Integrated Task and Motion Planning for Real-World Cooking Tasks

Jeremy Siburian*[1], Cristian C. Beltran-Hernandez*[2], and Masashi Hamaya[2]

***Abstract*— To fully realize robots in household environments, robots would need to be able to plan and execute a variety of tasks autonomously. However, task and motion planning for multi-step manipulation tasks is still an open challenge in robotics, particularly for long-horizon tasks in *dynamic* environments. This work proposes an integrated task and motion planning (TAMP) robotic framework for real-world cooking tasks using a dual-arm robotic system. Our framework integrates PDDLStream, an existing TAMP framework, with the MoveIt Task Constructor, a multi-stage manipulation planner, to enhance multi-step motion planning for interdependent tasks. We augment our framework with various cooking-related skills, such as object fixturing, force-based tip detection, and slicing using Reinforcement Learning (RL). As a motivating case study, we tackle the long-horizon task of preparing a simple cucumber salad, consisting of slicing and serving it on a plate. We demonstrate our framework both in simulation and real robot demonstration.**

I. INTRODUCTION

In order to realize general-purpose robots that can work collaboratively with humans, robots need to be able to understand instructions via natural language, then autonomously plan actions and execute motions based on these instructions. Consider a long-horizon task of preparing a meal or conducting a chemistry experiment, which may consist of multiple subtasks, such as cutting, stirring, or pouring [1], [2]. In such cases, the robot would need to simultaneously and continuously plan for a high-level sequence of actions and the corresponding low-level motions that must be executed to achieve a given goal. In addition, augmenting robots with the ability to understand linguistic instructions is essential for the daily user to intuitively work together with robots.

Our ultimate research goal is to construct a robotic system that can autonomously plan and execute long-horizon tasks based on natural language instructions, particularly for cooking tasks in a dynamic kitchen environment. To this end, we divide our problem into two parts: *natural language translation to symbolic planning* and *task and motion planning*. In a previous study, we addressed the first part by proposing the Vision-Language Interpreter (ViLaIn) [3] framework, which generates a *problem description* (PD) from linguistic instruction and scene observation. For a given planning problem, the generated PDs define the objects of interest, initial state, and desired goal state. A problem description is described using the Planning Domain Definition Language (PDDL) [4], which can then be solved by modern symbolic planners. This work focuses on the second part of the problem, solving task and motion planning.

*Equal contribution.

[1]Jeremy Siburian is with Waseda University, Tokyo 169-8555, Japan. Work done during internship at OMRON SINIC X Corporation. jeremydasa@akane.waseda.jp

[2]Cristian C. Beltran-Hernandez and Masashi Hamaya are with the OMRON SINIC X Corporation, Tokyo 113-0033, Japan. [cristian.beltran, masashi.hamaya]@sinicx.com

This work is supported by JST Moonshot R&D Grant Number JP-MJMS2236 and JSPS KAKENHI Grant Number 21H04910.

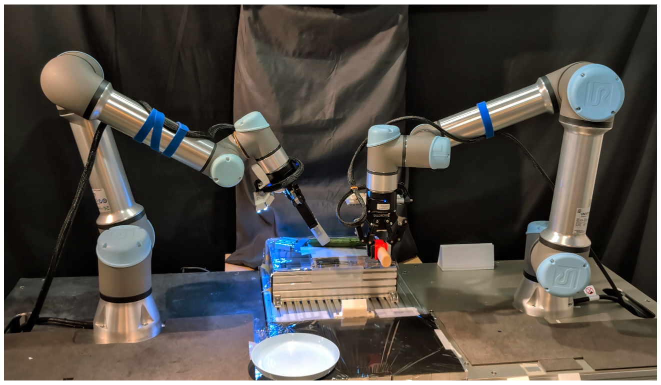


Fig. 1: **An overview of our dual-arm robotic system setup**. The dual-arm robot system places and fixtures the cucumber onto the cutting board, slices the cucumber using a knife, and places the slices on a plate.

