Synergistic Task and Motion Planning With Reinforcement Learning-Based Non-Prehensile Actions

Gaoyuan Liu, Joris de Winter, Denis Steckelmacher, Roshan Kumar Hota, Ann Nowe, and Bram Vanderborght, *Senior Member, IEEE*

***Abstract*—Robotic manipulation in cluttered environments requires synergistic planning among prehensile and non-prehensile actions. Previous works on sampling-based Task and Motion Planning (TAMP) algorithms, e.g. PDDLStream, provide a fast and generalizable solution for multi-modal manipulation. However, they are likely to fail in cluttered scenarios where no collision-free grasping approaches can be sampled without preliminary manipulations. To extend the ability of sampling-based algorithms, we integrate a vision-based Reinforcement Learning (RL) non-prehensile procedure, *pusher*. The pushing actions generated by *pusher* can eliminate interlocked situations and make the grasping problem solvable. Also, the sampling-based algorithm evaluates the pushing actions by providing rewards in the training process, thus the *pusher* can learn to avoid situations leading to irreversible failures. The proposed hybrid planning method is validated on a cluttered bin-picking problem and implemented in both simulation and real world. Results show that the *pusher* can effectively improve the success ratio of the previous sampling-based algorithm, while the sampling-based algorithm can help the *pusher* learn pushing skills.**

***Index Terms*—Task and motion planning, reinforcement learning, manipulation planning.**

I. Introduction

TASK and Motion Planning (TAMP) problems combine discrete task planning and continuous motion planning. The interplay between the two planning levels gives more comprehensive solutions which consider both logical and geometric constraints. Sampling-based algorithms have excellent generalization abilities when applied to new problem instances and it is proven probabilistically complete on robotic manipulation problems [1].

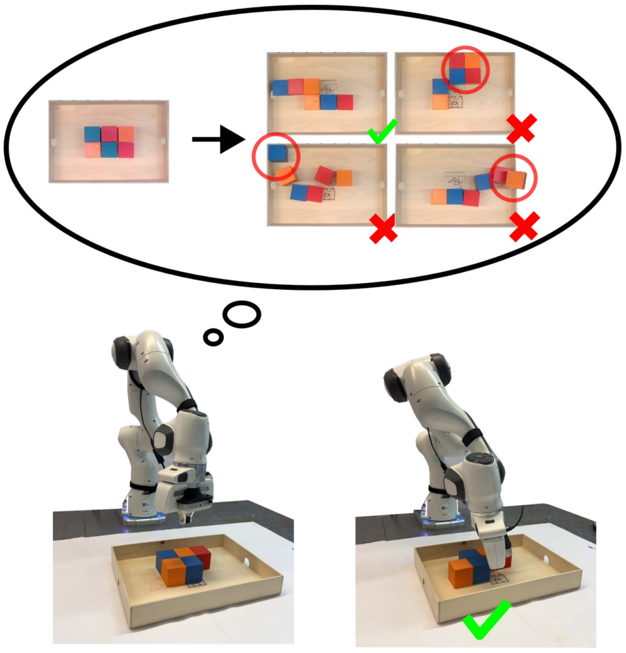

Fig. 1. For a cluttered bin pick-and-place task, the robot needs to push the objects into a situation where they can be grasped with certain task sequence and grasp position, and the TAMP solution can solve the problem by planning a rational task sequence and grasping pose. As for the pushing actions, dead-end situations such as pushing objects to the corner of the bin or colliding with the bin should be avoided.

Consider a cluttered bin-picking problem shown in Fig. 1, the robot has to pick objects from a narrow space in a bin, while the objects can be jammed together. Sampling-based TAMP algorithm, i.e., PDDLStream, can give task sequences and motion plans based on the logical relations among tasks and the geometric constraints such as physical collisions. Previous PDDLStream methods with deterministic action primitives can only provide solutions when there exists at least one collision-free task sequence and motion plan. However, cluttered bin-picking problems are likely to remain unsolvable because of the object proximity to the bin walls or to other objects [2]. Such situations require non-prehensile actions such as pushing to change the position or orientation of objects. However, planning a pushing

Manuscript received 8 February 2023; accepted 15 March 2023. Date of publication 24 March 2023; date of current version 31 March 2023. This letter was recommended for publication by Associate Editor R. Shome and Editor H. Kurniawati upon evaluation of the reviewers' comments. This work was supported in part by the Flemish Government through the Program *Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen*, and in part by China Scholarship Council (CSC) and EU-Project euROBIN under Grant 101070596. *(Corresponding author: Gaoyuan Liu.)*

Gaoyuan Liu and Bram Vanderborght are with the Brubotics, Vrije Universiteit Brussel, 1050 Brussels, Belgium, and also with the imec, B-3001 Leuven, Belgium (e-mail: gaoyuan.liu@vub.be; bram.vanderborght@vub.ac.be).

