Table Cleaning through Task and Motion Planning with Force Control

William Shen
MIT CSAIL
willshen@mit.edu

***Abstract*—We present a framework for solving table cleaning tasks which requires long-horizon planning and reasoning, representing a desirable skill towards building general household robots. Table cleaning requires (1) pick and place to clear clutter off the table, (2) wiping small objects (e.g. crumbs, diced vegetables) off the table with a sponge, and (3) the non-prehensile manipulation (e.g. dragging) of objects which are too large for the gripper to directly pick up. We model this problem using *search-then-sample*-based Task and Motion Planning (TAMP), which requires high-level symbolic task planning followed by low-level continuous motion planning. We investigate different sampling-based motion planning algorithms and force controllers to implement our low-level primitive behaviors. Our experiments in simulation demonstrate that, despite the limitations in our implementation, our TAMP-based system is able to solve long-horizon manipulation tasks.**

***Index Terms*—robotics, force control, task and motion planning**

I. Introduction

Robots are making an ever-increasing presence in both the home and work setting, allowing us to save time and become more productive by offloading menial yet important tasks to robots. Despite the prevalence of robots that are able to perform specialized tasks including vacuuming, pool cleaning, lawn mowing, or even home monitoring [1], we are far from building single robot systems capable of achieving a variety of tasks in unstructured environments. This may be attributed to several factors including the difficulty of developing algorithms which are able to reason about long-horizon behaviors and generalize to unseen environments.

Nevertheless, it is not impossible to imagine a (distant) future where it is common for a household to have a general robot capable of performing chores including cleaning, cooking, laundry, etc. In an effort towards making a small step to achieving that vision, this paper presents a system which allows a robot to clean a table. This could be a dining table with tableware, food containers and crumbs which need to be cleared and stored in their desired locations.

We consider a simplification of table cleaning, an example of which is depicted in Figure 1. Cleaning a table requires the ability to (1) pick and place to clear clutter off the table, (2) wipe small objects (e.g. crumbs, diced vegetables) off the table with a sponge, and (3) drag (i.e., non-prehensile manipulation) objects which are too large for the gripper to directly pick up. Subsequently, this paper will explore motion planning through rapidly-exploring random trees (RRTs) [2] for pick and place, and hybrid force-position control along with cartesian impedance control [3] for wiping and dragging.

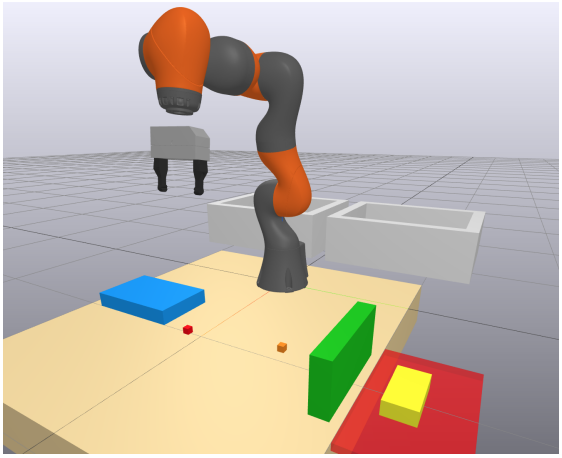

Fig. 1. A scenario where the goal is to place the green and blue box into the right bin, wipe the red and orange cube into the red dustpan using the yellow sponge, and place the sponge into the left bin (in any goal-satisfying order). Observe that the green box blocks the red cube from being wiped into the dustpan, and the blue box lies flat on the table and cannot be directly grasped.

We model our table cleaning problem using a simplified version of the "search-then-sample" Task and Motion Planning (TAMP) approach in [4]. At a high level, our framework:

1) Given a domain and problem definition described in PDDL [5], uses an off-the-shelf AI planner to **search** for a high-level symbolic plan.
2) Uses continuous motion planning with **sampling** to find a corresponding low-level plan that achieves the goal.

If low-level motion planning is unsuccessful, then we add additional constraints, search for a new task plan, and repeat the aforementioned process.

In the remainder of this paper, we firstly discuss related work in Section II and define the table cleaning problem in Section III. Section IV discusses our framework and the techniques we explored. Finally, Section V presents the results of our experiments which demonstrate that we can successfully solve table cleaning tasks which require reasoning.

We refer to our project website for additional details including our presentation and video demonstrations[1].

