学习冗余机器人的任务执行层次结构

Alessandro Adami *,‡ , Aris Synodinos † , Matteo Iovino † , Ruggero Carli * , Pietro Falco *

***Abstract*—现代机器人系统——例如移动操作器、人形机器人和带有机械臂的空中机器人——通常具有高度的冗余能力，使它们能够同时执行多个任务。管理这种冗余是实现可靠和灵活行为的关键。一个广泛使用的方法是任务堆栈（Stack of Tasks，简称 SoT），它在一个统一的框架内按优先级组织控制目标。然而，传统的 SoT 是由专家手动设计的，限制了其适应性和可访问性。本文介绍了一种新颖的框架，能够从用户定义的目标中自动学习 SoT 的层次结构和参数。通过结合强化学习和遗传编程，系统在无需人工干预的情况下发现任务优先级和控制策略。基于精度、安全性和执行时间等直观指标的成本函数引导学习过程。我们通过在高冗余的双臂移动操作器平台 mobile-YuMi 上的仿真和实验验证了我们的方法。结果表明，学习到的 SoT 使机器人能够动态适应变化的环境和输入，在保持健壮的任务执行的同时平衡竞争的目标。该方法为复杂机器人中的冗余管理提供了一种通用且用户友好的解决方案，推动了以人为中心的机器人编程，并减少了对专家设计的需求。**

***Note to Practitioners*—在现实世界的机器人应用中，如协同制造、检查或服务任务中，机器人通常拥有比单一任务严格需要的更多自由度。这种冗余使它们能够同时执行多个任务，例如在避免附近人类的同时保持工具稳定，或在最大化可操作性时到达目标。传统上，工程师必须手动编程任务优先级和控制策略以利用这种冗余，这是一个复杂且耗时的过程。本文介绍了一种基于学习的框架，使机器人能够自动生成和调整任务层次结构，结合遗传编程和强化学习。用户通过简单且可定制的成本函数来指定其目标（例如优先考虑障碍物避免或精度），系统演化出一个实时平衡不同目标的“任务堆栈”。该方法在真实的 ABB 移动 YuMi 平台上进行了验证，展示了学习行为如何使机器人在变化的条件下自适应协调多个目标。该框架具有普遍性，适用于任何冗余机器人平台，降低集成努力并在复杂环境中增加自主性。**

***Index Terms*—Genetic Programming, Reinforcement Learning, Learning Stack of Tasks, Redundancy, Task Prioritization, Redundant Robots**

I. 介绍

MODERN 机器人系统越来越多地部署在复杂、无结构和动态的环境中，在这些环境中，传统的刚性控制架构难以满足操作需求。特别是，移动操控器，如移动-YuMi 研究平台，展示了下一代机器人系统，这些系统结合了高灵活性与在不可预测环境中导航和与物体交互的能力。这些系统的特点是其高冗余度，这不仅提供了显著的灵活性和执行各种任务的能力，同时在控制、规划和任务优先级方面也带来了巨大的挑战。在机器人操纵器中，冗余意味着系统拥有比执行特定任务严格所需更多的自由度（DOF）。这种过剩的自由度可以用来优化多个目标，例如躲避障碍物、最大化操纵性 [**MultipleTaskPriority**]，或保持机械关节限制的安全距离。然而，管理这种冗余并非易事，尤其当机器人被要求同时执行多个任务时。决定采用哪种任务执行策略是一个极具挑战性的问题，也是一个活跃的研究领域。在 [**HierarchicalRedundancyResolutionUnderArbitraryConst** 中，作者提出了一个统一的控制框架，以解决具有冗余的机器人的运动学和动态控制，同时处理层次约束并通过广义零空间算法优化冗余。在 [**PickAndPlaceYumi**] 中，作者介绍了一种用于动态环境中移动双手操作的在线运动学控制方法，使用分布式距离传感器和优先约束优化来实现实时碰撞避免和任务执行。在当前实践中，冗余管理问题由专家程序员手动处理，他们对任务序列和决策规则进行硬编码。这种手动方法不仅耗时且成本高，而且不灵活：让机器人适应新的任务或环境通常需要重新编程整个任务逻辑。为了解决这些限制，我们提出了一种方法，通过使用代价函数来引导任务参数和优先级调整，自动学习完成高层任务所需的任务堆栈。这种方法减少了对手动调整的依赖，并能够在不同情况下实现可扩展、适应性的机器人行为。同时，基因编程作为一种强大的进化方法出现，用于自动生成控制程序 [**GPTutorial**]。与操作固定长度字符串的传统遗传算法不同 [**GAReview**]，基因编程进化可变长度的程序，从而捕获复杂任务的层次性和动态性 [**GAandGPComparison**]。本文提出了一个综合框架，利用强化学习和基因编程来解决移动操作器中的任务优先级和冗余管理的双重挑战（如图 **??** 所示）。我们方法的核心是引入了“任务堆栈”(SoT) 表示法到基因编程中，这是一种结构化格式，封装了构成高级任务的每个任务的优先顺序、相关控制参数和激活状态于单个复合实体中。SoT 不仅提供了一种自然而高效的方式来表示任务层次结构，还促进了基因操作（如变异和交叉）的应用，以进化和优化任务堆栈。SoT 的演变由用户定义的代价函数指导，这个代价函数是多个性能指标的加权组合。这些指标在算法开发期间定义，以便利用机器人在现实情况下可能有用的特征。它们包括精度（通过与期望位姿的偏差来衡量）、安全性（根据与障碍物的距离来评估）、可操作性（严格与运动学性能相关）和执行时间。在我们的框架内，强化学习与基因编程的结合提供了几个显著的优势。首先，它使得机器人系统能够自动处理冗余，而无需手动设计堆栈，增强了鲁棒性和灵活性。其次，遗传编程的进化特性使得能够自动发现最优的任务序列和控制参数，从而减少了手动调优和先验知识的需求。第三，通过显式地结合用户定义的成本函数，我们的方法确保了优化过程与应用程序的特定目标和约束保持一致。由于任务顺序和参数的学习独立于环境，因此它们可以推广到任何适用于相同任务和成本函数的真实世界环境。这使得实现可扩展、自适应的机器人行为成为可能，而无需进行手动重新编程。所提

These experiments were carried out in the WASP Research Arena (WARA)-Robotics, hosted by ABB Corporate Research Center in Västerås, Sweden and financially supported by the Wallenberg AI, Autonomous Systems, and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.
Project co-funded by the European Union –Next Generation Eu - under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2, Investment 3.3 –Decree no. 630 (24^{th} April 2024) of Italian Ministry of University and Research; Concession Decree no. 1956 del 05^{th} December 2024 adopted by the Italian Ministry of University and Research, CUP D93D24000270003, within the national PhD Programme in Autonomous Systems (XL cycle).

∗ University of Padova, Dept. of Information Engineering, Italy.
† Work performed while at ABB Corporate Research, Västerås, Sweden.
‡ Polytechnic of Bari Dept. of Electrical and Information Engineering, Italy.

出的框架通过包括模拟和实际实验在内的全面实验研究加以验证。模拟是在 Gazebo 环境中进行的，机器人在一个严格定义的动态环境中操作。这些模拟对于评估遗传编程算法的收敛特性以及在物理硬件部署之前微调 RL 参数至关重要。随后的在移动-YuMi 研究平台上的实验进一步展示了该方法的实际可行性，展示了其适应意外变化（如移动障碍物和任务优先级变化）并确保稳定和高效运行的能力。总之，本文的主要贡献有两个方面。

- 该工作的核心贡献是提出一种方法，用于学习冗余机器人的 SoT 结构——包括任务排序、参数调优和任务激活——基于高级用户定义的成本函数。特别是，我们将问题框架化为强化学习风格设置中的策略搜索，其中策略对应于结构化的 SoT。遗传编程被用作全局优化器，以发现有效的 SoT 配置。与传统的 RL 不同，我们不训练策略函数：相反，我们在离散-连续策略空间中直接进行搜索，使用周期性成本作为反馈。
- 由于任务顺序是通过任务参数自动学习的——而不是手动设计或者针对特定环境进行调整的——因此它可以直接推广到不同的现实环境中，在这些环境中适用于相同的高层任务和成本函数。这种全自动的方法消除了手动重新编程的需求，使机器人在各种部署场景下实现可扩展和适应的行为。我们通过模拟和实验室实验来展示我们方法的实际效能，突出了它在动态和不可预测环境中的稳健性，同时保持对安全和性能的强烈关注。

本文的其余部分组织如下。第 II 节回顾了机器人系统中冗余管理和任务优先级相关的工作。第 III 节和 **??** 节详细介绍了支持我们集成方法的理论框架，包括对强化学习（RL）和遗传编程方法的讨论。第 IV 节描述了实验设置，包括仿真环境和移动-YuMi 测试平台。第 V 节展示并分析了实验结果，第 VI 节总结了本文并指出了未来研究的方向。