Joris de Winter and Roshan Kumar Hota are with the Brubotics, Vrije Universiteit Brussel, 1050 Brussels, Belgium, and also with the Flanders Make, 3920 Lommel, Belgium (e-mail: joris.de.winter@vub.be; roshan1182.iitkgp@gmail.com).

Denis Steckelmacher and Ann Nowe are with the Artificial Intelligence (AI) Lab, Vrije Universiteit Brussel, 1050 Brussels, Belgium (e-mail: denis.steckelmacher@vub.be; ann.nowe@vub.ac.be).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3261708, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3261708

action remains challenging for sampling-based solutions because it is difficult to predict outcomes of the pushing actions due to substantial uncertainty in contact mechanics. Moreover, sampling pushing actions need continuous-time forward-simulation, which tends to be time-consuming without a well-designed heuristic guidance [3]. Especially, pushing objects in a bin with bad samplings can cause irreversible failures during manipulation. For instance, the objects can be pushed to corners of the bin and render further manipulations impossible, or the objects can be pushed against the bin wall and be damaged.

Previous work leverages the adaptability of RL to learn such non-prehensile actions. Learning synergistic actions (i.e., learning a combination of picking, placing, and pushing actions) can effectively solve the problem by learning both pushing and picking actions and separating the cluttered objects by pushing them until the goal object is feasible [4], [5]. On one hand, for deterministic actions such as pick-and-place, sampling-based methods are generally considered to be more efficient and reliable than RL-based methods since even grasping with small errors can damage both robot and object, and it's difficult for RL algorithms to fine-tune such an action. On the other hand, pushing actions are more tolerant for small deviations when the purpose of pushing actions is just to make grasping feasible.

Considering the different strengths of both sampling-based and RL-based strategies, we combine the previous work on both sampling-based TAMP and vision-based RL and introduce a hybrid planning method which is comprised of a *RL pusher* and a sampling-based *pick solver*. The sampling-based pick solver can efficiently solve deterministic problems such as pick-and-place when the situation is solvable, while the vision-based RL pusher can plan the non-prehensile actions with stochastic effects. The two modules work interactively: the pusher helps to unlock the current interlock situation that prevents the pick solver from making further moves, while the pick solver evaluates the pusher's actions during training by giving rewards. We train the RL agent in simulation and validate the hybrid planner in both simulation and real world. With our hybrid planner, the pick solver plans task sequences and motion trajectories for pick-and-place, requests pushing actions from the pusher when the objects are jammed together and no further pick-and-place actions can be planned. The main contributions of this work are as follows: (1) We coordinate the abilities of RL and PDDLStream and provide a novel perspective for the cluttered bin-picking problem; (2) We provide a novel reward shaping strategy for robotic RL; (3) We improved the ability of the PDDLStream method in a cluttered environment.

The remainder of the paper is organized as follows. Section II presents an overview of the related work, Section III details the methods and concepts in our framework, Section IV analyzes the learning performance of the presented framework and demonstrate the obtained results and, finally, Section V concludes the paper.

II. Related Work

Sampling-based planning algorithms give a generalizable solution for the robotic planning problems in different domains. The constrained sampling-based TAMP algorithm is introduced to solve multi-modal problems [6]. Such algorithms are proven to be probabilistically complete [1]. The TAMP problem is described and solved with the Planning Domain Definition Language (PDDL) [7] with a sampling preprocess, i.e., *stream* [6]. By doing so, off-the-shelf artificial intelligence planning algorithms, such as FastDownward [8], can be deployed seamlessly. Migimatsu and Bohg [9] extend the ability of the TAMP algorithm to the dynamic environment by sampling in the object-centered frames. By minimizing the task cost function, the strategies of TAMP can also be optimized [10].