6.843 Manipulation Project – 2021

[1]https://williamshen-nz.github.io/manipulation-project

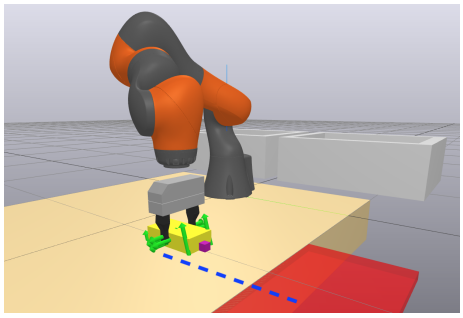

Fig. 2. The purple cube is too small to be grasped by the robot. Hence, we use a sponge to wipe it to the dustpan. The green arrows depict the contact forces and show that the robot is exerting a force into the table. The dotted blue line shows setpoints we could use to wipe the object into the dustpan for cartesian impedance control.

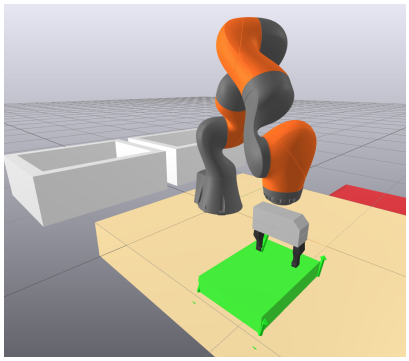

Fig. 3. The green box cannot be directly grasped by the robot. Hence we need to drag it to the side of the table so it may be grasped.

II. Related Work

We refer the reader to [6] for a review of classical and learning-based control approaches to solving a variety of cleaning tasks, such as sweeping, vacuuming and wiping. Although there are several existing approaches to general household cleaning including table cleaning and specific sub-problems (e.g. 'dirt' modelling), they do not model the problem as a TAMP problem and hence may not be able to generalize as well across different numbers and configurations of objects, or struggle with long-horizon problems.

Elliot and Cakmak [7] propose moving dirt particles in one region on a table to another region using heuristic search on a 2D grid-based representation of the table. Another work considered using Convolutional Neural Networks to detect litter on a table, and then used depth first search and Probabilistic Road Maps to generate a motion plan [8]. Yang et al. [9] use learning from demonstration to train a rhythmic Dynamic Movement Primitive to imitate a variety of primitive tasks including wiping and stirring. In contrast to the work discussed above, we focus on the composition of multiple 'skills' which are not learned, and use a TAMP-based approach. Additionally, to limit the scope of our project, we assume that perception is solved.

Task and Motion Planning (TAMP) solves the problem of planning for a robot operating in diverse environments with a potentially large number of objects and actions – this is usually achieved through a mix or interleaving of high-level task planning and low-level motion planning [10]. TAMP allows us to solve long-horizon problems that may require substantial reasoning, which is apt for our table cleaning task. For example, in the scenario depicted in Figure 1, our planner must reason that the green box must be moved out of the way before anything can be wiped into the red dustpan.

PDDLStream is a state-of-the-art TAMP framework which samples for continuous parameters and then reduces the TAMP problem into solving a sequence of PDDL problems [11]. We choose to focus on the approach presented in [4], in which task planning and motion planning are connected through a planner-independent interface layer. We discuss this in more detail in Section IV.

A recent work considered extending PDDLStream [11] for forceful manipulation tasks, such as opening a childproof medicine bottle [12]. Although our work uses a different flavor of TAMP, it is in a similar vein in that our problem requires behaviors that are based on forceful operations.

III. Table Cleaning

We consider the simplified task of cleaning a table with an arbitrary number of small, medium and large objects on top of it. To achieve the goal, the robot must clear all these objects off the table and place them into bins. The initial state of an example scenario is depicted in Figure 1.

We assume that the small sized objects on the table may consist of particles (e.g. crumbs or grains) and small shapes (e.g. diced vegetables) which are too small for the robot to directly manipulate with a gripper. Hence, we give the robot access to a sponge which can be used to **wipe** the small objects off the table into a dustpan. Note that in order to effectively clean the table, the robot must exert a force into the table, as depicted in Figure 2.

The medium and large sized objects (e.g. cereal box, soup can) need to be cleared off the table and put into bins using **pick** and **place**. However, we also consider situations where there may not exist a grasp for a given object due to its orientation and/or position. Hence we need to **drag** the object (i.e., non-prehensile manipulation) using force control techniques to 're-position' it so we may use standard pick and place – an example is shown in Figure 3.