Several challenges must be addressed to connect task planning and motion planning for cooking tasks. Combined task and motion planning, often referred to as *Integrated Task and Motion Planning* (TAMP) [5], requires consideration of both *discrete* and *continuous* parameters. For instance, discrete tool states (equipped or unequipped) and the continuous state of the robot joint configuration. Furthermore, as the dual robot often manipulates tools or food items during cooking tasks, collisions among the grasped objects, workspaces, and other robots must be considered for safety. However, most existing motion planners are often limited to pick-and-place pipelines and do not consider interdependence between subtasks, which is crucial for long-horizon manipulation.

In this study, we build upon our previous work by proposing an integrated TAMP framework developed for robotic cooking by leveraging an existing sampling-based TAMP framework and a multi-stage motion planning framework. To summarize, our paper has the following contributions:

- We present an integrated task and motion planning framework for executing real-world cooking tasks. Our framework integrates the MoveIt Task Constructor [6], a multi-stage manipulation planner, into an existing sampling-based TAMP solver, PDDLStream [7], as part of its motion samplers to further enhance multi-step motion planning for interdependent tasks.
- We demonstrate our framework in simulation and real-robot experiments using a dual-arm robotic system as

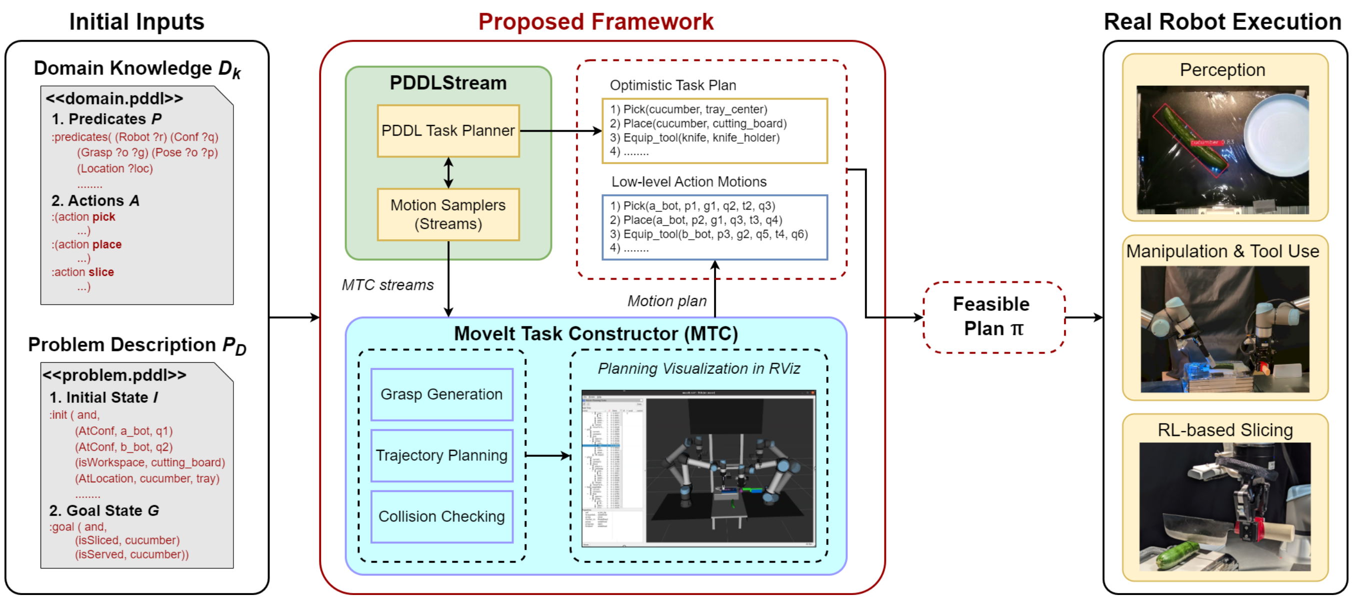


Fig. 2: **An overview of our proposed framework**. The framework integrates the MoveIt Task Constructor motion planning framework into PDDLStream as part of its motion samplers. Our framework receives a domain knowledge D_K and a PDDLStream problem description P_D as initial inputs. A classical PDDL planner then solves the problem optimistically. After an optimistic task structure is found, the planner samples for low-level action motions using the MTC-based streams. After a sequence of collision-free motion plans is found, the framework returns a complete plan π. The found plan is then executed on a real robot with a variety of cooking-related skills.

shown in Fig. 1. Through a case study of slicing a cucumber, we showcase cooking-related skills developed for our strategy, such as force-based tip detection and slicing using Reinforcement Learning (RL).