II. 相关工作

SoT 框架已广泛应用于机器人控制中以管理冗余和任务优先级。大量文献已证明 SoT 在各种背景下的灵活性和有效性。然而，任务的定义及其层次结构通常是手动完成的。 [**SoTInvertedKin**] 的作者引入了一种用于人形机器人控制的广义 SoT 框架，其中位置、方向和姿势等任务是手动指定和优先级排序的。同样，在动态动作捕捉的背景下， [**SoTDynMotion**] 中的 SoT 范式被应用于在机器人上再现和编辑人类动作。在后来的工作中 [**SoTUnilateralConstarints**] ，SoT 公式被扩展以纳入单侧约束，如接触力。虽然这允许更具表现力的控制，但任务和约束的定义和优先级排序仍然是手动的。 [**SoTVisuoTactile**] 的作者将视觉-触觉感知集成到 SoT 框架中以实现精细操作。最终的任务仍然基于传感器解读和任务启发式手动定义。最近， [**SoTWithBT**] 中提出将 SoT 与行为树结合以引入模块化和高级决策。尽管如此，即使在这种混合架构中，SoT 中的各个任务仍由开发人员明确指定。虽然任务堆栈（SoT）范式被广泛用于机器人中的冗余管理，任务优先级、控制参数和激活结构通常仍由专家手动设计，限制了在动态环境中的可扩展性和适应性。一些近期的工作引入了数据驱动的方法——例如强化学习或行为树——来自动化任务选择或控制策略生成，但它们止步于明确学习完整的 SoT 结构。近年来，遗传算法（GA）和结合行为树（BT）的遗传编程在机器人领域得到有效应用，以解决在具有多于所需自由度的机器人控制和路径规划中与冗余相关的挑战。这种特性允许很大的灵活性，但也给任务计算引入了复杂性。在 [**GARedundantManip**] 中，作者解决了在存在障碍物的环境中进行冗余机械手点到点运动的挑战。GAs 被用于寻找避免碰撞同时确保精确运动的最佳路径。在 [**IKRedRobotsGA**] 中，作者提出了一种改进的 GA 以解决在有障碍物的环境中冗余机器人操控手的逆运动学问题。该方法将问题表述为优化任务，旨在同时最小化末端执行器的位置误差和机器人的关节位移，比传统方法显示出更高的精度。此外，层次化任务优先级已经被广泛研究，诸如零空间投影和行为树的方法为管理机械臂中的冗余自由度提供了一个框架 [**LearningBTwithGPUmpredictEnv**, **LowLovelFlexiblePlanning**] 。这些方法表明，这样的层次结构不仅提高了任务执行的可靠性，还促进了多个并发任务的平滑集成。在 [**MerginGlobalAndLocalPlanners**] ，作者提出了一个双层运动规划框架，该框架结合了基于采样的笛卡尔空间规划器与基于集合的逆运动学，以实现动态、部分非结构化环境中冗余机械臂的高效反应性控制。然而，其性能可能受到对准确感知和环境建模的依赖的限制，并且在影响实时响应的复杂高维任务中可能出现可扩展性挑战。

新兴技术，包括深度强化学习（deep RL） ，在处理机器人的复杂决策问题中展示了显著的潜力 [**RLinRoboticsSurvey**] 。深度强化学习可以直接从高维传感器数据中学习稳健的控制策略。然而，这些方法通常缺乏任务优先级的显式表示，使得在安全或精度要求变化时难以解释和调整控制层次。然而，这些工作并不旨在自动发现或优化具有任务特定零空间投影和控制层次优先级的 SoT 层次结构。与现有方法相比，我们的工作引入了一个综合框架，该框架无需基于遗传编程的微调方法，通过仿真到实际转移自动处理冗余管理。在大多数最先进的机器人系统中，冗余管理是由专业程序员处理的，他们手动定义任务堆栈（SoT）。然而，这种手动方法既昂贵又缺乏灵活性。此外，手动为高度冗余的机器人（如双臂移动操控器）设计任务堆栈是不切实际的，因为随着任务数量的增加，它变得过于复杂且难以管理。我们的方法不仅可以演化出任务执行的最佳顺序，还能够实时调整控制参数以最小化用户定义的成本函数。通过动态学习任务优先级和相关参数，所提出的框架直接解决了在先前方法中静态任务层次结构和手动调整的局限性。据我们所知，此前没有工作通过性能驱动的方法共同学习任务顺序、控制参数和冗余机器人任务堆栈架构的任务激活标志。我们的工作填补了这一空白，将任务堆栈视为结构化策略表示，通过用户定义的成本函数进行演化，从而实现无需微调的模拟到真实转移和在复杂场景中自动的鲁棒行为适应。

III. 问题陈述和建议的解决方案

管理具有多个自由度（DOF）的机器人操作臂的冗余是一个关键挑战。机器人操作臂的冗余发生在执行给定任务所需的自由度多于严格必要的自由度时。正式地，对于一个在维度为 m 的任务空间中操作的拥有 n 个自由度的操作臂，冗余被定义为满足条件 $n > m$ 的情况。例如，考虑一个具有 7 个自由度的机器人臂（即具有七个回转关节的手臂），其任务是将其末端执行器定位在定义为 $SE(3)$ 的 3D 空间中的一个特定姿态中。由于在 $SE(3)$

中指定一个唯一姿态需要 $m = 6$ 个参数（其中三个用于位置，三个用于方向），因此完全限制这一任务只需要 6 个自由度。因此，7 自由度机械臂（$n = 7$ ）被称为运动学冗余，因为它拥有比任务所需更多的一个自由度。这样的冗余机械臂允许存在无限的关节配置来实现相同的末端执行器姿态，使其在受限或动态环境中具有很高的多功能性 [**RoboticsSiciliano**] 。为了从解决上述任务的无限关节配置中选择一个，我们可以同时引入一个次要任务来完成，例如，最大化可操作性。这样，我们定义一个由两个任务组成的简单任务堆栈。需要注意的是，SoT 方法允许同时处理利用冗余的多个任务。因此，可以利用冗余来优化多个目标，例如避开障碍物、最小化能耗或避免接近关节极限。然而，虽然冗余管理是一个研究充分的问题，我们的重点在于自动处理这一额外的自由度，以确定众多可能解决方案中哪个最符合整体系统目标。这不仅需要先进的控制策略，还需要一种机制来根据任务参数和任务目标自主选择和优先排序行动。尽管这种冗余提供了更大的灵活性和适应性潜力，但它在决策中也引入了复杂性，因为机器人可以以不同程度的效率、精度、负担和其他性能指标完成任务。在动态和不可预测的环境中，机器人调整其运动的能力对于实现最佳性能至关重要。如果没有稳健的方法选择最适当的配置，冗余带来的优势可能很容易丧失，导致结果低效甚至任务执行失败。我们要解决的核心问题是自动推导出一种最佳任务执行管理方式，称为任务堆叠（SoT），以使机器人在动态或未知的环境中成功完成高级任务。与其依赖手动定义的任务顺序，这通常是僵化且特定于环境的，我们的目标是通过模拟学习该堆栈，并使用任务参数和成本函数来指导优先级，从而直接将其转移到现实环境中。这使得无需传统的硬编码任务堆叠公式即可实现自适应、可扩展且具环境感知的冗余管理。所提出的解决方案（在图 **??** 中描述）包括强化学习和遗传编程技术，结合并增强这两种方法的特性，以找到问题的最佳解决方案。该框架优化了不同任务的执行和优先级，以执行全局任务，同时最小化与单个任务相关的成本。任务堆叠架构通过编码执行的优先顺序来表示机器人允许执行的任务，同时携带关于参数和适用性度量的信息。在我们的解决方案中，强化学习（RL）和遗传编程被用于优化高层任务的叠加任务（SoT）结构表示。其中，RL 组件利用遗传编程优化器通过环境反馈来使系统学习最佳的叠加任务。具体来说，智能体根据用户特定的成本函数定义的性能指标获得奖励。然后利用该奖励信号根据既定的算法更新叠加任务。需要最小化的成本函数由用户根据应用的具体优先级进行定制，例如最小化时间或最大化精度。遗传编程目前被用于演化叠加任务的结构。遗传操作符如变异和交叉被应用到候选解决方案，每个候选方案都编码了特定任务优先级和相关控制参数。在遗传编程过程中，适应性评估直接受到 RL 衍生的成本函数的影响。

