Tool- and Skill-based Robot Manipulation Task and Motion Planner

Himani Sinhmar, Guy Scher, Amy Fang, and David Gundana

I. Introduction and Related Work

Planning in robotics is traditionally divided into two: low-level, that deal with voltages to motors, obstacle avoidance and dynamically constrained trajectories; and high-level, which work with symbolic representations of actions and results. In this paper we focus on the latter in order to compose a series of actions to perform some high-level tasks never previously hard-coded to the robot.

In [1], the authors present a task and motion planner that uses a top-down approach to satisfy high-level tasks. They create a hierarchical tree that lead to the satisfaction of given tasks constructed using pre-conditions and post-conditions of actions. They can then exploit STRIPS and Planning Domain Definition Language (PDDL) to solve the task using automated and always-evolving artificial intelligence (AI) planning algorithms. Authors in [2] create a system to use modular robots to satisfy high level tasks that contain multiple parts described by Linear Temporal Logic (LTL). Using a set of known configurations for modular robots and a library of motion primitives, the authors create a system to automatically generate plans to satisfy the task. This work is extended in [3] to enable the system of modular robots to autonomously use structures found in the environment to satisfy otherwise infeasible tasks. Allowing robots to manipulate objects in the environment in order to use them to satisfy a task expands the set of feasible tasks that robots can satisfy. During execution it is possible for the environment to change, or tasks to not be completed as expected. In these cases, the system must be reactive to the environment in order to ensure the task is satisfied according to the given specification. Authors from [4] provide methods to satisfy task and motion planning with the ability to recreate plans on the fly in a computationally efficient manner for tasks where the human or the environment may be adversarial.

Affordances are defined as the possibility of an action on an object in the environment [5]. In robotics, and manipulation in particular, affordance is a critical factor to allow robots to acquire new possibilities to manipulate novel objects once its affordance information can be singled out. In this work we assume that the affordances of all objects are known. In other words, the different ways an object can be used, actions it can take or the way it extends the robot's skills. There is extensive work in defining the affordances of different objects for manipulation tasks [6], [7], [8]. The work in [9], [10] used the affordance of objects to modify a given task to replace missing or unreachable objects with objects with a similar affordance so that the task can be satisfied.

Our **contribution** is a system approach to Task and Motion Planning (TAMP) that uses the affordances of the objects in the environment to produce physically feasible plans to perform the desired task. Given the affordances of objects and a high-level specification (task, user-defined constraints, etc...), we find the symbolic actions that generate a sequence of motion primitives from the robot's skill set that are used to satisfy the task. Our goal is to find a solution that satisfies the task dependent on object affordances and that is not specific to individual objects and react if the environment changes. To validate our framework, we implement low-level controllers to execute our system on a physical robot and execute several examples.

II. Problem Definition

A. Assumptions

Assumption 1: The library of motion primitives (skill set) is given to the robot a priori
Assumption 2: All affordances and properties of the objects are known
Assumption 3: The robot is able to detect a change in the state of the environment

B. Robot Model

On the symbolic level, we model the robot as having a skill set S. Each skill is a tuple $(a, o, c, e) \in S$. a is an action mapped to a motion primitive in the continuous domain, e.g. the action grip is mapped to a motion produced by the end-effector gripper which closes its fingers. $o \in \mathcal{O}$ is the subset of objects from $\mathcal{O}$ that are participating in this operation, e.g. grip operates on one main object, and move moves the first object to the location of second object. c is the set of pre-conditions that must apply when wanting to perform the action. For example, grip cannot be executed if the gripper is already closed, or if it is not near the object it wants to grip, or if the object cannot be carried (due to weight or size constraints). We add an artificial pre-condition in the form of a proposition $\mathrm{man\_disable}(\mathrm{object1}, \mathrm{object2})$ so that we can manually impose other solutions. This is done to steer away from infeasible plans, and will be explained in detail later. Finally, e is the set of effects of performing this type of action. For the grip action, this means that the gripper is now closed and is grasping the object.

Every skill in the robot skill set is also associated with low-level controllers that are able to perform the action in the physical-space - e.g. move employs a feedback linearization controller in addition to waypoint generation to move the mobile manipulator base and end-effector to a different location (essentially solving the inverse kinematics). In addition

to the actual controls, we can use the skill in a "simulation mode" to check if some proposed action is feasible, i.e. safe.