This paper is organized as follows: a detailed description of our proposed framework is provided in Section II, the real-robot experiment is described in Section III, and a summary is provided in Section IV.

II. FRAMEWORK OVERVIEW

An overview of our proposed framework is shown in Fig. 2. As an initial input, the framework receives a domain knowledge D_K, which describes our cooking domain using predicates $\mathcal{P}$ and actions $\mathcal{A}$, and a problem description P_D, which represents the initial state $\mathcal{I}$ and desired goal state $\mathcal{G}$ of a planning problem. The proposed TAMP framework generates a high-level task plan while simultaneously planning for low-level motion actions. After a feasible plan π is found, the plan is executed on the real robot. As a motivating case study for our framework, we use the task of preparing a simple cucumber salad, which consists of slicing a cucumber and placing the slices on a plate.

A. PDDLStream Formulation

Our proposed system incorporates PDDLStream [7], an open-source sampling-based TAMP framework. PDDLStream extends PDDL [4] by introducing the concept of *streams*, declarative sampling procedures that sample continuous values such as robot joint configurations or candidate grasp poses. As a result, PDDLStream can perform planning over discrete and continuous parameters.

A PDDLStream problem can be represented as a tuple $(\mathcal{P}, \mathcal{A}, \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G})$ which is defined by a set of predicates $\mathcal{P}$, actions $\mathcal{A}$, streams $\mathcal{S}$, initial objects $\mathcal{O}$, an initial state $\mathcal{I}$, and a goal state $\mathcal{G}$. We use the following parameters to represent our PDDLStream problem: `?r` for an available robot, `?q` for a robot configuration, `?o` for the name of an object, `?tool` for the name of a cooking tool, `?p` for an object or tool pose, `?g` for a grasp pose, `?t` for a robot trajectory, and `?loc` for a location in the workspace.

We use *fluent* predicates to model continuous parameters that may change as actions are applied, such as robot configurations, object poses, and gripper status. `(AtConf ?r ?q)` represents an arm `?r` at a configuration `?q`. `(AtPose ?o ?p)` and `(AtLocation ?o ?loc)` represents an object `?o` at a location `?loc` in the workspace with a specific pose `?p`. `(HandEmpty)` and `(Holding)` represent whether or not an arm's gripper is holding something or is empty. If an arm is holding an object, `(AtGrasp ?o ?g)` represents the object `?o` that is attached to the arm using grasp `?g`.

In our cooking domain, we define several types of actions according to our proposed slicing strategy, such as *pick*, *place*, *equip-tool*, *fixture*, *check-extremity*, *slice*, *clean-up*, and *serve-slices*. An *action operator* is defined by a set of free parameters (**`:param`**), a precondition to be satisfied (**`:pre`**) before applying an action, and an effect formula (**`:eff`**) which describes the change of state caused by the action. For example, the following description shows the parameters, preconditions, and effects for the *slice* action.

```
(:action slice
 :param (?r ?q1 ?t ?q2)
 :pre (and (Equipped ?r ?tool) (isHeldDown ?o)
    (isWorkspace ?loc) (AtLocation ?o ?loc)
    (isWhole ?o) (AtConf ?r ?q1)
    (SlicingMotion ?r ?q ?tool ?o ?p ?g ?t ?q2)
 :eff (and (isSliced ?o) (not (isWhole ?o))
    (AtConf ?r ?q2) (not (AtConf ?r ?q1))))
```

We define several streams, including grasp sampling, motion planning, and collision checking. A *stream* is defined by a generator function that receives input values (**:inp**) to generate a potentially infinite sequence of output values (**:out**). For example, the `plan-slicing-motion` stream queries a motion planner to produce trajectories `?t` and an end configuration `?q2` that certify the `SlicingMotion` precondition of the *slice* action.

