Task and Motion Planning for Robotic Food Preparation
16-782 Project Report

Montiel Abello [mabello]
Travers Rhodes [traversr]
Sudharshan Suresh [sudhars1]
Himanshi Yadav [hyadav]

I. Introduction

Real-world planning requires reasoning about the high-level tasks and robot motion. They should not be considered in isolation due to the complexity of robots and their environments. We need planning techniques that interleave sequencing with motion planning for feasible plans in the configuration space. To that end, we implement a Task and Motion Planning (TAMP) system for robotic food preparation. We demonstrate this in both simulation and real-world, through the preparation of yogurt parfaits. We select this scenario, as there is merit in task sequencing while preparing multiple parfaits with common ingredients.

II. Task and Motion Planning

The task and motion planning problem extends task planning to the infinite domain. To explicitly define our planning problem, we employ the *Planning Domain Definition Language* (PDDL) [1]. This is similar to STRIPS, with additional features[1]. For the symbolic task planning problem, a PDDL description consists of:

Objects: Entities that influence the planning problem
Predicates: Logical properties of objects
States: Set of predicates describing the current world
Actions: Ways of changing states

Actions are defined by a set of *preconditions*—predicates on these variables that must be true for the action to be applied—and *effects*—changes to the state defined by a set of new and removed predicates. Actions become *grounded* when applied to objects, which substitute the action's variables.

A symbolic task plan consists of finding a sequence of actions that transforms an initial state s_0 to a goal state s_g. Each action is assigned a cost, and the cost of the plan is the sum of the costs of each action. In the *combined* TAMP problem, there is an infinite set of actions A, and a state cannot be defined only with a set of predicates on objects. In addition to predicates that describe the symbolic state of the world, objects also have physical characteristics that are defined in continuous domains. There is an infinite domain of physical motions that can define valid actions, rather than a discrete set of motions. Any physically valid action is also defined by a set of preconditions and effects, just as in the task planning problem. However, the objects these actions are defined on can have continuous properties, such as position in space, rather than just binary predicates.

[1]like type specifications for objects, negated preconditions, numeric variables, durative actions and conditional add/delete effects

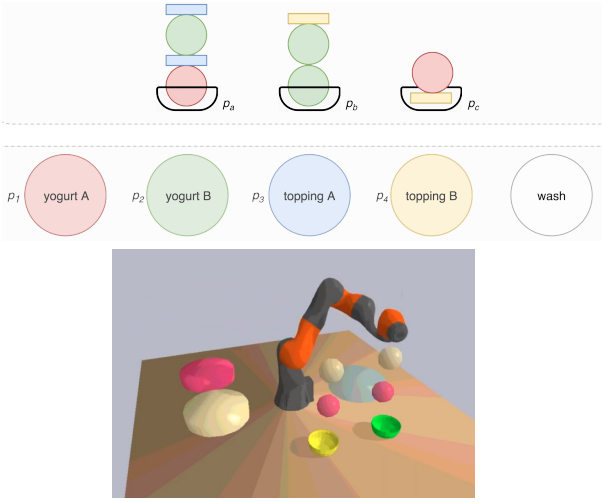

Fig. 1: **(top)** Schematic description of the parfait preparation problem, where the goal is defined by multiple bowls of different ingredient orderings. **(bottom)** Simulation of a 6 DoF KUKA arm performing TAMP towards the preparation of two parfaits.

Objects are limited to a set of bowls containing ingredients at known physical locations. We define our TAMP problem as finding a sequence of valid trajectories that transforms the initial bowl configuration to a goal configuration where bowls contain an ordered set of desired ingredients. We assume a constant angular velocity for joint angles in the arm, and thus define the cost of a trajectory as it's arc length in space. For a plan π defined as a sequence of actions with trajectories $\langle T_0, \ldots, T_i, \ldots, T_k \rangle$, the cost of the plan is: $(\pi) = \sum_{i=0}^{k} \text{arclength}(T_i)$.

III. Planning Algorithm

A. Streams

A naive approach to planning in the continuous domain would be to discretize the physical space and denote each valid path with a discrete action and employ standard task planning. This is computationally intractible due to the sheer number of possible actions. To choose valid actions from the continuous space, we use the concept of a *stream* [2]. We use the PDDLStream python library https://github.com/caelan/pddlstream for our implementation.

Candidate actions are generated from *streams* of possible actions. These candidate actions are evaluated

for validity in terms of physical configuration, and their effect on the state. Streams are polled until a feasible action is found, which is then taken in by the task planner for use in symbolic planning.

B. Symbolic Planning

An example of the PDDL problem file that involves two parfaits with three ingredients is in the Appendix Listing 1. The goal state is two parfaits with the ordering of ingredients as {vanilla, nuts, strawberry} and {vanilla, vanilla, nuts}. Appendix Listing 2 shows one of our actions—washing the scooper. For symbolic planning, our TAMP implementation uses the *FastDownward* planning algorithm [3], an efficient method based on heuristic search.

C. Motion Planning

Given an initial and final arm configuration pair that is determined to perform a valid action, the motion planner must find a sequence of joint angles to perform the required transformation. The sampling-based planning algorithm RRT-Connect [4] is used. It is efficient and probabilistically complete.