A. Task Planning Formulation

The Planning Domain Definition Language (PDDL) is used to describe symbolic planning problems. Problems in PDDL are defined in terms of a domain and problem file. The domain file specifies the predicates and actions achievable in the environment, while the problem file specifies the objects and the initial and goal state for a given problem. At a high level, PDDL allows us to model actions in terms of (1) the facts that must hold true in order for an action to be applied – i.e., preconditions, and (2) the facts that result from applying an

action – i.e., effects. We refer the reader to [5] for an in-depth explanation of PDDL.

For our simplification of the table cleaning task, we require four PDDL actions: pick, place, wipe and drag. For example, in order to `wipe` an object `x` from the table into the dustpan, one precondition is that the robot must be holding the sponge (i.e., `holding(sponge)` and `is_sponge(sponge)`), while an effect after wiping is that object `x` is no longer on the table but is on the dustpan (i.e., ¬`in_bin(x, table)` and `in_bin(x, dustpan)`). We link the domain PDDL along with the problem PDDL for the scenario in Figure 1 on our project website[2].

B. Assumptions

We assume that perception is solved, and that we have all the geometric information required to formulate the TAMP problem. This includes information to determine whether an object is too small to be grasped and hence needs to be wiped, or whether an object is to too large and requires non-prehensile manipulation. These properties are expressed in our PDDL problem files and in the simulation environment.

Additionally, an important assumption we make is that we have a sampling function that, for a given object, can sample for stable grasps if one exists. In our implementation, we provided manually defined 'good' grasps for objects due to time limitations and difficulties in extracting the geometry of objects from Drake without cameras.

IV. Our TAMP Framework

We consider a simplification of the search-then-sample framework presented by Srivastava et al. [4], in which high-level task planning is connected to low-level motion planning via a planner-independent interface. This interface performs sampling and manages the symbiosis between the task planner and the motion planner. The general process is depicted in Figure 4.

It is important to note that we compute a goal-satisfying plan offline and then pass the plan to our robot to execute. Hence, we do not do any replanning as we execute our low-level plan. In the remainder of this section, we firstly describe our simplification of Srivastava et al.'s approach, then discuss force control and motion planning techniques, and conclude by presenting the *primitive operators* we define.

A. TAMP Planner Logic (as shown in Figure 4)

We firstly feed the domain and problem PDDL for a TAMP problem to an off-the-shelf AI task planner to get a symbolic plan P with a sequence of actions $a_1, \ldots, a_n$. Note that the task planner operates in a discrete state space and completely ignores geometry – hence the task plan may not be successful at the motion planning level.

We assume that each action a_i can be mapped to a **primitive operator** which contains additional parameters required to perform motion planning and run our force controllers. For example, $Pick(box, table)$ can be mapped to a primitive controller $Pick(box, table, ?grasp\_pose)$.

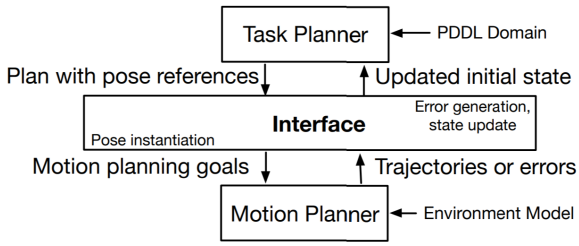

Fig. 4. A high level process diagram of the search-then-sample TAMP approach which we base our framework upon. Diagram from [4].

Now, for the first action a_1 in our task plan, the interface samples for any relevant parameters in its primitive operator. Then, we run motion planning with the sampled parameters to get a motion plan (i.e., a trajectory). For $Pick$, we sample for a $?grasp\_pose$ and then plan for a collision-free trajectory to move the gripper to the desired pose. If motion planning fails, we re-sample for new parameters and attempt to motion plan again.

If this does not succeed after r tries, then we assume that the current action a_i cannot be executed in the low-level continuous state of the environment. An example of a failure for $Pick(box, table)$ would be if another object near the box prevents us from grasping it. Our interface then adds additional constraints to the PDDL domain or problem and/or our task planner to generate a new task plan and repeat the previously mentioned steps.

For example, in the scenario in Figure 1, any plan in which we try to wipe an object before we pick up the green box would fail. Hence we could add an additional precondition to the Wipe PDDL action which states that the green box is no longer on the table. Given time constraints, we did not implement this logic in code but instead used a human to emulate this process by eliminating non-viable task plans as we ran our motion planners.