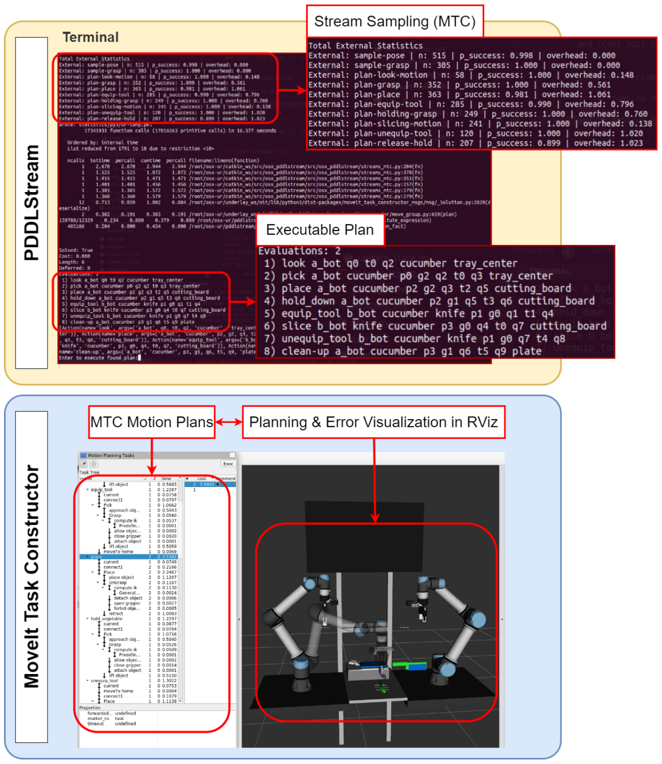
```
(:stream plan-slicing-motion
 :inp (?r ?q ?tool ?o ?p ?g)
 :dom (and (Robot ?r)(Conf ?q) (Pose ?o ?p)
    (EquipGrasp ?tool ?g))
 :out (?t ?q2)
 :cert (and (Conf ?q2) (Traj ?t)
    (SlicingMotion ?r ?q ?tool ?o ?p ?g ?t ?q2)))
```

An advantage of streams is that they allow for *programmatic* implementation of functions in a general programming language, such as Python. In our framework, we implement our streams using a hierarchical multi-stage manipulation planner described in more detail in Section II-B.

PDDLStream provides several algorithms that reduce continuous planning problems to a sequence of finite PDDL problems, which a traditional PDDL task planner can solve. In our implementation, we use the *Adaptive* algorithm, which produces an *optimistic* task sequence using hypothetical stream values before actually calling any stream procedures. In a dynamic environment where uncertainty exists, executing a found plan all at once may not be feasible. The initial object pose provided to the planner may be imprecise compared to the real scene, potentially causing errors when grasping. Certain actions may induce changes to the state of an object. For example, the *slice* action not only changes the state of the cucumber (whole or sliced) but also adds to the number of objects present (e.g., the number of slices produced). These changes can not be anticipated by the planner ahead of time, making one-time planning difficult to realize. Inspired by the approach in [8], we adopt two planning considerations: *optimistic planning* and *replanning from perception*. We first optimistically plan using an *initial belief* of the object pose to produce a full plan. Before any state-changing actions (e.g., pick, slice) can be applied on the object, the robot must first perform perception to *register* the actual state and real pose of the object. The planner then performs replanning to refine the task and motion plan using the updated values from perception. We model predicates `isWhole`, `isSliced`, `Uncertain`, and `Registered` to reflect these changes in the planning domain.