最初，所有叠加任务被执行，并且 RL 赋予相应的成本函数。遗传编程随后探索旨在最小化该成本的新潜在解决方案。这个迭代过程持续进行直到达到平稳点，此时最优的叠加任务被确定。这种集成确保了进化过程不仅偏好于实现低成本的叠加任务，还使其能够适应 RL 组件提供的动态反馈。在算法流程中，用户根据必须在特定用例中满足的要求定义一个成本函数 C 。为了便于使用，用户直接为给定情景指定所需的高级要求及其权重。然后，这些选择会自动映射到一个凸成本函数中。此外，最先进的方法可以被改编并用于从自然语言任务描述中直接生成奖励函数 [**TowardsAutonomousRLwithLLMs**] 。然后，大小为 n_T 的初始 SoTs 候选群体 — 其中 n_T 是任务数量 — 被随机创建，通过强化学习（RL）评估每一个候选的成本。相同的高级任务在所有候选中执行，初始条件为随机化，直到达到目标或执行失败为止。一旦当前群体的所有 SoTs 都被评估后，使用遗传编程技术创建新个体，并可能找到问题的更好解决方案。作为第一步，应用遗传选择，将 SoTs 配对，并在下一代中仅传播具有最低成本（即性能最佳）的个体，丢弃最不有效的解决方案，因为它不利于提高整体性能。因此，框架采用锦标赛选择 [**GPTournamentSelection**] ，这是一种在遗传编程中常用的方法，以进化出更好的任务优先策略。此二元锦标赛方法确保更佳性能的解决方案更有可能传播，引导搜索朝向更高效且具适应性的任务管理策略。在第二步中，通过遗传编程操作（交叉图 2 和变异图 3 ），初始个体数量将被恢复。然后，也将对新的 SoTs 个体进行评估。这个循环持续进行，直到适应度评分收敛到一个稳定值，这意味着进化的 SoT 个体之间几乎是相等的。当两个不同的 SoT 的距离（见第 **??** 节）低于一个基于经验观察手动选择的阈值时，我们认为它们几乎是相等的。

在这个背景下，遗传编程的使用提供了一种框架，使机器人能够学习任务优先级和参数。机器人探索各种潜在解决方案，不断完善其方法，以达到任务执行的最佳平衡。

之后，最佳的 SoT——即最终种群中成本最低的一个——将在真实实验室环境中进行测试，以证明算法在为高级任务找到最佳解决方案中的有效性。

提出的算法的伪代码在 Alg. **??** 中给出，而更多关于理论框架和使用的技术的细节将在以下章节中介绍。

在本节中，我们提供了上节中说明的理论框架的详细信息，重点在于方法和实施。

机器人可以执行从一个预定义字典中选择的任务。对于每个任务，我们定义要控制的任务变量为 $\boldsymbol{x} \in \mathbb{R}^m$ ，系统配置为 $\boldsymbol{q} \in \mathbb{R}^n$ 。让我们用函数 $\boldsymbol{k}$ 表示 $\boldsymbol{x}$ 和 $\boldsymbol{q}$ 之间的关系：

在假设 $\boldsymbol{k}$ 是可微的条件下，我们有：其中 $\dot{\boldsymbol{q}}$ 和 $\dot{\boldsymbol{x}}$ 分别是 $\boldsymbol{q}$ 和 $\boldsymbol{x}$ 的时间导数，而 J(q) 是与任务 T 相关的雅可比矩阵。给定一个期望的任务变量轨迹 $\boldsymbol{x_d}(t)$ ，理想地完成任务意味着生成一个轨迹 $\boldsymbol{q}_d(t)$ ，以便 $\boldsymbol{k}(\boldsymbol{q}_d(t)) = \boldsymbol{x}_d(t)$ 。从期望值 $\boldsymbol{x}_d(t)$ 和可能的 $\dot{x}_d(t)$ 开始，为机器人系统生成运动参考 $\boldsymbol{q}_d(t)$ 的有效方法是在微分级别通过计算机器人系统 [**antonelli2008null**] 的速度参考 $\dot{q}(t)$ 来实现。根据任务的特定类型，我们使用以下两种方法之一计算 $\dot{q}(t)$ 。

更一般地，我们可以用一个参数 θ 来描述每个任务 T ，该参数包括在学习目标中。根据这种参数化，在闭环任务中我们有 $\theta = \gamma_{CL}$ ，而在开环任务中我们有 $\theta = \gamma_{OL}$ 。

A. 零空间投影

任务可以堆叠在一个有序元组中，一旦部署在机器人上，该堆栈是不可变的。由此，一个优先级较低的任务可以迭代地投射到另一个优先级较高的任务的零空间中 [**MultipleTaskPriority**] [**RoboticsSiciliano**] 使用雅可比矩阵。以两个任务为例，该投影定义为：

$$\dot{\boldsymbol{q}}_d = \dot{\boldsymbol{q}}_1 + \left(\boldsymbol{I}_n - \mathrm{J}_1^{\dagger}\mathrm{J}_1\right)\dot{\boldsymbol{q}}_2. \tag{1}$$

其中 $\dot{\boldsymbol{q}}_1$ 和 $\dot{\boldsymbol{q}}_2$ 分别是通过求解主任务和次要任务获得的关节速度指令，而 J_1 是与高优先级任务相关的雅可比矩阵。当机器人必须同时执行多个任务时，零空间投影是一个至关重要的工具，以确保次要任务不会干扰主要任务。

B. 成本函数

成本函数 C 是该框架的一个关键方面，因为它定义了在同一场景中不同任务序列的性能。这可以嵌入用户偏好并增强易用性。此外，它是 RL 框架中遗传编程过程的关键组成部分，因为它是遗传选择和性能比较的基础。用户可以通过将成本函数定义为预定义成本操作符的加权组合来调节机器人在执行某些任务组合时的优先级。用户可以将成本与相对权重组合为：

$$C = \alpha_1 cost_1 + \alpha_2 cost_2 + ... + \alpha_n cost_n, \tag{2}$$

其中权重被归一化，使得 $\sum_{i=0}^{n} \alpha_i = 1$ 。此函数在学习阶段确定需优化的参数时至关重要，同时考虑了用户偏好。此实现允许根据应用的具体要求定制成本函数，确保优化过程与操作目标和安全约束一致。
用户被允许通过图形用户界面在训练阶段的开始选择定义成本函数的高层次需求和偏好，如在第 IV-C 节所述。因此，一旦用户定义了适应度测量的权重 C ，框架就会自动在一个仿真环境中进化出一个 SoF。

C. 任务堆叠

这项工作旨在通过学习一组任务的优化和优先顺序，自动管理复杂机器人系统的冗余。其采用遗传编程和强化学习技术，指导由用户定义的成本函数，反映与任务相关的特定性能标准，例如最小化执行时间或优先考虑感知到的安全性。
在下面，将遗传编程框架应用于一个 SoT 结构中。我们问题中潜在最佳解决方案群体中的每个个体都表示为一个堆栈，其中任务描述及其参数被存储在一起。数据结构中的特定顺序定义了任务的优先级，这通过零空间投影得到保证。
在实现中，堆栈的第一个位置，即图 1 数组中的索引 0 ，记录了与成本函数相关的值。因此，所有堆栈都携带了比较它们（一个成本函数 C ）和执行高层任务（一个参数集 $\boldsymbol{\theta}$ 和优先顺序 P ）所需的所有信息。

$$[\,\boldsymbol{C}\,,\,[\;\;\mathbf{1}^{st}\;\texttt{task}][\,,\,\mathbf{2}^{nd}\;\texttt{task}][\,,\,\mathbf{3}^{rd}\;\texttt{task}]\;\,,\,].$$