The robot we use has a differential drive mobile base with state (x, y, θ) and two prismatic joints extending up z in the $\hat{z}$ axis, and in the robot's $\hat{y}$ axis, extending the reach by d. We also assume in this work that the gripper's orientation with respect to the arm is fixed. To account for the change in the robots workspace when it is grasping an object, we extend the robot's configuration space with states t_x, t_y, t_z. These states are the position of the "tooltip" of the object, which represent the new "end effector" position. If the robot is currently not grasping any object, then $[t_x, t_y, t_z] = [0, 0, 0]$.

C. Object Definition

Each object is defined as $O = (\pi_O, P, A) \in \mathcal{O}$, where

- π_O is the name of the object
- $(p, q) \in P$ is a property name p and its value q. It includes both the physical property of the convex hull of the object and its affordance. For example the object ladle has, $P = \{(\text{length}, 25cm), (\text{width}, 3cm), (\text{height}, 3cm), (\text{mass}, 100g)\}$.
- $(a, \text{able}, \text{has}, \text{manipulation}) \in A$ are the affordances of this object, where a is the affordance name, e.g. liquid, able is whether the object can manipulate that affordance and has is whether the object also currently has it in its possession. manipulation is a tuple in itself that describes where is the location and orientation on the body of the object (in object's frame of reference) we need to manipulate in order to obtain this affordance, and the new end-effector (EE) position, again in object's reference frame.

On the symbolic level, we add a property of can_carry to each object that will depend on the mass and size of the object and the gripper constraints. This is set as the initial state of the system.

D. Specification

A specification is defined from both the user-defined objectives and constraints, and the auto-generated problem and domain descriptions. The user can define a goal state, such as "the bowl should have soup" and user-defined constraints such as "do not use a cup (even if the affordance allows it)" by setting the can_carry proposition of the cup to False.

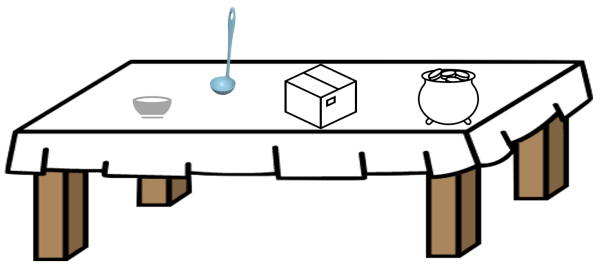

Fig. 1. *Example 1* environment - the goal is for the robot to transfer the soup from the pot to the bowl, while avoiding obstacles

Example 1: Consider the task of transferring liquid from a pot to a bowl. The environment is shown in Fig. 1. The ladle is defined as ("Ladle", P, A), with a property set $P = \{(\text{length}, 25cm), (\text{width}, 3cm), \text{height}, 3cm), (\text{mass}, 100g)\}$. The ladle has two relevant affordances for our example which are both liquid. The difference is in the position of grasp which also effects the end-effector pose. Let obj_pose and obj_R be the location and the rotation matrix along the object, w.r.t the object's origin, where the robot can perform an action. Affordance $A =$

- $(\text{liquid}_1, \text{able} = T, \text{has} = F, (obj\_pose = [-12cm, 0, 0], obj\_R = \text{I}_3, EE = [24cm, 0, 0]))$
- $(\text{liquid}_2, \text{able} = T, \text{has} = F, (obj\_pose = [0, 0, 0], obj\_R = \text{I}_3, EE = [12cm, 0, 0]))$

The bowl is defined as ("Bowl", P, A), with a property set $P = \{(\text{length}, 30cm), (\text{width}, 30cm), (\text{height}, 14cm), (\text{mass}, 300g)\}$ and affordance $A =$

- $(\text{liquid}_1, \text{able} = T, \text{has} = F, (obj\_pose = [0, 0, 0], obj\_R = \text{I}_3, EE = [0, 0, 0]))$

The pot is defined as ("Pot", P, A), with a property set $P = \{(\text{length}, 40cm), (\text{width}, 40cm), (\text{height}, 25cm), (\text{mass}, 1Kg)\}$ and affordance $A =$

- $(\text{liquid}_1, \text{able} = T, \text{has} = T, (obj\_pose = [0, 0, 0], obj\_R = \text{I}_3, EE = [0, 0, 0]))$

In this scenario, we assume that the maximum payload is small enough that the pot cannot be picked up, but ladle can. If the ladle is reachable, then the robot will pick up the ladle, place it into the pot to fill it with liquid, then place the ladel into the bowl.