B. Multi-Step Motion Planning

In long-horizon manipulation tasks, certain tasks are often strongly interrelated and cannot be considered independently from each other, particularly for dual-arm manipulation. For example, in the context of our cooking task, certain fixture grasps by an arm may interfere with slicing and cause collisions with another arm. The most widely popular motion planning framework is MoveIt [9], an open-source library designed for motion planning and mobile manipulation, commonly used with the Robotic Operating System (ROS). However, the current MoveIt manipulation pipeline presents



```
Total External Statistics
External: sample-pose | n: 515 | p_success: 0.998 | overhead: 0.000
External: sample-grasp | n: 305 | p_success: 1.000 | overhead: 0.000
External: plan-look-motion | n: 58 | p_success: 1.000 | overhead: 0.148
External: plan-grasp | n: 352 | p_success: 1.000 | overhead: 0.561
External: plan-place | n: 363 | p_success: 0.981 | overhead: 1.061
External: plan-equip-tool | n: 285 | p_success: 0.990 | overhead: 0.796
External: plan-holding-grasp | n: 249 | p_success: 1.000 | overhead: 0.760
External: plan-slicing-motion | n: 241 | p_success: 1.000 | overhead: 0.138
External: plan-unequip-tool | n: 120 | p_success: 1.000 | overhead: 1.020
External: plan-release-hold | n: 207 | p_success: 0.899 | overhead: 1.023
```

```
Evaluations: 2
 1) look a_bot q0 t0 q2 cucumber tray_center
 2) pick a_bot cucumber p0 g2 q2 t0 q3 tray_center
 3) place a_bot cucumber p2 g2 q3 t2 q5 cutting_board
 4) hold_down a_bot cucumber p2 g1 q5 t3 q6 cutting_board
 5) equip_tool b_bot cucumber knife p1 g0 q1 t1 q4
 6) slice b_bot knife cucumber p3 g0 q4 t0 q7 cutting_board
 7) unequip_tool b_bot cucumber knife p1 g0 q7 t4 q8
 8) clean-up a_bot cucumber p3 g1 q6 t5 q9 plate
```

Fig. 3: **PDDLStream and MTC in simulation**. After PDDLStream finds a task plan, the MTC-based streams are called to perform multi-step motion planning, producing motion plans for the previously found task plan. Both successful and failed motion plans are published and can be visualized in RViz before actual execution.

several drawbacks. The framework is mostly restricted to a single-arm pick-and-place pipeline, with limited functions for dual-arm coordination [10]. Another major drawback is that the provided pick and place stages are black-box implementations, reducing transparency and making it difficult to evaluate planning failures. To overcome these shortcomings, Gorner et al. [6] proposed the MoveIt Task Constructor (MTC), an open-source motion planning framework for planning multi-stage manipulation actions with *interdependent* tasks. MTC accounts for the interdependence of sub-tasks bypassing the world state and the result of a sub-solution between planning stages using the MoveIt Planning Scene as a common interface. MTC also provides introspection and failure visualization to allow for further analysis of successful and failed solutions, addressing the previous issue of transparency. However, the MTC framework currently assumes that a high-level sequence of tasks is known and defined in advance.

To address this current gap, we integrate MTC into our PDDLStream implementation as part of its streams for motion sampling. We modified the existing Pick and Place containers provided with the open-source implementation to satisfy our custom actions for a cooking task. A running example of PDDLStream and MTC in simulation is depicted in Fig. 3. First, PDDLStream performs symbolic planning using a PDDL task planner and finds an optimistic task sequence. After a task structure is found, PDDLStream calls upon the MTC-based streams to find low-level action motions for the

