;
    CLOSE $\leftarrow$ CLOSE $\cup\{N\}$ ;
    if $N.v$ near $V_g$ then
        Analytical expand path to $V_g$ ;
        if whole path has no collision then
            return path;
        end
    end
    foreach $s \in$ GetSuccessors($N$) do
        $N'.v \leftarrow s$ ;
        $N'.t \leftarrow N.t + 1$ ;
        if $N' \notin$ CLOSE then
            if $N'.v$ collide with obstacles then
                CLOSE $\leftarrow$ CLOSE $\cup\{N'\}$ ;
            end
            else if $N'.v$ exist in OPEN then
                Compute the optimum node cost ;
            end
            else
                OPEN $\leftarrow$ OPEN $\cup\{N'\}$ ;
            end
        end
    end
end
return no path;

无法达到精确的目标状态。为了解决这个精度问题同时进一步提高搜索速度，混合 A* 算法使用了一种被称为分析性扩展的方法。具体来说，在正常的子节点扩展过程中，对于某些节点，分析性扩展方法会额外计算出一条从当前节点到目标状态的最短 Reeds-Shepp 路径。然后算法会对该路径与障碍物地图进行碰撞检测，如果路径是可行的，算法就会将这条路径作为子节点添加到搜索树中，并由于该路径已经扩展了目标状态，因此搜索实际上完成了。但由于计算 Reeds-Shepp 曲线计算较为耗时，因此对于每一个节点都执行分析性扩展是不可取的，尤其是当前位置离目标点很远时，大多数分析性扩展的路径都会与障碍物发生碰撞。在混合 A* 算法中，分析性扩展会每隔 N 个节点才会执行一次。

CBS 算法的伪代码详见算法 2。需要注意的是，混合 A* 算法并不能保证能找到最优解，但其保证了输出路径能满足智能体的运动学约束。同时，在实际测试中，混合 A* 的路径通常位于全局最优解的附近，具有很高的应用价值[66]。

## 2.3 本章小结

本章首先介绍了经典多智能体路径规划问题，给出了明确的数学定义并提出了对规划方案的评价指标。随后阐释了两个相关算法：基于冲突搜索算法和混合 A* 搜索算法，为后续章节做准备。基于冲突搜索算法是一个非常先进的最优多智能体路径规划算法，而混合 A* 搜索算法也为本文将运动学约束引入多智能体路径规划求解器提供了思路。

# 3 考虑运动学模型的多智能体路径规划

## 3.1 概述

在2.2.1节中，本文介绍了基于冲突搜索（CBS）算法，它是一个非常先进的最优MAPF求解框架，近年来在多智能体路径规划领域被广泛使用，在物流分拣、机场拖曳等场景中都有着很好的表现。但是CBS算法是针对经典MAPF问题提出的，一般是在无向图尤其是栅格地图中进行规划，这导致CBS算法生成的规划方案一般是针对具有完整约束的全向移动多智能体系统。然而，当考虑为系统中包含具有非完整约束的智能体进行规划时，工作空间就无法再离散为无向图表征了，而是必须在一个连续空间下对多智能体系统进行规划，即问题转变为了本文称之为**考虑运动学约束的MAPF问题**。同时，经典MAPF求解器在无向图内生成的分段线性路径无法被具有非完整约束的智能体在连续工作空间内精确执行，智能体无法按计划在指定的时间到达既定位置，并可能导致执行时智能体之间发生碰撞。因此经典MAPF问题的求解器在考虑运动学约束的MAPF问题中求解时不仅无法保证算法最优性，对于一些问题实例甚至无法求出可行解。针对此情况的一个方法是使用额外的智能体控制器对路径进行跟踪[54]，但这个方法对最终执行效果没有任何保证，在障碍物复杂的环境中，智能体仍有几率与静态障碍物发生碰撞。因此，一个原生支持添加智能体运动学约束的MAPF求解器就显得尤为重要。

本章首先对考虑运动学约束的MAPF问题给出了严格的数学定义，并选取具有阿克曼转向模型的智能体作为一个具体实例进行研究。本章提出了一个全新的求解器：针对阿克曼模型的基于冲突搜索算法，来解决这个问题。该求解器采用了基于冲突搜索的分层搜索框架：在上层搜索中，本文提出了车身约束树的概念，求解器将冲突定义为在连续空间中考虑车身大小的实际碰撞，从而上层搜索树中只需要考虑规划方案的路径间是否存在车身约束，而不必考虑智能体具体的运动学模型；而在下层搜索中，本章提出了时空混合A*搜索算法来进行单智能体路径重规划，该算法能为智能体规划出同时满足运动学约束和时空车身约束的最短路径。因此，针对阿克曼模型的基于冲突搜索算法既继承了二叉约束树计算时间短的优势，又具备在连续空间内为大量阿克曼模型智能体提供运动学可行路径的能力。随后，本文还介绍了该算法的顺序规划版本，该版本在牺牲了一部分求解质量的情况下，大幅缩短了算法的规划时间，从而能在可以接受的运行时间内为包含更多

智能体的系统在更复杂的环境中进行路径规划。最后，本章提出了一个适用于连续空间内 MAPF 问题的基准测试集，随后使用该测试集进行了实验：将本章提出的方法与两个基准算法进行了对比试验，并对原版方法和其顺序规划版方法进行了性能测试来考察两者在智能体数量的可扩展性以及在求解质量上的差异。算法源代码已经在 Github 上开源：https://github.com/APRIL-ZJU/CL-CBS。

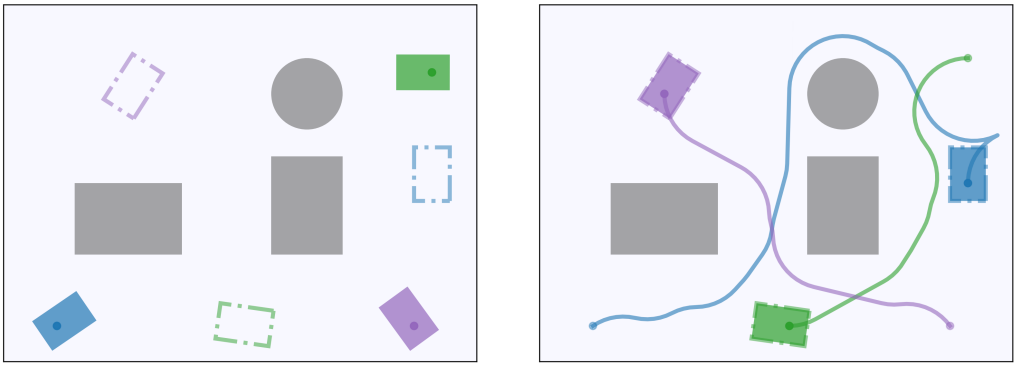
综上，本章提出的算法贡献如下：

1. 本章提出了一种针对阿克曼模型的基于冲突搜索方法，这个分层搜索框架原生支持求解考虑运动学约束的 MAPF 问题。该方法扩展了传统基于冲突搜索算法中对冲突的定义，使上层约束树能够处理连续空间内的智能体碰撞情况，同时仍然具有搜索时间短的优势。
2. 本章提出了时空混合 A* 算法作为基于冲突搜索的下层单智能体路径重规划，通过在搜索节点状态中增加时间维度，该算法能够在连续空间内规划出同时满足运动学约束和时空车身约束的智能体最短路径。
3. 本章还提出了使用优先级规划的思想，提出了上述方法的顺序规划版本。使用顺序规划方法虽然会牺牲少许的规划方案质量，但可以大幅减少算法中约束树的节点扩展量，从而有效缩短程序运行时间。
4. 本章在仿真环境中进行了实验，结果表明本章提出的方法在所有六个场景中都保持着 98% 以上的成功率，而两种基线算法的规划成功率都低于 50%。同时该方法的规划方案质量也超过了两个基线算法的求解质量。此外，本章提出的方法在可接受的运行时间内能够为多达 100 个阿克曼模型智能体求解考虑运动学约束的 MAPF 问题。

## 3.2 问题定义

经典的 MAPF 问题的一条重要假设是智能体被抽象为一个全向移动的质点并在一个无向图 $G=(V,E)$ 中运行，这个假设导致经典 MAPF 问题的规划结果往往无法被现实中具有运动学约束的机器人执行。在本章中，本文去除这条假设，研究在连续工作空间内的多智能体路径规划问题，同时要求规划结果路径符合智能体的运动学约束。本文将这个问题称为**考虑运动学约束的 MAPF 问题**，该问题的定义如下。

考虑一个包含 $N$ 个智能体 $a_1,a_2,\dots,a_N$ 的多智能体系统在一个二维连续的工作空间 $\mathcal{W}$ 中运行。假设工作空间中的障碍物是已知的且静态的，它们占据了工作空间内的指定的区域 $\mathcal{O}$，$\mathcal{O}\subset\mathcal{W}$。则智能体的可通行工作空间可以表示为 $\mathcal{F}=\mathcal{W}\backslash\mathcal{O}$。该系统中的智能体状态表示为 $\mathbf{z}=[x,y,\theta]^{\mathrm{T}}$。$(x,y)$ 表示智能体的位置，一般设置为其刚体框架的几何中



(a) 一个考虑运动学约束的 MAPF 问题实例 (b) 对于该问题的一个解

图 3.1 考虑运动学约束的 MAPF 问题示意图

心。$\theta$ 表示智能体的偏航角，定义为智能体自身坐标系 X 轴和全局坐标系 X 轴的夹角。

本文假设在该问题中，时间是离散的而非连续的。每个智能体 $a_i$ 一直以一个恒定速度 $v_i$ 移动，除非它在该时间步内处于停止状态。同时本文假设该系统中的智能体不考虑加减速的过程，即智能体的移动状态与停止状态可直接进行切换而无需多余的加减速时间。对于一个系统中的智能体 $a_i$，其车身所占据的区域用 $\mathcal{R}(\mathbf{z}^i)$ 来表示，车身形状函数 $\mathcal{R}$ 是一个关于智能体 $a_i$ 状态 $\mathbf{z}^i$ 的函数，其由机器人的大小尺寸等因素共同决定。本文还需考虑智能体的运动学约束，令 $u$ 表示智能体 $a_i$ 的控制输入，$\mathcal{U}$ 表示包含 $a_i$ 所有控制输入的集合。则 $a_i$ 的运动学模型可以定义为：

$$\frac{d\mathbf{z}^i}{dt} = f(\mathbf{z}^i, u), u \in \mathcal{U}. \tag{3.1}$$

令 $\mathbf{z}_i(t)$ 为智能体 $a_i$ 在 $t$ 时刻的状态，一个被分配给 $a_i$ 的路径规划任务会要求它从起始状态 $s_i \in \mathcal{F}$ 移动到目标状态 $g_i \in \mathcal{F}$。任务的起始和目标状态都应位于自由工作空间内，以保证该任务是有效的，即 $\mathcal{R}(s_i) \cap \mathcal{R}(s_j) = \emptyset, \mathcal{R}(g_i) \cap \mathcal{R}(g_j) = \emptyset, \forall i \neq j$。

智能体 $a_i$ 的路径可以表示为 $\pi_i = [\mathbf{z}_i(0), \mathbf{z}_i(1), \ldots, \mathbf{z}_i(T_i), \ldots]$。如果路径满足以下三个条件，则该路径是**可行的**：

- $\pi_i$ 应该从它的起始状态开始 $\pi_i[0] = s_i$，并在有限的时间步 $T_i$ 后达到它的目标状态并随后一直保持在目标状态处，即 $\pi_i[t] = g_i, \forall t \geq T_i$。
- $\pi_i$ 中的每次移动都应满足智能体 $a_i$ 的运动学模型，$\pi_i[t+1] = \pi_i[t] + f(\pi_i[t], u(t))$。
- 智能体 $a_i$ 在沿路径移动时不应与空间内的障碍物发生碰撞，$\mathcal{R}(\pi_i[t]) \subset \mathcal{F}, \forall t$。

本文使用元组 $\langle a_i, a_j, t \rangle$ 来表示智能体 $a_i$ 和 $a_j$ 之间的一个冲突（Conflict）[68]，它意味着这两个智能体的路径在 $t$ 时刻发生了碰撞，即 $\mathcal{R}(\pi_i[t]) \cap \mathcal{R}(\pi_j[t]) \neq \emptyset$。考虑运动学约束的 MAPF 问题的解应当是一个包含所有 $N$ 个智能体的可行路径的集合，其中任意两个智

能体的路径在任何时刻都没有冲突，即

$$\mathcal{R}\left(\pi_i[t]\right) \cap \mathcal{R}\left(\pi_j[t]\right) = \emptyset,\ \forall t \geq 0, i \neq j \tag{3.2}$$

图3.1a展示了一个考虑运动学约束的 MAPF 问题实例，其中深灰色的区域为障碍物区域 $\mathcal{O}$，浅灰色区域为可通行区域 $\mathcal{F}$，三个不同颜色的智能体分别要前往目标位置（用对应颜色虚线框表示）。图3.1b则给出了该示例的一个可行解，三个智能体的路径分别用它对应颜色的实线标出，三个智能体的路径在任意时刻都不会发生碰撞。

很明显，对于同一个 MAPF 实例，一些解会比其他解表现更好。参照经典的 MAPF 问题[51]，本章提出了两个目标函数来评估 MAPF 问题解的性能：总运行时间（Makespan）和平均到达时间（Average flowtime）。

总运行时间，或称为最长运行时间，被定义为多智能体系统中所有智能体都完成任务到达目标状态的时长：

$$\max_{1 \leq i \leq N} T_i \tag{3.3}$$

平均到达时间则是用于评估整个规划方案的平均性能的指标，它被定义为系统中智能体到达其目标的平均时间:

$$\frac{1}{N} \sum_{i=1}^{N} T_i \tag{3.4}$$

## 3.3 阿克曼转向几何

有若干种运动学模型可以描述非完整约束智能体，其中一种应用非常广泛也是本文使用的模型被称为阿克曼转向几何（Ackermann steering geometry）。该模型描述了大部分现代汽车和四轮类车机器人的转向方式，即前轮转向后轮直行，内外轮转过的角度不同。该模型在运动学上表现为禁止智能体的横向移动和原地旋转，本节对阿克曼转向模型进行简单介绍。

一个阿克曼转向模型如图3.2所示，智能体状态可以被表示为 $\mathbf{z} = [x, y, \theta]^T$。智能体自身刚体坐标系的原点 $(x, y)$ 位于其后轴中心位置，自身坐标系的 $X'$ 轴指向偏航角 $\theta$，坐标系 $Y'$ 轴指向智能体右侧。用 $v$ 来表示智能体的线速度，用 $\phi$ 来表示智能体前轮的转向角。当智能体转向角固定为 $\phi$ 时，其转向半径通过计算 $r = L/\tan\phi$ 求得。

智能体转向角 $\phi$ 和其转向角速度 $\omega$ 之间有如下运动学关系：

$$\omega = \dot{\theta} = \frac{v}{L} \tan\phi \tag{3.5}$$

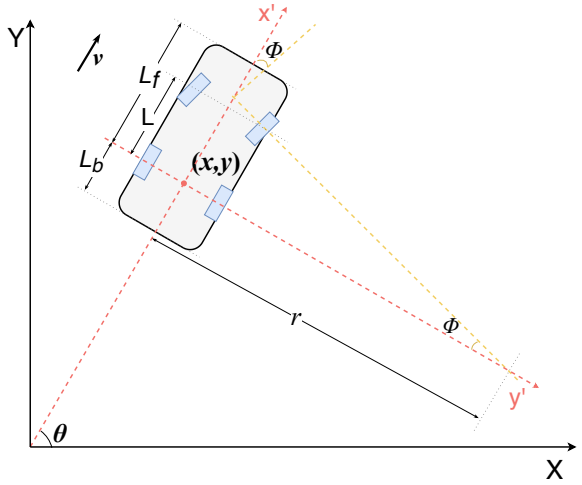



图 3.2 阿克曼转向运动学模型示意图

阿克曼转向模型的智能体的控制输入被定义为 $\mathrm{u} = [v, \omega]^T$。考虑到多智能体系统的时间离散特性，通过离散化和递归积分，可以计算出智能体在时间步 $t$ 的状态如下：

$$\mathrm{z}_t = [x, y, \theta]^{\mathrm{T}} = \mathrm{z}_{t-1} + [v\cos\theta, v\sin\theta, \omega]^{\mathrm{T}} \tag{3.6}$$

同时，阿克曼模型中智能体速度被限制为 $v_B \leq v \leq v_F$，其中，$v_F > 0$ 和 $v_B < 0$ 分别代表智能体前向和倒车移动时的最大速度。智能体转向角也有着上限 $\phi_{max}$，这意味着具有阿克曼转向模型的智能体在移动中具有一个最小转弯半径 $r_{min} = L/(\tan\phi_{max})$。

## 3.4 针对阿克曼模型的基于冲突搜索

在本节中，本文提出了一个全新的求解器，称为针对阿克曼模型的基于冲突搜索算法。该求解器采用了分层搜索框架，在上层搜索中，本文将冲突的概念扩展为在连续空间内的实际车身碰撞，并提出了车身约束树的概念。在下层搜索中，本文提出了时空混合 A* 算法（SHA*），该算法可以为具有阿克曼模型的智能体规划出既符合运动学约束又满足时空车身约束的最短路径。在本节的最后，本文还介绍了该方法的优先级规划版本，通过引入优先级规划的思想，该版本虽然牺牲了少许求解质量，但显著减少了约束树中搜索节点扩展数量，大幅缩短了程序运行时间。

### 3.4.1 车身约束树

如2.1节所述，经典 MAPF 求解器常常需要应用顶点冲突、边冲突和调换冲突等不同类型的冲突来表征两个单智能体路径之间的碰撞。但这些冲突只能在无向图中适用，同时它们也无法代表所有可能发生的碰撞情况。而受益于考虑运动学约束的 MAPF 问题是

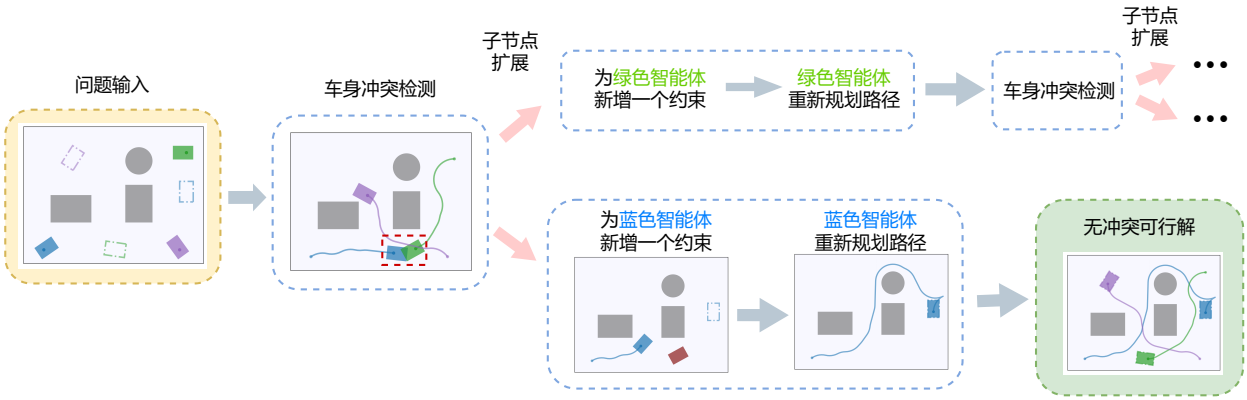



图 3.3 针对阿克曼模型的基于冲突搜索方法示意图

在连续空间中进行规划的特性，本文直接使用**车身冲突**来描述智能体之间的所有碰撞情况。车身冲突被定义为 $(a_i, a_j, t)$，该冲突意味着智能体 $a_i$ 和 $a_j$ 会在 $t$ 时刻发生碰撞，即 $\mathcal{R}(\pi_i[t]) \cap \mathcal{R}(\pi_j[t]) \neq \emptyset$。

本文提出了一个二叉车身冲突树（Body Conflict Tree，BCT），并对其进行最佳优先搜索。同基于冲突搜索算法类似，车身冲突树中的每个节点 $N$ 内都包含了一个智能体间的约束集 $N.constraints$ 以及一个满足这些约束条件的规划方案 $N.solution$。本文将节点的成本 $N.cost$ 定义为规划方案中所有路径的长度之和。

算法上层搜索的伪代码如算法3所示。首先，车身冲突树的根节点中智能体约束集为空集，下层单智能体路径规划器会在忽略其他智能体影响的前提下为每个智能体生成一条满足运动学模型的最短路径。随后算法对车身冲突树进行最佳优先搜索，算法将冲突树中所有可扩展节点中成本最小的节点 $N$ 弹出，并对 $N$ 中的规划方案 $N.solution$ 进行一次车身冲突检测。如果 $N.solution$ 中没有车身冲突，这意味着 $N.solution$ 就是该 MAPF 问题的一个可行解，同时最佳优先搜索的特性也保证了这是冲突树中能找到的最好的规划方案。反之，如果节点 $N$ 中存在一个或者多个车身冲突，算法考虑最先发生的冲突作为该节点需要处理的冲突，而将其他冲突交给更深层的节点进行处理。假设最先发生的冲突为 $(a_i, a_j, t)$，算法会生成两个约束条件：$(a_i, N.\pi_j[t], t)$ 和 $(a_j, N.\pi_i[t], t)$；前者约束意味着智能体 $a_i$ 在时间步 $t$ 需要避开指定区域 $\mathcal{R}(N.\pi_j[t])$，从而避免冲突 $(a_i, a_j, t)$ 的发生；后者约束也是类似，即智能体 $a_j$ 在时间步 $t$ 需要避开指定区域 $\mathcal{R}(N.\pi_i[t])$。接着两个节点 $N$ 的子节点被生成，每个子节点包含上述两个约束条件之一，算法在每个子节点中对新增约束相关的智能体执行下层单智能体路径规划器，对其路径重新规划，其他智能体的路径则继承节点 $N$ 中的路径。最后两个子节点被加入车身冲突树中，算法继续对车身冲突树进行最佳优先搜索寻找下一个处理的节点 $N$。

车身冲突树的扩展方式如图3.3所示。智能体的起始状态用彩色实心矩形表示，目标

**算法** 3: 针对阿克曼模型的基于冲突搜索算法

Root.constraints← ∅;
Root.solution← lowlevel_planner($a_i, \emptyset$), for each $a_i$ in system;
BCT← {Root};
while BCT $\neq \emptyset$ do
    $N \leftarrow \min_{cost} N'$, $\forall N' \in$ BCT;
    BCT ← BCT $\setminus \{N\}$;
    $C \leftarrow$ search for first body conflict in $N.solution$;
    if C $= \emptyset$ then
        return $N.solution$;
    end
    foreach $a_i$ appears in $C$ do
        $N' \leftarrow N$;
        Add $(a_i, N.\pi_j[t], [t, t+k])$ to $N'.constraints$;
        $N'.\pi_i \leftarrow$ lowlevel_planner($a_i, N'.constraints[i]$);
        if $N'.\pi_i \neq \emptyset$ then
            BCT ← BCT $\cup \{N'\}$;
        end
    end
end
return ∅;

状态用对应颜色的虚线表示，深灰色区域为障碍物 $\mathcal{O}$。首先算法会在忽略其他智能体影响的前提下为每个智能体生成最短路径，然后执行车身冲突检测时发现，蓝色智能体和绿色智能体的路径在时间 $t$ 存在车身冲突，如左二图红色框中所示。随后算法为节点进行扩展出两个子节点，每个子节点都包含一个新的约束。下层单智能体路径规划期分别在两个子节点中为蓝色智能体和绿色智能体重新规划满足新增约束的运动学可行路径，并在此基础上不断扩展并进行最佳优先搜索，直到找到一个无冲突可行解。

当然，节点 $N$ 在时间步 $t$ 也存在有多于两个智能体同时发生碰撞的可能，尤其是当系统内包含较多智能体时。虽然本文将车身冲突定义只能描述两个智能体之间的碰撞，但是上述多于两个智能体连环碰撞的情况可以被拆分为若干个智能体两两之间的车身冲突。