If motion planning succeeds for a_1, then we repeat the sampling then motion planning process for all the remaining actions $a_2, a_3, \ldots, a_n$ in the high level plan. If low-level planning for the entire task level plan succeeds, then we execute the plan using the sampled parameters in our simulator.

For a better understanding of this overall approach, we refer the reader to the original paper [4]. It is important to note we made several assumptions in our final implementation which restricted the generality of our approach, such as pre-defining good grasps and fixing end positions for wiping. This was to ensure our project was feasible in the limited time frame – we detail these assumptions in Section IV-D and V.

B. Force Control

We use force control to refer to any control technique which commands forces – e.g., direct force control, stiffness control, impedance control, etc. Table cleaning requires force control

[2]https://williamshen-nz.github.io/manipulation-project

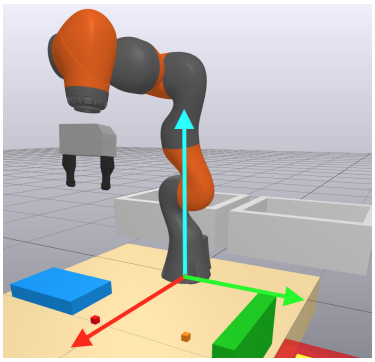

Fig. 5. The axes we use for our table cleaning problem, depicted as colored arrows. The positive x axis is red, y axis is green, and z axis is blue.

for wiping and dragging, as we wish to exert forces into the table or an object.

We consider a robot manipulator whose dynamics can be described by:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) + \tau_{ext} \tag{1}$$

where M is the mass matrix, C is the Coriolis term, $g(q)$ are joint torques due to gravity, and τ_{ext} are the external torques commanded to the robot [13]. By the principle of virtual work, we can write the external joint torques in terms of a desired Cartesian spatial force command F_d, such that:

$$\tau_{ext} = J^T(q) \cdot F_d \tag{2}$$

where $J^T(q)$ is the transpose of the Jacobian for the current configuration q, and $F_d = [\tau_r, \tau_p, \tau_y, f_x, f_y, f_z]^T$. τ_r, τ_p, τ_y are the desired torques for roll, pitch and yaw, respectively; while f_x, f_y, f_z are the desired forces in the x, y and z axis, respectively [13], [14].

Based on Equation 2, we can view both hybrid force-position control and Cartesian impedance control as methods for computing a Cartesian force F_d given desired cartesian position/force inputs.

Hybrid Force-Position Control allows us to command desired positions in certain axes and desired forces in others. The main advantage of this approach is that it allows us to regulate the exact amount of force we wish to exert (e.g. we wish to exert 100N of force into the table).

Our axes of reference are shown in Figure 5. For wiping with a sponge, we wish to exert a force into the table by commanding a negative force in the z axis, while we wish to command positions in the x and y axes in order to move the sponge around the table.

We attempted to implement an open-loop PD control method, where we provided desired x and y positions, a desired force in the z axis, and additionally a desired orientation with the gripper facing down as shown in Figure 5. This control method computes a desired Cartesian force F_d, which we then use to compute the external torque τ_{ext} through Equation 2. For the sake of brevity, we refer to Chapter 7.1.5 in the Manipulation Course Notes [13] for the full derivation of a PD-based hybrid-force position controller. Our implementation was based off the hybrid force-position control notebook in the problem set for the course.

Although we were able to exert forces into the table, we found that it was very difficult and time consuming to (1) tune the proportional and differential gains of the PD controller to smoothly move across the table whilst maintaining contact, and (2) maintain the desired roll, pitch and yaw of the end-effector. For these reasons, we decided to focus on a Cartesian impedance controller.

Cartesian Impedance Control allows our robot to act like a mass-spring-damper system with adjustable stiffness and damping parameters [14], [15]. For wiping and dragging, we wish to be compliant in the z axis to ensure we can exert a force into the table/object, and stiff in the x and y axes to allow horizontal movement. We implemented an open-loop cartesian impedance controller. For simplicity in notation, we assume a pose $X = [\theta_r, \theta_p, \theta_y, p_x, p_y, p_z]^T \in \mathbb{R}^6$ describes the roll-pitch-yaw angles and x, y and z positions, respectively.