Figure 1. 任务堆栈（SoT）表示法。与 SoT 执行相关的成本位于表示法的开头，后面是根据优先级排列的任务。

在图 1 中，报告了一个简单的 SoT 示例。在这种情况下，标记为 1^{st} 的任务以最高优先级完全执行，然后标记为 2^{nd} 的任务被投影在第一个任务的零空间中，而 3^{rd} 则被投影在两个具有更高优先级的任务的组合零空间中 [**MultipleTaskPriority**] 。注意，其他任务可能被串联到 SoT 结构中。可以通过改变和结合优先顺序、活动任务和参数，并结合遗传编程来找到使成本函数最小化的解决方案。
参数，表示为 $\boldsymbol{\theta}$ ，限制在由任务的先验知识所规定的特定范围内。例如，逆运动学增益被限制为正且低于 2 ，以满足安全原因所需的稳定性条件 [**StabilityClosedLoopIK**] 。我们引入了一种受生物启发的布尔激活机制，用于优先控制框架中的任务包含。这个机制基于内含子 [**GPIntronsBenefit**] ，允许保留潜在的任务模块——类似于遗传学中的隐性等位基因——从而在不破坏结构的情况下进行探索 [**GPExplicitelyDefinedIntrons**] 。据我们所知，这种方法尚未正式应用于运动控制或任务堆栈框架，提供了一种新颖的方法，在进化优化过程中保留潜在有益的行为。控制层次结构中的每个任务都与一个布尔激活标志相关，该标志决定它是否对整体控制输出有贡献。当标志被设为 `True` 时，该任务被视为激活状态，并通过其对应的任务堆栈投影参与期望关节速度的计算。如果标志为 `False` ，则该任务在投影计算中被跳过，不影响结果运动，但其参数保存在基因型中。这些不活跃的任务作为潜在特征存在；它们在当前行为中不表达，但可能通过突变或重组在将来的世代中变为活跃。这一机制使得进化算法不仅能够改变任务的参数，还能够改变控制堆栈的有效长度和组成，而不会永久丢弃潜在有用的行为。通过允许解决方案保留并在不同的环境背景或任务配置中重新引入可能变得有利的任务，它支持了更灵活的搜索过程。因此，种群随着时间的推移保持更大的行为多样性，这可以降低早熟收敛的风险，并提高最终解决方案的稳健性和泛化能力。
成本一旦达到总体高级任务的目标、或超过最大时间限制、或者遇到模拟的关键点（如机器人撞到障碍物或机械臂配置达到奇异性），就会被评估。一旦计算出成本，就可以用于遗传选择。随后，将遗传操作应用于任务堆栈结构。在发生变异的情况下，任务的优先级顺序或任务在堆栈中的存在可以随机改变，产生新后代并加入到下一个种群中。如果应用交叉操作，则另一个幸存的任务堆栈会被随机选择为第二个父代，然后派生出后代。因此，所提出的框架结合了遗传编程和强化学习，以找到优化优先级顺序 P 和参数 $\boldsymbol{\theta}$ 的集合，从而最小化成本函数 C ：

$$\hat{P}, \hat{\boldsymbol{\theta}} = \underset{P,\boldsymbol{\theta}}{\operatorname{argmin}}\, C. \tag{3}$$

每个任务堆栈可以形式化地表示为一个映射 S ，它将用户输入的成本函数映射为所需的关节速度，包括优先级顺序、参数和任务 T 从字典 D ：

$$S : (P, \boldsymbol{\theta}, T|C) \longrightarrow \dot{\boldsymbol{q}}_d. \tag{4}$$

总之，任务堆栈是一种结构化表示，堆栈的每个元素对应一个单独的任务。每个任务条目包括：

- 优先级：决定执行顺序，优先级较高的任务优先执行。
- 控制参数：任务执行的具体设置。
- 激活标志：指示任务是激活还是不激活，允许动态修改堆栈而不丢失潜在的任务信息。

给定两个不同的 a 和 b （长度为 n ）的 SoT，它们之间的距离——用于量化两个解决方案之间的差异程度——可以计算为：对于列表中的所有 n 任务 T^i 。$\delta_\theta(\theta_a^i, \theta_b^i) = \|\theta_a^i - \theta_b^i\|$ 返回在不同 SoT 中，同一任务的两个不同实现的参数差异的总和（即 T_a^i 和 T_b^i ），和

$$\delta_{prior}(T_a^i, T_b^i) = \begin{cases} |\pi_a^i - \pi_b^i| & \text{if } T_a^i \text{ and } T_b^i \text{ are both active} \\ n & \text{otherwise} \end{cases} \tag{5}$$

，其中 π_a^i 和 π_b^i 是在每个 SoT 的活动任务集中任务 T_a^i 和 T_b^i 的优先级位置。

D. 遗传编程作为强化学习的成本最小化器

基因编程 [**GPTutorial**] 是一种受自然进化原理启发的计算方法。其主要目标是自动生成能够有效解决特定任务的计算机程序。与以固定长度字符串编码基因组运作的基因算法（GA）[**GAReview**] 不同，基因编程进化的是可

变长度的程序，从而允许更灵活和动态的解决方案。
对于我们的目的，基因编程作为开发 RL 算法 [**RLSurveyShakya**, **RLSutton1998**] 的基础，使我们能够最小化适应度测量。与依赖标记数据集的监督学习不同，RL 代理通过获得反馈并在迭代试错过程中以奖励的形式调整其策略。因此，适应度评估是 RL 的关键组成部分，并随着基因编程范式的整合而得到显著改进 [**HybridNNGP**, **EvolutionaryApproachestoExpensiveOptimisation**, **SurrogatesinGP**, **FitnessApproxinML**, **EvolvingSimpleSymbolicRegressionGP**, **CompetitiveCooperativeCoEvAlgorithms**] 。基因编程与 RL 技术的结合为在动态环境中开发自适应策略和战略开辟了新途径 [**RLSutton1998**] 。基因编程可以被用来直接进化策略，将其表示为程序。我们的框架可以解释为一种 RL 方法，其中选择的策略（SoT）在环境（模拟或真实机器人）中执行，并通过标量成本来评估。然而，与其利用梯度或价值函数学习，我们使用基因编程作为策略优化器。这使得在可解释的任务层级中进行策略搜索成为可能。在我们的研究中，我们使用该方法作为一种零样本模拟到现实学习方法，因为机器人在模拟中学习，无需在真实环境中进行微调。传统上，遗传编程将程序表示为基于树的结构（如图 2 和图 3 所示），反映了许多生物语言的层次性质。此外，遗传编程能够进化出时间上扩展的动作，这增强了代理学习长期策略的能力，这在许多现实世界的应用中是一个关键因素。在实际应用中，遗传编程已经在广泛的领域中展示了其多功能性和有效性。在机器人领域，例如，遗传编程被用来进化控制程序，使机器人能够在动态环境中表现出适应性和韧性的行为。

遗传编程中的一个基本机制是应用遗传算子，特别是在遗传选择之后的交叉和变异，通过在种群中产生变异来推动进化过程。遗传选择允许选择出在过程中表现优异的算法，而表现最差的则被淘汰。这个过程将基于与个体相关的适应度测量。交叉是一种重组算子，通过在两个父程序之间交换遗传物质来创建后代。在基于树的遗传编程中，交叉通常通过在每个父代中随机选择一个子树并交换这些子树以产生新的个体来实现。这个机制使得在解决方案之间交换功能构建块成为可能，促进了新颖程序结构的发现。数学上，给定两个父程序 P_1 和 P_2 ，交叉操作（如图 2 显示）可以表示为：

$$O = crossover(P_1, P_2) \tag{6}$$

其中，产生的后代 O 从两个父代中继承特征，保留有用特性，同时可能引入多样性。

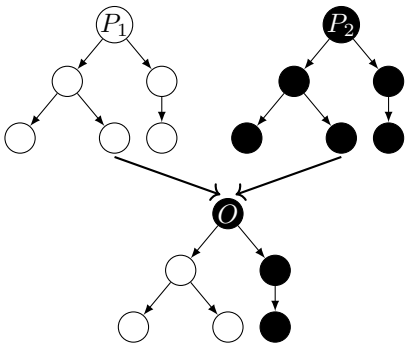

Figure 2. 树结构交叉的例子。从两个父代 P_1 和 P_2 ，通过混合它们的特征生成一个后代 O 。

另一方面，变异是一个操作符，它通过引入小的、随机的变化来增强程序的多样性，并探索搜索空间中的新区域，同时防止过早收敛到局部最优。在基于树的遗传编程中，变异通常涉及选择一个随机节点，并用新生成的节点或子树替换它。正式地，变异（在 Fig. 3 中示意化）可以定义为：

$$O = mutation(P_1) \tag{7}$$

其中 O 是 P_1 的一个变体，其结构经过修改，以便在连续的世代中促进适应和精炼。

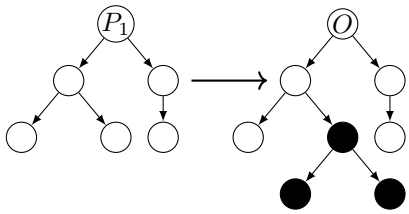

Figure 3. 具有树结构的变异示例。从一个父代 P_1 生成一个子代 O ，其特征发生改变。

IV. 实验装置

在这一部分中，我们介绍学习阶段的设置——使用模拟环境——以及测试阶段的设置——在一个可控的实验室环境中使用真实机器人进行的。

测试阶段选定的机器人模型是移动式 YuMi 研究平台，可在瑞典 Västerås 的 ABB 公司研究部门获得。其架构基于双臂 ABB YuMi，是一个具有每个臂 7 个自由度的双手操作器。操作器安装在一个移动底座的顶部，增加了导航功能。底座本身被视为 3 个额外的自由度元素（沿 x 和 y 轴的平移以及围绕 z 轴的旋转），可以添加到运动链中或用作单个实体来移动平台。只有移动底座配备了距离传感器（一对 LiDAR 扫描仪，一个在移动底座的前部，另一个在后部）。因此，仅在底座参与的情况下才能处理避障任务，因为机械臂没有配备距离传感器。
学习阶段通过在 Gazebo 环境中进行的多次模拟进行，而现实世界的测试则在 ABB 企业研究中心进行。模拟的和真实的机器人都可以通过对手臂施加关节速度和对底座施加三个速度来进行控制 — 沿 x 和 y 的线速度以及围绕 z 的角速度 — 这些速度通过机器人软件转换为车轮的转向和旋转速度。真实和模拟机器人的接口在 ROS2 Humble 中实现。