因此对于节点 $N$，基于冲突搜索算法仍然只需要处理一个车身冲突，将其余因碰撞产生的车身冲突留给节点 $N$ 更深层的子节点进行处理。

由于通信干扰、执行精度等因素，智能体在实际场景中往往无法按照规划方案精确执行。例如，智能体 $a_i$ 的规划路径在 $t$ 时间步会到达 $\pi_i[t]$，但由于 2 个时间步的因通讯干扰导致的执行延迟，导致智能体 $a_i$ 在 $t+2$ 时间步才到达 $\pi_i[t]$。因此，算法也需要具有一定程度的鲁棒性来应对执行延迟。假设多智能体系统内的每个智能体存在一个最多 k 时间步的执行延迟，那么当一个冲突 $(a_i, a_j, t)$ 发生时，搜索算法需要确保在时间 $t$ 及之后的 $k$ 个时间步内都不会发生因执行延迟而导致的碰撞。因此在算法实现中，代理间的约束需要持续一定的时间窗口 $(a_i, N.\pi_j[t], [t, t+k])$ 来保证整体规划方案的安全（算法 3第 13行）。

### 3.4.2 时空混合 A* 搜索

如上所述，上层的车身冲突树搜索要求下层单智能体路径规划器能够规划满足阿克曼模型运动学约束的路径的同时，也要求这些路径能满足节点中关于这个智能体的所有约束条件。混合 A* 算法能够为阿克曼运动学模型的智能体在连续空间内规划路径，但是其不能处理同时包含时间和位置的智能体间约束条件。为此，本文提出了一个扩展版本的算法，称为时空混合 A* 算法（Spatiotemporal Hybrid-State A*，SHA*），作为基于冲突搜索方法的下层单智能体路径规划器。

时空混合 A* 算法维护一个开放列表（Open list）并基于它做进行路径搜索和扩展。每个开放列表中的节点 $N$ 都包含一个元祖 $(state, t, g, h, f)$，意味着在当前节点 $N$，智能体在 $t$ 时间步扩展到了 $state = (x, y, \theta)$ 的状态。$g$，$h$ 和 $f$ 函数和混合 A* 算法的原始定义相同，$g$ 函数计算从起始状态到当前状态 $(x, y, \theta)$ 的路径成本，一般直接按路径长度计算；$h$ 函数为启发式函数，用于估算当前节点到目标状态的路径成本；$f$ 函数是当前节点的总路径成本，为 $g$ 函数和 $h$ 函数之和，即 $N.f = N.g + N.h$。

当开放列表中具有最小 $f$ 值的节点被弹出时，时空混合 A* 算法首先会检查节点状态 $N.state$ 是否在目标状态附近。如果在附近，算法会调用分析性扩展函数（Analytical_Expansion）来计算一条从当前状态到目标状态、忽略障碍物的最短路径。若该路径不与静态障碍物发生碰撞，则表明找到了一条从起点到目标状态的可行路径。如果分析性扩展路径与静态障碍物发生了碰撞或当前状态不在目标状态附近，那么算法会正常扩展当前路径的子节点并对每一个生成的子节点进行碰撞检测。对于可行的子节点，时空混合 A* 算法会将其时间步加一并计算该节点状态的 $g$、$h$ 和 $f$ 函数，最后将该节点加入开放列表中。详细的伪代码请参见算法 4。

**算法 4**: 时空混合 A* 搜索（以智能体 $a_i$ 为例）

Root.state$\leftarrow s^i$;
Root.t$\leftarrow 0$;
OpenList$\leftarrow \{$Root$\}$;
while OpenList$\neq \emptyset$ do
    $N \leftarrow \arg\min N'.\text{f},\ \forall N' \in$OpenList;
    if $N$.state near $g^i$ then
        Analytic_Expansion$(N.state, g^i)$;
        $\pi_i \leftarrow$ Backtracking search from $N_{goal}$;
        if No collision in $\pi_i$ then
            return $\pi_i$;
        end
    end
    States $\leftarrow$ Get_ChildNode($N$.state);
    foreach s $\in$ States do
        $N'$.state $\leftarrow$ s;
        $N'$.t $\leftarrow$ $N$.t $+1$;
        if Check_Collison($N'$.state ,$N'$.t) $= \emptyset$ then
            $N'.h \leftarrow$ Admissible_Heuristic($N'$.state,$g^i$);
            Update $N'.g, N'.f$;
            if $N'$.state not appear in OpenList then
                OpenList $\leftarrow$ OpenList $\cup \{N'\}$;
            else if $N'.g < N_{inOpen}.g$ then
                Update $N_{inOpen}$ with $N'$.state, $N'.f$;
            end
        end
    end
end
return $\emptyset$;

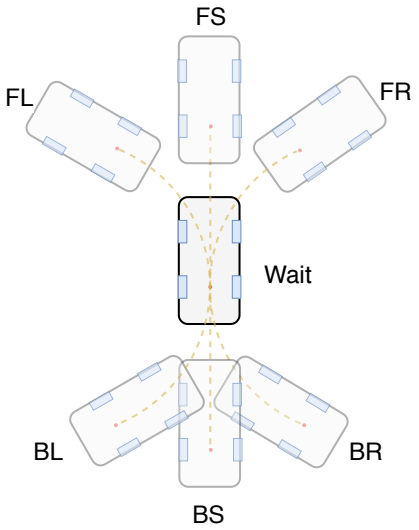
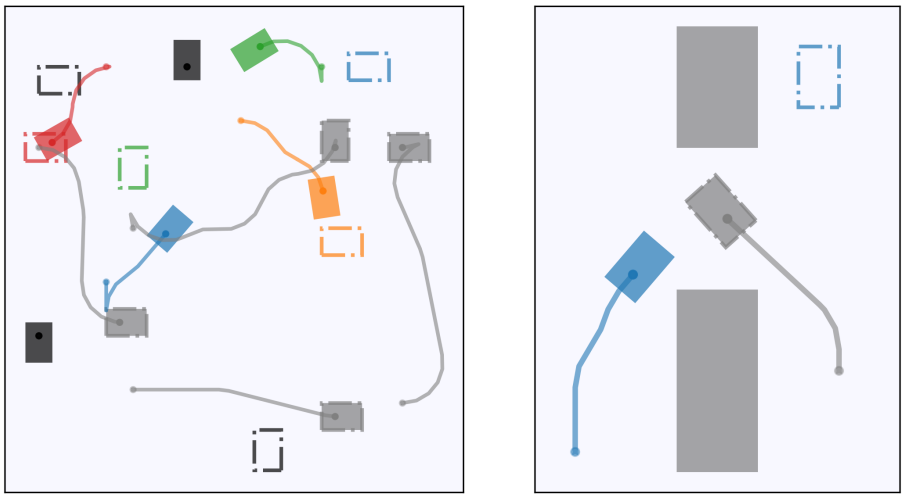



图 3.4 时空混合 A* 算法扩展动作集 $\mathcal{U}$ 示意图

对于开放列表中的节点 $N$，一个重要的问题就是判断其状态 $N.state$ 是否可行。算法会调用碰撞检测函数（Check_Collision）来检查该状态时车身是否位于可行区域 $\mathcal{F}$ 内以及是否符合所有当前时间点的上层约束条件。具体来说，一个节点状态 $N.state$ 是可行的当且仅当：$\mathcal{R}(N.\text{ state}) \cap C = \emptyset, \forall C \in \mathcal{C}_{\text{N.time}}$ 且 $\mathcal{R}(\text{ N.state}) \subset \mathcal{F}$，此处 $\mathcal{C}_{\text{N.time}}$ 表示对于当前智能体的约束集中约束时间为 $N.time$ 的所有约束。

考虑到时空混合 A* 算法中工作空间的连续性和时间的离散性，本文采用了运动基元[69] 的概念，将阿克曼运动模型的控制离散为一个动作集 $\mathcal{U}$，其中有七个动作，分别为：向前左转（FL），向前直行（FS），向前右转（FR），向后左转（BL），向后直行（BS），向后右转（BR）和原地等待（Wait），如图 3.4所示。除了原地等待之外，其余六个移动动作都应符合公式 3.6中的阿克曼运动学约束。算法会调用扩展子节点函数（Get_ChildNode）来使用动作集 $\mathcal{U}$ 中的每个动作 $u$ 来扩展节点 $N$ 的状态并生成新的子节点状态 $N'.state$。值得注意的是，本文更希望智能体进行向前直行，因为这个动作既是最符合人类直觉（没有人会希望一个智能体一直倒车前往终点），同时向前直行也是大部分机器人执行精度最高的一个动作。因此，当智能体选择转弯动作、后退以及变化移动方向（即上个时刻是前行当前时刻选择后退动作，或者相反）时，算法会对该时间步的成本函数 $g$ 增加了惩罚系数。

同时，当将动作离散化时，本文希望每次扩展的长度足够小，以便组合出一条精细的路径。因此，时空混合 A* 算法将节点每次延伸长度限制在小于阿克曼模型智能体的车身长度 $L$。这意味着智能体在相邻时间的两个状态之间会存在车身重叠，$\mathcal{R}(\pi_i[t]) \cap \mathcal{R}(\pi_i[t+1]) \neq \emptyset$，这个限制同时保证了碰撞检测过程的有效性。此外，时空混合 A* 算法的启发式函数（Admissible_Heuristic）和分析性扩展函数（Analytical_Expansion）的设计都与原始混合 A*



(a) 顺序规划版本算法的示意图 (b) 一个顺序规划无法求解的实例

图 3.5 针对阿克曼模型的基于冲突搜索算法的顺序规划版本

算法一致，具体详见2.2.2节。

### 3.4.3 顺序规划版本

由于工作空间从离散空间扩展到了连续空间，同时还考虑了智能体的运动学约束，针对阿克曼模型的基于冲突搜索算法会天然比针对经典 MAPF 问题的求解器搜索时间更长。当障碍物数量增加或者工作空间变大时，下层单智能体规划器的计算时间还会相应的增加。此外，当多个智能体在相近的时间密集访问同一个区域时，车身冲突也会大量增加，导致上层搜索树需要扩展更多的节点来解决该冲突。这些问题都会使得整个 MAPF 方法的搜索时间进一步增加。

尽管下层时空混合 A* 的扩展性问题是不可避免的，但是利用文献[58] 中的思想，本文提出了一种顺序规划方法来减少上层冲突树的扩展节点数，从而减少整个框架的搜索时间。顺序规划方法将系统内的 $N$ 个智能体分为 $K_b$ 个批次，除了最后一个批次每个批次将包含 $\lceil N/K_b \rceil$ 个智能体。顺序规划的思想是先依次解决每一批次的 MAPF 子问题，再将所有批次的规划结果整合起来最终得到整个 MAPF 问题的规划方案。对于第 $b$ 个批次，所有后续批次中的智能体将在该批次规划时被忽略；而第 1 到第 $b-1$ 个批次中已经规划完成的智能体的路径将被当做工作空间内的动态障碍物进行规避，这些路径会被转换为约束集的形式直接添加到当前批次的车身冲突树根节点中。

图 3.5a展示了顺序版本方法的示意。该系统内包含 10 个阿克曼模型的智能体，分为 3 个批次。每批次分别包含 4 个、4 个和 2 个智能体。图中展示的是第二批规划时的场景，当前批次正在规划的智能体用彩色矩形表示。第一批已经规划好的智能体作为动态障碍

物以灰色矩形的路径标识，意味着当前批次的智能体需要在规划时避开它们。而第三批次的两个智能体还没有被规划，它们将在被本次规划忽视，在图上用黑色矩形来标识。然而，需要注意的是，顺序规划的方案会在某些原本方法可解的情况中遭遇无法求解的情况。图3.5b展示了一个典型的顺序规划无法求解的例子。图中障碍物将工作区域分为了两部分，中间只留了一个通道相连。蓝色智能体需要从左侧区域通过通道到达右侧区域，而灰色智能体的目标状态就位于通道中央。当灰色智能体在蓝色智能体之前的批次规划时，它会在蓝色智能体还没有通过障碍物时就停在了通道中间的目标位置上。又因为灰色智能体路径已经成为动态障碍物，无法进行修改，因此在当前批次内规划的蓝色智能体无法到达其位于障碍物另一侧的目标位置，顺序规划失败。

但是，虽然顺序规划方法可能在少数情况下无法求解，但是在本文的实际测试中，这种极端情况较少出现。而由于避免了上层冲突树同时处理过多的智能体的碰撞，顺序方法相比于原版算法的搜索时间缩短了近一个数量级。本文认为该顺序规划方法是值得在实际场景中进行应用的。

## 3.5 实验

在本节中，本文使用 C++ 实现了前文所述的针对阿克曼模型的基于冲突搜索算法，其中代码使用 Boost 库[①]用于数学计算，使用 OMPL 库[②]来生成 Reeds-Shepp 最短路径。由于目前使用较为广泛的基准测试集[51] 是针对经典 MAPF 算法的，该测试集中工作空间都是以栅格地图来表示，无法应用于考虑运动学约束的 MAPF 问题。因此，本节首先提出了一个全新的连续空间下的 MAPF 问题基准测试集以供后续算法测试。随后本章将针对阿克曼模型的基于冲突搜索算法和两个基准算法在仿真环境中进行对比实验。最后由于本章提出的方法有原版和顺序规划两个版本，本节最后对这两个版本进行了相同环境下的性能测试。本节提出的基准测试集和算法源代码已经开源在 Github 上：https://github.com/APRIL-ZJU/CL-CBS。

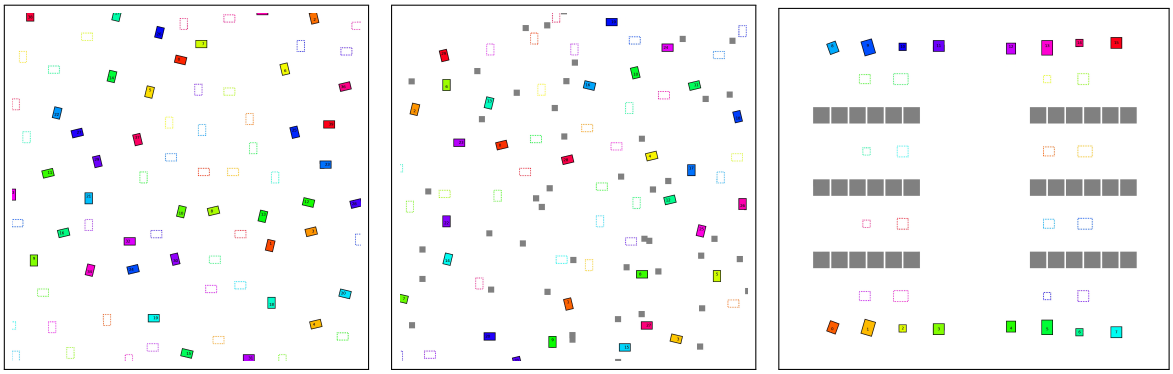
在本节实验中，本文假设在仿真实验中的所有智能体都是具备阿克曼运动学模型且能按规划精准执行。阿克曼智能体具有如下参数：其形状为一个 2 米 ×3 米的矩形，其中 $L_f = 2m, L_b = 1m$；其前进最大速度 $v_F = 2m/s$，后退最大速度 $v_B = -1m/s$；其前轮最大转向角 $\phi_{max} = 45°$，即其最小转弯半径 $r_{min} = 3m$。所有的程序都是在一台配有 Intel i7-8700@3.20GHz 处理器、8G 内存的运行有 Ubuntu 16.04 操作系统的计算机上执行的。

[①] https://www.boost.org/

[②] https://ompl.kavrakilab.org/

### 3.5.1 基准测试集

经典的 MAPF 基准测试集，如 DAO 地图集，都是基于栅格地图来表征工作空间的，因此对于考虑运动学约束的 MAPF 问题并不适用。本文生成了一个全新的基准测试集，其中包含三个不同场景：空旷（Empty）、障碍物（Obstacle）和仓储（Warehouse）。每个场景又包含了三种不同的地图大小，分别为 300 米 ×300 米、100 米 ×100 米和 50 米 ×50 米。空旷场景中除了智能体外没有任何障碍物，是最为简单的场景。障碍物场景中，工作空间内会随机布置 100 个方形障碍物，在从小到大的地图中，障碍物半径分别为 0.5 米、1 米和 2 米。相比于空旷场景，障碍物场景更为复杂，对算法的可扩展性挑战更大。而仓储场景则模拟了物流仓的环境，在工作空间中会放置若干排的物流架，该场景在在线 MAPF 实例中更常出现。三个不同场景如图 3.6所示。



(a) 空旷（Empty）场景 (b) 障碍物（Obstacle）场景 (c) 仓储（Warehouse）场景

图 3.6 基准测试集中不同场景示意图。实心矩形为智能体起始位置，空心矩形为相应颜色智能体终止位置，灰色为障碍物区域。

本节使用不同大小的空旷和障碍物共计六个场景进行实验。基于这六个场景，本文扩展出 3000 个的考虑运动学约束的 MAPF 问题测试实例。在这些实例中，场景中智能体数量从 5 个到 100 个不等。在每组具有相同智能体数量的子场景中，通过设置智能体不同的起终点，最终每组子场景都会包含 60 个互不相同的测试实例。对于每个测试实例，本文都保证如下三点：i）该实例描述了一个连续的工作空间；ii）所有智能体的开始和目标状态保证不相互碰撞；iii）单个智能体的开始状态和目标状态之间的直线距离大于地图宽度的 1/4。下面本章基于这些测试实例进行算法的对比测试和性能测试。

### 3.5.2 与基线算法对比试验

本节使用两种方法来作为实验的基线算法：i）文献[70] 中提出的方法（下称 CBS-MPC），该方法是一个集中式算法，其使用原版基于冲突搜索算法生成分段线性的路径，并且在后

端使用模型预测控制来生成带运动学约束的智能体最终轨迹。ii）是优先级规划[35]的时空混合 A* 搜索方法（下称 PP-SHA*），即先为系统内的每个智能体分配一个优先级，再根据优先级对每一个智能体依次进行规划，具有低优先级的智能体需要避开更高优先级的规划路径，同时使用时空混合 A* 算法作为单智能体路径规划器，以满足运动学约束的需要。

本节选取了三个不同的场景进行实验，分别为：包含 50 个智能体的 300 米 ×300 米场景、包含 30 个智能体的 100 米 ×100 米场景以及包含 20 个智能体的 50 米 ×50 米场景。同时每个场景都会在空旷和有障碍物的情况下分别进行测试。本节使用顺序版本的针对阿克曼模型的基于冲突搜索算法（下称 Ours）来与两个基线算法进行对比实验，其中顺序版本的批次数量设置为 $K_b = 2$。本文在每个场景中对三个算法各进行 30 次运行，并将每次程序运行时间限制设定为 90 秒，若程序在 90 秒内未能运行完成给出可行解或给出的规划方案内存在碰撞，则认为本次运行失败。本文比较了每个算法的成功率、总运行时间和平均到达时间，其中算法成功率越高越好，而总运行时间和平均到达时间都是越小越好。实验结果见表 3.1。

表 3.1 与基线算法对比试验

| 类型 | 地图大小 | 智能体数量 | 规划方法 | 成功率（%） | 总运行时间（秒） | 平均到达时间（秒） |
|---|---|---|---|---|---|---|
| 空旷场景 | 300×300 | 50 | CBS-MPC | 7.5 | 206.626 | 144.90 |
| | | | PP-SHA* | 45.0 | 188.27 | 137.66 |
| | | | Ours | 100 | 179.36 | 134.77 |
| | 100×100 | 30 | CBS-MPC | 6.7 | 69.9192 | 49.27 |
| | | | PP-SHA* | 36.7 | 70.90 | 50.18 |
| | | | Ours | 100 | 83.73 | 56.32 |
| | 50×50 | 20 | CBS-MPC | 28.2 | 46.38 | 35.64 |
| | | | PP-SHA* | 36.7 | 49.88 | 32.15 |
| | | | Ours | 100 | 46.25 | 30.25 |
| 障碍场景 | 300×300 | 50 | CBS-MPC | 3.3 | 205.451 | 143.13 |
| | | | PP-SHA* | 10 | 189.47 | 138.44 |
| | | | Ours | 98.3 | 181.27 | 134.68 |
| | 100×100 | 30 | CBS-MPC | 3.3 | 67.91 | 47.00 |
| | | | PP-SHA* | 8.33 | 70.38 | 48.67 |
| | | | Ours | 100 | 85.76 | 56.60 |
| | 50×50 | 20 | CBS-MPC | 8.3 | 55.35 | 34.51 |
| | | | PP-SHA* | 3.33 | 54.38 | 32.54 |
| | | | Ours | 98.3 | 47.32 | 30.81 |

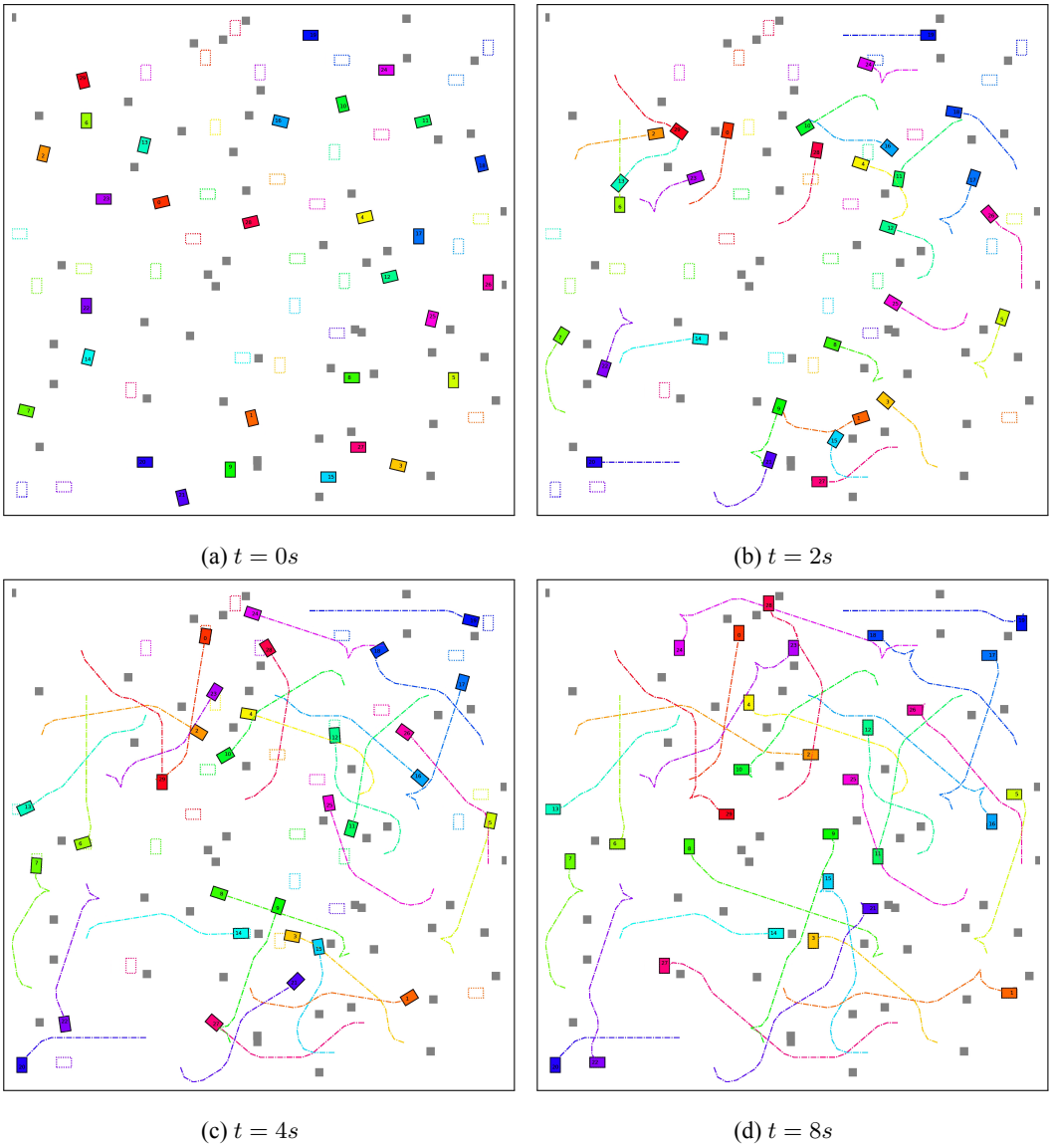
在所有六个场景中，CBS-MPC 方法在五个场景中成功率都低于 10%。这是因为该方法前端使用了不考虑运动学约束的原版 CBS 算法，后端的模型预测控制会使得智能体在执行时偏离原始路径，从而导致智能体之间发生碰撞并最终使得规划任务失败。使用优先级规划的时空混合 A算法表现稍好，在 300 米 ×300 米空旷场景中可以取得 45% 的成功率。然而，由于障碍场景中的工作空间更加复杂，在优先级规划框架中，低优先级的智能体无法找到前往终点的可行路径。因此 PP-SHA* 方法的成功率在所有三个障碍场景中均不足 10%，甚至在 50 米 ×50 米障碍场景中，其成功率甚至不如 CBS-MPC 方法。相比之下，针对阿克曼模型的基于冲突搜索算法在四个场景中都达到了 100% 的成功率，在 300 米 ×300 米障碍场景和 50 米 ×50 米障碍场景中也达到了 98.3% 的成功率，大大超过了基线算法的表现。