E. Problem Statement

Given a high-level task (Sec. II-D), a set of object properties and affordances (Sec. II-C) and their state in the environment, and a set of robot skills and their associated post-conditions (Sec. II-B), we automatically seek a plan to generate safe controls to satisfy the specification in a potentially "dynamic" environment, or return that it is infeasible.

III. Methods

A. Overview

The visual understanding of the scene, the reasoning about the affordance of the objects, and the transition between raw sensor data to the symbolic conterparts are the so-called "signal-to-symbol gap" and are not considered in this work. The affordance and the effects of actions are assumed to be known. The set of objects in the environment and their properties are also given. In practice, this could come from many papers dealing with perception of affordances [11], [12]. Fig. 2 depicts the system approach of the high and low-level planners, the inputs and the hardware. We will discuss each of the building blocks in the following sections.

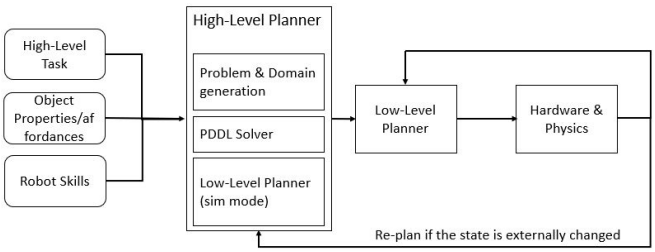

Fig. 2. System view of the affordance Task and Motion Planner

B. High-level planner

The high-level planner pipeline consists of three phases: (1) generating the domain and problem description files based on the task, the objects in the environment and their affordances; (2) running available open-source PDDL solvers to find optimal plans; (3) simulating feasibility for the entire plan and adding constraints to the problem description in case the plan is infeasible and re-iterate. We now describe each section in detail:

1) Problem and Domain description: The description files are automatically generated for PDDL version 2.1 with action costs to enable optimal solution search (in the experiments presented, each action has a value of 1, so the planner finds a plan with the minimum number of actions).

Propositions assumes a value of True or False, based on its arguments. There are 4 hard-coded propositions that deal with the symbolic state of the environment:

- $\mathrm{gripped(obj)}$
- $\mathrm{at(obj, loc)}$
- $\mathrm{can\_carry(obj)}$
- $\mathrm{is\_robot\_carrying}$

which capture "is the robot gripping obj", "is the robot at obj's (and more specifically loc) location", "can the robot carry obj" and "whether the robot is carrying anything at the moment", respectively. Another proposition $\mathrm{man\_disable(obj1, obj2)}$ is added to support disabling of actions and is checked in the precondition of the actions.

The robot's skills are hard-coded as well. Our robot manipulator (discussed in the next section) has 4 skills:

- $\mathrm{gofromto(obj\_from, from\_loc, obj\_to, to\_loc)}$
- $\mathrm{move(obj\_from, from\_loc, obj\_to, to\_loc)}$
- $\mathrm{grip(obj, loc)}$
- $\mathrm{ungrip(obj)}$

The difference between $\mathrm{gofromto}$ and move is to differentiate between the mobile base moving by itself to when it is also changing the location of some object, which is important to keep track when simulating the actions for the feasibility check. An example for the preconditions of the action $\mathrm{grip(obj, loc)}$ are $\neg\mathrm{gripped(obj)} \wedge \neg\mathrm{is\_robot\_carrying} \wedge \mathrm{at(obj, loc)} \wedge \mathrm{can\_carry(obj)}$ while its effects are $\mathrm{gripped(obj)} \wedge \mathrm{is\_robot\_carrying}$ and total plan cost increasing by 1 units.

Auto-generated actions fill the rest by looping through the properties JSON file containing all the objects and extracting the unique set of affordances and locations. Every action (e.g. liquid) will have a do and cease actions which encapsulate the action of taking from an object and transferring the affordance to the self object, and the other way around, respectively (e.g. $\mathrm{do\_liquid}, \mathrm{cease\_liquid}$). The preconditions are that the receiving object is "able" that affordance, and that the sending object "has" it.