Previous work improves the efficiency of PDDLStream with learning strategies. To speed up the searching process, a neural network is trained to predict object importance for a planning tasks [11]. More rigidly, learned constraints are imposed in the planning process to reduce the searching space [12]. To improve the efficiency of the sampling process and leverage past experience, Kim et al. design score-space representation for TAMP problem instances, therefore sampling constraints can be learned from past experience to boost the sampling process in similar problems [13]. Chitnis et al. formulate the local search as a Markov decision process (MDP), and use RL to guide sampling in the motion level [14]. Similarly, a neural network trained on prior planning experience is integrated to score the relevance of streams [15]. Domain uncertainty such as a congested area for a mobile robot can lead to a sub-optimal planning solutions. With RL, TAMP can deal with the domain uncertainties by optimizing the cost value [16]. Curtis et al. extend the ability of PDDLStream to unknown environments which does not require knowledge about the objects [17]. Another way to speed up the TAMP process is to train a neural network with a data-set which contains pre-planned policies. Therefore, the trained neural network can make decisions quickly without planning from scratch [18]. Moreover, a feasibility checking neural network is trained to avoid unfeasible motion planning processes [19]. However, most of the tasks in previous work can be solved by prehensile actions, i.e., pick-and-place. Wang et al. [20] consider pushing as one of the action primitives, but only single object pushing is considered and the consequence uncertainty of the pushing action is neglected. Another learning approach focuses on integrating search-based task planning with learned skill effect model (SEM) [21]. An SEM predicts the terminal state and costs of a skill execution given a start state and skill parameters. However, they do not consider a skill that leads to non-deterministic outcomes.

Recently, vision-based RL has been utilized on different manipulation tasks [22], [23]. End-to-end RL algorithms require an enormous amount of data to show effects. Zeng et al. narrow down the problem to learning the synergistic actions between pushing and grasping [4]. While their work focuses on learning instant actions between grasping and pushing, our work focuses on using pushing actions to support a unpacking task, which presents a long horizon TAMP problem. Also, their method assumes that the cluttered objects are on a flat holding surface, thus no collision or dead-end situations are considered. Following research focused on goal-oriented domains, where manipulating a specific object is set as the goal to be reached [5]. In order to avoid the time-consuming searching process, a pushing prediction network is trained to imitate the Monte-Carlo

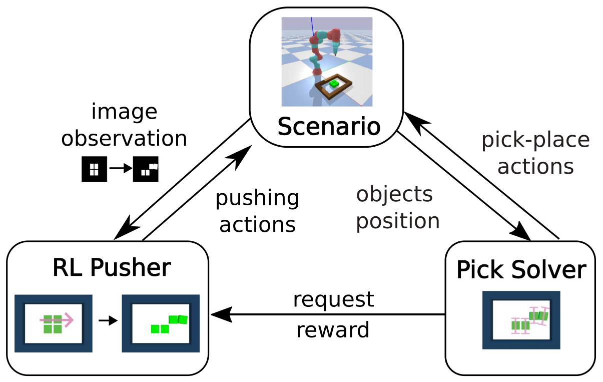

Fig. 2. Hybrid TAMP planning method structure. The scenario is the cluttered bin-picking environment in simulation or in real world. During training, the pick solver sends requests when there is no solvable objects, and evaluates the pusher's behaviour by giving rewards. Therefore, the pusher can effectively learn to create a solvable situation for the pick solver.

search process [24]. This module evaluates pushing actions by predicting the grasping feasibility with a separate network. Instead, PDDLStream can evaluate the grasping feasibility by sampling all the possible grasping poses. A one finger sliding motion is added to isolate objects even when the object is in the corner [25], the cornered objects can be released by a sliding action, thus no dead-end situation is considered in their work. A user-rewarded pushing proposal network is used to singulate all objects [26]. However, with rational picking sequences and grasping poses, the problem can be solved before complete singulation. Danielczuk et al. use pushing motions to increase the bin-picking success ratio [2]. Similarly, suction grasps are used to reach the cornered objects. In our work, we try to solve the similar problem without requiring alternative end-effectors, and also consider collision issues.

III. Methodology

The learned policy provides a promising alternative to a hand-crafted policy when multiple constraints need to be considered. In our case, actions with deterministic effects, such as picking with known object positions, can be solved by the sampling-based pick solver, which avoids the lengthy training process. For actions with stochastic effects such as pushing, sampling-based algorithm requires continuous forward simulation to sample a valid action. Without rational guidance, the sampling process with forward simulation can be time-consuming. Instead, the RL-based data-driven algorithm leverages previous experience and provides valid actions as split-second reactions from observation. The structure of our hybrid planner is shown in Fig. 2, it is composed of a RL pusher and a PDDLStream pick solver, the two parts function interactively.