而在算法的求解质量方面，针对阿克曼模型的基于冲突搜索算法表现也明显由于两个基线算法。在所有 300 米 ×300 米和 50 米 ×50 米的四个场景中，本章提出的方法在总运行时间和平均到达时间两项指标都小于 CBS-MPC 方法和 PP-SHA* 方法。这是由于该方法原生支持智能体的运动学约束，而两种基线算法中：CBS-MPC 方法前端规划在离散空间内进行，本身的分段线性路径就比连续空间内规划路径要更长；PP-SHA* 方法虽然使用时空混合 A* 算法考虑了运动学约束，但是由于优先级规划会使用固定的优先级顺序并要求低优先级智能体避让高优先级智能体，导致整体规划方案中平均路径到达时间不断增加。此外，在 100 米 ×100 米的两个场景中，本章提出的方法在两项总运行时间和平均到达时间指标上都略高于基线算法。值得注意的是，这并不意味着该方法在该场景上表现差于基线算法，而是由于基线算法的成功率较低（基本都在 10% 以下），规划失败的实例并不会纳入总运行时间和平均到达时间两项指标的计算。一般来说，较为简单的实例中方案的总运行时间和平均到达时间均小于复杂实例，基线方法只能求解较少简单的实例，平均的规划方案指标自然就会比可以规划所有问题实例的方法低。

综上所述，针对阿克曼模型的基于冲突搜索方法与两种基线算法相比，在规划成功率方面有显著优势，同时该方法求出的多智能体规划方案的质量也更好。

### 3.5.3 性能测试

本小节测试了在本章提出的方法在扩展到大规模智能体系统的表现。实验选择四个场景进行测试：300 米 ×300 米大小和 100 米 ×100 米大小的空旷和障碍场景。实验测试包括两种算法：原版的针对阿克曼模型的基于冲突搜索算法和其顺序规划的版本，其中顺序版本的批次数量设置为 $K_b = 2$。在每个场景中，实验对两种算法从包含 10 个智能体系



(a) $t = 0s$ (b) $t = 2s$

(c) $t = 4s$ (d) $t = 8s$

图 3.7 30 个阿克曼模型的智能体在一个 100 米 ×100 米障碍场景实例的规划方案示意图

统开始，一直测试到系统内包含 100 个智能体为止（对于 100 米 ×100 米的场景测试到 50 个智能体为止）。一个由原版方法求出的在一个 100 米 ×100 米的障碍场景实例中的规划方案如图所示。图 3.7展示了一个由原版方法为 20 个智能体在 100 米 ×100 米的障碍场景实例中的规划方案。

在上述四个场景的每个智能体数量下，本文对两个算法各进行 30 次运行，并将每次程序运行时间限制设定为 90 秒，若程序在 90 秒内未能运行完成给出可行解或给出的规划方案内存在碰撞，则认为本次运行失败。本文计算了两种方法在不同场景下的成功率（Success Rate）、平均算法运行时间（Runtime）、规划方案的总运行时间（Makespan）和平均到达时间（Average flowtime），结果见图 3.8和图 3.9。

在 300 米 ×300 米大小的场景中，当智能体数量较少的时候，所有算法的规划成功率

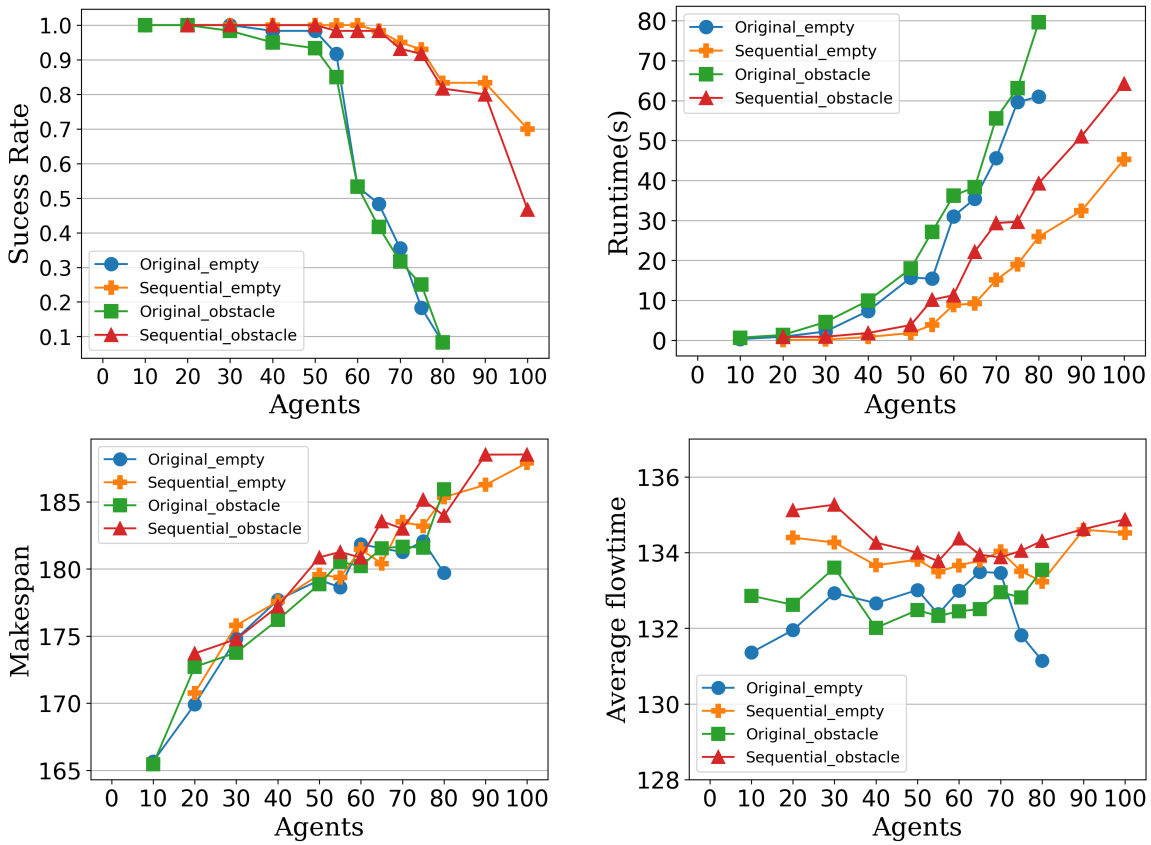



图 3.8　在 300 米 ×300 米场景中两种算法的实验结果。其中蓝、橙、绿、红线分别代表空旷场景内原版方法（Original_empty）、空旷场景内顺序方法（Sequential_empty）、障碍场景内原版方法（Original_obstacle）和障碍场景内顺序方法（Sequential_obstacle）。

都 100%。而当智能体数量超过 50 个时，由于车身冲突的增加，原始版本的算法在某些实例中无法在规定的时间内求出可行解，规划成功率开始下降。在系统内超过 70 个智能体的情况下，原始版本的算法只在不到 20% 的实例中在时间限制内找出了规划方案。相对的，无论是在空旷场景还是在障碍场景中，顺序规划版本在系统内包含 90 个智能体的情况下都保持了 80% 的成功率。同时可以发现，由于障碍场景比空旷场景更复杂，在相同情况下，障碍场景的规划成功率普遍低于空旷场景。随着智能体数量增加，原始版本算法的程序运行时间呈指数级增长，而顺序规划版本的方法程序运行时间则更短。在空旷场景中，原始版本的算法为 75 个智能体计算规划方案平均需要 59.62 秒，而顺序规划版本平均只需要 18.98 秒，是前者的三分之一。这是由于顺序规划的方法部分解耦了智能体之间的车身约束，减少了上层冲突树中扩展节点数，从而降低了程序搜索时间。

此外，在 300 米 ×300 米大小的场景中随着智能体数量的增加，规划方案中智能体路径中的规避和绕路行为也在增加，最终表现为规划方案总运行时间的增加（从不到 170 秒到最高的 187.9 秒）。在同一场景的相同实例中，原版算法和顺序版本生成的规划方案的

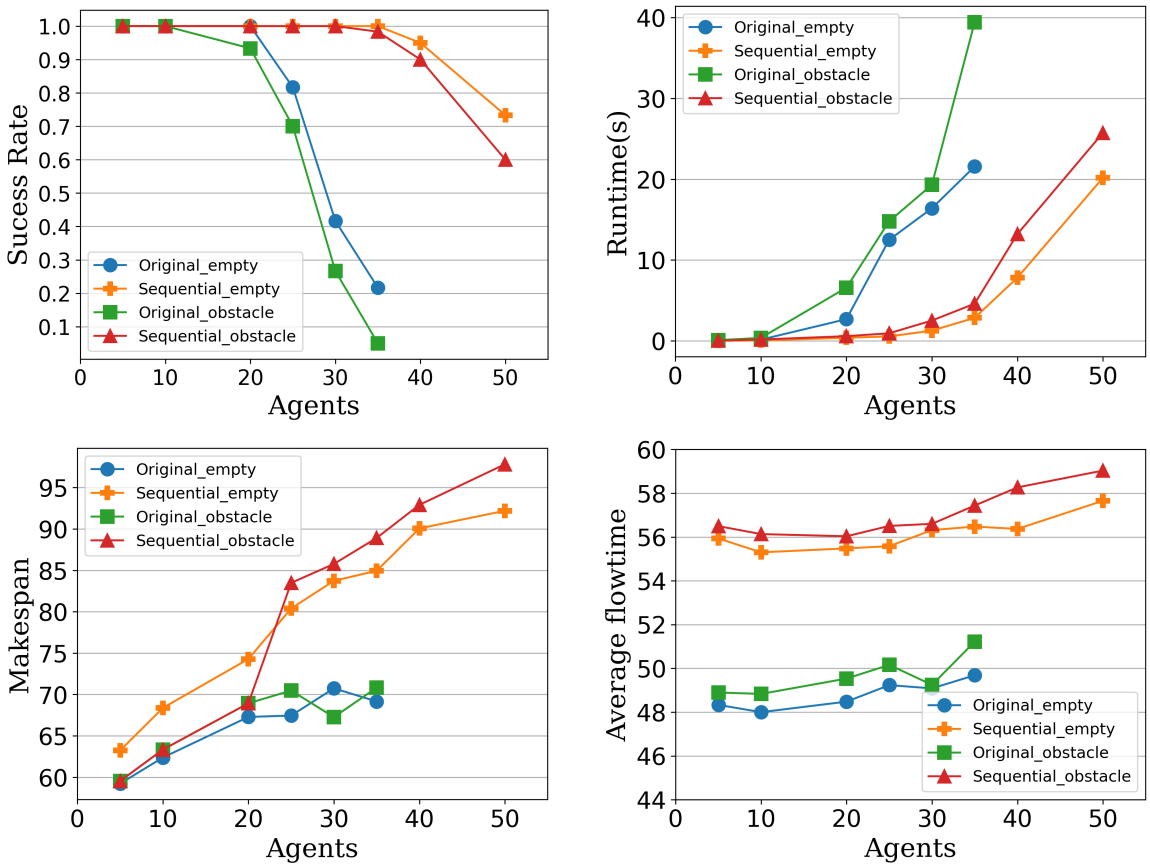



图 3.9 在 100 米 ×100 米场景中两种算法的实验结果。其中蓝、橙、绿、红线分别代表空旷场景内原版方法（Original_empty）、空旷场景内顺序方法（Sequential_empty）、障碍场景内原版方法（Original_obstacle）和障碍场景内顺序方法（Sequential_obstacle）。

总运行时间差别不大。但顺序版本方案的平均到达时间均略高于原版算法生成的方案。这是由于顺序规划版本中存在智能体批次之间的优先级关系，因而其牺牲了一定的求解质量以换取更短的程序搜索时间。但是，顺序规划版本的方法生成的规划方案的平均到达时间最大差异不超过原版方案的 101.5%，差距几乎可以忽略。

图 3.9显示了在 100 米 ×100 米大小的场景中的实验结果。随着系统内智能体数量的增加，两种算法在成功率（Success Rate）、平均算法运行时间（Runtime）、规划方案的总运行时间（Makespan）和平均到达时间（Average flowtime）四项指标上的趋势和在 300 米 ×300 米场景中相似。在 100 米 ×100 米大小、智能体数量超过 20 个后，原始版本的方法规划成功率便开始急剧下降，而顺序规划版本的方法为多达 50 个智能体进行规划的情况下仍然能保持 73.3% 的规划成功率。同时，由于空旷场景中的可行工作空间较大，空旷场景下的规划成功率通常大于障碍场景的成功率。

在 100 米 ×100 米大小的场景中，原版的针对阿克曼模型的基于冲突搜索算法和顺序

规划版本生成的规划方案在方案总运行时间和平均到达时间上产生了差异。显然，原版方法在相同情况下具有更小的平均到达时间和更好的求解质量，但顺序版本生成的规划方案将平均到达时间控制在了原版方案的 117% 以内。因此本文认为，顺序版本的方法通过牺牲少许的解决方法质量来实现运行时间的大幅减少和成功率的显著提高是值得的。

## 3.6 本章小结

本章主要介绍了本文提出的针对阿克曼模型的基于冲突搜索方法。首先本章对考虑运动学约束的 MAPF 问题给出了严格定义，并简单介绍了阿克曼转向模型。本章针对多阿克曼模型的智能体系统构建了一个分层搜索框架，在上层搜索中本文提出了车身约束树的概念来处理在连续工作空间内的智能体间碰撞的情况，而在下层搜索中，本文提出了时空混合 A* 搜索算法来作为单智能体的路径规划器，该算法可以规划出同时满足智能体动力学约束和车身时空约束的最短路径。此外本章还介绍了该方法的顺序规划版本，通过对智能体分批次的解耦，其牺牲一定求解质量换取了更短的程序搜索时间。

本章提出了一个适用于连续空间内 MAPF 问题的基准测试集，并基于该测试集进行了实验。本章将针对阿克曼模型的基于冲突搜索算法与两个基准算法进行了对比试验，结果显示本章提出的方法在规划成功率和规划方案质量上都超越了两个基线算法。此外，本章还对原版方法和其顺序规划版方法进行了性能测试来考察两者在智能体数量的可扩展性以及在求解质量上的差异。

# 4 异构多智能体系统的在线路径规划

## 4.1 概述

在第三章中，本文介绍了一种改进的基于冲突搜索方法来解决考虑运动学模型的MAPF问题，据本文所知，这是学术界第一个原生支持在连续空间内进行规划的基于搜索的多智能体路径规划框架。但是，要想在实际工业系统中进行应用，这个方法还远远不够。因此在本章中，本文继续对MAPF问题进行扩展，以进一步缩小多智能体路径规划学术研究与现实场景应用之间的差距。具体来说，本章的扩展主要有两个部分：异构多智能体系统中的规划以及在线多智能体路径规划。

无论是目前大多数关于MAPF问题的研究还是本文第三章的提出的方法，都是集中在同质（homogeneous）的多智能体系统中的，即系统中所有智能体完全相同，各模型参数完全一致。而在某些实际场景中，可能需要不同种类的机器人协作来完成复杂的任务，即系统中的智能体是异构的（heterogeneous）。目前解决异构多智能体系统中的MAPF问题的主要有两类方法，基于优化和基于搜索的方法。基于优化的方法一般采用非线性优化技术来生成符合智能体运动学约束的轨迹，如文献[71]中提出的在二维非凸环境中为异构四旋翼无人机群生成时间最优轨迹的方法，以及在文献[72]和文献[59]中提出的对无人机和非完整约束机器人更高效的路径规划方法。但是基于优化的方法存在不能保证算法完整性且存在优化时间过长的问题。基于搜索的方法包括文献[55]提出考虑不同大小的智能体MCCBS算法、文献[73]中提出的针对异构多智能体系统的考虑最多k步延迟的G-CBS算法。但基于搜索的方法虽然在运算时间上优于基于优化的方法，但是其在离散的无向图上进行规划而忽略了智能体自身的运动学模型，因此无法应用于现实工业场景中。

此外，上述方法都是针对一次性（One-shot）的MAPF问题，即算法只需要在任务开始前为当前智能体系统规划一次路径，智能体到达目标位置后就会一直停在原地，这类问题也被称为离线MAPF问题（Offline MAPF）。但在实际工业系统中，智能体在到达终点位置后会被分配一个新的任务，并要求其继续移动到新的目标位置。这种智能体在执行过程中不断被分配新的寻路任务的问题被称为在线MAPF问题（Online MAPF），在有些文献[60]中也称为全生命周期MAPF问题（Lifelong MAPF）。对于在线MAPF问题，文献[74,75]将问题分解为一连串的MAPF实例并在每一个时间步为所有智能体重新规划路径。尽管使

用了增量搜索的技术，但该方法在每个时间点都为所有智能体重新规划还是非常耗时的，因此也无法扩展到包含大量智能体的系统中。文献[63,76] 中提出将在线 MAPF 问题分解为一连串离线 MAPF 实例，在每个时刻只为具有新目标的智能体重新规划路径。该类方法可以扩展到更多的智能体系统中，但是它需要地图具有一个特殊的结构来保证算法完整性，只针对部分在线 MAPF 实例有效。

针对上述问题，本章对异构多智能体系统的在线 MAPF 问题进行研究。首先本章对要研究的问题给出了数学定义，并拆分为异构智能体系统规划和在线多智能体规划两个子问题分别进行解决。对于异构智能体系统，本章扩展了3.4节中提出的基于冲突搜索的方法：得益于该方法上层搜索使用的车身冲突树并不直接考虑智能体的运动学模型，其无需修改仍能在异构多智能体系统中进行冲突的搜索和解决；而在下层搜索中本文借鉴了时空混合 A* 搜索算法中离散动作空间的思想，提出了一个通用模型的单智能体规划器，其具备在连续空间中为不同模型的智能体规划满足运动学约束的最短路径的能力。而对于在线 MAPF 问题，本文使用了滚动时域碰撞解决算法（Rolling-Horizon Collision Resolution，RHCR）[62] 并针对连续工作空间进行了算法优化。该算法不再每个时刻都重新为所有智能体规划路径，而是将在线 MAPF 问题拆分成一连串的时间窗口 MAPF（Windowed MAPF）实例，并每隔一定的时间重新规划一次路线，交替进行路径规划和路径执行。最后，本章进行了异构多智能体系统 MAPF 实验以验证上述方法的可行性，测试了该方法在不同场景中表现并与现有先进方法进行了对比。

综上，本章提出的算法贡献如下：

1. 本章提出了一个适用于异构多智能体系统的基于冲突搜索方法。该方法可以在连续空间内为具有任意运动学模型、尺寸大小和形状的智能体生成可行的规划方案，同时具备基于搜索方法求解时间短的优势。

2. 本章在基于冲突搜索框架内应用了滚动时域碰撞解决算法，每隔一定的时间调用时间窗口 MAPF 规划器重新规划一次路线。通过交替进行路径规划和路径执行，解决了在线路径规划问题中存在寻路任务流的问题并降低了智能体的闲置时间。

3. 本章进行了仿真实验，并与现有先进方法进行了对比。实验结果显示，本章提出的方法可以为多达 100 个异构智能体进行运动学可行的路径规划，并保持 90% 以上的算法成功率。而对于在线 MAPF 问题，本章提出的方法可以将在线 MAPF 规划器的速度提升六倍而几乎不影响规划方案的质量。

## 4.2 问题定义

在3.2节中，本文将经典 MAPF 问题扩展到了连续工作空间上并考虑智能体的运动学约束，本章在此基础上进一步扩展。考虑异构多智能体系统中的在线 MAPF 问题。具体来说，此处对问题扩展分为两点：异构系统，即系统中的智能体不再是完全相同的，而是具有不同的大小形状、速度限制甚至运动学模型；在线规划，即存在一个任务流，算法需要在线为新加入的任务进行规划。本文将这个问题称为**异构系统中的在线 MAPF 问题**，该问题的定义如下。

考虑一个包含 $N$ 个智能体 $a_1, a_2, \ldots, a_N$ 的异构多智能体系统在一个二维连续的工作空间 $\mathcal{W}$ 中运行。障碍物区域 $\mathcal{O}$、可通行区域 $\mathcal{F}$、智能体状态 $\mathbf{z}$ 都与前文定义一致。$m_i$ 表示异构系统中智能体 $a_i$ 的模型，智能体的不同模型意味着智能体的车身形状函数 $\mathcal{R}_i(\mathbf{z}^i)$、移动速度 $v_i$、智能体控制输入集合 $\mathcal{U}_i$ 和运动学模型 $f_i(\mathbf{z}^i, u), u \in \mathcal{U}_i$ 都不尽相同。

在多智能体路径在线规划系统中，需要设置一个任务分配器负责把任务流中的新增寻路任务分配给系统中具体的某个智能体进行执行。本文考虑任务分配器不在算法控制范围内的情况，即任务分配器和路径规划算法是解耦的。这样做的好处在于在线 MAPF 算法可以适用于不同场景，对于特定场景，可以使用一个分层框架将特定场景中的任务分配器[60,63,75]和不依赖于场景的在线 MAPF 算法结合起来应用。

在线 MAPF 问题中，对于系统中的智能体 $a_i$，其寻路任务为一个目标状态序列 $\mathbf{g}_i = \{\mathbf{g}_i[0], \mathbf{g}_i[1], \ldots\}$，其中每个目标状态都是可行的，即 $\mathbf{g}_i[k] \in \mathcal{F}, \forall 0 \leq k \leq |\mathbf{g}_i|$。随着多智能体系统的运行，任务分配器会不断地在目标状态序列 $\mathbf{g}_i$ 中加入新的寻路目标任务。同时，智能体也需要从当前状态（一般是上个任务的目标状态）规划出一条可行路径 $\pi_i^k$ 到达目标状态 $\mathbf{g}_i[k]$，其中路径可行的定义与前文类似，但是本文要求智能体在到达目标状态后不能保持静止，而是应当继续执行目标状态序列 $\mathbf{g}_i$ 中的下一个寻路任务 $\mathbf{g}_i[k+1]$，除非智能体 $a_i$ 已经执行完 $\mathbf{g}_i$ 中的所有任务。此外，在线 MAPF 算法同样应当保证在任意时刻系统中的任意两个智能体不发生冲突，即

$$\mathcal{R}_i\left(\pi_i[t]\right) \cap \mathcal{R}_j\left(\pi_j[t]\right) = \emptyset, \ \forall t \geq 0, i \neq j \tag{4.1}$$

与离线 MAPF 问题不同，用于评价在线 MAPF 算法性能的指标主要有平均路径成本（average flowtime），每百秒吞吐量（throughput），程序运行时间（planning time）三种。假设截止到截止时刻 $T$，系统中智能体 $a_i$ 完成了任务集 $\mathcal{T}_i$ 中前 $K_i$ 个任务，上述三个指标的具体定义如下：