IV. Simulation Results

We demonstrate our results in two simulation cases, using a *KUKA* 6DoF robot arm. The results are visualized in PyBullet, a Python module for physics simulation (Fig. 1). The robot arm and models for all objects in the world take the form of URDF files. To physically illustrate the process of parfait preparation, we consider the scoops as *graspable* objects in the world (circumventing the complexities of modeling scooping). The two cases we present are **(i)** our classroom demo: preparation of one double scoop parfait, **(ii)** preparation of two double scoop parfaits.

We are able to successfully generate task and motion plans for both cases, where planning takes just a few seconds. We encountered problems in our simulations occasionally, when the grasp sampling takes arbitrary amounts of time. Demonstration **(ii)** shows the merit in task planning as we are able to add scoops of similar flavors prior to washing the scooper.

V. Physical Demonstration

We perform a physical demonstration on the Niryo One arm for the task of creating two yogurt parfaits (Fig. 2). Though we recognize the superiority of combined TAMP when the ordering of tasks depends on results of motion planning, we found it was satisfactory to perform a hierarchical structure for the physical demo. With the task planning results from simulation, we perform motion planning separately for our Niryo One arm. This was done since we were unable to successfully find kinematic solutions for the Niryo One arm in simulation. Taking just task-level results from the simulation results gave us the task-level plan for the ordering of motions necessary to generate the desired parfaits.

Fig. 2: Our Niryo One arm performing the task plan from simulation, using a local gradient planner.

Given the task plan, to perform the motion planning for the Niryo One arm, we want to have smooth motions for the robot so that the held food is not dropped during motion. We are also fortunate that the majority of the workspace is obstacle-free. Thus, a local gradient-based planner (constrained to keep the spoon level) is sufficient for path planning between the ingredients, the wash basin, and bowls in which to place ingredients. We have the local planner follow lines drawn in cylindrical coordinates to avoid paths that collide with the base of the robot. Therefore, if the robot tries to move the spoon from one side of itself to the other side, it won't try to have the spoon pass through the base.

For manipulation tasks, such as picking up yogurt or nuts, we use a simplified learning from demonstration procedure where we record a human trajectory kinesthetically and convert the recording to pose information of the spoon over time. We are then able to translate that pose information to new locations, so we can use the same recording of *"scoop yogurt"* to scoop from any bowl of yogurt in the workspace. A successful video demonstration was presented in class.

VI. Project Deliverables

With this report, we submit the PDDL domain and problem files, along with the python run files. These are integrated with the PDDLStream library and cannot be run standalone. We presented three demonstration videos—(i) The Niryo One physical demo[2], (ii) simulation TAMP problem I[3], (iii) simulation TAMP problem II[4].

VII. Conclusion

In this work, we employ a TAMP towards preparing yogurt parfait. We formally define our TAMP problem, utilizing prior work in this field. The planning algorithm is successfully evaluated in simulation on a KUKA arm and physically demonstrated on the Niryo One arm. In the future, we aim to include time constraints and on-the-fly food orders into the proposed framework.

[2] https://www.youtube.com/watch?v=VBhGjcgPVqA
[3] https://www.youtube.com/watch?v=mJocimdTLtA
[4] https://www.youtube.com/watch?v=BADt_yy_Lvw

References

[1] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," 1998.

[2] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Strips planning in infinite domains," *arXiv preprint arXiv:1701.00287*, 2017.

[3] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[4] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, pp. 995–1001, IEEE, 2000.

Appendix

Listing 1: Two parfait problem definition. It defines the objects initial state and the goal state. The goal state comprises of two derived predicates that represent the ordering of the parfaits.

```
(define (problem pb2)
   (:domain parfaits)
   (:objects vanilla strawberry nuts cup1
             cup2)
   (:init (is-cup cup1)
(is-cup cup2) (clear cup1)
     (clear cup2) (is-vanilla vanilla)
     (is-straw strawberry) (is-nuts nuts))
   (:goal ( and
    (Order cup1 vanilla nuts strawberry)
    (Order cup2 vanilla vanilla nuts))))
```

Listing 2: The wash action—preconditions and effects.

```
(:action wash
:parameters  ()
:precondition (or (DirtyVanilla)
    (DirtyNuts) (DirtyStraw))
:effect (and (not(DirtyVanilla))
    (not (DirtyNuts)) (not(DirtyStraw))))
```

Output task sequence for single parfait with ordering of strawberry, vanilla:

1) move free
2) scoop straw
3) move holding
4) dump first
5) move free
6) wash
7) move free
8) scoop vanilla
9) move holding
10) dump second
11) move free

Output task sequence for 2 parfaits with ordering of strawberry, vanilla and strawberry, vanilla:

1) move free
2) scoop straw
3) move holding
4) dump first1
5) move free
6) scoop straw
7) move holding
8) dump first2
9) move free
10) wash
11) move free
12) scoop vanilla
13) move holding
14) dump second2
15) move free
16) scoop vanilla
17) move holding
18) dump second1
19) move free