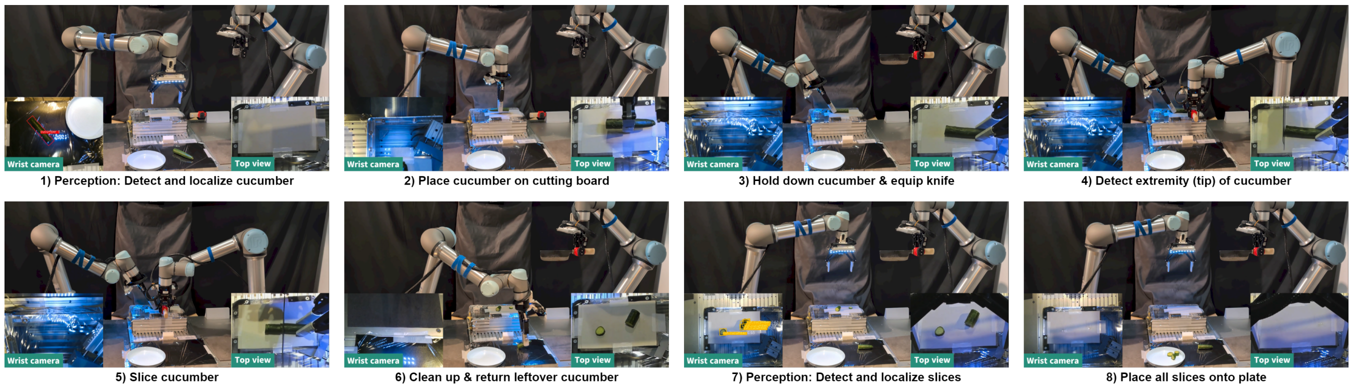

Wrist camera
Top view
1) Perception: Detect and localize cucumber
2) Place cucumber on cutting board
3) Hold down cucumber & equip knife
4) Detect extremity (tip) of cucumber
5) Slice cucumber
6) Clean up & return leftover cucumber
7) Perception: Detect and localize slices
8) Place all slices onto plate

Fig. 4: **Full workflow of the cucumber slicing task**. 1) The robot initially registers the cucumber pose using perception. 2) The robot then picks and places the cucumber onto the cutting board. 3) A robot holds down the cucumber, while another robot equips a knife. 4) The robot detects the extremity of the cucumber using force-based tip detection. 5) Once the extremity is reached, the robot performs RL-based slicing. 6) After slicing is successful, the robot cleans up by returning the leftover cucumber to the tray. 7) Finally, the robot performs perception again to detect the cucumber slices and 8) repeatedly performs pick and place until all slices are on the plate.

previously found task plan. Both successful and failed motion plans are published, which can be visualized and validated in RViz before the actual execution on the real robot.

III. ROBOT EXPERIMENTS

We demonstrate our framework in simulation and a real robot experiment. The following sections describe in detail the experimental setup, our learning-based slicing skill, and the full workflow of our slicing task.

A. *Experiment Setup*

We demonstrate our framework on a real robotic system initially introduced in [11]. The system is comprised of two Universal Robot UR5e robotic arms. Each arm is equipped with a parallel gripper and a force-torque sensor at the arm's end. The arm that is allocated for the slicing task is equipped with a custom finger adapter on its fingertips to facilitate attaching and detaching the kitchen knife.

B. *Slicing using Reinforcement Learning*

In our approach, our slicing method leverages Reinforcement Learning (RL) and compliance control. We utilize a framework that learns compliant control slicing motions using deep RL in a real2sim2real formulation [12]. In a previous study, we show that our method can adapt to unseen objects and require as little as a single slicing motion to acquire the force profile of the object.

C. *Real-World Slicing Strategy*

In the real world, there are numerous strategies for slicing a vegetable. In general, the task consists of placing and holding down the vegetable on the cutting board, locating the extremity or tip of the vegetable, and slicing. Holladay et al. [13] described slicing as a forceful manipulation task and discussed various methods of holding down the objects, which is referred to as *fixturing*. In forceful manipulation, fixturing is essential to prevent the motion of the object that force is being exerted to. In our strategy, we use a dual-arm approach where a robot grasps and fixtures the object onto the cutting board, while another robot performs the slicing operation. Fig. 4 describes the full workflow of our slicing task. Given an initial location of the cucumber at the tray, the robot first performs perception and retrieves the initial pose of the cucumber. After the cucumber pose has been updated, the robot picks and places the cucumber onto the cutting board. The robot fixtures the cucumber onto the cutting board to prevent its motion, while another robot equips a knife to perform slicing. Afterward, the robot locates the extremity of the cucumber by performing force-based tip detection. Once the extremity has been reached, the robot performs the RL-based slicing until it reaches the desired number of slices. After slicing is successful, the robot performs a clean-up operation by returning the leftover cucumber to the tray. As the state of the cucumber is now uncertain, the robot again performs perception to detect the slices and update their respective poses. Finally, the robot performs pick and place repeatedly until all slices are served on the plate. Once the cutting board is empty and all slices are on the plate, the goal is achieved, and the task is deemed successful.