- 平均路径成本是目前所有智能体完成的任务的平均路径长度，即

$$\frac{1}{N}\sum_{i=1}^{N}\frac{1}{K_i}\sum_{t=1}^{T}|\pi_i[t]-\pi_i[t-1]| \tag{4.2}$$

- 每百秒吞吐量是指系统截止 $T$ 时刻每百秒平均完成的任务数，即

$$\frac{100}{T}\sum_{i=1}^{N}K_i \tag{4.3}$$

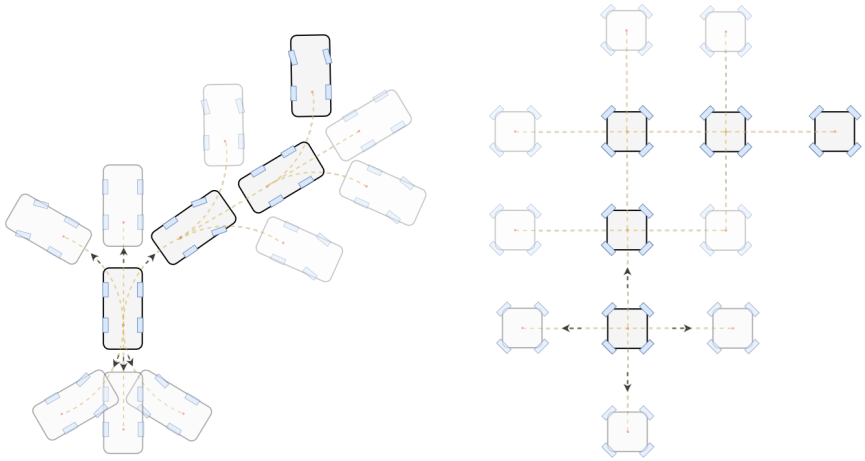
- 程序运行时间是指到 $T$ 时刻为止每次在线 MAPF 算法的平均运行时间。

其中，平均路径成本和每百秒吞吐量主要用来评价在线 MAPF 算法的求解质量，程序运行时间则用来评价在线 MAPF 算法的运行效率。

## 4.3 异构多智能体系统的基于冲突搜索

在本节中，本文把3.4节中介绍的基于冲突搜索方法扩展到异构多智能体系统中。针对阿克曼模型的基于冲突方法使用了一个分层搜索框架，上层使用了车身冲突树来解决连续空间内的智能体间的碰撞，下层搜索使用时空混合 A* 算法来进行单智能体符合运动学约束的路径重规划。可以注意到，异构多智能体系统中 MAPF 问题的核心在于每个智能体都拥有不同的模型，因此求解器需要规划出符合智能体各自模型的可行路径。对于上层车身冲突树而言，其只判断规划方案中的路径之间是否存在车身冲突而不关心每个智能体具体的运动学模型。因此在扩展到异构多智能体系统中时，车身冲突树只需在检测路径间车身冲突时考虑智能体不同的车身形状函数 $\mathcal{R}_{\rangle}(\mathbf{z}')$ 即可。而对于下层搜索而言，时空混合 A* 算法只适用于阿克曼转向模型的智能体。借鉴混合 A* 离散控制输入空间的思想，本节提出了一种通用模型的单智能体路径规划算法，它可以为具有任意模型的机器人在连续工作空间中生成可行的路径。下面本文对这个方法进行详细介绍。

对于上层搜索中的车身冲突树，其算法流程基本无需改变即可直接应用于异构多智能体系统中。唯一需要注意的是，由于异构系统中智能体 $a_i$ 具有各自的模型 $m_i$，因此对于车身冲突生成的约束，需要进行如下修改。车身冲突仍然定义为 $(a_i, a_j, t)$，该冲突表示智能体 $a_i$ 和 $a_j$ 会在 $t$ 时刻发生碰撞，即 $\mathcal{R}_i(\pi_i[t]) \cap \mathcal{R}_j(\pi_j[t]) \neq \emptyset$。如果节点 $N$ 中存在一个冲突 $(a_i, a_j, t)$，算法会生成两个冲突条件：$(a_i, N.\pi_j[t], m_j, t)$ 和 $(a_j, N.\pi_i[t], m_i, t)$。前一个约束意味着智能体 $a_i$ 在时间步 $t$ 需要避开模型 $m_j$ 的车身区域 $\mathcal{R}_j(N.\pi_j[t])$，从而避免冲突 $(a_i, a_j, t)$ 的发生；后一个约束也是类似，即智能体 $a_j$ 在时间步 $t$ 需要避开模型 $m_i$ 的车身区域 $\mathcal{R}_i(N.\pi_i[t])$。接着节点 $N$ 的两个子节点被生成，每个子节点包含上述两个约束条件之一。在每个子节点中算法对新增约束相关的智能体执行下层搜索以对其路径进行重



(a) 阿克曼转向模型 (b) 全向移动模型

图 4.1 不同运动模型的路径扩展。（本图有意放大了每个节点的扩展长度以便示意）

规划，其他智能体的路径则继承节点 $N$ 中的路径。最后算法将两个子节点加入车身冲突树中，并继续对车身冲突树进行最佳优先搜索寻找下一个处理的节点 $N$。

接下来本文详细介绍通用模型的单智能体路径规划算法，这个算法流程与3.4.2节中的时空混合 A* 算法类似，但本章进行了一些关键性的修改使该算法适用于异构智能体系统中的任意模型。在时空混合 A* 算法中，不难发现与阿克曼转向模型有关的部分主要集中在三个函数中：碰撞检测函数（Check_Collision）、子节点扩展函数（Get_ChildNode）和启发式函数（Admissible_Heuristic）。因此，本节介绍的通用模型单智能体路径规划算法，对每个模型设置好各自合理的碰撞检测、子节点扩展和启发式函数后，就可以为具有任意模型的智能体在连续工作空间中生成可行的路径。下面依次介绍这三个函数：

(1) 碰撞检测函数：对于一个开放列表中的节点，一个基本的问题就是确定该节点状态是否合法。在异构系统中，即使处于同一状态下，由于智能体不同模型的大小和形状不同导致其占据的区域 $\mathcal{R}_i(\mathrm{z}^i)$ 也是不同的。因此异构系统中的智能体模型需要自定义碰撞检测函数来确定当前节点状态是否位于可行工作区域 $\mathcal{F}$ 内，并符合所有当前时间点的上层约束条件。具体来说，一个关于模型 $m_i$ 的智能体 $a_i$ 的节点状态 $N.state$ 是可行的当且仅当：$\mathcal{R}_i(N.\text{ state}) \cap C = \emptyset, \forall C \in \mathcal{C}_{\text{N.time}}$ 且 $\mathcal{R}_i(\text{ N.state}) \subset \mathcal{F}$，此处 $\mathcal{C}_{\text{N.time}}$ 表示关于当前智能体的约束集中约束时间为 $N.time$ 的所有约束。

(2) 子节点扩展函数：由于不同的模型具有各异的运动学约束，节点在开放列表中的扩展方式自然也需要进行不同的设置。考虑到异构智能体系统 MAPF 问题中工作空间的连续性和时间的离散性质，本文参考文献[69] 中运动基元（motion primitives）的概念，将模型的控制输入离散为动作集合 $\mathcal{U}$。对于一个节点 $N$，算法尝试使用动作

集合 $\mathcal{U}$ 中的每一个动作 $u$ 来扩展节点状态 $N.state$ 并生成子节点 $N'$。当使用离散运动时，本文希望每次扩展步长较小以便最终可以结合为一条精细的路径。因此，算法限制节点每次扩展的长度小于该智能体的模型在一个时间步能移动的最大距离 $v_{max} \cdot T_{step}$，同时也应当保证智能体在两个相邻时间的状态之间车身区域存在重叠 $\mathcal{R}_i(\pi_i[t]) \cap \mathcal{R}_i(\pi_i[t+1]) \neq \emptyset$。这一限制保证了在大多数情况下，只要在任意时间步时路径上的节点状态是合法的，那么智能体按照路径在节点之间移动也是不会有碰撞的。因此，其子节点的扩展应该遵循：

$$\begin{aligned} &N'.state = N.state + f_i(N.state, u), \forall u \in \mathcal{U} \\ &s.t. \quad |N'.state - N.state| \leq v_{\text{max}} \cdot T_{\text{step}} \end{aligned} \tag{4.4}$$

其中，$f_i$ 是具有模型 $m_i$ 的智能体 $a_i$ 的运动学方程。

图4.1展现了两个常见的模型，全向移动模型（omnidirectional）和第三章提到的阿克曼转向模型（Ackermann steering），的动作集合。全向移动模型有 5 个可用动作，而阿克曼转向模型有 7 个动作，包括原地等待动作。需要强调的是，相同模型的动作集合是不唯一的，动作集和扩展步长的设置只需满足前述限制条件，即可应用在通用模型的单智能体路径规划框架内。

(3) 启发式函数：对于连续工作空间内不同模型的移动智能体，算法也应该设置不同的启发式函数来对路径探索提供指导。启发式函数的设置需要具备可接纳性（admissible），即 $N.h \leq |N'.state - N.state| + N'.h$。举例来说，对于全向移动模型，原始 A* 算法中使用的欧氏距离或者曼哈顿距离就是一个很好的选择；而对于阿克曼模型智能体，Reeds-Shepp 路径长度[67] 就是一个更为合适的选择。

除此之外，由于在异构多智能体系统的基于冲突搜索方法中仍然沿用了分层搜索的框架，因此本文在3.4.3节提出的顺序规划技术仍然适用于该方法。即，将系统内的 $N$ 个智能体分为 $K_b$ 个批次，先依次解决每一批次的 MAPF 子问题，再将所有批次的规划结果整合起来最终得到整个 MAPF 问题的规划方案。对于第 $b$ 个批次，所有后续批次中的智能体将在该批次规划时被忽略，而前序批次中已经规划完成的智能体的路径将被当做工作空间内的动态障碍物进行规避。

## 4.4 在线多智能体路径规划方法

### 4.4.1 时间窗口 MAPF 规划器

在解决在线 MAPF 问题之前，本小节首先研究一种被称为时间窗口 MAPF（Windowed MAPF）规划器的算法。与前述离线 MAPF 问题的求解算法相比，时间窗口 MAPF 求解器在两个方面有差异：i）每个智能体的路径都需要访问一个目标状态序列。ii）智能体路径只需要在最初的 $w$ 个时间步内保证是无碰撞的。下面本文依次对这两个方面进行研究。

前述基于冲突搜索的方法只能处理每个智能体只有一个目标位置的情况，而时间窗口 MAPF 问题要求路径规划算法为每个智能体规划出一条依次访问序列 $\mathrm{g}_i$ 中所有目标状态的路径。尽管存在这种差别，前述通用模型的单智能体规划器只需要稍作修改即可应用于时间窗口 MAPF 求解器中的下层搜索。如算法5所示，本文将通用模型的单智能体路径规划算法应用于时间窗口 MAPF 求解器的下层规划。对于每个开放列表节点 $N$，算法额外添加一个属性 $N.label$，表示当前状态已经访问过的目标序列 $\mathrm{g}_i$ 的状态数量。例如，$N.label = 2$ 表示当前路径已经访问了 $\mathrm{g}_i[0]$ 和 $\mathrm{g}_i[1]$ 两个目标状态，但尚未到达 $\mathrm{g}_i[2]$ 目标状态。同时对于节点的启发式函数 $N.h$，算法应当按照公式4.5计算出完成全部目标状态的剩余距离，而不能只计算到当前目标状态的距离。算法的具体流程中，首先创建 $label = 0$ 的根节点 $Root$，并将其加入开放列表（OPEN）中。当开放列表不为空的时候，算法选取具有最小 $f$ 值的节点 $N$ 进行扩展，如果节点 $N$ 的状态已经接近当前目标状态 $\mathrm{g}_i[N.label]$ 时，算法进行分析性扩展；若扩展路径可安全到达目标点无碰撞，则节点 $N.label$ 会递增，即继续前往目标状态序列中的下一个目标状态。若 $N.label$ 等于目标状态序列 $\mathrm{g}_i$ 的总状态数量 $|\mathrm{g}_i|$，意味着路径以依次访问过所有目标，算法终止并返回路径；否则，算法将继续生成符合时空约束条件和智能体模型约束的子节点进行扩展。

时间窗口 MAPF 求解器的另一个特点是其只需解决智能体在前 $w$ 个时间步中的碰撞。前述基于冲突搜索的离线 MAPF 算法可以很容易地满足时间窗口 MAPF 求解器的这个要求：算法只需要关注前 $w$ 个时间步中发生的碰撞，而忽视对于大于 $w$ 时间步的碰撞，并假定每个智能体都按照其最短路径访问目标列表中的所有状态。具体到基于冲突算法流程中，只需要修改车身冲突检测部分，令算法只检测 $N.solution$ 规划方案中前 $w$ 个时间步内的冲突 $(a_i, a_j, t), t \leq w$，然后解决其中最早的一个车身冲突。而对于超过 $w$ 个时间步的冲突，算法会将其保留在规划方案中不进行处理。由于时间窗口 MAPF 问题中基于冲突搜索需要解决的碰撞较少，它生成的上层车身冲突树也相对较小，故而该方法较离线 MAPF 问题中的基于冲突搜索方法运行得更快。

**算法 5**: 时间窗口 MAPF 下层规划器（以具有模型 $m_i$ 的智能体 $a_i$ 为例）

Root.state$\leftarrow s^i$; Root.t$\leftarrow 0$;
Root.label$\leftarrow 0$;
OPEN$\leftarrow \{$Root$\}$;
while OPEN$\neq \emptyset$ do
    $N$ arg min $N'$. f, $\forall N' \in$OPEN;
    if $N$.state near g$_i[N$.label$]$ then
        Invoke model $m_i$'s Analytic_Expansion($N.state$, g$_i[N$.label$]$);
        $\pi_i \leftarrow$ Backtracking search from g$_i[N$.label$]$;
        if No collision in $\pi_i$ then
            $N$.label $\leftarrow$ $N$.label$+1$;
        end
        if $N$.label $=$ |g$_i$| then
            return $\pi_i$;
        end
    end
    Invoke model $m_i$'s Get_ChildNode($N$.state, $\mathcal{U}$);
    foreach s $\in$ States do
        $N'$.state $\leftarrow$ s; $N'$.t $\leftarrow$ $N$.t $+1$;
        Invoke model $m_i$'s Check_Collison($N'$.state ,$N'$.t);
        if $N'$.state is valid for $a_i$ then
            $N'.h \leftarrow$ Invoke model $m_i$'s Admissible_Heuristic($N'$.state,$g^i$) using Equation 4.5;
            Update $N'.g, N'.f$;
            if $N'$.state not appear in OPEN then
                OPEN $\leftarrow$ OPEN $\cup \{N'\}$;
            end
    end
end
return $\emptyset$;

### 4.4.2 滚动时域碰撞解决算法

下面本文研究在线多智能体路径规划问题。与离线 MAPF 算法相比，在线 MAPF 问题中智能体 $a_i$ 的目标状态不再只有一个，而是有一个目标状态序列 $\mathrm{g}_i$。智能体需要依次到达这些目标状态，同时随着任务进行，该目标序列还会增加新的目标状态。本文使用滚动时域碰撞解决算法（Rolling-Horizon Collision Resolution，RHCR）来解决该问题，该算法将在线 MAPF 问题拆分成一连串的时间窗口 MAPF 实例，并每隔一定的时间重新规划一次路径。滚动时域碰撞解决算法有两个用户指定的参数：时间跨度 $w$ 和重规划周期 $h$。时间跨度 $w$ 指定时间窗口 MAPF 求解器必须能够解决当前 $w$ 个时间步内的智能体间碰撞，而重规划周期 $h$ 则规定了时间窗口 MAPF 求解器需要每隔 $h$ 个时间步为当前系统中的智能体进行重新规划路径。为了避免智能体之间发生碰撞，时间窗口 MAPF 求解器需要比时间跨度 $w$ 更加频繁地重新规划路径，即 $w \geq h$。

对于一个从 $t$ 时间步开始的时间窗口 MAPF 实例，该实例所覆盖的时间窗口应为 $[t, t+h]$。滚动时域碰撞解决算法会首先更新系统内每个智能体 $a_i$ 的当前状态 $s_i$ 和目标状态序列 $\mathrm{g}_i$。然后算法会计算出智能体到达序列 $\mathrm{g}_i$ 中所有剩余目标状态所需的最短距离 $d$：

$$d = \mathrm{dist}\left(s_i, \mathrm{g_i}[0]\right) + \sum_{j=1}^{|\mathrm{g_i}|-1} \mathrm{dist}\left(\mathrm{g_i}[j-1], \mathrm{g_i}[j]\right) \tag{4.5}$$

其中 $dist$ 函数用于计算两个状态之间的距离，对于异构系统中不同模型的智能体可以使用其模型对应的启发式函数来进行近似计算，$|\mathrm{g_i}|$ 表示目标状态序列 $\mathrm{g}_i$ 当前的总元素个数。因此对于智能体 $a_i$，若其以 $v_i$ 的速度移动，则其最少需要 $d/v_i$ 个时间步才能完成所有目标状态。若有 $\frac{d}{v_i} < h$，则意味着智能体 $a_i$ 可能在当前时间窗口内完成对所有目标位置的访问并在下一个时间窗口 MAPF 实例开始之前处于空闲状态。为避免这种情况，降低智能体的闲置率，算法会给 $a_i$ 分配新的目标状态直到 $\frac{d}{v_i} \geq h$。一旦所有智能体的完成目标序列最短距离 $d$ 都满足要求不再需要更新，滚动时域碰撞解决算法就会调用时间窗口 MAPF 规划器为系统内所有的智能体寻找路径，使其按照目标序列 $\mathrm{g}_i$ 中的顺序从当前位置依次移动到所有目标状态，并保证前 $w$ 个时间步内没有碰撞发生。随后系统开始执行生成的规划方案，每个智能体沿着当前路径移动 $h$ 个时间步，并在其目标序列 $\mathrm{g}_i$ 中删除已经完成访问的目标状态。

对于时间跨度 $w$ 与重规划周期 $h$ 的设置问题，可以根据不同的场景自定义设置。当重规划周期 $h$ 确定时，重规划周期 $w$ 不应过于接近 $h$。当 $w = h$ 时，每次时间窗口 MAPF 规划器都只考虑规划方案前 $h$ 个时间步的碰撞，这意味着当前时间窗口的规划方案将完全不考虑下一时间窗口后的智能体间碰撞。当下一时间窗口开始后，智能体的状态可能并

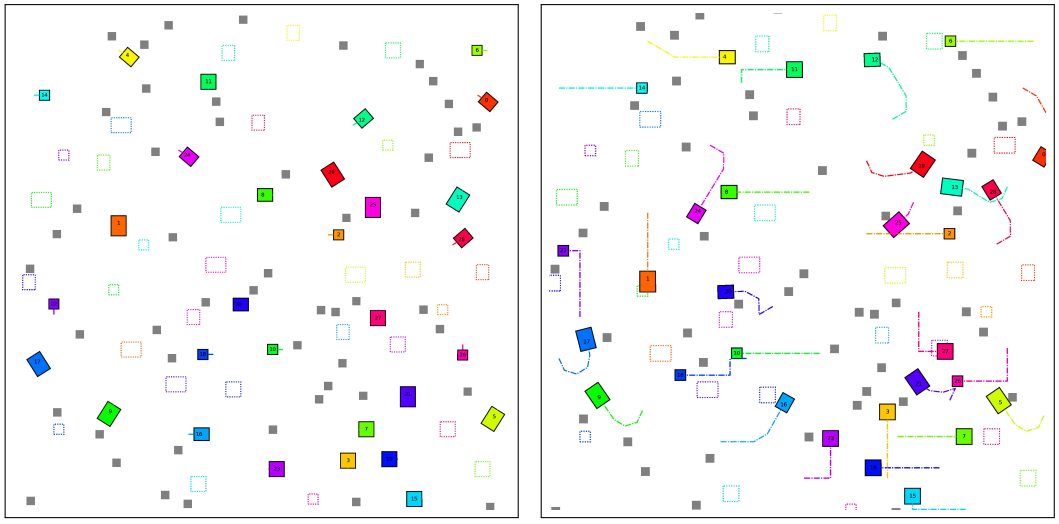
不是当前时刻的最优解，而规划器需要重新解决智能体之间的约束。而另一方面，当时间跨度远大于重规划周期，即 $w \gg h$ 时，算法中自然不会存在时间窗口规划器考虑不周的情况。但是由于重规划周期 $h$ 相对于规划时间跨度 $w$ 过小，实际上每次规划的方案只有一小部分被执行，导致大量的程序搜索时间实际上是浪费的。结合下文所做实验结果，当时间跨度 $w$ 选择 2 到 4 倍重规划周期 $h$ 时，可以兼顾规划方案的质量与程序运算时间。

## 4.5 实验

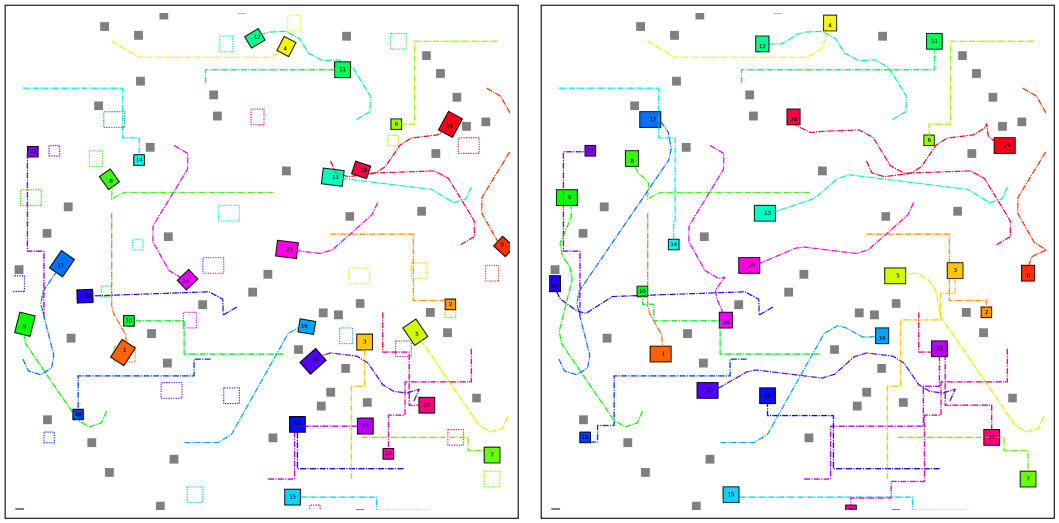
本节使用 C++ 实现了适用于异构多智能体系统中的在线 MAPF 方法，其中使用 Boost 库用来进行数值计算，使用 OMPL 库来计算部分智能体模型的启发式函数。本文在3.6节提出的连续空间内的基准测试集上进行仿真实验，首先在空旷和障碍场景中测试了本章提出的方法扩展到大规模异构多智能体系统上的表现，随后在仓储场景中将本章提出的方法与文献[59] 中提出的方法进行了对比实验。最后，本文构建了一个在线 MAPF 问题的实验环境，进行了长时间的在线规划问题的测试。所有的程序都是在一台配有 Intel i7-8700@3.20GHz 处理器、8G 内存的运行有 Ubuntu 16.04 操作系统的计算机上执行的。本章提出方法的源代码已经在 Github 上开源：https://github.com/zijinoier/Hetero-MAPF。