The problem description file describes the objects in the environment and the initial state - the pot has liquid, the ladle and the bowl do not have liquid but are able to hold it. Also, $\mathrm{can\_carry(bowl)}$ and $\mathrm{can\_carry(pot)}$ are undefined which sets them as False on default and disables the trivial solution of transporting the pot to the bowl or vice-versa. Here, we also manually disable actions based on infeasible actions we witness using the low-level planner as a feasibility check. Although this may be too restrictive (in the sense that we restrict an action for the entire plan, whereas perhaps given a sequence of other performed actions the disabled action could become feasible again), it at least prevents from a plan that cannot be finish, to start executing. Perhaps a better way to deal with this was to create a graph where the nodes are the environment state, edges are actions and construct the tree breadth-first while searching for an accepting node (goal), but we decided to focus on the other parts of the work.

Finally, the target state is defined by the user, $\mathrm{has\_liquid(bowl)} \wedge \neg\mathrm{is\_robot\_carrying}$ and the optimization objective of minimizing the total cost.

2) PDDL Solver: There are plenty[1] of available PDDL solvers implementing different versions of PDDL standards. We chose to use the singularity container of the submissions to IPC2018[2] because it bundles many algorithms in a convenient interface where it can find the suitable algorithm to try and find the best solutions. This is on the expense of performance, however, our problems are sufficiently small, $\ll$ 1sec wall time. If there are several objects with similar affordances it will choose the one with minimal number of actions. In case of a tie, it will output one plan in random.

3) Feasibility check: As mentioned, we use the exact same low-level algorithm (Sec. III-C) that finds the controls (path, joint parameters) to perform each action, in a simulated fashion where we keep track of the location and state as every action completes. The low-level planner is probabilistically complete due to the usage of sampling trajectories to find feasible paths.

Note: as mentioned earlier, obtaining a valid plan does not guarantee that executing this plan will necessarily produce a feasible plan. Due to the non-determinism of the system, some actions may render the rest of the problem infeasible. For example, executing ungrip action and the ladle drops in a way that cannot be gripped again. Or, if some adversarial moved an item to outside of the robot's reachable set.

4) Reactivity: This system approach to the TAMP is reactive to some extent. First, every object has a symbol and as such, the high- and low-level planners are invariant to the actual location of the objects, and will adjust accordingly

[1] https://planning.wiki/ref/planners/atoz
[2] https://ipc2018-classical.bitbucket.io/

even if they move. A small caveat is that once a trajectory to an object began executing, it is static, so it will not reach a moving object. This can be relaxed in future work. If a change is made to the state of the environment, e.g. someone spilled the soup from the pot so now $\text{has\_liquid}(\text{pot}) = \text{False}$, the system needs to recognize that and call the high-level planner with the new state as initial conditions.

C. Low-level planner

When the high-level planner decides to take an action such as *move* between two places, it still needs to find a safe trajectory considering the robot and object combined configuration and the current workspace state (obstacles). The simpler motions, such as grasp, are implemented using a position controller based on the properties of the main manipulated object (e.g. width).

Because of the way we have defined our robot model (outlined in II-B), the low-level planner implements a "drive then extend" scheme that can be broken down into two main steps: navigation and grasping.

D. Controller for Navigating Robot Base

Algorithm 1: Navigation Controller

Input : $\mathbf{x_{curr}}$, $\mathbf{x_{goal}}$, Env, Obj, EE_pose
Output: w

$F = \text{FINDFREESPACE}(Env, Obj)$
$reachable = \text{False}$
// Check if current pose can reach the target position
$reachable, w_{arm} = \text{GENERATEARMWP}(\mathbf{x_{curr}}, EE\_pose, \mathbf{x_{goal}}, Obj)$
// If current pose cannot reach, sample a new pose for the base
while *not reachable* **do**
  $\mathbf{x_{sample}} = \text{SAMPLEPOSE}(\mathbf{x_{curr}}, F)$
  $reachable, w_{arm} = \text{GENERATEARMWP}(\mathbf{x_{sample}}, EE\_pose, \mathbf{x_{goal}}, Obj)$
$w_{base} = \text{VISIBILITYROADMAP.PLANNING}(\mathbf{x_{curr}}, \mathbf{x_{sample}}, Obj)$
$w = [w_{base}, w_{arm}]$

Given the goal point the robot wants to reach, the robot needs to move its base to a point such that the arm can reach the goal point while avoiding obstacles. Alg. 1 outlines the process for the robot to find this new base point and the corresponding waypoints to reach it.