Now, given a desired pose of the end-effector ${}^W X^D$ in the world frame W, we use the following control method to compute the cartesian force F_{des}.

$$F_{des} = K_s({}^W X^D - {}^W X^C) + K_d(V_d - V)$$

where $K_s \in \mathbb{R}^{6\times6}$ is the diagonal stiffness matrix, ${}^W X^C$ is the current pose, $K_d \in \mathbb{R}^{6\times6}$ is the diagonal damping matrix, V is the current spatial velocity, and V_d is the desired spatial velocity which we set to 0.

We used quaternion differences to compute errors in the rotations. This allows us to overcome issues with computing angle differences using a roll-pitch-yaw formulation (e.g. 0 and 2π are the same angle but lead to an error of 2π, not 0). We computed errors in the translation with $p_d - p$, where p_d is the desired translation and p is the current translation.

Note that we may specify unique stiffness/damping coefficients for each roll-pitch-yaw angle and x, y, z position. Similar to the approach in [12], we fix damping to be critically damped with $K_d = 2\sqrt{K_s}$ and only adjust the stiffness parameters K_s in our final implementation.

Now, in order to exert a force into the table, we can simply define a desired pose of our end-effector with some z position that is underneath the surface of the table. Then, to move across the table while exerting that force, we can simply vary the x and y position. An example of these setpoints is depicted by the dotted blue line in Figure 2.

We argue that cartesian impedance control is a simpler approach than hybrid force-position control – the latter requires us to carefully tune the coefficients of the PD controller which may take several hours if not days of experimentation. However, cartesian impedance control comes at a disadvantage in that we cannot command a desired value for force, and hence there is no guarantee on the magnitude of the applied force.

C. Motion Planning

We wish to find a collision-free path of configurations between an initial q_{init} and goal configuration q_{goal}.

We consider the popular sampling-based Rapidly-Exploring Random Tree (RRT) algorithm. At a very high level, RRT incrementally constructs a tree by randomly sampling for configurations and growing a tree by biasing the search into the largest Voronoi region. This allows the graph to grow in a uniform manner and leads to impressive performance, considering the randomness of the approach [2]. We based our implementation of the RRT algorithm on the code provided in Exercise 8.2 of the Manipulation Textbook [13]. We do not discuss the core algorithm behind RRT, as it is well documented and studied in several papers [2], [16].

In our initial experiments, we found that vanilla RRT was very slow and the solutions it returned were of poor quality and resulted in the RRT 'dance'. To address these issues, we firstly added goal-biased sampling where, with a probability of x, we would sample the goal configuration. This can be thought of as biasing the RRT search towards the goal – we found that using $x = 0.05$ was helpful and sped up planning significantly.

Vanilla RRT returns the first solution when is reached q_{goal} – hence, there is no guarantee on the quality of this plan. We improve the quality of our RRT plans by maintaining a collection of goal-achieving paths, and run RRT up to a pre-defined maximum number of iterations or time limit [2]. Once we exhaust the maximum iterations or time, we return the best plan in terms of path length. We also found that re-running RRT multiple times allowed us to achieve better quality solutions. By re-initializing the tree, we remove any potential biases induced by existing random samples.

We note that path length is not an amazing metric, given the differences between each step of the path is not fixed and may vary. Despite this, we found that this approach provides better plans and prevents the robot from 'dancing' as much.

Finally, to further speed up planning time, we additionally implemented bidirectional RRT. At a high level, we grow two trees at the same time – one tree is rooted at q_{init} while the other is rooted at q_{goal}. If we are able to extend both trees to connect to a sampled configuration q_{sample}, then we can connect both trees together to get a valid path from q_{init} to q_{goal} [17].

The planning time improved significantly when using bidirectional RRT over RRT, but we surprisingly discovered that the path length was worse. Although we do not have a good explanation for this, we theorize that we are more susceptible to randomness when growing two trees instead of one, and hence increase potential noise in the final plan. Because of this, we used RRT over bidirectional RRT in our final experiments.

One significant disadvantage of RRT to consider is that it is not a complete algorithm, meaning that it will run forever if there is no path from q_{init} to q_{goal}. Thus, if we exhaust our maximum number of attempts for each run of RRT, we conclude that motion planning has failed.

D. Primitive Operators

Now that we are equipped with our Cartesian impedance controller and RRT algorithm for collision-free motion planning, we discuss how to implement the pick, place, wipe and drag primitives.