### 4.5.1 异构系统中性能测试

首先，本文考察本章所提出的方法在离线异构多智能体系统中的求解性能和规划方案质量。本文在3.6节中的 300 米 ×300 米大小的空旷和障碍场景中分别进行实验，以衡量算法在不同情况下的性能表现。值得强调的是，在障碍场景中工作空间内存在 100 个随机放置的 2 米 ×2 米大小的障碍物。在每个场景中的问题实例内，本文都创建一个包含 100 对各异状态的集合。在每次实验过程中，每个智能体都会从这个集合中随机选择一对状态，并将其中一个状态设置为起始状态，另一个则作为目标状态。该实验考虑异构系统中包含 4 种不同模型的智能体：两个具有阿克曼运动学模型的智能体、两个具有全向运动模型的智能体。本文假设尺寸越小的智能体模型具有更高的运动速度。两种阿克曼模型的智能体中，一种尺寸为 3 米 ×2 米，最大移动速度为 $v_{max} = 2m/s$；另一种尺寸为 2 米 ×1.5 米，最大移动速度为 $v_{max} = 2.5m/s$。而至于两种全向运动模型的智能体，一种尺寸为 3 米 ×3 米，最大移动速度为 $v_{max} = 2m/s$；另一种全向模型尺寸为 2 米 ×1.5 米，最大移动速度也是 $v_{max} = 2.5m/s$。图4.2中展示了该实验设置中包含 30 个异构智能体的系统在 300 米 ×300 米的障碍场景实例中的一个规划方案。



(a) $t = 0s$ (b) $t = 2s$

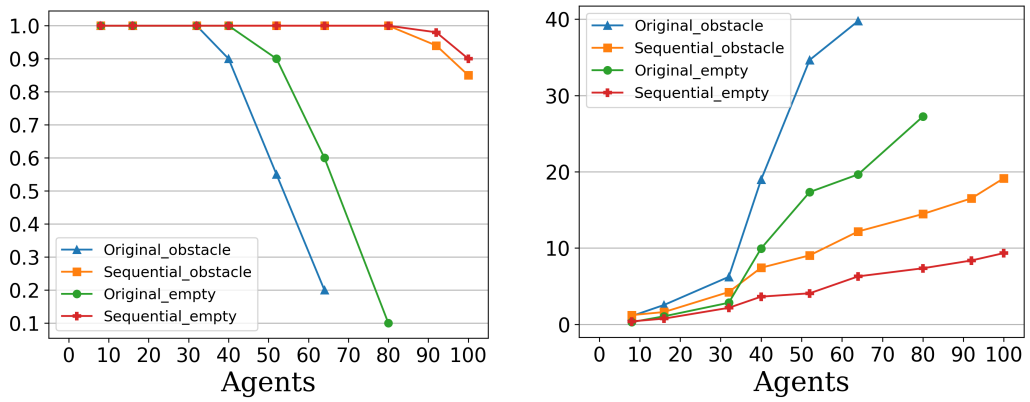


(c) $t = 4s$ (d) $t = 8s$

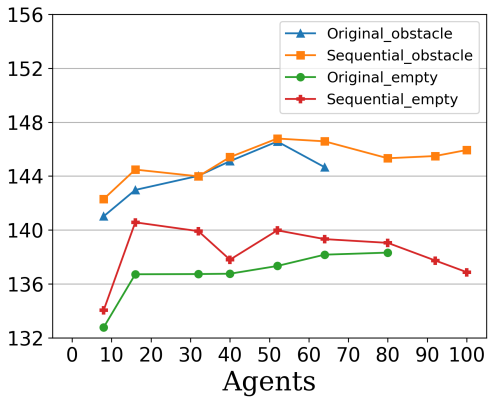
图 4.2 包含 30 个异构智能体的系统在一个 300 米 ×300 米障碍场景实例的规划方案示意图

实验测试本章提出的两种算法：原版异构多智能体系统的基于冲突搜索算法和其顺序规划的版本，其中顺序版本的每批次包含的智能体数量设置为 $\lceil N/K_b \rceil = 10$ 个。在每个场景中，实验对两种算法从异构系统内包含 10 个智能体系统开始，一直测试到系统内包含 100 个智能体为止。在每个智能体数量下，本文对两个算法各进行 10 次运行，并将每次程序运行时间限制设定为 60 秒，若程序在 60 秒内未能运行完成给出可行解，则认为本次运行失败。本文计算了两种方法在不同场景下的成功率、平均算法运行时间、规划方案的平均路径成本，结果如图 4.3所示。

如图 4.3a所示，无论是原版还是顺序规划版本，在智能体数量较少时，算法成功率都能达到 100%。当机器人数量超过 50 个时，由于车身冲突的增加导致程序搜索时间变长，原始版本的规划成功率开始下降，在系统内超过 70 个异构智能体的情况下，原版算法的




(a) 算法成功率 (b) 程序运行时间



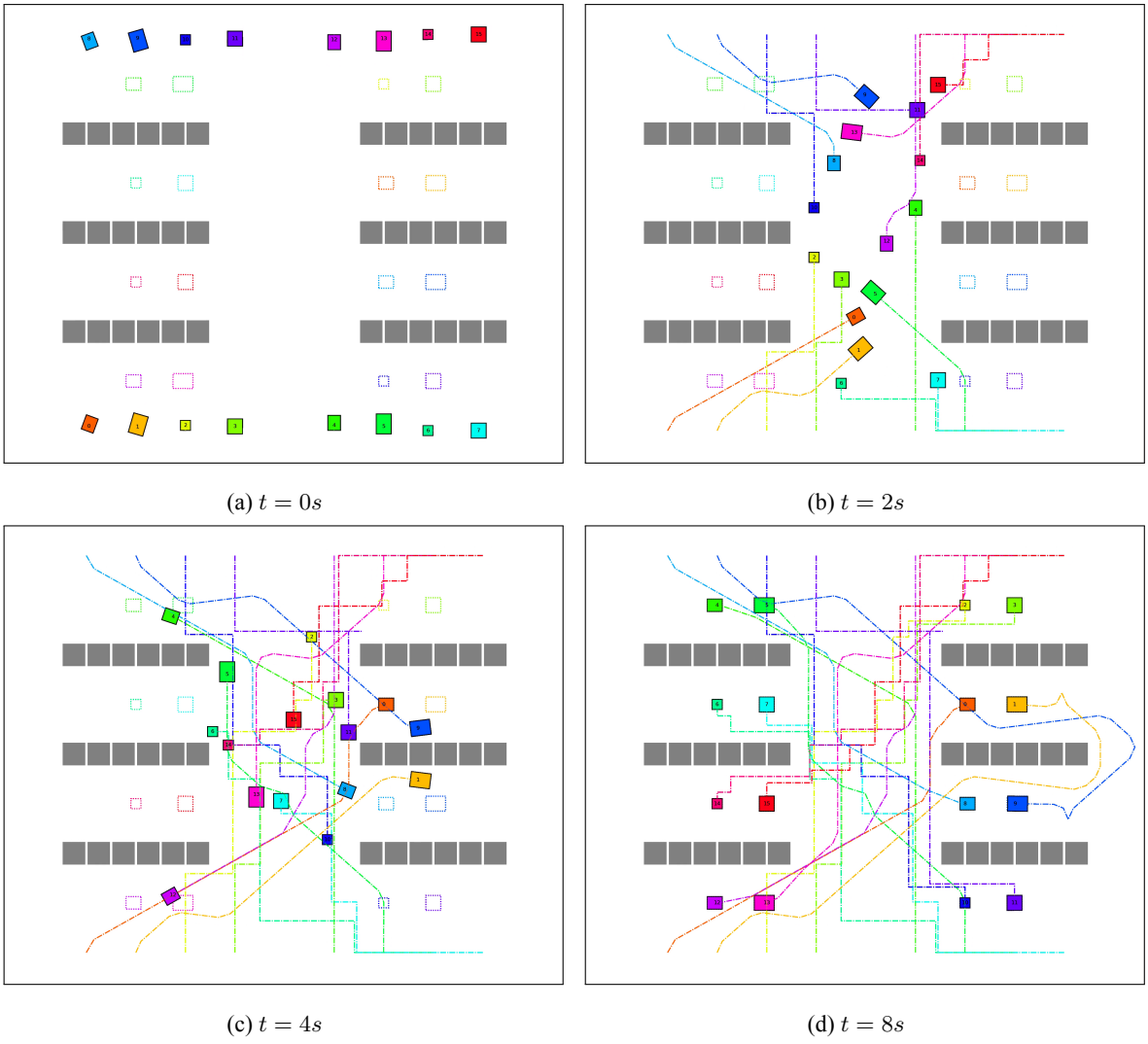

(c) 平均路径成本

图 4.3 在 300 米 ×300 米场景中两种算法的实验结果。其中蓝、橙、绿、红线分别代表障碍场景内原版方法（Original_obstacle）和障碍场景内顺序方法（Sequential_obstacle）、空旷场景内原版方法（Original_empty）、空旷场景内顺序方法（Sequential_empty）。

成功率降到了 20% 以下，失败情况都是由运行超时导致的。于此相反的是，顺序规划版本在系统内包含 100 个智能体的情况下也保持了 80% 以上的成功率，对于大规模系统的扩展性更好。同时，由于可行工作空间更大、场景相对简单，空旷场景的算法成功率要略高于同智能体情况下的障碍场景。

程序运行时间的结果如图 4.3b所示。随着系统内智能体数量增加，原版方法的运行时间呈指数级上升，相比而言，顺序规划版本的方法运行时间偏短且基本呈线性增长。这是由于顺序规划版本通过不同批次智能体的解耦，缩小了车身冲突树的扩展范围，使得最终运行时间较短。至于图 4.3c所示的规划方案平均路径成本，由于空旷场景中智能体不需要避让静态障碍物，因此空旷场景的规划方案平均路径成本普遍小于相同智能体数量在障碍场景中的规划方案。同时，顺序规划版本在相同情况下表现比原版要差一些，但是该指标的差距被控制在了原版平均路径成本的 2.8% 以内，差距并不大。

与 3.5.3小节中同构智能体系统的性能测试结果相比，本节的实验结果在趋势上与上



(a) $t = 0s$ (b) $t = 2s$

(c) $t = 4s$ (d) $t = 8s$

图 4.4 包含 16 个异构智能体的系统在一个 120 米 ×100 米仓储场景实例的规划方案示意图

一节基本相同，由于都是用了分层的基于冲突搜索框架，因此无论是在同构或者异构多智能体系统中，本文提出的方法都可以在 300 米 ×300 米的场景中为高达 100 个智能体进行满足运动学约束的路径规划，并保持 90% 以上的成功率。同时从实验结果中可以发现，由于异构系统中的全向移动模型比阿克曼更容易进行路径搜索，因此在相同场景中，包含全向模型的异构系统甚至比全由阿克曼模型组成的同构系统路径规划时间更短。此外，两个性能测试都说明，本文提出的顺序规划方法同构牺牲少量的规划方案质量质量来实现运行时间的大幅减少和成功率的大幅提高是值得的。

### 4.5.2 异构系统中对比测试

仓储环境中的货物运送是 MAPF 问题，尤其是在线 MAPF 问题的重要应用场景之一。Li 等人在文献[59] 中提出了一个高效的基于优化的方法来生成符合智能体运动学约束的路径规划方案，该方法具体可分为两个版本的求解器：耦合版（下称 Li's coupled）和优先

级版（下称 Li's prioritized）。优先级版求解器相比于耦合版的规划器具有更短的计算时间，但其生成的规划方案质量会有所损失。

本节在仓储场景中将本章提出的方法与上述两个求解器进行对比，由于 Li 等人的方法只针对离线 MAPF 问题，因此本节在离线异构 MAPF 场景下进行对比试验。仓储场景的大小为 120 米 ×100 米，其中有 6 个 30 米 ×6 米的货架被放置在场地中央，智能体在移动时需要注意避让，不能与这些货架发生碰撞。场景中的异构系统同样包含 4 种不同模型的智能体：两种阿克曼模型的智能体，一种尺寸为 3 米 ×2 米，$v_{max} = 2m/s$；另一种尺寸为 2 米 ×1.5 米，$v_{max} = 2.5m/s$。以及两种全向运动模型的智能体，一种尺寸为 3 米 ×3 米，$v_{max} = 2m/s$；另一种全向模型尺寸为 2 米 ×1.5 米，$v_{max} = 2.5m/s$。图 4.4中展示了包含 16 个异构智能体的系统在该仓储场景中的一个规划方案。

本文在仓储场景中创建了一个均匀分布在可行区域内的 40 对各异状态集合。在每次实验过程中，每个智能体都会从这个集合中随机选择一对状态，作为起始和目标状态。同时集合保证任意两个智能体的起始和目标状态均不会发生碰撞。在 Li 等人的耦合版规划器中，一组智能体的数量被设置为 3，其他规划器被设置为程序默认值。本节使用顺序规划版本的异构多智能体系统的基于冲突搜索算法进行对比实验（下称 Ours），每批次中包含的智能体数量同样设置为 3 个。本文在场景内包含 20 个、30 个和 40 个智能体的情况下分别进行了对比实验。每个规划器的运行时间都被限制在了 60 秒，若算法在 60 秒内未能运行完成给出可行解或给出的规划方案内存在碰撞，则认为本次求解失败。在每次测试中，本文对每个方法进行 10 次试验，并计算算法成功率、平均算法运行时间、规划方案的平均路径成本，如表 4.1所示。

表 4.1 仓储场景中规划方案比较

| 智能体数量 | 方法 | 算法成功率 (%) | 程序运行时间 (秒) | 平均路径成本 (米) |
|---|---|---|---|---|
| 20 | Li's coupled | 100 | 4.04 | 76.73 |
| | Li's prioritized | 100 | 1.12 | 76.39 |
| | Ours | 100 | 1.87 | 66.69 |
| 30 | Li's coupled | 100 | 16.73 | 84.70 |
| | Li's prioritized | 100 | 8.06 | 80.15 |
| | Ours | 100 | 6.13 | 71.82 |
| 40 | Li's coupled | 60 | 57.32 | 91.45 |
| | Li's prioritized | 90 | 17.63 | 89.22 |
| | Ours | 100 | 11.22 | 69.90 |

随着异构系统内智能体数量的增加，本章提出的方法一直可以保持 100% 的成功率，但 Li 等人的方法在智能体数量达到 40 个时开始有失败的情况产生，所有规划失败均是由于他们求解器中的 ECBS 模块运行时间超过了 60 秒时限。同时可以看到，Li 等人的耦合版求解器随着智能体数量增加运行时间呈指数级增长，而其他两个规划器则保持了线性增加。当对 40 个异构智能体进行规划时，本章提出的方法只需要 11.2 秒就可以求出可行的规划方案，而 Li 等人的优先级版求解器需要 17.6 秒，耦合版求解器则需要 57.32 秒。此外，本章提出的方法在规划方案的最优性方面优势更加明显。随着智能体数量增多，仓库场景变得愈发拥挤时，本章方法生成的规划方案的平均路径成本几乎没有增加，而 Li 等人的两个版本的求解器生成的规划方案质量则有显著下降。当系统中有 20 个智能体时，Li 等人求解器的平均路径成本比本章提出的方法大 15.1%，而当系统内存在 40 个智能体时，该差异达到了 30.8%。

产生这个结果的原因在于 Li 等人的方法先在离散空间内规划路径，再在连续空间中进行轨迹优化。其最终规划方案非常依赖前端 ECBS 模块在栅格地图中规划的路径。而栅格地图是对连续工作空间进行离散采样，因此最终优化出的轨迹会存在不必要的绕路。同时，随着场景内智能体数量变多，Li 等人的方法生成的规划方案中智能体在路径中原地等待的行为也显著增加。但是值得注意的是，本章提出的方法忽略了智能体的加减速过程，并假设智能体移动速度是均匀的，而 Li 等人的方法则提供了详细的速度规划。

### 4.5.3 在线 MAPF 测试

本节进行在线多智能体路径规划场景的实验。本实验主要考察本章提出的方法在实际在线 MAPF 问题中的表现以及不同大小的时间跨度 $w$ 对于实验结果的影响。实验在 3.6节中的 300 米 ×300 米大小的障碍场景中进行，在障碍场景的工作空间内存在 100 个随机放置的 2 米 ×2 米大小的障碍物。与上节实验相同，本场景中的异构系统同样包含 4 种不同模型的智能体：两种阿克曼模型的智能体，一种尺寸为 3 米 ×2 米，$v_{max} = 2m/s$；另一种尺寸为 2 米 ×1.5 米，$v_{max} = 2.5m/s$。以及两种全向运动模型的智能体，一种尺寸为 3 米 ×3 米，$v_{max} = 2m/s$；另一种全向模型尺寸为 2 米 ×1.5 米，$v_{max} = 2.5m/s$。

本节使用 4.4节提出的在线多智能体路径规划方法进行实验，其中，滚动时域碰撞解决算法中的重规划周期 $h$ 设置为 10 秒，即每隔 10 秒会调用时间窗口 MAPF 规划器进行重规划。实验分别测试系统内智能体规模从 8 个开始，一直扩展到包含 52 个智能体的异构系统。系统内每个机器人都被分配有一个目标状态序列，目标序列保证了机器人完成所有目标状态的时间 $T_i \geq 1000$ 秒。本文对每个规模的智能体系统都进行长达 1000 秒的在

线多智能体路径规划实验，并计算每次试验的平均系统每百秒吞吐量和平均程序运行时间，实验结果如表4.2和表4.3所示。

表 4.2 在线 MAPF 场景中的系统每百秒吞吐量

| 智能体数量 | 8 | 16 | 32 | 40 | 52 |
|---|---|---|---|---|---|
| $w=5$ | 16.22 (-1.58%) | 31.98 (-2.14%) | 46.86 (-2.92%) | 53.84 (-2.78%) | 60.51 |
| $w=10$ | 16.46 (-0.72%) | 32.53 (-0.45%) | 48.17 (-0.20%) | 55.10 (-0.55%) | 61.95 |
| $w=20$ | 16.54 (-0.24%) | 32.65 (-0.09%) | 48.19 (-0.16%) | 55.22 (-0.34%) | 61.87 |
| $w=\infty$ | 16.58 | 32.68 | 48.27 | 55.41 | - |

* "-" 表示时间窗口 MAPF 规划器需要运行 60 秒以上。括号中的数字表示与时间跨度 $w=\infty$ 相比的吞吐量百分比差异。

表 4.3 在线 MAPF 场景中的程序运行时间 (秒)

| 智能体数量 | 8 | 16 | 32 | 40 | 52 |
|---|---|---|---|---|---|
| $w=10$ | $0.29\pm0.00$ | $0.53\pm0.00$ | $1.06\pm0.01$ | $2.95\pm0.03$ | $4.75\pm0.03$ |
| $w=20$ | $0.37\pm0.00$ | $0.68\pm0.01$ | $1.31\pm0.01$ | $3.67\pm0.03$ | $6.06\pm0.05$ |
| $w=40$ | $0.74\pm0.01$ | $1.31\pm0.01$ | $2.75\pm0.02$ | $7.57\pm0.08$ | $12.11\pm0.13$ |
| $w=\infty$ | $1.09\pm0.01$ | $2.54\pm0.02$ | $6.22\pm0.04$ | $18.96\pm0.15$ | - |

* "-" 表示时间窗口 MAPF 规划器需要运行 60 秒以上。"±" 后面的数字表示运行时间的标准差。

从实验结果中可以看出，在大多数情况下和时间跨度 $w=\infty$ 相比，较小的时间跨度 $w$ 对重规划时间也都有着很大的影响，而对每百秒吞吐量的影响则基本小于 1%。时间跨度 $w=\infty$ 意味着每次重规划时规划器都会为所有智能体规划出后续完成所有目标状态序列的避碰路径，这无疑会使得程序运行时间很长，尤其是在系统内智能体数量达到 52 个的时候，MAPF 规划器无法在 60 秒内完成重规划，在表格内以 "-" 显示。而适当减小到合适的时间跨度 $w$，滚动时域碰撞解决算法可以将规划器的速度提升六倍而几乎不影响规划方案的精度，即每百秒吞吐量。同时，合理设置较小的时间跨度也增加了算法对系统规模的可扩展性，对于系统内包含 52 个智能体的情况，$w=10$ 只需要 4.75 秒进行路径重规划，$w=20$ 也只需平均运行 6.06 秒即可得出结果。从实验结果来看，本文认为将滚动时域碰撞解决算法中时间跨度 $w$ 设置为重规划周期 $h$ 的 2-4 倍，可以兼顾算法的运行效率和求解质量。

## 4.6 本章小结

本章对异构多智能体系统的在线路径规划问题进行了研究。首先本章对该问题给出了严格的数学定义并将其拆分为异构智能体系统规划和在线多智能体规划两个子问题分别进行解决。对于两个子问题，本章分别提出了适用于异构多智能体系统的基于冲突搜索方法和滚动时域碰撞解决算法。上述方法可以使得系统内智能体持续参与移动，避免了过多限制时间，同时能够产生灵活的规划方案以应对不断到来的新目标位置。

本章随后进行了异构多智能体系统路径规划的实验，首先在空旷和障碍场景中分别测试了本章提出的方法扩展到大规模异构多智能体上的表现，随后在仓储场景中将该方法与文献[59]中提出的方法进行了对比实验。最后本文进行了在线多智能体路径规划实验，并考察了使用不同大小的时间跨度 $w$ 对于实验结果的影响。实验结果显示，本章提出的方法都可以在 300 米 ×300 米的场景中为多达 100 个异构智能体进行满足运动学约束的路径规划，并保持 90% 以上的算法成功率。同时，该方法的规划方案质量也显著优于对比的现有先进方法。而对于在线 MAPF 问题，本章提出的方法可以将在线 MAPF 规划器的速度提升六倍而几乎不影响规划方案的质量。

# 5 多智能体路径规划方案后处理框架

## 5.1 概述

在第三章和第四章中，本文使用基于冲突搜索算法解决了考虑智能体运动学模型的MAPF问题，再结合滚动时域碰撞算法解决了异构多智能体系统中的在线MAPF问题，不断地缩小多智能体路径规划研究与实际工业场景之间的差距。然而在实际运行阶段，多智能体系统还面临着针对工业控制网络安全的严峻挑战。由于大多数工业机器人的芯片都是由特定供应商提供，缺乏严格的身份认证机制，攻击者可以通过固件或网络漏洞远程对多智能体系统中的通信信息进行篡改甚至对智能体进行恶意控制。这不仅会导致失控智能体无法按照原定规划方案准确执行路径，严重时还会与其他智能体或物理工作环境发生碰撞，导致系统大范围瘫痪，给工厂带来巨大的经济损失，甚至对员工的人身安全造成危害。因此在工业级多智能体路径规划系统中，一个能在运行阶段实时监测系统内异常情况，主动识别潜在网络安全威胁的框架尤为关键。

在现有的研究中，关于此类MAPF问题后处理（Post-processing）框架的研究尚且不多。文献[54]中提出了一个被称为MAPF-POST的框架，其可以在多项式时间内对MAPF求解器生成的规划方案进行处理并创建一个可以在系统运行时中使用的计划-执行时间表。该框架可以对异构多智能体系统进行异常行为检测，但是其需要每个智能体持续广播自己的当前位置，对系统的通信质量和通信带宽都有较高的要求。文献[77]中提出的RMTRACK方法也是基于类似的想法，但其是作为一个控制律而非后处理框架进行应用的。该方法只在一些重要的状态变化时才在系统中发送信息，从而减少了多机器人系统的通信成本。但RMTRACK方法只考虑了系统中可能发生的通讯信号和执行延迟，并没有考虑由于网络攻击造成系统内智能体被恶意控制从而导致轨迹偏差的情况。此外，以上两种方法均只能针对离线MAPF规划方案而不能应用到在线MAPF问题中。

针对上述问题，本章提出了一个适用于异构多智能体系统MAPF问题的后处理框架。本文使用行动依赖图（Action Dependency Graph，ADG）[65]来分析基于冲突搜索的MAPF求解器生成的规划方案中的智能体行动优先关系。利用这些优先级关系，系统可以实时监测系统内受到攻击偏离原有规划的智能体，并对该次攻击可能影响到的其他智能体进行预警。同时，通过重叠执行过程和规划过程，该框架可以进一步应用于在线MAPF场景。