The free space that the robot can navigate through is an alley around the objects, where Env is the outer circumference of the free space, and Obj is the set of 3D object meshes and their locations. The alley can be tuned such that the robot is not too close to the objects, but close enough that the arm does not need to overextend.

The robot first checks if its current position can reach the goal point. If it can, it will reach from there. Otherwise, it samples a new base point. The sampling is a Gaussian distribution with the mean at its current pose to encourage the robot to find a point closer to its current position and limits "jumping" across the free space.

To determine whether or not the sampled point is valid, we check if the arm can reach the goal point from the sampled point. The details are outlined in Section III-E. If the arm can reach, then we find a series of waypoints for the robot base to navigate to by building a visibility roadmap. The algorithm then outputs waypoints w for the full trajectory, which includes waypoints for both the base and the arm.

E. Controller for Robot Arm

The goal for this controller is to find the required arm height, z_{reach}, and arm extension, d_{reach}, such that the end effector can reach the goal point from the sampled position. We take advantage of the robot's Cartesian motion by first rotating the robot's base such that the pose of the robot is directly in line with the goal point, and then finding the proper extension and lift motion for the robot to perform.

To find a collision-free trajectory for the arm to reach the goal point, we create 3D object meshes for all the objects in the scene as shown in Fig 3. To account for the volume of the arm we bloat the objects by an appropriate factor. We consider three scenarios: first, the gripper is not holding any object and the objective is to reach for a object and grasp it; second, retract the arm after grasping the object; and third, the gripper is holding an object, in which case the objective is to reach the goal point and release the object. For the latter case, we consider the objects tip as the "new end effector".

We sequentially find the required arm height and arm extension by discretizing the arm's motion in z direction by taking steps of 1 cm. We start by setting the arm height at the goal point's height and check for the required arm extension. If a collision is detected of end effector with any of the mesh, we move the arm up by step size and repeats the process until either we find a collision free d_{reach} or the arm can no longer move in z direction. When the gripper is holding an object we check for collision of the "new end effector" with any of the meshes in the scene while extending the arm to reach goal point. For retraction we set z_{reach} as the total arm height and check if the arm can be retracted without any collisions.

For example, say the robot is holding a ladle and the objective is to put the ladle at the bottom of the pot. We start by setting the arm height such that ladle's tip (which is now the new end effector, EE_pose) matches the goal point height (in this case, the bottom of the pot). If the arm tries to reach the pot's bottom by directly extending we would register a collision with the pot's mesh. Thus, we keep on moving the arm by step size until the ladle's tip stops colliding with the pot's mesh, which happens just at the top of the pot. At this point, we store the corresponding z of the arm and extend the arm such that arm is just above the goal point in xy plane. We then lower the arm in step sizes until reached the bottom of the pot. The arm trajectory for this example would consist of two waypoints:

$w_{arm} = [z_{reach_1},\ d_{reach};\quad z_{reach_2},\ d_{reach}]$ where z_{reach_1} and z_{reach_2} corresponds to the pot's top and bottom respectively.

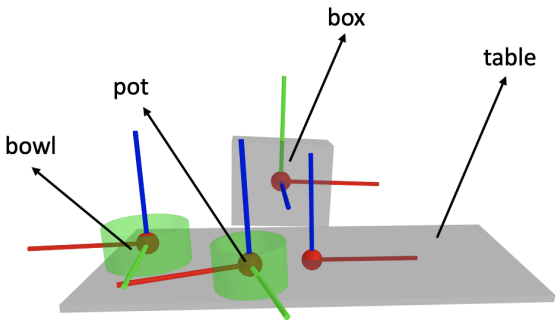

Fig. 3. Real time scene modeled by creating 3D meshes of each object

IV. Experiments

A. Hardware

We implement our planner on on the Stretch robot by Hello-robot [13] (Fig. 4). The low-level planner results in position commands sent to the robot in closed loop in a "drive and extend" manner. This means that the differential base of the robot will navigate and align to a goal position and then attempt to manipulate objects using the three degree of freedom manipulator. For our demonstrations, we assume that the manipulator has two degrees of freedom. The arm can raise or lower in the z-direction and extend outwards in the robots y-direction (perpendicular to the direction of forward motion). The gripper has the ability to rotate about its axis similar to a wrist, but for our demonstrations we have assumed that this degree of freedom is locked.