IV. SUMMARY

In this paper, we introduced an integrated task and motion planning framework for executing real-world cooking tasks. The key contribution of our framework is by integrating PDDLStream, a sampling-based TAMP solver, with the MoveIt Task Constructor, a multi-stage manipulation planner, to further enhance multi-step motion planning for interdependent, long-horizon tasks. We demonstrated our framework in simulation and a real-robot experiment through a case study of slicing a cucumber. We incorporated a variety of cooking-related skills into our framework, such as tool use, force-based tip detection, and RL-based slicing. In future work, we plan to fully integrate our TAMP framework with our vision-language interpreter [3] to realize the execution of natural language instructions. We also plan to consider replanning from error or failure feedback to improve the overall robustness of the framework [14].

REFERENCES

[1] N. Saito, J. Moura, T. Ogata, M. Y. Aoyama, S. Murata, S. Sugano, and S. Vijayakumar, "Structured motion generation with predictive learning: Proposing subgoal

for long-horizon manipulation," in *IEEE International Conference on Robotics and Automation*, 2023, pp. 9566–9572.

[2] N. Yoshikawa, A. Z. Li, K. Darvish, Y. Zhao, H. Xu, A. Kuramshin, A. Aspuru-Guzik, A. Garg, and F. Shkurti, "Chemistry lab automation via constrained task and motion planning," *arXiv preprint arXiv:2212.09672*, 2022.

[3] K. Shirai, C. C. Beltran-Hernandez, M. Hamaya, A. Hashimoto, S. Tanaka, K. Kawaharazuka, K. Tanaka, Y. Ushiku, and S. Mori, "Vision-language interpreter for robot task planning," *arXiv preprint arXiv:2311.00967*, 2023.

[4] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[5] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 265–293, 2021.

[6] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "Moveit! task constructor for task-level motion planning," in *IEEE International Conference on Robotics and Automation*, 2019, pp. 190–196.

[7] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.

[8] C. R. Garrett, C. Paxton, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, "Online replanning in belief space for partially observable task and motion problems," in *IEEE International Conference on Robotics and Automation*, 2020, pp. 5678–5684.

[9] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[10] P. M. Fresnillo, S. Vasudevan, W. M. Mohammed, J. L. M. Lastra, and J. A. P. Garcia, "Extending the motion planning framework—moveit with advanced manipulation functions for industrial applications," *Robotics and Computer-Integrated Manufacturing*, vol. 83, p. 102559, 2023.

[11] F. von Drigalski, C. C. Beltran-Hernandez, C. Nakashima, Z. Hu, S. Akizuki, T. Ueshiba, M. Hashimoto, K. Kasaura, Y. Domae, W. Wan, *et al.*, "Team o2ac at the world robot summit 2020: towards jigless, high-precision assembly," *Advanced Robotics*, vol. 36, no. 22, pp. 1213–1227, 2022.

[12] C. C. Beltran-Hernandez, N. Erbetti, and M. Hamaya, "Sliceit! - a dual simulator framework for learning robot food slicing," in *IEEE International Conference on Robotics and Automation*, 2024, p. accepted.

[13] R. Holladay, T. Lozano-Pérez, and A. Rodriguez, "Robust planning for multi-stage forceful manipulation," *The International Journal of Robotics Research*, vol. 43, no. 3, pp. 330–353, 2024.

[14] M. Skreta, Z. Zhou, J. L. Yuan, K. Darvish, A. Aspuru-Guzik, and A. Garg, "Replan: Robotic replanning with perception and language models," *arXiv preprint arXiv:2401.04157*, 2024.