随后本章在 Gazebo 三维仿真环境中对上述后处理框架的有效性进行了测试，实验中异构系统包含了轮式移动机器人和腿足机器人。同时本文在仿真中模拟了系统内一个机器人遭受恶意入侵失控的情况以测试后处理框架对于异常情况的鲁棒性。最后本文将后处理方法与前两章提出的 MAPF 规划方法进行整合，形成全流程多智能体路径规划解决方案，并在室内场景和室外场景下分别进行了实物实验。在室内场景中，本文构建了一个由 7 个阿克曼机器人组成的同构多智能体系统，并在空旷和障碍场景中进行了路径规划实验；而在室外场景中，本文为一个包含四足机器人和轮式移动机器人的异构多智能体系统进行了路径规划并在 10 米 ×16 米的场地中进行了实际测试。

综上，本章的主要贡献如下：

1. 本章提出了一个基于行动依赖图 MAPF 问题后处理框架。该方法通过多智能体路径规划方案分析智能体间动作的优先级关系，仅需智能体和规划器之间的少量通信即可实时监测系统内受到网络安全攻击偏离原有规划的智能体，并对该次攻击可能影响到的其他智能体进行预警。同时，通过重叠执行过程和规划过程，该方法可以扩展到在线 MAPF 问题中。
2. 本章在 Gazebo 仿真环境中对一个包含三辆不同大小的轮式机器人和一台四足机器人的异构多智能体系统进行了实验。实验模拟了系统内一个机器人遭受恶意入侵失控的情况，但在后处理框架的干预下系统很快修复了偏差并保证了智能体路径执行过程中的安全避碰，验证了后处理框架在实际 MAPF 应用场景中的有效性和必要性。
3. 本章将后处理方法与前两章提出的 MAPF 规划方法进行整合，构建了一套工业级全流程多智能体路径规划解决方案并在室内场景和室外场景下分别进行了实物实验。在不同的场景及各异的系统中都验证了该解决方案的实际可行性和执行鲁棒性。

## 5.2 基于行动依赖图的 MAPF 后处理框架

由 MAPF 求解器生成的路径规划方案中实际上隐式编码了智能体之间的依赖关系，例如，哪个智能体应当优先通过一个狭窄通道。这些依赖关系可以在后处理阶段中以多项式时间提取。在 MAPF-POST[54] 中，这种依赖关系是以智能体路径的每个状态建立的，这种方法对智能体系统内的通信带宽要求较高。而本文则利用行动依赖图（Action Dependency Graph，ADG）的概念，将这种依赖关系建立在智能体路径中的一系列关键行动上，既减少了所需的系统通信带宽，同时也可以拓展到在线 MAPF 的工业场景中。

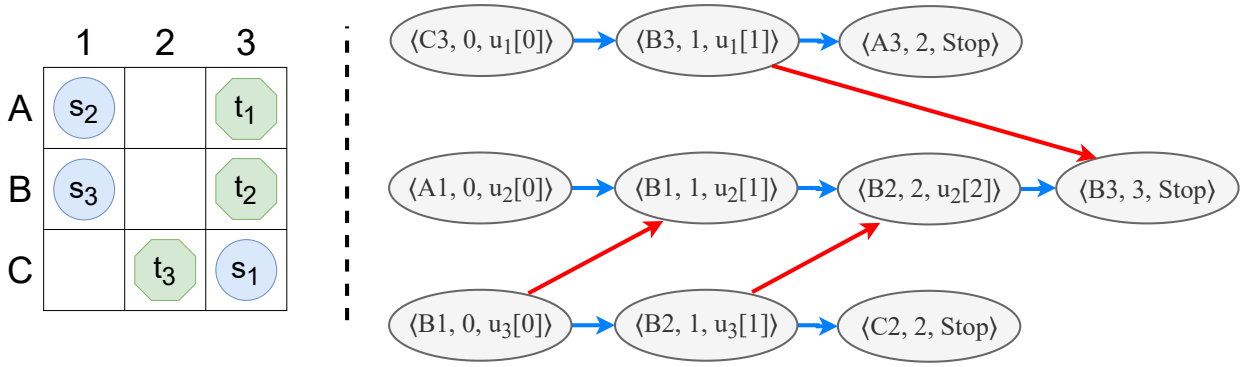



图 5.1 一个三智能体系统规划方案的行动依赖图。（为了方便示意，在图中对工作空间进行了离散）

### 5.2.1 行动依赖图

本文定义一个在连续工作空间内的行动依赖图 $\mathcal{G}_{ADG} = (\mathcal{V}_{ADG}, \mathcal{E}_{ADG})$。行动依赖图是一个有向图，图中的一个行动 $A_i^t \in \mathcal{V}_{ADG}$ 被定义为一个元组 $A_i^t = \langle \pi_i[t], t, u \rangle$，意味着智能体 $a_i$ 计划在 $t$ 时间步通过控制输入 $u$ 到达 $\pi_i[t]$ 这个状态。行动依赖图中的一个边 $(A_i^{t_1}, A_j^{t_2}) \in \mathcal{E}_{ADG}, t_1 < t_2$ 代表智能体行动之间的依赖关系，即智能体 $a_j$ 需要在智能体 $a_i$ 完成 $A_i^{t_1}$ 动作后才能开始执行 $A_j^{t_2}$ 动作。根据不同情况可以将集合 $\mathcal{E}_{ADG}$ 中的边分为两类：i）一类边，代表同一智能体自身路径之间的依赖关系。按照规划，智能体当然应先完成前序动作才能开始进行当前动作。具体来说，一类边内的两个行动都属于同一个智能体，且两个行动应该是时间上相邻的，即形如 $(A_i^t, A_i^{t+1})$。ii）二类边，代表不同智能体之间的依赖关系。当前序智能体状态与当前状态有冲突时，在前序智能体未完成该状态时，当前智能体不能开始行动。具体来说，二类边内两个行动属于两个不同智能体，同时两个行动的状态之间应当有冲突发生，即对于 $(A_i^{t_1}, A_j^{t_2})$，$\mathcal{R}_i(\pi_i[t_1]) \cap \mathcal{R}_j(\pi_j[t_2]) \neq \emptyset$。一个三智能体系统规划方案的行动依赖图如图5.1示意，其中蓝色箭头表示一类边，红色箭头表示二类边。需要注意的是，为了方便示意本图对工作空间进行了栅格离散化。在实际算法中，行动依赖图是根据连续工作空间中的多智能体路径规划方案构建的。

行动依赖图的构建流程如算法6所示。首先算法遍历所有智能体的路径，将路径上每一个状态 $A_i^t$ 都加入到行动依赖图的顶点集合 $\mathcal{V}_{ADG}$ 中。同时，沿着每个智能体的路径找到所有一类边 $(A_i^{t-1}, A_i^t)$ 并加入行动依赖图的边集合 $\mathcal{E}_{ADG}$。接下来算法开始寻找二类边，遍历所有不相同的智能体对 $(a_i, a_j)$，依次考察两个智能体的路径上的状态是否存在 $\mathcal{R}_i(\pi_i[t_1]) \cap \mathcal{R}_j(\pi_j[t_2]) \neq \emptyset, t_1 < t_2$ 的情况，若有，算法就将这两个状态构建出一个二类边并加入图的边集合 $\mathcal{E}_{ADG}$，直到算法遍历完所有智能体对。可以看出，该构造的算法复杂度是一个关于系统中智能体数量的多项式时间。

在系统实际运行阶段，后处理框架追踪行动依赖图 $\mathcal{G}_{ADG}$ 中的每一个顶点（即行动）

**算法** 6: 构建行动依赖图

Input: A MAPF solution

Output: $\mathcal{G}_{ADG}$

// Create $\mathcal{V}_{ADG}$ and Type 1 edges

foreach $a_i$ in the system do
    Add $A_i^0$ to $\mathcal{V}_{ADG}$;
    for $t \leftarrow 1$ to $T_i$ do
        Add $A_i^t$ to $\mathcal{V}_{ADG}$;
        Add edge $(A_i^{t-1}, A_i^t)$ to $\mathcal{E}_{ADG}$;
    end
end

// Create Type 2 edges

foreach $a_i, a_j$ in the system, $a_i \neq a_j$ do
    for $t_1 \leftarrow 0$ to $T_i$ do
        for $t_2 \leftarrow t_1$ to $T_j$ do
            if $\mathcal{R}_i(\pi_i[t_1]) \cap \mathcal{R}_j(\pi_j[t_2]) \neq \emptyset$ then
                Add edge $(A_i^{t_1}, A_j^{t_2})$ to $\mathcal{E}_{ADG}$;
                Break;
            end
        end
    end
end

return $\mathcal{G}_{ADG}$;

的完成状态。本文将顶点的状态分为三类：未开始的，进行中的和已完成的。在初始情况下，每个顶点都处于未开始的状态。只有当一个顶点满足如下两个条件时，算法才会将其标记为进行中并将该顶点中对应的指令传递给机器人 $a_i$：i）该顶点的前序顶点（传入一类边连接）已被标记为已完成或者进行中；ii）所有向该节点传入二类边的顶点已经被标记为已完成。而当机器人实际准确完成了该顶点的动作，算法就会将该顶点标记为已完成。

当多智能体系统受到网络攻击并发生机器人被恶意控制的情况时，该机器人会偏离原有路径，从而导致机器人当前的进行中状态无法被完成。当后处理框架探测到有机器人在

规定时间内未完成当前顶点后，即将该异常发送至多智能体控制系统并将该受攻击机器人暂时停止以免其对环境造成进一步的破坏。同时，由于该顶点一直处于未完成状态，其余要前往该区域的机器人也会暂停移动，从而避免与该失控智能体的后续碰撞。当工业控制系统处理完当前网络攻击，收回对受攻击智能体的控制权后，后处理框架会调用 MAPF 求解器进行重规划，最终发送给机器人新的命令。由于本文提出的后处理框架只需要跟踪每个机器人已经完成的行动，而无需机器人在任何时刻都汇报当前位置，因此该框架拥有比 MAPF-POST 框架更广泛的安全距离以及更少的通信量。

### 5.2.2 在线 MAPF 问题中的后处理框架

上述后处理框架同样可以扩展到在线 MAPF 问题中。在4.4节中，本文提出滚动时域碰撞解决算法将在线 MAPF 问题拆分成一连串的时间窗口 MAPF 实例，并每隔 $h$ 时间步重新规划一次路径，每次规划的方案能够保证在 $w$ 时间步内是安全无冲突的，$w \geq h$。因此，对于后处理框架来说，其也可以把 MAPF 问题解耦为连续的时间窗口 MAPF 实例。事实上，后处理框架不需要考虑智能体需要连续到达一个目标状态序列，而只需考虑如何鲁棒的执行当前规划方案，保证实际执行中智能体稳定有序运行。借由滚动时域碰撞解决算法的特点，后处理框架只需保证当前 $h$ 时间步的规划方案的执行。具体来说在线 MAPF 场景中，当 $t$ 时刻在线 MAPF 求解器生成了一个保证 $(t, t+w)$ 时间段内不发生碰撞的规划方案后，后处理框架接受该方案并将当前依赖图内所有节点标记为已完成。随后，后处理框架针对该方案的前 $h$ 个时间步，即 $(t, t+h)$，生成行动依赖图，并下发给具体机器人进行实际执行。当机器人反馈当前的行动依赖图完成时，后处理框架会调用 MAPF 求解器生成下一个时间段即 $(t+h, t+h+w)$ 的规划方案，并重复上述过程。

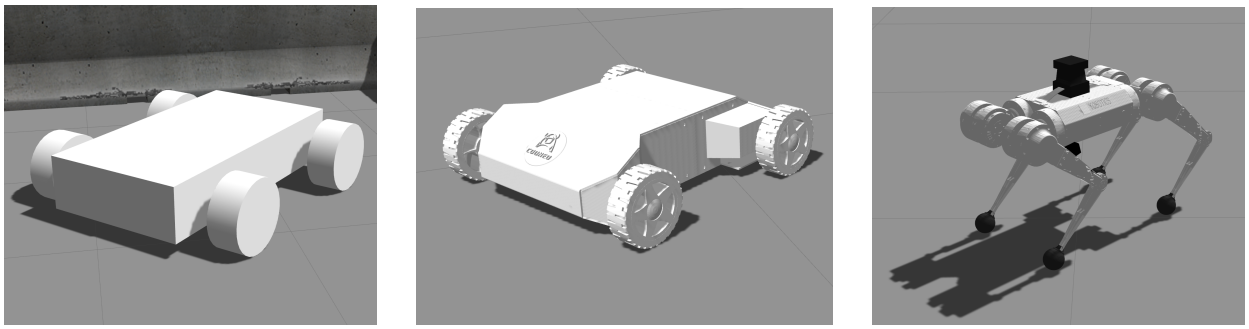
显然，在线 MAPF 问题中旧规划方案和新规划方案之间可能存在着依赖关系，但是通过滚动时域碰撞解决算法的解耦，本文提出的这套流程确保了新计划和旧计划之间没有任何依赖关系，新计划可以被直接添加到行动依赖图中并根据算法6计算出依赖关系。最后，为了保持算法只占用一小部分内存，在行动依赖图中状态为已完成的顶点可以被安全的删除。

## 5.3 实验

### 5.3.1 Gazebo 仿真实验

首先本文在一个具有物理引擎的三维仿真环境中来测试使用 MAPF 异常检测方法的全流程多智能体系统规划系统的实际效果。本文选取 Gazebo[78] 作为仿真平台，Gazebo 是

一款功能强大的开源三维物理仿真器，支持包括 ODE、Bullet、DART 在内的多款高性能物理引擎，具有高质量的图形化渲染能力，可以显示具有逼真光影和纹理的三维环境。同时，Gazebo 还提供了多种机器人模型可供使用，自定义的机器人模型也可以非常方便地加入仿真环境中。机器人的传感器信息也可以通过插件的形式加入仿真环境。



(a) 阿克曼移动机器人 (b) NEOR Mini 移动机器人 (c) Mini Cheetah 四足机器人

图 5.2 Gazebo 环境中使用的机器人模型

在本节仿真实验中，多机器人系统由四辆异构机器人组成，其中包括两辆阿克曼模型移动机器人，一辆 NEOR Mini 移动机器人[①]和一台 Mini Cheetah 四足机器人[②]，如图 5.2所示。其中，两辆阿克曼模型移动机器人大小为 1.6 米 ×1.3 米，最大移动速度设置为 1.5 米每秒，其运动学模型完全符合阿克曼转向几何，最小转弯半径为 4 米。NEOR Mini 机器人是酷牛科技推出的一款针对高校教学和科研的智能开放式移动平台，其具备有稳固的主体框架结构并使用 3D 打印的外壳，性能稳定且体积小巧，并可根据应用场景搭载超声波雷达、深度相机和激光雷达等多种传感器。在 Gazebo 仿真环境中，该机器人模型大小为 0.6 米 ×0.4 米，最大移动速度设置为 0.6 米每秒，模型最小转弯半径为 1.5 米，同样符合阿克曼转向几何。

Mini Cheetah 四足机器人[79] 是麻省理工学院仿生机器人实验室（MIT Biomimetic Robotics Lab）于 2018 年推出的一个小型四足机器人，其只有 9 公斤的重量但是具备跑步，倒退和后空翻的能力。其具备强大的执行机构，可以在不平坦的地形上保持平衡并进行小跑，是目前世界范围内最先进的四足机器人之一。同时，它除软件、固件（Firmware）和机器人本体之外的硬件已经全部开源。值得注意的是，四足机器人实际上是在一个三维空间内进行运动，具有较为复杂的动力学模型，本文将其模型简化为在二维平面上的运动，该简化模型允许四足 Mini Cheetah 机器人以最快 2.4 米每秒的速度前进，1 米每秒的速度倒退以及在原地以最快 5 弧度进行转向，本文接着使用专用控制器将二维平面的速度控制

[①] https://github.com/COONEO/neor-mini

[②] https://github.com/mit-biomimetics/Cheetah-Software

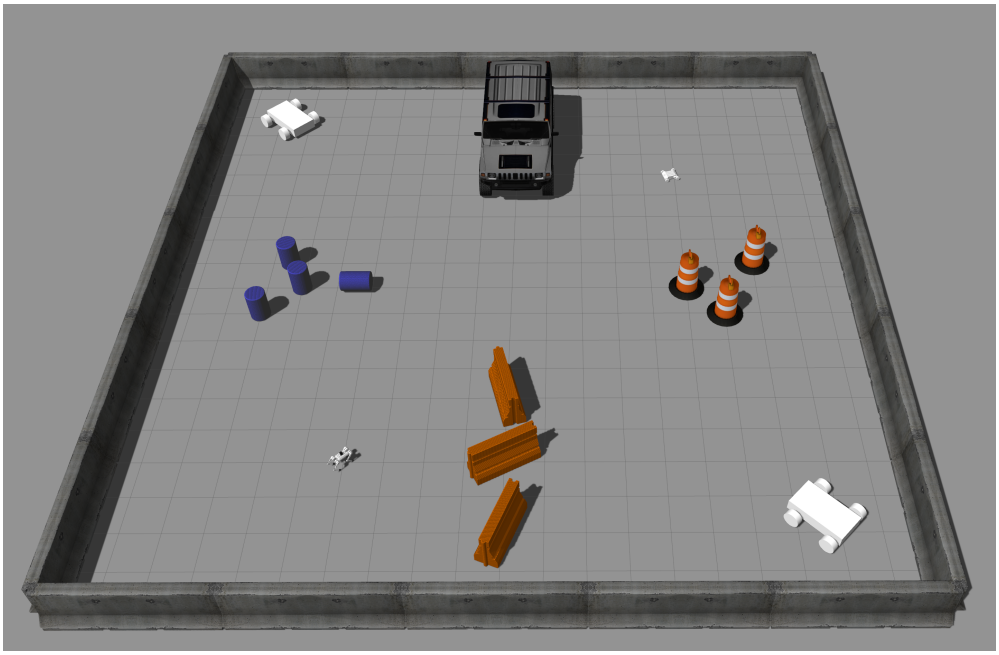


图 5.3 Gazebo 仿真场景示意图

转化到腿足上的电机控制。

本文在 Gazebo 仿真器中建立了一个仿真工业场景，如图 5.3所示。该场景大小为 20 米 ×20 米，四周有围栏进行围挡。在场地中，本文设置了包括一辆越野车，若干油桶、交通锥桶和路障在内的不同障碍物。四个机器人分别在场地的四角上，以场地左上角为原点，两辆阿克曼机器人分别位于 $(2, 2)$ 和 $(18, 18)$ 的位置，NEOR Mini 移动机器人位于 $(6, 16)$ 位置，而 Mini Cheetah 四足机器人则位于 $(15, 5)$ 的位置。各个机器人都需要前往场地对角的机器人的位置，即系统中的四个机器人按照场地对角线两两互换位置。同时，由于场地内的障碍物排列，四个机器人都需要前往场地中央位置进行相互避让最终到达终点。

本文使用针对异构多智能体系统的基于冲突搜索方法为该场景进行规划。本文直接将地图信息输入 MAPF 规划器中，并指定系统中每个机器人的起始状态和终止状态，程序使用 C++ 并在一台配有 Intel i7-8700@3.20GHz 处理器、8G 内存的运行有 Ubuntu 16.04 操作系统的计算机上执行。规划程序对该场景进行五次尝试，每次求解时限为 60 秒。五次尝试获得了 100% 的成功率，程序平均运行时间为 3.762 秒。生成的规划方案的总运行时间为 62.7 秒，平均到达时间为 44.2 秒。

本文将规划方案输入给后处理框架，并在 Gazebo 中进行仿真实验。实验结果如图 5.4所示。可以看出，后处理框架成功地接受了规划器生成的规划方案，并保证了方案安全避碰地完成执行。在实验中，由于简化了四足机器人的运动学模型，导致实际过程中四足机器人实际行走轨迹并不能完全符合规划路径。在 $t = 23.1$ 秒时，实验模拟了四足机器人遭受

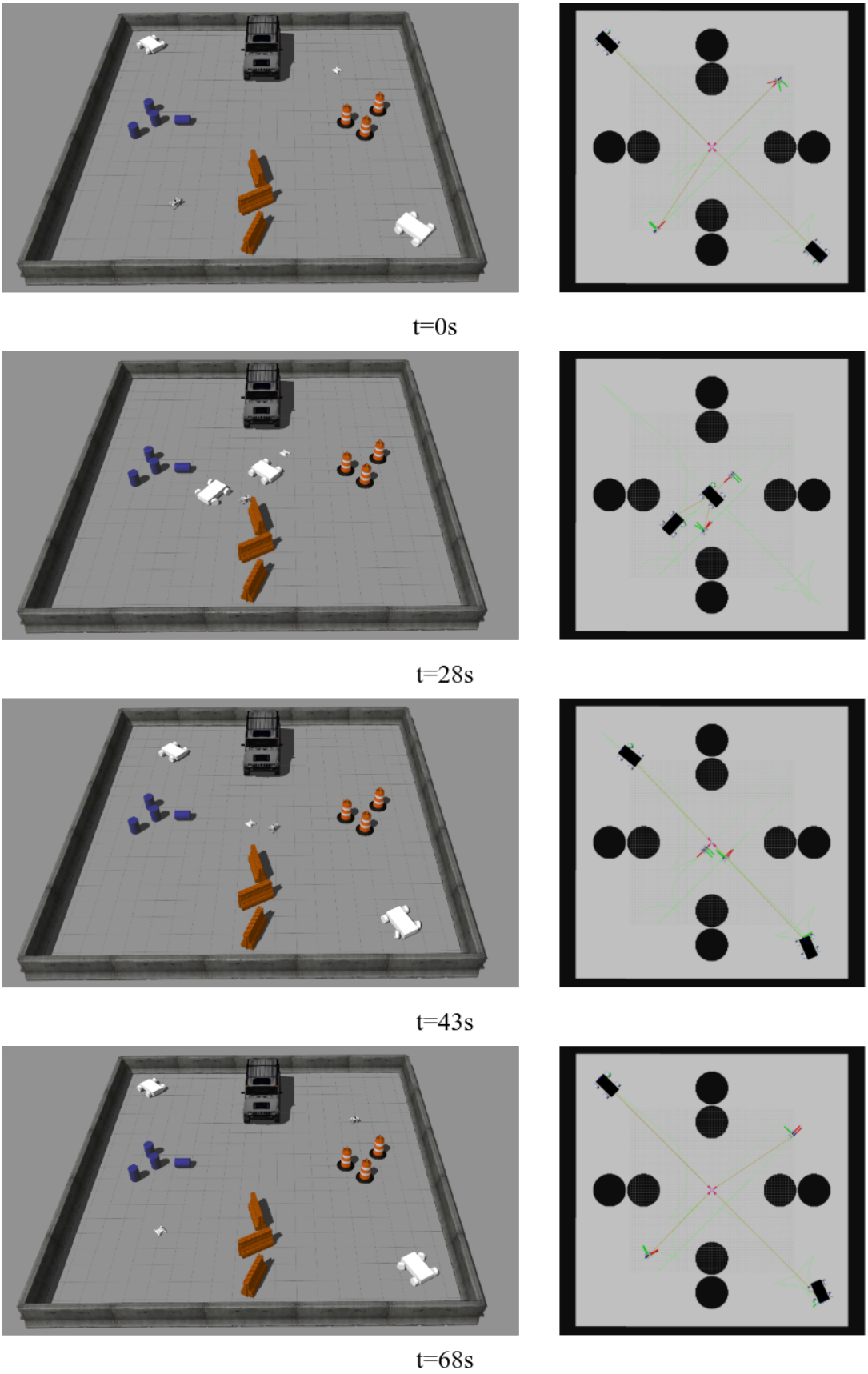
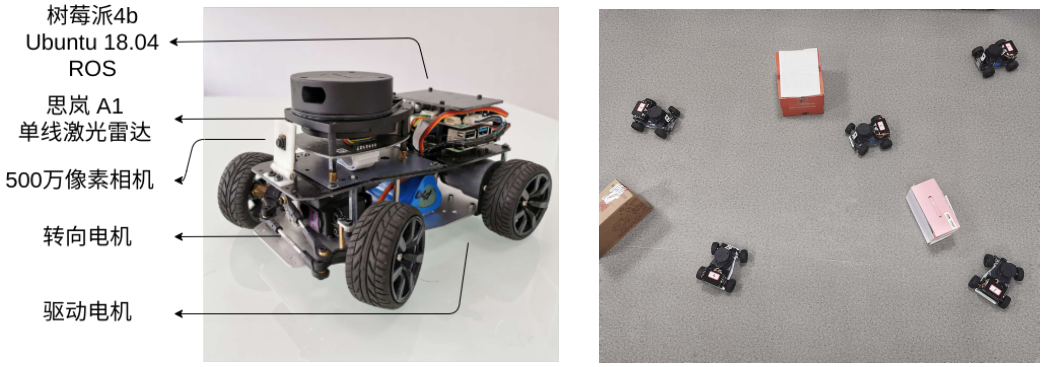