Fig. 4. Stretch robot by Hello-robot

During our executions we assume that the system has knowledge of the state of the Stretch robot and other objects in the environment. This is achieved using Optitrack, a motion capture system. Each object is tracked in real-time and information on the objects dimensions and weight are given a-priori. The system provides the object's context to the robot to decide the state of the system and the appropriate low-level controls. The Optitrack system is comprised of 22 cameras with a resolution of 1.3MP, a 3D accuracy of $\pm 0.2mm$, and a frame rate of 240 FPS.

Fig. 5. View of several Optitrack cameras in the workspace

Fig 6 shows a possible initial configuration of objects in the physical demonstration. The Optitrack system is able to determine the positions of all objects and using the physical attributes create representations of the objects that are used by the high and low level planners. This real-time information allows for reactivity in our model.

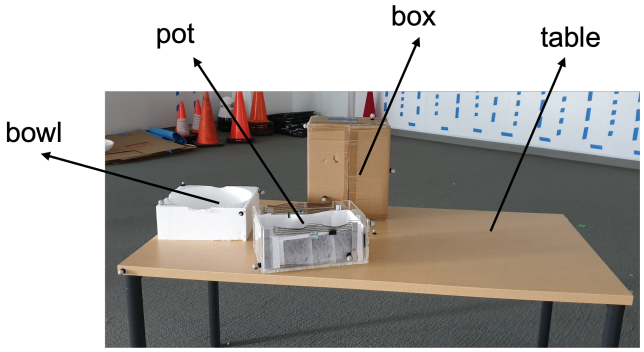

Fig. 6.

B. Experiments

We demonstrate our proposed framework by executing physical experiments of our example in three different scenarios [3]:

1) The ladle is reachable and outside the pot
2) The ladle reachable and inside the pot
3) The ladle is unreachable, but a cup with the same affordance is reachable (can_carry_liquid = *True*)

Demonstration 1: In the first demonstration the ladle is in a position in the environment that is reachable by the robot in the initial state. The high-level planner uses this information

[3]Video available `https://youtu.be/IccXzYfV-ko`

to automatically generate a plan to satisfy the task of moving liquid from the pot to the bowl. The output is as follows:

1) gofromto idle loc1 ladle loc1
2) grip ladle loc1
3) move ladle loc1 pot loc1
4) do_liquid ladle loc1 pot loc1
5) move ladle loc1 bowl loc1
6) cease_liquid ladleloc1 bowl loc1
7) ungrip ladle

Where gofromto, grip, move, and ungrip are motion primitives that the robot should take to complete the task and do_liquid and cease_liquid are affordances of objects. For this task the ladle contains two locations, loc1 and loc2, which represent points that the robot can grip the ladle that result in different affordances of the ladle. During execution the robot follows this plan in sequence until the last final step where, once completed, the task is satisfied. The robot first completes gofromto from the idle state to the ladle in loc1. Once in position the robot will grip the ladle in loc1 and move to the pot in the environment. Using the affordance of the ladle do_liquid, the robot now contains the liquid in the pot. The robot then moves the ladle to the bowl and, using the affordance cease_liquid, the ladle deposits the liquid into the bowl. Finally, the robot ungrips the ladle and the task of moving liquid from the pot to the bowl is complete.

Demonstration 2: In the second demonstration the ladle begins inside of the pot. This means that liquid that needs to be transferred from the pot to the bowl is in the ladle at initialization. The high-level plan is similar to the plan generated for demonstration 1 except after griping the ladle, the robot moves directly to the bowl to deposit the liquid and ungrip the ladle.

Demonstration 3: In the final demonstration the ladle is not available for use in the environment. However, a cup is present and it has the same affordance which allows it to carry liquid. To satisfy the task of transferring liquid, the high-level planner instructs the robot to use the cup instead of the ladle. This shows a strength of our approach in that a plan is able to be found using different objects with similar affordances.