Pick. To pick an object from a given bin, we firstly need to determine a grasp pose. We sample for a pose using a grasp sampler, and then use Inverse Kinematics (IK) to compute a joint configuration which achieves the desired pose. Next, we use our modified RRT algorithm described in Section IV-C to compute a collision-free path from the robot's current configuration. At the goal configuration, we can then close the gripper to grasp the object.

In practice, our grasp sampler was a pre-defined good 'grasp' that we manually defined. That being said, it is not too difficult to write a grasp sampler, but it is a time consuming engineering problem. We could do so with point clouds and antipodal grasps, a topic we explored in the course.

Place. We follow a similar methodology to that for Pick. We determine a desired pose by manually dividing our bin into 2 areas for placing objects, and open the grippers once we reach the desired configuration.

Wipe. To simplify wiping, we only consider wipes in a straight line directly to the dustpan along the y axis. We do not expect that it would be wildly difficult to extend our approach to movement along both the x and y axes, as it only requires additional interpolation to determine desired setpoints and checking those setpoints for collisions.

We split wiping into (1) moving to a pre-wipe pose, (2) going down to exert a force into the table with the given sponge, (3) wiping the object across the table, and (4) going up to a post-wipe pose. We implement (1) with position control and (2)-(4) with impedance control.

To determine a pre-wipe pose, we compute a fixed offset from the object to be wiped (we offset in the y axis by -0.2m and z axis by 0.2m). We then use IK and RRT to determine a collision-free path to get to the pre-wipe pose.

Next, we compute a trajectory of desired Cartesian setpoints in order to go down and exert a force into the table, move across the table to the dustpan, and then go up. We specify a fixed z position of -0.05m to exert a force into the table (note the end-effector frame is not at the tips of the gripper but between the tips and the last joint of the robot).

To determine whether this trajectory is collision-free, we run a simple grid-based search along the proposed path of setpoints and check if there are any other non-wipeable objects blocking the way - if so, then we consider motion planning as failed. If there are no collisions, we pass the desired setpoints to our Cartesian impedance controller at simulation time.

Drag. The required methodology for drag is similar to wipe in which we have a desired pre-drag pose, specify setpoints to drag the object across the table, and go back up into a post-drag pose. Now, instead of using a sponge we exert forces into the object using the tips of the gripper.

We also need to ensure that we do not drag the object such that its center of mass is no longer on the table because it

would fall off. We compute the end setpoint for dragging by considering the size of the object in the x and y axes.

We manually tuned stiffness parameters for wiping and dragging and decided on a stiffness of 100 for the x and y position, and 10 for the z position as we wish to be compliant in the z-axis. We used a value of 10 for rotational stiffness.

In theory, many of these parameters for wiping/dragging can be sampled but it did not make sense given the limited time we had to complete this project, and the fact that we are able to provide sensible choices in the first place (e.g., for a pre-wipe and pre-drag pose).

Summary. We have now specified how we leverage our cartesian impedance controller and RRT motion planner to define our primitive actions. Although we have made many manual engineering decisions and assumptions, our approach is able to generalize easily across many problem instances that satisfy our assumptions using our TAMP formulation.

V. Experimental Results

A. Setup

We consider a 7-DOF KUKA iiwa robot arm with a Schunk WSG 50 gripper in simulation. We implement our simulator, motion planner and controllers in the Drake robotics framework [18] with Python, and use Pyperplan with A* search with and the h^{max} heuristic for task planning [19]. Since h^{max} is an admissible heuristic, A* will give a least-cost plan [20].

For a given scenario, we specified the problem PDDL files by hand. We model all free objects in our environment as rigid bodies with varying mass and friction coefficients for simplicity in modeling and implementation,

One important thing to note is that we switch between our position controller and force controller while we execute our plan. However, it is not possible to switch between controllers on the real KUKA iiwa without rebooting the robot. Thus, if we wish to transfer our code in simulation to reality, we may consider switching to a Franka Emika Panda which supports switching controllers on-the-fly.

In terms of RRT parameters, we run each iteration for a maximum of 10 seconds or 100 iterations. We re-run RRT 5 times for each primitive in Section IV-D and select the plan with the shortest length. We implement collision checking for RRT by checking if there exists any unexpected contact forces between objects in the environment for a given configuration q. We note that our implementation is still buggy and needs work, and we occasionally still observe collisions.

B. Results

It is not feasible to compare our approach against other techniques, given the significant amount of engineering and research effort required to get another TAMP method such as PDDLStream [11] working, or training a reinforcement learning based approach.