A. 拟议任务

在我们的实验评估中，机器人允许执行的任务集在一个专用的任务字典中被明确定义。这个字典作为一个结构化的参考，枚举并分类所有可允许的任务。出于清晰和分析的目的，我们将这些任务分为两大组，每组分别对应于评估框架中的一种不同的交互类型或目标。
一种包括有助于执行高级别任务的任务，另一种包括那些对完成高级别目标无贡献的任务，因此会扰乱执行。不相关的任务被引入到框架中，以证明所提出解决方案的鲁棒性，给定的成本函数。两组任务都由 4 个不同的任务组成，它们都返回关节的一个期望速度矢量 $\dot{\boldsymbol{q}}_d \in \mathbb{R}^n$ 。
有用任务集由以下组成：

- 如图 4 所示，具有轨迹时间 t 的逆运动学（IK）。在这项工作中，逆运动学（IK）任务的期望速度并不是

仅仅由期望姿态和当前姿态之间的姿态误差 $e_{IK} = \boldsymbol{k}_d - \boldsymbol{k}(\boldsymbol{q}(t)) \in \mathbb{R}^6$ 决定。相反，通过在预定义的时间段内从初始姿态到目标姿态选择一系列中间点生成轨迹（作为任务参数设置）。然后在每个时间步计算出期望速度，以驱动系统沿此轨迹向下一个点移动。因此，在每个时间步，末端执行器（或移动基座）尝试达到此轨迹的特定点。任务的输入由时间变化的期望末端执行器姿态 $\boldsymbol{k}_d$（由于通过轨迹计算得到）和期望的笛卡尔空间速度组成。任务的期望速度由以下方程的解给出：

$$\dot{\boldsymbol{q}}_d(t) = \mathrm{J}^{\dagger}(\boldsymbol{q}(t)) \left(\dot{\boldsymbol{k}}_d(t) + \gamma_{CL}(\boldsymbol{k}_d(t) - \boldsymbol{k}(\boldsymbol{q}(t))) \right), \tag{8}$$

其中 $\boldsymbol{k}(t) \in \mathbb{R}^6$ 是描述时间 t 的末端执行器姿态的向量，$\boldsymbol{k}_d \in \mathbb{R}^6$ 是期望目标姿态，$\dot{\boldsymbol{k}}_d$ 是其导数，它们根据图 5 的期望轨迹计算得出。$\mathrm{J} \in \mathbb{R}^{6\times n}$ 是参与任务的运动链的雅可比矩阵。

Figure 4. 移动-YuMi 研究平台的逆运动学序列。

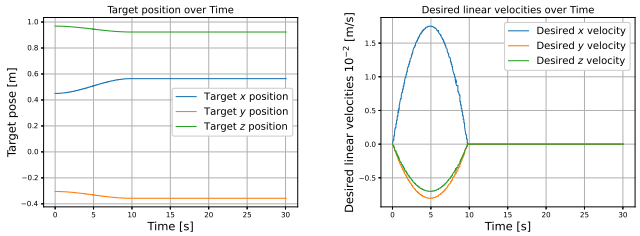

Figure 5. 计算所得所需逆运动学位置的示例图作为轨迹（左）和所需的线速度（右），轨迹时间为 10 s 。

- 利用图 6 描述的轨迹进行障碍回避。类似于 [**LowLevelFlexiblePlanning**]，传感器被视作具有静止长度 r_l 和相关伪能量的 l 个虚拟弹簧。一旦检测到障碍物，当检测到的距离 d 小于弹簧的静止长度时，该任务通过推开移动基座来实现零伪能量的目标。如果一个传感器被激活，即当 $d \leq r_k$ 时，与其相关的伪能量为 $\epsilon_l = \frac{1}{2}(r_l - d)^2$，否则为 $\epsilon_l = 0$ 。与 IK 任务类似，此情况下也会有一个轨迹被跟随。由于机器人使用的是 LiDAR 扫描仪而不是单光束测距仪，因此将值数组进行抽样以获得一个 180° 地图（即使扫描仪的范围是 270°），通过地图以移动基座的零位置 (x, y) 为中心。每个角度的感测值被视为单一的独特测距传感器。角度值被限制在 $[-90°, 90°]$ 角度范围内，因为在同一方向两个不同传感器的重叠会被机器人解释为两次检测到障碍物，从而导致不稳定的行为。在需要 0° 和 180° 的情况下，两者传感器都应激活时，前面的传感器优先于后面的传感器，后者被关闭。这不会影响任务的有效性，因为检测到的障碍物是相同的，但保证了解决方案的稳定性。使用该解决方案，机器人可以感知围绕基座的所有障碍物，将原始 270° 扫描仪的范围缩减为 180° 的范围。对于此任务，期望速度由下式给出：

$$\dot{\boldsymbol{q}}_d(t) = \mathrm{J}_o^{\dagger}(\boldsymbol{q}(t)) \left(\dot{\boldsymbol{\sigma}}_d(t) + \gamma_{CL}(\boldsymbol{\sigma}_d(t) - \boldsymbol{\sigma}(\boldsymbol{q}(t))) \right), \tag{9}$$

，其中 $\boldsymbol{\sigma}(\boldsymbol{q}(t)) \in \mathbb{R}^l$ 是当前状态的全局伪能量，由与所有 m 传感器相关的伪能量 $\epsilon_l(\boldsymbol{q}(t))$ 的总和给出，即 $\boldsymbol{\sigma}(\boldsymbol{q}(t)) = \sum_{l=1}^{m} \epsilon_l(\boldsymbol{q}(t))$ 。$J_o(\boldsymbol{q}(t))$ 是与此任务相关的 Jacobian 矩阵，通过伪能量导数 [**LowLevelFlexiblePlanning**] 计算得出。
与雅可比矩阵 J 不同，由于机械臂在任何时候都有笛卡尔空间和操作速度之间的映射，因此始终非空，而该任务的雅可比矩阵 $\mathrm{J}_o \in \mathbb{R}^{l\times n}$ 可能为空。如果没有障碍物比 d 更接近机器人，该矩阵不会阻止后续任务的完全执行。由于传感器在任务执行期间始终处于激活状态，并且被看作是它们描述的半圆的每一度的测距传感器，所以在此特定情况下 $l = 181 + 179 = 360$ 。因此，传感器描述了围绕机器人的完整圆周。
该任务的输入由传感器的有效样本、目标伪能量及其导数给出。与 IK 的情形一样，目标值和期望速度是时间相关的，因为它们是作为连续点的轨迹计算得出的。

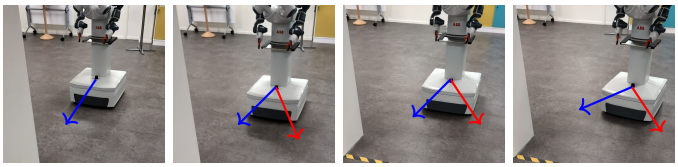

Figure 6. 移动-YuMi 研究平台的障碍规避顺序。机器人移动到障碍物后面的一个位置，寻找能使其远离障碍物的路径。蓝色箭头显示基础在仅执行 IK 任务时将采取的行为，红色箭头显示应用的整体控制，包括障碍规避任务。请注意，如果没有障碍规避任务，机器人将会撞到障碍物。

- 最大化操作性度量。通过这个任务，机器人可以通过远离奇异点来最大化手臂的操作性。
期望速度的解由以下方程给出：

$$\dot{\boldsymbol{q}}_d(t) = \gamma_{OL} \left(\frac{\partial w(\boldsymbol{q}(t))}{\partial \boldsymbol{q}(t)} \right)^T, \tag{10}$$

其中 $w(\boldsymbol{q}(t))$ 是定义为

$$w(\boldsymbol{q}(t)) = \sqrt{det(\mathrm{J}(\boldsymbol{q}(t))\mathrm{J}^T(\boldsymbol{q}(t)))}, \tag{11}$$