C. Failure Modes

During demonstrations most failures came from the robot being unsuccessful in the "grip" motion primitive. In our "drive and extend" approach, we assume that the robot navigates and aligns to a point within a threshold. We use this threshold to avoid overshoot and chatter caused by hardware and control frequency constraints. Because of this, we are not able to exactly reach the desired 2D point or alignment required to grip the object and a failure occurs. In addition to this, the Stretch robot does not perfectly rotate in place. This means that during the aligning process, the robot translates further then the desired goal point. This problem could be alleviated with an omni-directional or holonomic robot that would allow us to manipulate the position and orientation of the robot directly.

V. Conclusion

We present a TAMP framework that uses the affordances of objects in the environment to produce physically feasible plans to perform the desired task. We automatically find the symbolic actions that generate a sequence of motion primitives from the robot's skill set that are used to satisfy the task. Depending on the task, the outputted plan may include commands for the robot to use objects in the environment based on their affordances, thus expanding the robot's skill set. We create the motion primitives by running low-level controllers and validate our framework by executing three examples on the Stretch robot.

Our planner is reactive to environment changes; if a symbolic action from the high-level planner can no longer be satisfied, the high-level planner generates a new sequence of actions. The low-level planner can also react to environmental changes between each symbolic action. However, it is unable to react if a series of waypoints has already been generated. Because the state of the robot and all of the objects are fully known, it is easy to implement this aspect of reactivity in the future.

In future work, we plan to add capabilities to plan with actions that do not have a deterministic outcome. For example, using a stick to push an object to make it reachable from a different place - there are infinite number of displacement values the robot can push and we need to know when to stop for it to be reachable. In addition, we plan to add temporal constraints, like "the ladle should hold the soup within n-steps (otherwise the pot is taken away)".

VI. Contributions and scores

In the paper and class presentation Guy worked on creating the high-level planner, Himani and Amy worked on the low-level planner, and David worked on the environment abstraction for the experiments and code integration. We believe that all four team members contributed equally and we should each receive a 5 out of 5 for our contributions.

References

[1] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[2] G. Jing, T. Tosun, M. Yim, and H. Kress-Gazit, "An end-to-end system for accomplishing tasks with modular robots.," in *Robotics: Science and systems*, vol. 2, p. 7, 2016.

[3] T. Tosun, J. Daudelin, G. Jing, H. Kress-Gazit, M. Campbell, and M. Yim, "Perception-informed autonomous environment augmentation with modular robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6818–6824, IEEE, 2018.

[4] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12618–12624, IEEE, 2021.

[5] J. J. Gibson, "The theory of affordances. perceiving, acting and knowing," *Eds. Robert Shaw and John Bransford*, 1977.

[6] L. Jamone, E. Ugur, A. Cangelosi, L. Fadiga, A. Bernardino, J. Piater, and J. Santos-Victor, "Affordances in psychology, neuroscience, and robotics: A survey," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 1, pp. 4–25, 2016.

[7] T. E. Horton, A. Chakraborty, and R. S. Amant, "Affordances for robots: a brief survey," *AVANT. Pismo Awangardy Filozoficzno-Naukowej*, vol. 2, pp. 70–84, 2012.

[8] H. Min, R. Luo, J. Zhu, S. Bi, *et al.*, "Affordance research in developmental robotics: A survey," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 8, no. 4, pp. 237–255, 2016.

[9] I. Awaad, G. Kraetzschmar, and J. Hertzberg, "Finding ways to get the job done: An affordance-based approach," in *Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014.

[10] I. Awaad, G. K. Kraetzschmar, and J. Hertzberg, "The role of functional affordances in socializing robots," *International Journal of Social Robotics*, vol. 7, no. 4, pp. 421–438, 2015.

[11] Y. Xie, F. Zhou, and H. Soh, "Embedding symbolic temporal knowledge into deep sequential models," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4267–4273, IEEE, 2021.

[12] S. Reich, M. J. Aein, and F. Wörgötter, "Context dependent action affordances and their execution using an ontology of actions and 3d geometric reasoning.," in *VISIGRAPP (5: VISAPP)*, pp. 218–229, 2018.

[13] C. C. Kemp, A. Edsinger, H. M. Clever, and B. Matulevich, "The design of stretch: A compact, lightweight mobile manipulator for indoor human environments," *CoRR*, vol. abs/2109.10892, 2021.