图 5.4 Gazebo 仿真场景实验。左列图为展示了实验中的四个关键帧，右列图中为对应帧在 Rviz 中显示的机器人姿态。




(a) 维嵌科技智能小车 (b) 室内障碍实景

图 5.5 室内场景实物试验

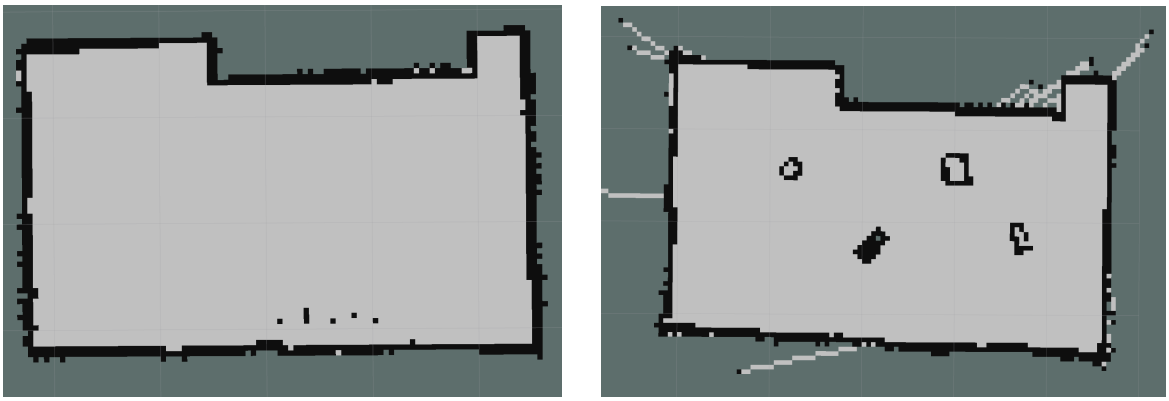
到了网络攻击而被恶意控制，此时四足机器人不再接收控制系统发送的指令，偏离了原有路径并向一台阿克曼机器人冲去。在 $t = 24.1$ 秒，后处理框架探测到该机器人出现异常，并通知控制系统该机器人失控。$t = 24.7$ 秒时，控制系统重新控制四足机器人，同时后处理框架调用 MAPF 求解器重新为该机器人规划路径。最终后处理框架的干预下，四足机器人在 $t = 28$ 秒就弥补了之前的偏离，回到了正常的路径上。最终，方案总运行时间为 68.1 秒，同时没有发生任何机器人之间的碰撞情况。实验证明，MAPF 后处理框架在实际工业场景中是非常必要且有效的。完整的实验视频参见如下链接：https://youtu.be/MI1VmkJ2ovc。

### 5.3.2 室内场景实物试验

接下来本文进行了室内场景的实物实验。在该实物实验中，由 7 个阿克曼模型的移动机器人组成的同构多智能体系统在一个 5 米 ×3 米的房间内开展实验。本文选用维嵌科技智能 ROS 小车作为多智能体系统中的移动机器人，如图 5.5a所示。该智能小车为矩形外形，边长为 23 厘米 ×20 厘米。该机器人基于阿克曼转向模型，能够以 0.3 米每秒的速度移动，最小转弯半径为 0.26 米。机器人上配备有一个思岚科技 A1 单线激光雷达，一个 500 万像素的单目摄像头，以及一个运行有 Ubuntu18.04 操作系统率和 ROS Melodic 的树莓派 4b 嵌入式芯片。此外，本文使用一台同样运行 ROS Melodic 的装有 Intel i5-9300@2.40GHz 处理器的笔记本电脑作为中央计算单元，负责运行 MAPF 求解器和后处理框架，并使用 2.4GHz WiFi 与系统内所有移动机器人组成局域网，使用 ROS 通讯功能对系统内所有移动机器人进行控制。

实验在一个 5 米 ×3 米的房间内开展，并包含空旷场景和障碍场景两种情况。在空旷场景中，房间内除了房间内墙外没有任何障碍物，而在障碍场景中，房间中央摆放了 4 个大小形状各异的纸箱作为静态障碍物。由于机器人上配备了激光雷达，本文首先使用

Gmapping 算法[80] 对两个情况的房间进行二维建图。Gmapping 算法是基于滤波框架的常用开源 SLAM 算法，其基本原理基于 RBPF（Rao-Blackwellized Particle Filters）粒子滤波算法，将定位和建图过程分离，先进行定位再进行建图。该算法可以实时构建室内环境地图，在小场景中计算量少且生成地图精度较高，其有效利用了车轮里程计信息，因此可以提供机器人的位姿先验，对激光雷达频率要求较低。Gmapping 的建图结果如图 5.7所示，建图完成后，194 个直径为 0.05 米的障碍物构成了房间的边界，86 个同样直径的障碍物则代表了放置在障碍场景中的纸箱。



(a) 室内空旷场景 (b) 室内障碍场景

图 5.6 室内场景建图结果

本文在两个场景内人工指定了每个机器人的起点和目标状态，并在中央处理单元上进行求解。在本实物实验中，求解器使用原版的针对阿克曼模型的基于冲突搜索方法，在空旷和障碍场景中，分别进行五次尝试，每次尝试的时限为 60 秒。在空旷房间场景下，所有五次尝试都是成功的，程序平均运行时间为 2.204 秒。在该场景中，车身冲突树会生成 7 个运动学约束，与系统内智能体的数量相等，在加入七个车身时空约束后，规划方案有最小的平均到达时间为 16.98 秒。至于室内障碍场景，五次尝试中也有 100% 的算法成功率，其中程序平均运行时间为 4.774 秒。在障碍场景下，车身冲突树构建了七个运动学约束，并在添加了九个智能体间的车身约束后，获得了规划方案的最小的平均到达时间，18.67 秒。与前者相比，室内障碍场景更加复杂，机器人的路径更容易发生碰撞，因此需要更多的车身冲突才能完成求解。

后处理框架接受求解器生成的规划方案后生成行动依赖图，并按照行动依赖图的信息依次在恰当的时间将控制指令发送给每个机器人。在实际执行过程中，本文采用自适应蒙特卡洛定位（Adaptive Monte Carlo Localization，AMCL）算法[81] 对智能体进行环境中的全局定位。AMCL 算法是机器人在 2D 中移动的概率定位系统，它实现了自适应（或

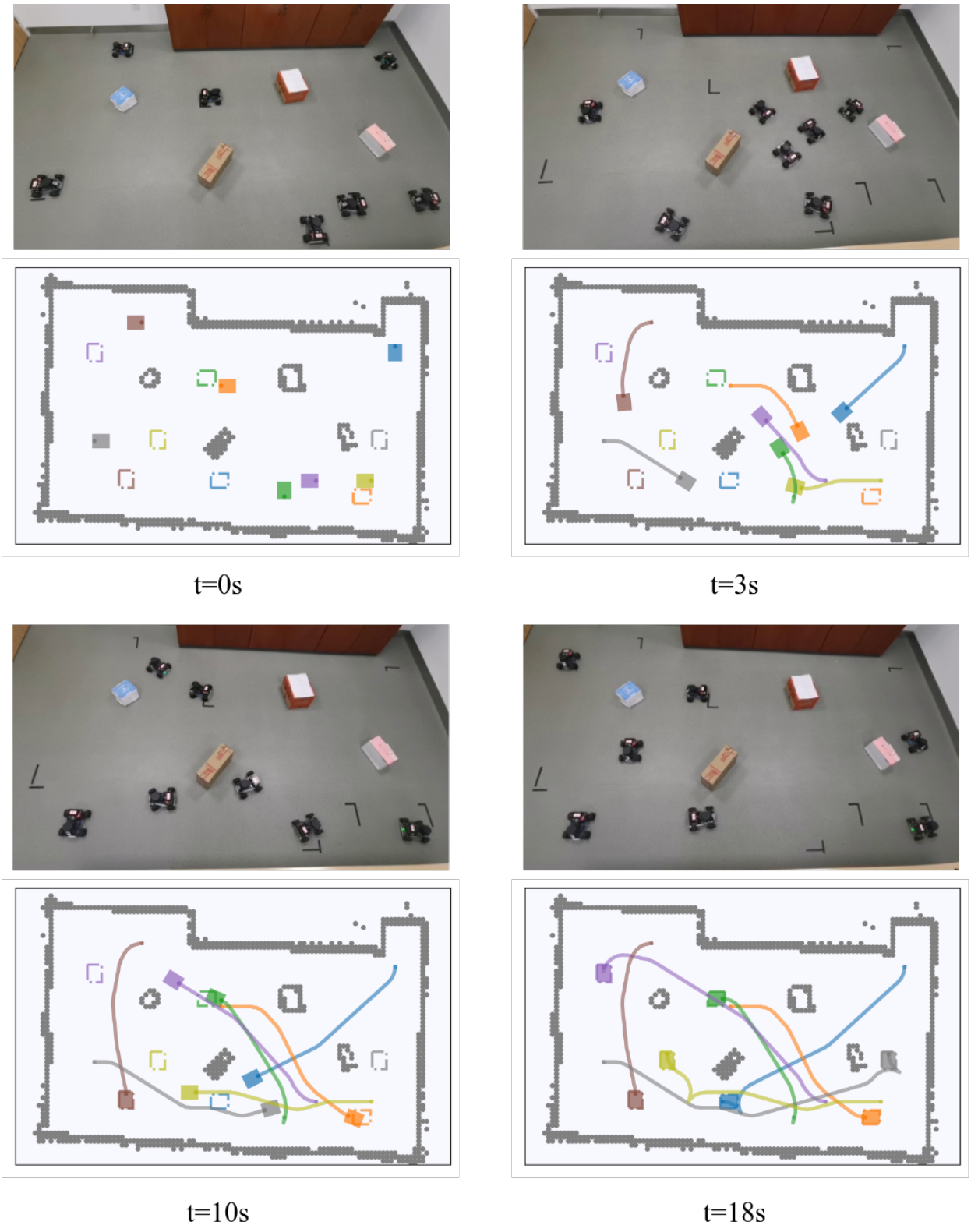


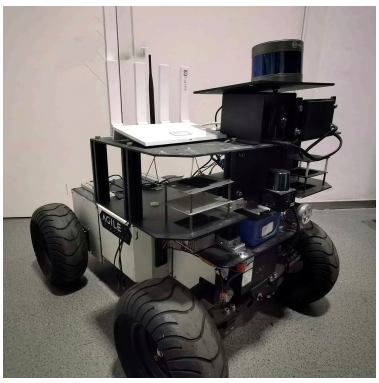

图 5.7 室内障碍场景实物试验。上部分的视频截图展示了实验中的四个关键帧。下行的图片描绘了智能在相应帧中的行驶轨迹。（最好以彩色浏览）

KLD 采样）蒙特卡罗定位方法，其使用粒子滤波器来针对已知的地图跟踪机器人的姿态。同时，AMCL 算法在机器人遭到绑架的时候，会随机注入粒子，使得机器人可以从全局定位失败中恢复过来。图5.7展示了在室内障碍场景中实验的 4 个关键帧截图，时间分别为 $t = 0, 3, 10, 18s$。可以看到，在后处理框架的配合下在 $t = 3s$ 时，地图左侧区域汇聚了 5 个机器人集中通过，仍然保持了没有冲突产生。完整的实验视频参见如下链接：https://youtu.be/KThsX04ABvc?t=106。

### 5.3.3 室外场景实物实验

在本节中，本文进行室外场景的实物实验。相较于室内场景的实验，室外场景更大更复杂，建图难度更大，同时，室外场景中的机器人也更贴近工业场景的实际应用。在该实物实验中，多机器人系统由四个异构机器人组成，其中包括一辆松灵 Scout Mini 移动机器人[①]，两辆松灵 Hunter 移动机器人[②]和一台云深处绝影四足机器人[③]，如图 5.8所示。

(a) 松灵 Scout Mini 移动机器人



(b) 松灵 Hunter 移动机器人

(c) 云深处绝影四足机器人

图 5.8 室外场景实物实验中使用的机器人

松灵 Scout Mini 移动机器人是一款全地形移动机器人，具备四轮差速驱动、独立悬挂、原地自转等特点。该机器人尺寸为 0.61 米 ×0.58 米，总重为 23 千克，通过自主研发的轻量动力系统解决方案，Scout Mini 移动机器人最大速度可以达到 2.5 米每秒。在实际实验过程中，本文在该机器人上装配了一台速腾聚创 16 线激光雷达 RS-LiDAR-16[④]用于环境感知和定位，以及一台配备有 Intel i3-8145U@2.1GHz 处理器和 Intel AC 9560 无线网卡的英特尔普罗沃峡谷迷你电脑（NUC）进行系统通讯和机器人轨迹控制。松灵 Hunter 移动机器人是一款前转向移动机器人平台，其前轮为阿克曼转向结构并搭载有摇摆臂悬挂，可

[①] https://www.agilex.ai/product/3

[②] https://www.agilex.ai/product/9

[③] https://www.deeprobotics.cn/products_jy_2.html

[④] https://www.robosense.ai/rslidar/RS-LiDAR-16

以轻松跨越各种崎岖路面。同时其相比 Scout Mini 具有更强的动力系统，具备更强的承载能力和续航能力。该型机器人尺寸为 0.98 米 ×0.74 米，总重为 65 千克，最大移动速度可达 2 米每秒，其最小转向半径为 1.6 米，最大爬坡角度为 10 度。本文在每台松灵 Hunter 机器人上都装备有一台速腾聚创 16 线激光雷达 RS-LiDAR-16 和一台与前述相同配置的英特尔普罗沃峡谷迷你电脑进行机器人轨迹控制。云深处绝影四足机器人是国内第一款面向行业应用的灵敏型四足机器人，其支持行走、小跑、跳跃和奔跑等多种步态，并具备抗外力扰动平衡控制，摔倒后自主爬起等能力。该机器人尺寸为 0.85 米 ×0.45 米，总重为 42 千克，最大负载 20 千克，最快行走速度为 2.1 米每秒。绝影四足机器人上配备有一台速腾聚创 16 线激光雷达 RS-LiDAR-16、两台英特尔 Realsense D435i 深度摄像头和一台配有 Intel i7@2.2GHz 处理器的工业控制计算机。

系统中所有 4 台机器人均基于 Linux 操作系统并运行 ROS 框架。在一台松灵 Hunter 机器人上配有一个华为 AX3 无线路由器，为系统内所有机器人建立无线局域网进行通讯。在场地中还有一台配有 Intel i5-9300@2.40GHz 处理器的笔记本电脑作为中央计算单元，负责运行 MAPF 求解器和后处理框架，也加入无线局域网并使用 ROS 通讯功能对系统内所有移动机器人进行控制。

(a) LIO-SAM 建图结果　　(b) 滤除地面后的实验场地

图 5.9　室外场景建图结果

实验在浙江大学玉泉校区第十八教学楼外空地开展。首先需要对试验场地进行建图，本文使用一台配备有速腾聚创 32 线激光雷达 RS-LiDAR-32[①]和 Xsens MTi-300 惯性测量

[①]https://www.robosense.ai/rslidar/RS-LiDAR-32

单元（IMU）[①]的松灵 Hunter 移动机器人对第十八教学楼外场地进行三维建图。本文采用 LIO-SAM 算法[82] 进行激光雷达建图，该算法通过紧耦合激光雷达和惯性测量单元，并优化包含激光雷达里程计因子、惯性测量单元预积分因子、GPS 因子和回环因子在内的因子图实现了高精度且实时的移动机器人轨迹估计和地图构建。为了确保算法的实时性能，LIO-SAM 算法使用当前激光雷达帧和局部地图进行匹配而不是全局地图匹配，从而进一步提高了帧与地图的匹配效率。LIO-SAM 算法的建图结果如图 5.9a所示，可以看到建图结果准确度非常高，无论是对楼宇还是街边树木都能很好地还原出来。经过考察，本文选用图 5.9a右侧楼宇间空地作为试验场地。由于激光建图时不可避免地会对地面产生回波，因此需要在建图后对地面进行滤除，滤除地面后的实验场地地图如图 5.9b所示，其中本文选择的实验场地为图中红色矩形区域，大小为 10 米 ×16 米。

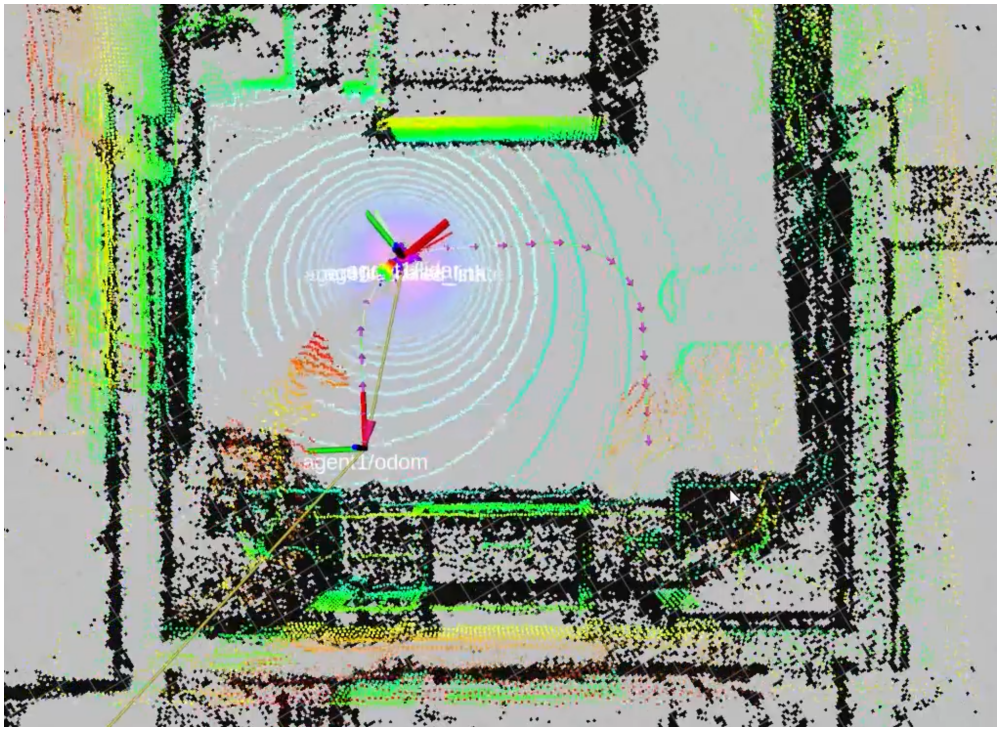
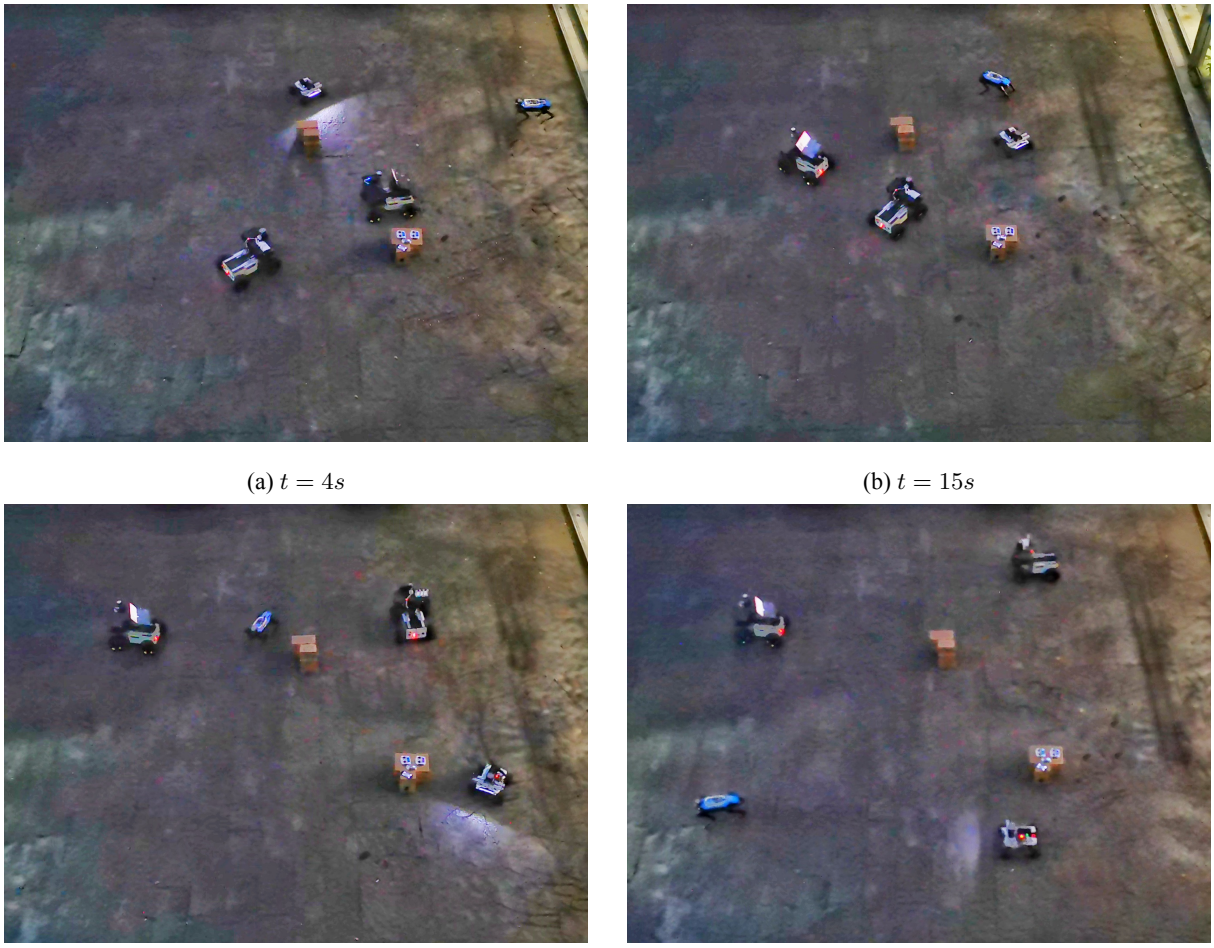


图 5.10 NDT 点云配准结果

有了建好的三维地图后，系统内的机器人需要根据已知的三维地图在环境中进行定位。在本实验中，机器人上并未配备差分 GPS 传感器，而使用 IMU 和里程计 (Odometry) 根据上一时刻的位置和方位推断现在的位置和方位，虽然输出频率高，但是定位误差会随着时间不断累积。本文使用 NDT（Normal Distribution Transform）点云配准算法[83]，利用激光雷达进行基于环境特征的定位，用每一帧获取的点云和预先制作的三维地图进行匹配，从而得到实时的车的位置和姿态。相比于其他点云配准算法，NDT 算法运行更加稳