的操作性度量，其中 $\mathrm{J}(\boldsymbol{q}(t))$ 是任务中涉及的运动链雅可比矩阵，作为实际关节配置的函数。操作性的导数可以被分析计算为 [**MaxManip**]：

$$\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}} = w \cdot tr\left((\mathrm{J}\mathrm{J}^T)^{-1} \frac{\partial \mathrm{J}}{\partial \boldsymbol{q}} \mathrm{J}^T \right), \tag{12}$$

其中雅可比矩阵的偏导数被分析计算以加快计算速度。
因此，该任务的输入 $z(t)$ 简单地由一个标量值的操作性度量给出。
- 如图所示，最大化机械关节极限（M.J.L.）与距离的关系 7 。此任务使每个关节尽可能远离其机械极限，以最小和最大角度表达。因此，关节趋于向其平均位置移动。重要的是要注意，如果达到了最小点，保证是全局最小点而不是局部最小点。其他任务使用的优化过程，例如可操作性最大化，依赖于基于梯度的方法。

这些方法逐步调整关节角度，遵循每一步中最能提高可操作性的方向。通过这种方式，可操作性最大化方法不会像在后面的情况中那样探索整个配置空间，而仅考虑邻近配置。因此，如果没有小的变化导致进一步改善，算法便会停止。这些点是局部最小值，即在小范围内的最佳解决方案，但不一定是总体上的最佳可能解决方案（全局最小值）。找到全局最小值需要对整个配置空间进行更彻底的搜索，对于高自由度系统（如冗余机器人手臂）来说，这通常在计算上是不可行的。因此，这些任务的解决方案以及最终与之相关的成本严格依赖于手臂的初始配置。另一方面，在最大化从 M.J.L. 情况的距离时，任务所达到的最大值，即最小成本，不依赖于初始配置，并且是一个全局点。这是因为不同于操作性，其依赖于关节角度和末端执行器运动之间复杂的非线性相互作用，而机械关节限制的距离是由一个凸函数定义的。每个关节到其限制的距离可以独立计算，整体成本函数具有明确定义的平滑结构。这使得优化是凸的，从而更容易求得全局解。所需的速度按照先前任务（方程 10 ）计算，其中

$$w(\boldsymbol{q}(t)) = -\frac{1}{2n}\sum_{i=1}^{n}\left(\frac{q_i(t)-\bar{q}}{q_{i_{max}}-q_{i_{min}}}\right)^2 \tag{13}$$

是距离机械关节限制的度量。$q_{i_{max}}$ 和 $q_{i_{min}}$ 分别是 i^{th} 关节在 n 元素的运动链中的最大和最小机械限制值。在这种情况下，同样，通过利用导数的解析表达式来计算所需的关节速度：

$$\frac{\partial w(\boldsymbol{q})}{\partial \boldsymbol{q}} = -\frac{1}{n}\sum_{i=1}^{n}\left(\frac{q_i-\bar{q}}{q_{i_{max}}-q_{i_{min}}}\right). \tag{14}$$

Figure 7. 最大化移动-YuMi 的机械关节极限距离的序列。

该任务的输入 $\boldsymbol{u}(t)$ 由度量 $w(t)$ 和常数值 $\boldsymbol{q}_{max}$ 和 $\boldsymbol{q}_{min}$ 给出。

解决任务的所有函数返回期望的速度、用于应用零空间投影的雅可比矩阵 [**MultipleTaskPriority**] ，以及计算成本函数或研究执行行为的所有有用参数。除了避障任务外，所有其他任务都是按照 [**RoboticsSiciliano**] 中描述的方式实现的。

第二组的分心任务旨在移动手臂或基座，而不实现有意义的目标。其中两个是特定于基座的，执行圆形轨迹或沿着 z 轴绕初始姿态旋转。另两个控制手臂，描述末端执行器的圆周轨迹或使关节彼此独立地移动，轨迹在一个机械边界和另一个之间振荡。不相关的任务可以返回不同类型的雅可比矩阵，可以选择。有可能得到一个雅可比矩阵，使后续任务不能完全或部分执行，或者在投影阶段导致不可预测行为的随机雅可比矩阵。

B. 拟议成本

实验设置中使用的成本集合由以下组成：

- 精度，由姿态误差的平方范数给出

$$cost = ||\boldsymbol{k}_d - \boldsymbol{k}||^2. \tag{15}$$

可以分为位置精度和方向精度。

- 安全性，可以相对于所有小于静止长度的距离来计算，应用与障碍回避任务中相同的伪能量，r_k ，

$$cost = \sum_{l=1}^{m}\begin{cases}\frac{1}{2}(d-r_l)^2, & \text{if } d_i < r_l\\ 0, & \text{otherwise}\end{cases} \tag{16}$$

，或者相对于最小长度

$$cost = \begin{cases}\frac{1}{2}(d-r_{min})^2, & \text{if } d < r_{min}\\ 0, & \text{otherwise}\end{cases}. \tag{17}$$

。

- 操作性的最大化，可以表示为

$$cost = \frac{1}{w^2_{max.manip}} \tag{18}$$

其中 $w_{max.manip}$ 是操作性度量，当机械臂远离奇异点时，该度量会增加。

- 机械关节极限距离的最大化，由

$$cost = w^2_{M.J.L.} \tag{19}$$

给出，其中 $w_{M.J.L.}$ 是机械关节极限距离的度量，当关节角度达到其平均值时，该度量趋于零。

- 时间，由

$$cost = t^2 \tag{20}$$

给出，其中 t 是从任务开始到结束的模拟时间。使用 t^2 会对更长的持续时间更剧烈地增加成本，这惩罚了较慢的执行并鼓励更快地完成任务，这是在优先考虑效率时优化中的常见方法。

在实验中，我们对成本进行归一化以使其可比较，并按照方程式 III-B 中的方法结合。

C. 图形用户界面 G.U.I.

为了简化使用并扩大可访问性，系统开发了一个专用的图形用户界面（GUI）。该 GUI 旨在提供一个直观的环境，使用户能够与应用程序互动，而不需要深入的技术专业知识或命令行操作。通过将复杂操作抽象成简单的视觉组件，如按钮、下拉菜单和输入字段，GUI 显著简化了工作流程。图 **??** 中的 GUI 允许用户配置模拟的关键参数，而无需直接修改代码。它分为两个主要部分：成本参数和模拟设置。在成本参数部分，用户可以为各种评估标准分配相对权重，例如准确性和安全性。每个标准都有一个相关的数值输入字段，用户可以在其中指定其权重，从而根据其优先级自定义学习目标。单选按钮指示哪些成本标准是活跃的，引导用户关注并选择参数。模拟设置部分使用户能够定义模拟的结构方面。用户可以通过明确标记的输入字段设置初始种群中的堆栈数量、迭代次数和模拟时间（以秒为单位）。界面简约，旨在清晰和功能性，方便实验的配置。我们开发了一个框架，在受控条件下模拟任务，允许我们分析它们的行为和相关性。每个任务与一组学习参数相关联，如估计持续时间、成功概率、资源消耗和上下文限制等，这些参数用于基于可配置的成本函数计算优先

级顺序。通过反复模拟，框架不仅学习到单个任务的特性，还学习到它们如何对整体任务效率的贡献。这种方法可以在不同的现实环境中实现泛化。由于优先级顺序是从任务参数及其在完成给定高级任务中的角色中得出的——而不是从特定环境特征中得出的——因此在相同任务和成本函数适用的任何环境中都有效。这种将任务逻辑与环境细节分离的方式使得该框架具有鲁棒性、灵活性，并适用于各种部署场景。

在模拟中，我们使用了 ABB 企业研究中心实验设施的 Gazebo 模型。环境和机器人模型（图 **??** 左侧）均由 WARA Robotics 提供。机器人控制功能和遗传编程算法是用 Python 开发的。