We focus instead on discussing our solution to the scenario in Figure 1, which requires a composition of all the pick, place, wipe and drag primitive skills. We also ran our algorithm on several simpler scenarios as we were building our system – some of these videos are showcased on our project website[3].

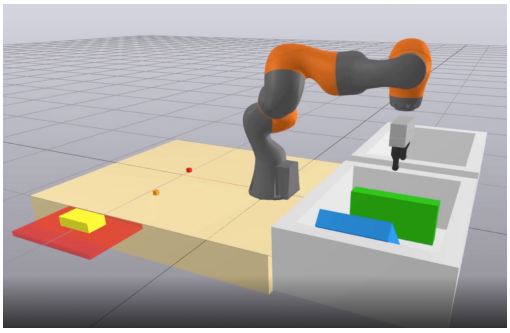

Fig. 6. Execution of the final plan for the scenario in Figure 1 in progress. We depict the state after the robot has executed $Place(box, right\_bin)$ and is now executing $Pick(sponge, dustpan)$

The scenario in Figure 1 requires us to reason about two things: (1) that the flat blue box is blocking the red cube from being wiped and hence needs to be dragged to the side of the table before we wipe the red cube, and (2) the green box must be removed before we can wipe any of the cubes into the dustpan. In the goal state, the green and blue box must be in the bin on the right, the cubes on the dustpan, and the sponge in the left bin. The problem PDDL file is linked on our project website. We note that the optimal plan requires nine actions, which we believe is reasonably long-horizon for a manipulation problem.

The first plan that our task planner returned was to pick up the sponge and then wipe the red cube (a tomato) and orange cube (a carrot), and then place the sponge in the left bin. This plan obviously failed because the green box was blocking the path to the dustpan and hence motion planning for wiping failed. Our human oracle then removed all plans where we wiped before picking and placing the green box, emulating adding a precondition to the wipe action that the green box is not on the table.

We then considered a plan in which the green box was first picked and placed, and then we picked up the sponge to wipe the cubes. Unfortunately, the blue flat box is blocking us from 'going down' from the pre-wipe position. Similarly, our human oracle takes this geometric constraint into account.

Now, we have successfully reasoned about the restrictions in this scenario and require both the book to be dragged out of the way but not necessarily picked and placed, and the green box to be removed from the table in order to wipe both cubes into the dustpan.

The final plan that was executed was:

1) $Drag(book, table)$
2) $Pick(book, table)$
3) $Place(book, right\_bin)$
4) $Pick(box, table)$
5) $Place(box, right\_bin)$
6) $Pick(sponge, dustpan)$

[3]https://williamshen-nz.github.io/manipulation-project

7) $Wipe(carrot, table, dustpan)$
8) $Wipe(tomato, table, dustpan)$
9) $Place(sponge, left\_bin)$

Figure 6 demonstrates our system execution in-progress. As it is difficult to express this plan purely with images given it involves several actions and sub-actions within the low-level primitive operators, we provide the video demonstration here: https://www.youtube.com/watch?v=j5t2DlDHMoU&t=147s.

Although we did use a human to add geometric constraints to our task plans, it should be clear that this could all be automated in code with significant engineering, as Srivastava et al. did [4]. Using the TAMP formulation described in Section IV, even if we did not add constraints if motion planning failed, then we an show that our system will eventually find a valid plan (in which case we would be doing task *then* motion planning, not task *and* motion planning).

C. Limitations

As mentioned previously we made several assumptions to ensure our project was feasible in the limited time frame and as aimed to focus on motion planning and force control and bring them together with TAMP. It is important to understand these limitations as it restricts the current performance of our system.

That being said, these assumptions may be lifted through further research and engineering effort. We relist the most limiting assumptions below:

- We assume perception is solved and that we know the precise pose of each object.
- We use a human who is given a list of task plans to 'filter out' unfeasible plans when we fail to motion plan at the low-level.
- We use manually determined grasps as our grasp sampler.
- We only support straight line wiping in the y axis.

VI. Conclusion and Future Work

We have developed a search-then-sample Task and Motion Planning framework based on [4], which is capable of solving table cleaning tasks which require reasoning and long-horizon planning. We explored motion planning techniques through RRT and bidirectional RRT, and force control through hybrid force-position control and cartesian impedance control for wiping and dragging. Our experimental results demonstrate that we are able to successfully solve table cleaning problems in simulation.