[①]https://www.xsens.cn/products/mti-100-series/



(a) $t = 4s$ (b) $t = 15s$

(c) $t = 24s$ (d) $t = 42s$

图 5.11 室外场景实物实验

定，实际定位测试效果也更加鲁棒。如图5.10所示，即为系统内一台移动机器人在场景中使用 NDT 算法进行全局定位的结果，结果显示使用算法定位精度较高。

本文使用针对异构多智能体系统的基于冲突搜索方法为该场景进行规划。在场地中，除了机器人之外，还放有两个由纸箱构成的静态障碍物。本文将包含障碍物的地图信息输入到 MAPF 规划器中，并指定系统内四台机器人的起始状态和目标状态。程序由 C++ 编写并在场地内配有 Intel i5-9300@2.40GHz 处理器的笔记本电脑上运行。规划程序进行 5 次尝试，每次求解时限 60 秒。五次尝试均获得了成功，程序平均运行时间 8.62 秒，生成的规划方案的总运行时间为 39.86 秒，平均到达时间为 31.78 秒。本文将规划方案输入后处理框架并在场地内进行实物实验，实验结果如图5.11所示。场地内所有四台机器人均成功到达终点并没有发生任何碰撞。实验中网络环境较好，并未出现丢包指令延迟等因素。但由于四足机器人运动模型的简化，导致实际执行过程中，四足机器人对路径的追踪没有轮式机器人追踪精度高。而得益于后处理框架，四足仍然成功的进行了避障并到达终点。最终，规划方案实际运行时间为 41.86 秒。完整的实验视频参见如下链接：

https://youtu.be/Ufjs9q10H18。

该实物实验成功验证了本文提出的规划方法和后处理框架可以实际应用于真实的异构多智能体寻路场景中，同时能够轻易地与建图、定位和控制等算法一起整合为一套完整的机器人系统软件。

## 5.4 本章小结

本章主要介绍了一个利用行动依赖图的 MAPF 问题后处理框架，该框架可以利用求解器生成的多智能体规划方案确定智能体之间的动作优先级关系。利用这些优先关系，仅需智能体和规划器之间进行少量通信，多机器人系统即可防御网络攻击者利用安全漏洞对系统中智能体的恶意控制，确保系统在实际执行阶段长时间安全鲁棒地执行规划方案。

本章在一个 Gazebo 三维仿真环境中进行了异构多智能体系统寻路实验，实验的异构系统中既包含轮式移动机器人也包含腿足机器人。仿真实验模拟了系统内一个机器人遭受恶意入侵失控的情况，但在后处理框架的干预下，系统很快修复了偏差并保证了智能体之间的安全避碰，验证了前述后处理框架在实际 MAPF 场景中的有效性和必要性。随后本章进行了同构多机器人系统的室内场景实物实验以及异构多机器人系统的室外场景实物实验，在不同的场景和多机器人系统中都验证了本文提出的包含后处理框架的全流程多智能体路径规划解决方案的可行性和高效性。

# 6 总结与展望

## 6.1 总结

近年来随着多智能体系统在机器人和人工智能领域的广泛应用，多智能体路径规划问题被越来越多的学者关注和研究。为系统内所有智能体在避开环境中的障碍物并在保证彼此之间不发生碰撞的前提下规划出前往目标的路径，是多智能体系统协作完成复杂任务的基础保障。现有的方法可以求解一些基础的 MAPF 问题，但距离部署到实际工业场景中还存在较大的差距。同时，如何抵御潜在的工控网络攻击并识别攻击者对系统内智能体的恶意控制，保障多智能体系统安全鲁棒地执行既定路径仍然是一个开放性问题。

针对以上问题，本文提出了一套可应用于工业场景的全流程多智能体路径规划解决方案。本文首先以阿克曼运动学模型为例，研究了使用基于冲突搜索算法解决连续空间内考虑智能体运动学模型的 MAPF 问题。随后，本文将该方法一般化扩展到异构多智能体系统内，并利用滚动时域碰撞解决算法应对在线 MAPF 问题中不断输入的寻路任务流。最后，本文提出了一个 MAPF 问题的后处理框架，以此确保多智能体系统在执行阶段对于异常情况的鲁棒性。本文的主要研究成果如下：

1. 提出了一种针对阿克曼模型的基于冲突搜索算法，该求解器采用了分层搜索框架：在上层搜索中，本文提出了车身约束树的概念，只需要考虑规划方案的路径间是否存在车身冲突，而不必考虑具体的运动学约束；而在下层搜索中，本文提出了时空混合 A* 算法来进行单智能体路径规划。该方法具备连续工作空间中为大量阿克曼运动模型机器人规划安全避碰路径的能力并保持了较短的运行时间。本文还提出了一个上述方法的顺序规划版本，其牺牲了部分求解质量进一步减少了算法搜索时间。在仿真环境中的实验结果表明该方法在所有六个场景中都保持着 98% 以上的成功率，而两种基线算法的规划成功率都低于 50%，同时本文提出方法生成的规划方案质量也超过了两个基线算法的求解质量。
2. 提出了一种异构多智能体系统的在线路径规划方法。在前述连续空间阿克曼模型 MAPF 求解器的基础上，通过对下层单智能体路径规划算法的一般化，将方法进一步扩展到更接近于实际工业场景的异构多智能体系统中。同时，本文将滚动时域碰撞解决算法加入基于冲突搜索框架中，通过交替进行路径规划和路径执行降低了智能

体的空闲等待率并能够高效应对在线 MAPF 问题中不断输入的寻路任务流。本文对异构多智能体系统进行了仿真实验并与现有先进方法进行了对比，结果表明本文方法可以为多达 100 个异构智能体进行运动学可行的路径规划，并可以将在线 MAPF 规划器的速度提升六倍而几乎不影响规划方案的质量。

3. 提出了一种基于行动依赖图的多智能体路径规划后处理框架。该框架利用求解器生成的规划结果来分析智能体间移动的优先级关系，仅需智能体和规划器之间进行少量通信，即可防御利用安全漏洞对系统中智能体的恶意控制。通过该主动检测机制，本文保障了多智能体系统在长时间内安全鲁棒地执行规划方案。本文在 Gazebo 仿真环境中进行了实验，模拟了后处理框架对智能体被恶意劫持情况的处理。最后本文将后处理框架与前述 MAPF 规划器进行整合，形成了全流程多智能体路径规划解决方案。在室内场景和室外场景下分别进行了实物实验，在不同的应用场景和各异的系统中都验证了该解决方案的实际可行性和执行鲁棒性。

## 6.2 展望

本文对考虑运动学约束的异构多智能体路径规划方法进行了系统化的研究。提出了在连续空间内考虑运动学模型的规划方法，并进一步扩展到异构多智能体系统中，解决了此类系统的在线路径规划问题。最终形成了全流程多智能体路径规划解决方案，并将其应用到实际多机器人系统中，取得了阶段性的成果。但本文的研究仍然存在着一些限制和简化假设，未来的研究可以在以下几个方面展开：

- 精细的速度规划：本文提出的规划方法可以在连续空间内进行路径规划，但问题假设了智能体忽略加减速过程且能保持恒定速度移动，因此路径可能无法满足智能体二阶动力学约束。尽管本文后续提出了后处理框架来部分解决了该简化假设，但如果规划器原生支持速度规划无疑是更好的选择。未来的工作中可以考虑在保证较短搜索时间的前提下考虑速度规划，进一步提高解决方案的质量。
- 融合任务分配和路径规划：本文提出的在线路径规划方法假设了任务分配器和路径规划算法是解耦的。而在合适的场景中耦合考虑任务分配和路径规划算法可以获得更优的解决方案质量[84]，这种问题被称为融合任务分配和路径规划问题（combined Target-Assignment and Path-Finding，TAPF）[85]。未来的工作中，可以考虑将本文的方法进一步扩展到 TAPF 问题中。

# 参考文献


[1] 刘佳, 陈增强, 刘忠信. 多智能体系统及其协同控制研究进展 [J]. 智能系统学报, 2010, 5(1) : 9.

[2] SARTORETTI G, KERR J, SHI Y, et al. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning[J]. IEEE Robotics and Automation Letters, 2019, 4(3) : 2378 – 2385.

[3] DEBORD M, HONIG W, AYANIAN N. Trajectory Planning for Heterogeneous Robot Teams[J]. IEEE International Conference on Intelligent Robots and Systems, 2018 : 7924 – 7931.

[4] DOBREV Y, VOSSIEK M, CHRISTMANN M, et al. Steady Delivery: Wireless Local Positioning Systems for Tracking and Autonomous Navigation of Transport Vehicles and Mobile Robots[J]. IEEE Microwave Magazine, 2017, 18(6) : 26 – 37.

[5] MORRIS R, PASAREANU C S, LUCKOW K, et al. Planning, scheduling and monitoring for airport surface operations[C] // Workshops at the Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[6] MA H, YANG J, COHEN L, et al. Feasibility study: Moving non-homogeneous teams in congested video game environments[C] // Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment : Vol 13. 2017.

[7] YU J, LAVALLE S M. Structure and intractability of optimal multi-robot path planning on graphs[C] // Twenty-Seventh AAAI Conference on Artificial Intelligence. 2013.

[8] 杨婷, 张嘉元, 黄在起, 等. 工业控制系统安全综述 [J]. 计算机研究与发展, 2022 : 1 – 19 [2022-02-15].

[9] 徐伟, 孔坚, 毛庆梅, 等. 工业控制系统安全现状及应对策略 [J]. 网络安全技术与应用, 2021(9).

[10] HART P E, NILSSON N J, RAPHAEL B. A formal basis for the heuristic determination of minimum cost paths[J]. IEEE transactions on Systems Science and Cybernetics, 1968, 4(2) : 100 – 107.

[11] STANDLEY T. Finding optimal solutions to cooperative pathfinding problems[C] // Proceedings of the AAAI Conference on Artificial Intelligence : Vol 24. 2010.

[12] GOLDENBERG M, FELNER A, STERN R, et al. Enhanced partial expansion A[J]. Journal of Artificial Intelligence Research, 2014, 50 : 141 – 187.

[13] GOLDENBERG M, FELNER A, STURTEVANT N, et al. Optimal-generation variants of EPEA[C] // Sixth Annual Symposium on Combinatorial Search. 2013.

[14] WAGNER G, CHOSET H. M*: A complete multirobot path planning algorithm with performance bounds[C] // 2011 IEEE/RSJ international conference on intelligent robots and systems. 2011 : 3260 – 3267.

[15] WAGNER G, CHOSET H. Subdimensional expansion for multirobot path planning[J]. Artificial Intelligence, 2015, 219 : 1 – 24.

[16] FERNER C, WAGNER G, CHOSET H. ODrM* optimal multirobot path planning in low dimensional search spaces[C] // 2013 IEEE International Conference on Robotics and Automation. 2013 : 3854 – 3859.

[17] SRINIVASAN A, HAM T, MALIK S, et al. Algorithms for discrete function manipulation[C] // 1990 IEEE international conference on computer-aided design. 1990 : 92 – 93.

[18] SHARON G, STERN R, GOLDENBERG M, et al. The increasing cost tree search for optimal multi-agent pathfinding[J]. Artificial Intelligence, 2013, 195 : 470 – 495.

[19] SHARON G, STERN R, FELNER A, et al. Conflict-based search for optimal multi-agent pathfinding[J]. Artificial Intelligence, 2015, 219 : 40 – 66.

[20] BOYARSKI E, FELNER A, STERN R, et al. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding[C] // Twenty-Fourth International Joint Conference on Artificial Intelligence. 2015.

[21] FELNER A, LI J, BOYARSKI E, et al. Adding heuristics to conflict-based search for multi-agent path finding[C] // Proceedings of the International Conference on Automated Planning and Scheduling : Vol 28. 2018.

[22] LI J, FELNER A, BOYARSKI E, et al. Improved heuristics for multi-agent path finding with conflict-based search[J]. IJCAI International Joint Conference on Artificial Intelligence, 2019, 2019-Augus(2) : 442 – 449.

[23] COHEN L, WAGNER G, CHAN D, et al. Rapid randomized restarts for multi-agent path finding solvers[C] // Eleventh Annual Symposium on Combinatorial Search. 2018.

[24] LI J, HARABOR D, STUCKEY P J, et al. Disjoint splitting for multi-agent path finding with conflict-based search[C] // Proceedings of the International Conference on Automated Planning and Scheduling : Vol 29. 2019 : 279 – 283.

[25] FELNER A, STERN R, SHIMONY S E, et al. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges[C] // Tenth Annual Symposium on Combinatorial Search. 2017.

[26] SURYNEK P, FELNER A, STERN R, et al. Efficient SAT approach to multi-agent path finding under the sum of costs objective[C] // Proceedings of the Twenty-second European Conference on Artificial Intelligence. 2016 : 810 – 818.

[27] SURYNEK P. Makespan optimal solving of cooperative path-finding via reductions to propositional satisfiability[J]. arXiv preprint arXiv:1610.05452, 2016.

[28] BARTÁK R, ZHOU N-F, STERN R, et al. Modeling and solving the multi-agent pathfinding problem in picat[C] // 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI). 2017 : 959 – 966.

[29] ZHOU N-F, KJELLERSTRAND H, FRUHMAN J. Constraint solving and planning with Picat[M]. [S.l.] : Springer, 2015.

[30] VAN ROY T J, WOLSEY L A. Solving mixed integer programming problems using automatic reformulation[J]. Operations Research, 1987, 35(1) : 45 – 57.

[31] ERDEM E, KISA D G, OZTOK U, et al. A general formal framework for pathfinding problems with multiple agents[C] // Twenty-Seventh AAAI Conference on Artificial Intelligence. 2013.

[32] SURYNEK P. Multi-Agent Path Finding with Continuous Time and Geometric Agents Viewed through Satisfiability Modulo Theories (SMT)[C] // SURYNEK P, YEOH W. Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019. 2019 : 200 – 201.

[33] 舒翔翔. 多机器人最优路径规划研究 [J]. 信息与电脑, 2017(15) : 62 – 64.

[34] SILVER D. Cooperative pathfinding[C] // Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005. 2005 : 117 – 122.

[35] VAN DEN BERG J P, OVERMARS M H. Prioritized motion planning for multiple robots[C] // 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2005 : 430 – 435.

[36] BNAYA Z, FELNER A. Conflict-oriented windowed hierarchical cooperative A*[C] // 2014 IEEE International Conference on Robotics and Automation (ICRA). 2014 : 3743 – 3748.

[37] ANDREYCHUK A, YAKOVLEV K S. Two Techniques That Enhance the Performance of Multi-robot Prioritized Path Planning[C] // Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018. 2018 : 2177 – 2179.

[38] LUNA R, BEKRIS K E. Efficient and complete centralized multi-robot path planning[C] // 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2011 : 3268 – 3275.

[39] SAJID Q, LUNA R, BEKRIS K E. Multi-Agent Pathfinding with Simultaneous Execution of Single-Agent Primitives[C] // Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012. 2012.

[40] DE WILDE B, TER MORS A W, WITTEVEEN C. Push and rotate: a complete multi-agent pathfinding algorithm[J]. Journal of Artificial Intelligence Research, 2014, 51 : 443 – 492.

[41] WEI C, HINDRIKS K V, JONKER C M. Altruistic coordination for multi-robot cooperative pathfinding[J]. Applied Intelligence, 2016, 44(2) : 269 – 281.

[42] POHL I. Heuristic search viewed as path finding in a graph[J]. Artificial intelligence, 1970, 1(3-4) : 193 – 204.

[43] BARER M, SHARON G, STERN R, et al. Suboptimal Variants of the Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem[C] // Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014. 2014.

[44] THAYER J T, RUML W. Bounded suboptimal search: A direct approach using inadmissible estimates[C] // Twenty-Second International Joint Conference on Artificial Intelligence. 2011.

[45] GILON D, FELNER A, STERN R. Dynamic Potential Search - A New Bounded Suboptimal Search[C] // Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, Tarrytown, NY, USA, July 6-8, 2016. 2016 : 36 – 44.

[46] PEARL J, KIM J H. Studies in semi-admissible heuristics[J]. IEEE transactions on pattern analysis and machine intelligence, 1982(4) : 392 – 399.

[47] COHEN L, URAS T, KOENIG S. Feasibility Study: Using Highways for Bounded-Suboptimal Multi-Agent Path Finding[C] // Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel. 2015 : 2 – 8.

[48] 王毅然, 经小川, 田涛. 基于强化学习的多 Agent 路径规划方法研究 [J]. 计算机应用与软件, 2019, 36(8) : 165 – 171.

[49] 郑延斌, 李波, 安德宇. 基于分层强化学习及人工势场的多 Agent 路径规划方法 [J]. 计算机应用, 2015, 35(12) : 3491 – 3496.

[50] LIU S, WEN L, CUI J, et al. Moving Forward in Formation: A Decentralized Hierarchical Learning Approach to Multi-Agent Moving Together[C] // IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021, Prague, Czech Republic, September 27 - Oct. 1, 2021. 2021 : 4777 – 4784.

[51] Stern R, Sturtevant N R, Felner A, et al. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks[C] // Proceedings of the International Symposium on Combinatorial Search : Vol 10. 2019 : 151 – 158.

[52] PHILLIPS M, LIKHACHEV M. Sipp: Safe interval path planning for dynamic environments[C] // 2011 IEEE International Conference on Robotics and Automation. 2011 : 5628 – 5635.

[53] YAKOVLEV K, ANDREYCHUK A. Any-angle pathfinding for multiple agents based on SIPP algorithm[C] // Twenty-Seventh International Conference on Automated Planning and Scheduling. 2017.

[54] HÖNIG W, KUMAR T K S, COHEN L, et al. Summary: Multi-Agent Path Finding with Kinematic Constraints[C] // Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. 2017 : 4869 – 4873.

[55] LI J, SURYNEK P, FELNER A, et al. Multi-Agent Path Finding for Large Agents[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2019, 33 : 7627 – 7634.

[56] ATZMON D, ZAX Y, KIVITY E, et al. Generalizing Multi-Agent Path Finding for Heterogeneous Agents Generalizing Multi-Agent Path Finding for Heterogeneous Agents[C] // Thirteenth Annual Symposium on Combinatorial Search. 2020.

[57] ROBINSON D R, MAR R T, ESTABRIDIS K, et al. An Efficient Algorithm for Optimal Trajectory Generation for Heterogeneous Multi-Agent Systems in Non-Convex Environments[J]. IEEE Robotics and Automation Letters, 2018, 3(2): 1215–1222.

[58] PARK J, KIM J, JANG I, et al. Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee using Relative Bernstein Polynomial[C] // 2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020. 2020: 434–440.

[59] LI J, RAN M, XIE L. Efficient trajectory planning for multiple non-holonomic mobile robots via prioritized trajectory optimization[J]. IEEE Robotics and Automation Letters, 2020, 6(2): 405–412.

[60] MA H, LI J, KUMAR T K S, et al. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks[C] // Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017. 2017: 837–845.

[61] ŠVANCARA J, VLK M, STERN R, et al. Online multi-agent pathfinding[C] // Proceedings of the AAAI Conference on Artificial Intelligence: Vol 33. 2019: 7732–7739.

[62] LI J, TINKA A, KIESEL S, et al. Lifelong Multi-Agent Path Finding in Large-Scale Warehouses[C] // Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. 2021: 11272–11281.

[63] LIU M, MA H, LI J, et al. Task and path planning for multi-agent pickup and delivery[C] // Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS). 2019.

[64] MA H, HÖNIG W, KUMAR T K S, et al. Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery[J]. Proceedings of the AAAI Conference on Artificial Intelligence, 2019, 33: 7651–7658.

[65] HÖNIG W, KIESEL S, TINKA A, et al. Persistent and robust execution of MAPF schedules in warehouses[J]. IEEE Robotics and Automation Letters, 2019, 4(2): 1125–1131.

[66] DOLGOV D, THRUN S, MONTEMERLO M, et al. Practical search techniques in path planning for autonomous driving[J]. Ann Arbor, 2008, 1001(48105): 18–80.

[67] REEDS J, SHEPP L. Optimal paths for a car that goes both forwards and backwards[J]. Pacific journal of mathematics, 1990, 145(2): 367–393.

[68] HÖNIG W, PREISS J A, KUMAR T S, et al. Trajectory planning for quadrotor swarms[J]. IEEE Transactions on Robotics, 2018, 34(4): 856–869.

[69] BOTROS A, SMITH S L. Computing a Minimal Set of t-Spanning Motion Primitives for Lattice Planners[C] // 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2019: 2328–2335.

[70] NEGENBORN R R, DE SCHUTTER B, HELLENDOORN J. Multi-agent model predictive control: A survey[J]. arXiv preprint arXiv:0908.1076, 2009.

[71] ROBINSON D R, MAR R T, ESTABRIDIS K, et al. An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments[J]. IEEE Robotics and Automation Letters, 2018, 3(2): 1215–1222.

[72] PARK J, KIM J, JANG I, et al. Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial[C] // 2020 IEEE International Conference on Robotics and Automation (ICRA). 2020: 434–440.

[73] ATZMON D, ZAX Y, KIVITY E, et al. Generalizing Multi-Agent Path Finding for Heterogeneous Agents[C] // Thirteenth Annual Symposium on Combinatorial Search. 2020.

[74] WAN Q, GU C, SUN S, et al. Lifelong multi-agent path finding in a dynamic environment[C] // 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV). 2018: 875–882.

[75] GRENOUILLEAU F, van HOEVE W-J, HOOKER J N. A multi-label A* algorithm for multi-agent pathfinding[C] // Proceedings of the International Conference on Automated Planning and Scheduling: Vol 29. 2019: 181–185.

[76] CÁP M, VOKRÍNEK J, KLEINER A. Complete Decentralized Method for On-Line Multi-Robot Trajectory Planning in Well-formed Infrastructures[C] // Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015. 2015: 324–332.

[77] GREGOIRE J, CÁP M, FRAZZOLI E. Locally-optimal multi-robot navigation under delaying disturbances using homotopy constraints[J]. Autonomous Robots, 2018, 42(4): 895–907.

[78] KOENIG N, HOWARD A. Design and use paradigms for gazebo, an open-source multi-robot simulator[C] // 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) : Vol 3. : 2149 – 2154.

[79] KATZ B, DI CARLO J, KIM S. Mini cheetah: A platform for pushing the limits of dynamic quadruped control[C] // 2019 international conference on robotics and automation (ICRA). 2019 : 6295 – 6301.

[80] GRISETTI G, STACHNISS C, BURGARD W. Improved techniques for grid mapping with rao-blackwellized particle filters[J]. IEEE transactions on Robotics, 2007, 23(1) : 34 – 46.

[81] DELLAERT F, FOX D, BURGARD W, et al. Monte carlo localization for mobile robots[C] // Proceedings 1999 IEEE International Conference on Robotics and Automation : Vol 2. 1999 : 1322 – 1328.

[82] SHAN T, ENGLOT B, MEYERS D, et al. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping[C] // 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2020 : 5135 – 5142.

[83] BIBER P, STRASSER W. The normal distributions transform: a new approach to laser scan matching[C] // Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems : Vol 3. 2003 : 2743 – 2748 vol.3.

[84] HENKEL C, ABBENSETH J, TOUSSAINT M. An optimal algorithm to solve the combined task allocation and path finding problem[C] // 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2019 : 4140 – 4146.

[85] MA H, KOENIG S. Optimal Target Assignment and Path Finding for Teams of Agents[C] // Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016. 2016 : 1144 – 1152.

# 作者简历

温力成，男，1996 年 12 月出生于江苏省南京市。2015 年 9 月进入浙江大学工信大类，2019 年 6 月本科毕业，获得浙江大学自动化（控制）专业的工学学士学位。2019 年 9 月推荐免试进入浙江大学控制科学与工程学院攻读硕士学位，师从刘勇教授。

# 攻读学位期间取得的科研成果

1. “CL-MAPF: Multi-Agent Path Finding for Car-Like robots with kinematic and spatiotemporal constraints”, Robotics and Autonomous Systems, Elsevier. 已发表，SCI，控制学院高水平论文，第一作者。
2. “Collision-free Trajectory Planning for Autonomous Surface Vehicle”, 2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). 已发表，EI 会议，第一作者。
3. “Moving Forward in Formation: A Decentralized Hierarchical Learning Approach to Multi-Agent Moving Together”, 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 已发表，CCF C 类会议，第二作者。
4. “TorPeDo: A Smooth and Safe Trajectory Planner for Differential Unmanned Surface Vehicles”. 论文在投，SCI，第一作者。
5. 发明专利“一种基于阿克曼模型的移动机器人避障路径规划方法”已公开。（除导师外）学生第一发明人。