对于每个不同的成本函数，学习阶段用最大代数初始化，具体取决于学习的复杂性，以及种群中的个体数量。然后，每个个体是一个随机生成的优先任务次序（SoT）。初始种群可以用随机优先级顺序、随机参数或两者来创建，视范围而定。所有参数都被限制在一个基于任务先验知识定义的范围内，以维持模型的稳定性并获得合理的结果。然后，遗传编程算法进化个体，直到达到一个稳定点，即所有存活的 SoT 在成本上差异很小的迭代。在每个学习过程的末尾，只有最佳的 SoT（即最终种群中成本最低的那个）被视为算法的最终结果，并选中进行测试。在执行过程中，如果两个竞争的 SoT 具有相同的关联成本，则随机选择一个存活。以这种方式，我们确保在每一代中种群中个体的数量保持恒定。在第一阶段，任务参数是固定的，只有任务的顺序可以通过遗传操作来改变。通过这种方式，可以学习到架构的优先顺序。然后，优先顺序被固定，参数被允许在由任务先验知识决定的边界约束范围内发生变化。比如，用于定义逆运动学中期望速度 $\dot{\boldsymbol{q}}_d$ 的增益被限制在范围 $\gamma_{CL} = (0, 2]$ [**StabilityClosedLoopIK**, **StabilityAnalysisDiscreteTimeSensorBase**] 内取值，因为在这个范围之外，任务没有稳定性保证。在学习阶段，手臂的初始位置可以随机初始化，但要遵循约束，即保持末端执行器在避免碰撞的区域内开始模拟。此外，施加到手臂的速度会朝向位置限制减小，以避免超出并有损坏机器人的风险。每个 i^{th} 关节的

$$\epsilon(q_{i_{min}} - q_i) \leq \dot{q}_i \leq \epsilon(q_{i_{max}} - q_i) \tag{21}$$

，其中 $q_{i_{min}}$ 是其允许的最小位置，$q_{i_{max}}$ 是其最大位置。为了让机器人能够对环境中不可预测的变化作出反应，开发了静态和动态变化的环境。在第二种情况下，在 Gazebo 环境中引入了一个移动物体，以模拟接近机器人的操作员的存在，以测试算法的安全措施。

最后，非相关任务被引入到可能任务的词典中，以显示算法对与其主要目标无关的多余和分散注意力的任务的适应能力。即使用户没有定义，机器人发生碰撞时也会负担高成本，并且在发生碰撞时始终包括在内。

由于 LiDAR 扫描仪安装在移动底盘的外缘之外且居中位置，因此对测量的距离应用了可变偏移。此修正考虑了扫描仪位置与底盘之间的物理位移，提供了从障碍物到机器人实际车体的准确距离估计。如果此距离低于某一阈值，则机器人被认为因碰撞而损坏，成本相应增加。

D. 实验室测试

每个特定成本函数的堆栈在受控的实验室环境中进行了测试，证明了所提出的框架使我们能够将学习到的解决方案从模拟环境转移到现实世界中。机器人通过 ROS2 接口进行控制，提供所需的手臂关节速度，以及移动底座的线性 $[x, y]$ 和角速度 z 。

测试在以下环境中进行：

- 静态环境中的移动底座平台
- 动态变化环境中的移动基平台
- 一个静态环境中的机械臂
- 移动基底平台和一个机械臂在静态环境中，
- 移动基座平台和一个机械臂在动态变化的环境中。

为创建一个动态变化的环境，允许一个人在机器人执行任务时靠近行走。由于安全保证，用户在现实场景中执行任务时可以随时停止机器人。

V. 实验结果

在本节中，报告了学习过程中及后续实验室测试期间获得的结果，评估了所提出的方法的绩效和有效性。

A. 任务堆栈的学习

学习阶段分为两个部分：第一部分，我们学习任务的优先顺序，第二部分，我们学习参数。

1) 优先顺序学习： 在这个阶段，参数被固定为不最小化用户给定的成本函数的值，但它们保证了稳定性和所有任务的执行。

在只使用移动底座的情况下，问题更简单，因为只有两个不同的任务相关，因为可操纵性最大化和远离机械关节限制是特定于操作器的。在模拟过程中，碰到障碍物时成本很高，因此在几代（取决于初始分布，2 或 3 代）中，机器人独立于全球任务优先考虑障碍物避免任务，而不是逆运动学。另一方面，当堆栈中的任务数量增加时，算法需要更多的迭代才能达到最终的优先级顺序（图 **??** ）。最后，最终学习的顺序如下所示：

- 1^{st} 障碍物规避
- 2^{nd} 逆运动学
- 操控性测度的最大化
- 最大化与 M.J.L. 的距离

。有时，根据具体的成本函数，3^{rd} 和 4^{th} 的位置可能会交换。然而，这种配置有助于避免奇异性，使其成为一般情况的合适选项。此外，这一顺序可以作为下一个阶段的初始基准，允许任务在学习参数的同时切换它们的位置并适应任何成本函数。这样，由于顺序已经几乎学会，参数的学习将更快。

2) 参数学习： 一旦初始优先级顺序确定好，就可以让参数在预先定义的范围内假定任何值。这样，算法可以利用尽可能多的组合来寻找成本函数的最小值。在此阶段，算法可能需要不同的代数来达到一个最小点，具体取决于初始的 SoTs 集合。在此基础上，流程结束时得出的解决方案对于相同任务可能会有所不同。所有仿真在达到稳定行为时停止。在这种情况下，不可能像优先级顺序那样找到一个可以适应所有成本函数的通用解决方案。一个任务的参数在很大程度上依赖于用户在期望性能方面定义的成本函数。为了展示所采用方案的工作原理，我们提出以下两个示例。

首先，决定一个关注某些用户需求的成本函数 C_1 ： 然后，从导出的初始序列开始，创建了一个随机种群。之后，算法运行，并在第 7 代找到一个平稳点。这个成本函数鼓励机器人在尽量避免障碍物的同时，努力达到所需姿态并保持较高的可操控性。在计算结束时，SoT 的顺序保持与之

前找到的相同。所有与适应性测量相关的任务仍然处于激活状态，而最后一个任务（即最大化与 M.J.L. 的距离）被标记为不活跃，因为它对于执行高级任务并不重要。机器人的目标是在环境中到达一个特定姿态，同时避免动态和静态障碍物。在这种情况下，学习是在动态环境中进行的。在图 **??** 中，选择了由优化算法挑选出的最佳 SoT。可以观察到，避障任务处于第一位置，因为在发生碰撞的情况下，机器人所付出的代价要高于未达到期望位姿的代价。注意，图 **??** 中报告的堆栈顺序和参数与用户所描述的成本函数一致。避障是优先级最高的任务，因为机器人希望避免碰撞并远离障碍物。由于环境是动态的，避开障碍物所需的时间相当短，并且增益足够高以确保快速响应。接下来是逆运动学的优先级。在这种情况下，增益足够高以确保收敛而不产生不稳定，同时轨迹时间并不重要。操控性任务的最大化排在逆运动学任务之后，允许在不影响期望位姿的情况下最大化测量值。在这种情况下，增益允许机器人在保持稳定性的同时达到目标。最大化与机械关节限制的距离排在最后，甚至处于非激活状态，因为它在最小化成本函数中并不相关。为了进行比较，可以定义第二个成本函数：在这种情况下，机器人不需要满足与障碍物的距离，而只需精确且快速地实现这一结果。从学习阶段中获得的结果顺序任务，在经过 6 代后如图 **??** 所示。在这种情况下，顺序与之前相同。尽管用户没有严格地指定，但机器人仍然以最高优先级避免障碍物以防止碰撞。实际上，在发生碰撞时，机器人被认为是损坏的，并且所分配的代价很高。在逆运动学任务的情况下，增益略高，轨迹时间更短，因此末端执行器更快地收敛到期望的姿态。需要注意的是，这两个最大化任务在成本函数中的相关性不大，因此一个任务不活动，而另一个任务的增益较低，因此在高层次任务的解决中其参与度相当弱。

3) 对字典任务中存在的非相关任务的稳健性： 在这个阶段，字典任务中最多有 4 个不相关任务被添加到可能的 SoT 选项中，以随机位置出现，以展示算法抵抗噪声的能力。这些任务如果具有足够高的优先级顺序以阻止其他有用任务，将会阻止机器人实现最终目标。因此，算法的目标是让这些任务失效或将它们移到 SoT 的最后几个位置。我们可以说，结果总能达到，即不相关的任务总是被拒绝，但算法在学习阶段执行这项任务的性能有所不同。实现这一目标的生代数很大程度上取决于初始随机配置或交叉与变异操作的随机性质。如果群体中有很高比例的个体将不相关任务排列在前几个位置，则收敛速度会较慢。例如，报告了一个 10 自由度运动链的案例，其中不相关任务从第一个优先级位置开始。

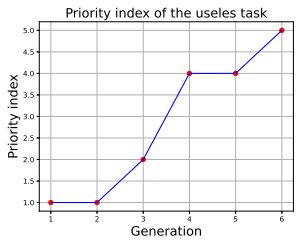

Figure 8. 非相关任务的最低优先级顺序索引。

如图 8 所示，任务被拒绝并移至最后一个位置。请注意，5^{th} 是最后可能的索引。获得的曲线并没有表现出特定的行为，它只是一个非递减函数，因为所有改进都取决于算法的随机选择。