As future work, we may consider building a perception pipeline which measures the poses, keypoints [21], or geometry of objects so we can determine their shape, size and whether an object requires wiping, dragging or pick and place. This would allow us to automatically generate problem PDDL files. Using the geometry of an object, we could also implement an algorithm that samples for stable grasps. This could be achieved through training Convolutional Neural Network for grasping [22] or using geometric techniques such as analytical antipodal grasping [23].

Additionally, we could consider variants of RRT which provide guarantees regarding optimality including RRT* and PRM* [16], and hence reduce the amount of energy a robot expends executing a plan. Finally, we may consider building more robust controllers and integrate the sampling of their parameters, such as stiffness and damping for cartesian impedance control, into our TAMP algorithm. We could also investigate and compare other TAMP techniques including PDDLStream [11].

Disclaimer – Overlap with Research

I am a first year PhD student in the Learning and Intelligent Systems (LIS) lab supervised by Leslie Kaelbling and Tomás Lozano-Pérez. Although I have a background in classical/probabilistic planning, I had zero experience in robotics (TAMP, motion planning, control, kinematics, etc.) prior to taking this class.

This project has allowed me to explore topics that have been covered in the lectures in more depth (motion planning, force control) and also build a simple TAMP system which has assisted in my 'on-boarding' for potential future research in the LIS lab.

Acknowledgments

We thank Professor Russ Tedrake, Rachel Holladay and Dani White for their support in this project and throughout the 6.843 Manipulation course.

References

[1] Amazon, "Meet Astro, a home robot unlike any other," https://www.aboutamazon.com/news/devices/meet-astro-a-home-robot-unlike-any-other, 2021.

[2] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 473–479 vol.1.

[3] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, "Modelling, planning and control," *Advanced Textbooks in Control and Signal Processing. Springer,*, 2009.

[4] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[5] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, "An introduction to the planning domain definition language," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 2, pp. 1–187, 2019.

[6] J. Kim, A. K. Mishra, R. Limosani, M. Scafuro, N. Cauli, J. Santos-Victor, B. Mazzolai, and F. Cavallo, "Control strategies for cleaning robots in domestic applications: A comprehensive review," *International Journal of Advanced Robotic Systems*, vol. 16, no. 4, p. 1729881419857432, 2019.

[7] S. Elliott and M. Cakmak, "Robotic cleaning through dirt rearrangement planning with learned transition models," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1623–1630.

[8] J. Yin, K. G. S. Apuroop, Y. K. Tamilselvam, R. E. Mohan, B. Ramalingam, and A. V. Le, "Table cleaning task by human support robot using deep learning technique," *Sensors*, vol. 20, no. 6, p. 1698, 2020.

[9] J. Yang, J. Zhang, C. Settle, A. Rai, R. Antonova, and J. Bohg, "Learning periodic tasks from human demonstrations," *arXiv preprint arXiv:2109.14078*, 2021.

[10] 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.

[11] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.
[12] R. Holladay, T. Lozano-Pérez, and A. Rodriguez, "Planning for multi-stage forceful manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
[13] R. Tedrake, "Robot Manipulation: Perception, Planning, and Control (Course Notes for MIT 6.881)," http://manipulation.csail.mit.edu/, 2021.
[14] B. Siciliano, O. Khatib, and T. Kröger, *Springer handbook of robotics*. Springer, 2008, vol. 200.
[15] N. Hogan, "Impedance control: An approach to manipulation," in *1984 American Control Conference*, 1984, pp. 304–313.
[16] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
[17] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," 2013.
[18] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu
[19] Y. Alkhazraji, M. Frorath, M. Grützner, M. Helmert, T. Liebetraut, R. Mattmüller, M. Ortlieb, J. Seipp, T. Springenberg, P. Stahl, and J. Wülfing, "Pyperplan," https://doi.org/10.5281/zenodo.3700819, 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3700819
[20] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
[21] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, "kpam: Keypoint affordances for category-level robotic manipulation," *arXiv preprint arXiv:1903.06684*, 2019.
[22] S. Kumra and C. Kanan, "Robotic grasp detection using deep convolutional neural networks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 769–776.
[23] I.-M. Chen and J. W. Burdick, "Finding antipodal point grasps on irregularly shaped objects," *IEEE transactions on Robotics and Automation*, vol. 9, no. 4, pp. 507–512, 1993.