B. 学习任务堆栈的实验室测试

所有在学习阶段获得的 SoT 都在实验室环境中经过适当的安全措施在机器人上进行了测试。为此，移动基座的速度被限制在线性速度为 $[-0.2, 0.2]\ m/s$ 的范围内和角速度为 $[-0.2, 0.2]\ rad/s$ 的范围内。此外，关节速度被限制在 $[-1, 1]\ rad/s$ 的范围内。这种选择显然会阻止机器人在指定时间太低的情况下满足特定的轨迹。因此，在测试阶段的所有轨迹时间都进行了固定偏移量的缩放。这种选择不会影响我们所提出的框架的有效性。为了进行实验验证，首先测试了一个具有固定障碍物的静态环境。然后，一个人被允许在机器人附近行走（图 10），测试在动态环境中的性能，并模拟操作员和机器人之间的协作情况。

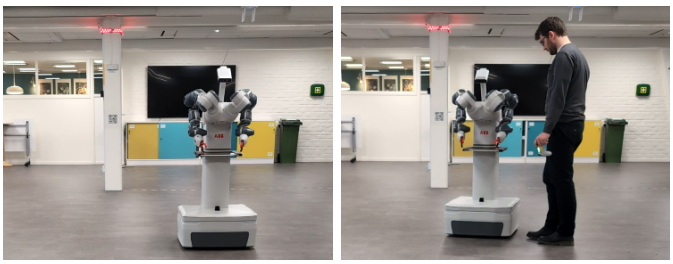

Figure 10. 移动式 YuMi 研究平台在环境中导航时远离人类操作员。左侧显示它在实验室中达到期望的姿态，然后它移开，右图显示了它避开与用户碰撞的场景，在模拟中它学会了优先考虑安全，即使这意味着可能错过目标。

正如预期的那样（因为移动平台和双臂 YuMi 机器人的 Gazebo 仿真足够准确），学习的 SoT 在实际实验室环境中也能工作。

不过，由于仿真中的一些不准确性，结果略有不同但仍然一致。所产生的 SoT 还在一些不同的条件下进行了测试，例如不同的初始配置、不同的目标或不同的障碍。结果总是满足要求，达到了成本函数方面的最佳性能。

我们的实验室实验结果证实，完全在模拟环境中学习的任务堆栈可以成功应用于真实世界环境中。尽管环境细节存在差异，机器人能够在自动导出的任务优先级指导下有效执行相同的高级任务。这个结果验证了我们方法的核心前提：基于任务参数和成本函数驱动的模拟学习过程，能够产生具有鲁棒性和可推广性的控制策略。通过将任务逻辑与环境特定约束分离，该框架在各种部署情境中实现了自适应行为，显著减少了工程工作量——一旦设计了成本函数，SoT 即可以自动学习——并提高了机器人系统的可拓展性和可重用性。

然而，在真实机器测试中出现了一些限制。首先，由于传感器噪声和测量延迟，机器人对环境中动态障碍物的反应不如模拟中快速。另一个在模拟中出现的问题是缺乏用于障碍物规避任务的全局规划器。没有它，机器人无法智能地绕过障碍物，而是尽可能被推离这些障碍物——通常会不必要地远离其预定目标。在图 9 中，展示了一些在测试过程中获得的结果，其中两个成本在前面的章节中已定义。一个重要的点是关于与现有方法的比较。据我们所知，与之前的 SoT 方法进行直接的定量比较是不可行的，因为现有的工作依赖于针对特定场景手动定义和硬编码的任务层级。然而，我们方法自主学习的任务优先级在质量上与那些手动定义并在类似机器人任务的经典框架中采用的优先级相一致，例如在 [**LowLevelFlexiblePlanning**] 中。这种与专家设计策略的一致性支持了我们所学习的配置的实用有效性，尽管它们是完全自动化导出的。

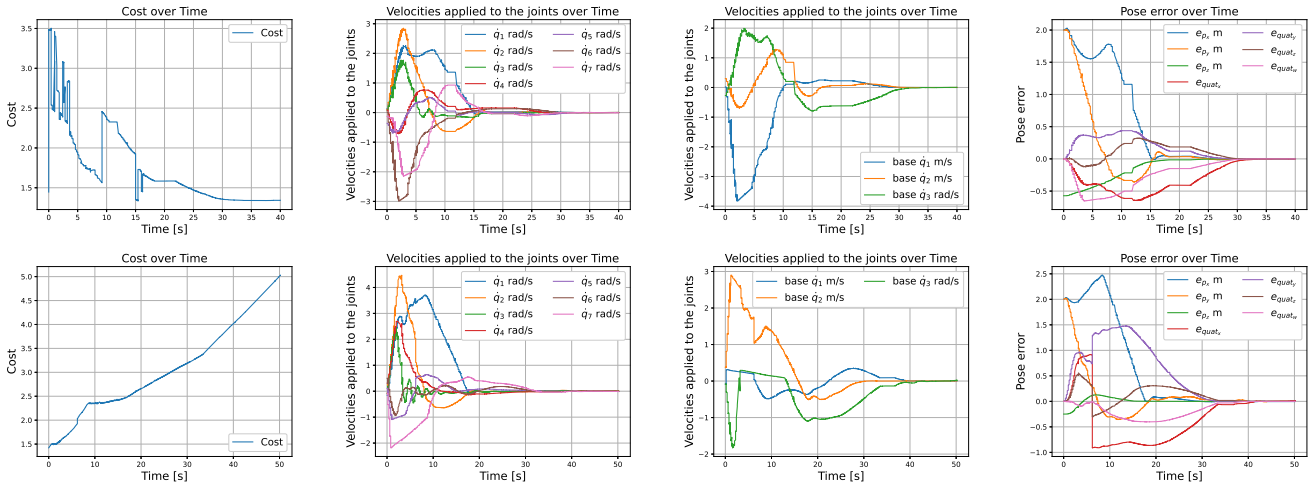

Figure 9. 在这个图中，报告了在移动-YuMi 研究平台上进行的实验室测试中获得的数据，分别为上面的 C_1 和下面的 C_2 。从左到右，我们报告了成本函数 C ，应用于手臂关节和底座的速度 $\dot{q}$ ，以及逆运动学的姿态误差。请注意，在 C_1 的情况下，由于 LiDAR 测得的最小距离包含在成本函数中，因此适应度测量有时会呈阶梯状。当物体进入或离开传感器范围时，测量值会更新。此外，在这种情况下，成本在执行过程中达到最低成本。对于 C_2 而言，环境中物体的距离不包含在内，但由于任务是活跃的，机器人在任何情况下都不会发生碰撞。在这种情况下，由于时间的平方项，适应度测量值总是增加。对于这种情况，如果目标达到一定阈值，考虑任务完成并停止计时是至关重要的。事实上，在 $t = 10$ 秒之后，可以看到时间成本函数的二次行为，因为精度成本几乎为零。

VI. 结论与未来工作

在这项工作中，据我们所知，我们首次提出了一种自动学习完整任务堆栈（SoTs）的方法——包括任务优先级、控制参数和激活逻辑——直接源于用户直观偏好定义的成本函数。我们的方法结合了遗传编程和强化学习，以进化出可解释且适应性强的冗余机器人系统任务执行层次结构。

我们证明了遗传编程是一个有效的框架，用于优化任务顺序和控制增益，以响应高级目标函数。该集成使机器人能够动态管理多项任务，优先考虑诸如避障等安全关键行为，同时保持整体的精确性和性能。

实验验证——在模拟环境（Gazebo）和真实双臂移动机器人（mobile-YuMi）上进行——确认了所学习的任务序列在不同和不可预测的条件下能够泛化，而无需人工调节。重要的是，我们的框架支持零样本的从模拟到真实的迁移：完全在模拟环境中学习的任务序列成功地在真实系统上部署，无需微调。这种环境特定的建模与控制策略的分离标志着实现可扩展和鲁棒的机器人自治的关键进步。尽管有这些令人鼓舞的结果，仍然存在一些挑战。模拟环境虽然对安全和快速学习至关重要，但受制于执行速度并需要人工设置，限制了完全自动化。弥合模拟和现实之间的差距仍然是一个开放的问题，差异的来源包括传感器噪声、通信延迟以及缺乏全局避障规划器。此外，虽然成本函数可以有效指导简单场景中的任务学习，复杂任务互动和参数依赖关系需要深入探索。未来工作将集中在：(i) 设计更丰富的成本模型，(ii) 集成全局规划层 [**Nav2Tool**] 以实现更为鲁棒的导航，(iii) 扩展框架用于多机器人协调，以及 (iv) 基于传感反馈调查控制参数的自适应实时调整。

我们也计划探索更复杂的进化策略和替代的遗传算子来加速收敛。最终，我们预计遗传编程和 RL 的进步将显著增强冗余机器人系统在复杂现实环境中任务执行架构的通用性、鲁棒性和可扩展性。