A. Markov Decision Process

We formulate the problem of pushing augmented TAMP as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, A, \mathcal{R}, \mathcal{T}, \gamma, \mathcal{O})$, where $\mathcal{S}$ is the state space, A is the RL action space, $\mathcal{R}$ is the reward function, $\mathcal{T}$ is an unknown transition function, $\mathcal{O}$ is the observation space and γ is a discount factor. Below we describe the details of the observation space, the action space, and the reward function.

B. Reinforcement Learning Pusher

The pusher is defined as a RL agent which observes the current state of objects and provides a pushing action to separate the jammed objects while avoiding irreversible failures. We explain the action space, observation space, reward function and how we generate the training data and launch the training process in the following sections.

1) Action Space: The pushing action is defined by a line segment $[\boldsymbol{p}_1, \boldsymbol{p}_2] \in \mathbb{R}^2$ parallel to the support surface. Since the pushing actions only need to solve the interlock rather than transfer objects to precise positions, we relax the accuracy requirement for pushing actions by dividing the workspace to a 2D grid world, where $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ can be the center point of any grid. Therefore, the action space is a 4D discrete array $[s_x, s_y, e_x, e_y]$ where $\boldsymbol{p}_1 = [s_x, s_y]$ represents the grid position whose center point is the start point and $\boldsymbol{p}_2 = [e_x, e_y]$ represents the grid position whose center point is the endpoint. After getting the start and end points, we use the constrained sampling-based motion planner [27] to plan a straight line, which is the trajectory followed by the end-effector.

2) Observation Space: We use the camera image as observation space, the motivation of such a choice includes: (1) Image observation space provides a unified solution for scenarios with different numbers of objects. Moreover, it can keep the size of the observation space consistent during training, even though the observed objects number is changing during each episode. (2) Image observation can provide both pose and 2D geometric information of objects. (3) Recent research on PDDLStream leverages computer vision methods to relax the requirements for geometric model of objects [17], an image observation space will help the pusher to integrate with vision-aided PDDLStream in the future.

We assume one level of objects in the cluttered space (no stacking allowed). The RGB-D images produced by the RGB-D camera pass through a lightweight image processing pipeline before being fed as observations to the agent: the color information is discarded, and the depth information is thresholded to obtain a simple binary mask that, for each pixel, indicates whether there is an object there or not. An example of observation is shown in Fig. 3. During training, the simulator produces a depth map that we process in exactly the same way, to produce the same binary observations.

3) Reward: The reward generated for pushing actions is given by the sampling-based pick solver. It follows a simple rule: if the pushing action increases the solvability of the current state, then a positive reward will be given, otherwise no reward will be given. In our case, PDDLStream tries to find a plan for picking as many objects as possible after every pushing action, and the solvability is defined by how many objects can be picked with the plan. The details of the sampling-based pick solver are explained in Section III-C.

4) Data Generation: Data for training contains different situations of cluttered objects. In RL, the training data set decides

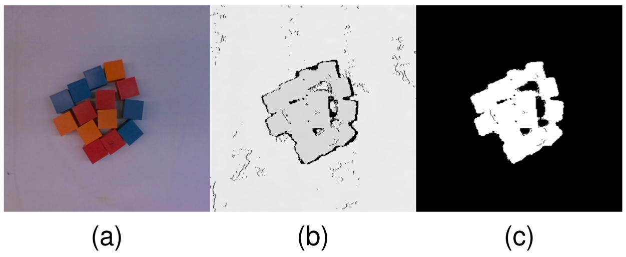

Fig. 3. Observation of the RL agent. (a) Cluttered objects we observe. (b) Gray scale depth image received from the RGB-D camera. (c) Binary image we generate by depth detection, which is used as the final observation for the RL agent. The white pixels indicate 1 and the black pixels indicate 0.

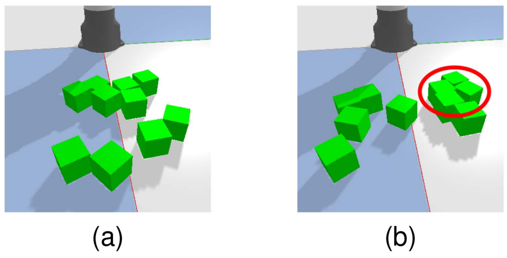

Fig. 4. Training data generation. When we randomly distribute the objects on the workspace, it can generate a solvable or an unsolvable situation. (a) Solvable situation, in which all the objects are accessible with rational task sequence and grasping approach. Such situation will not be added in the training data set since no pushing is required; (b) Unsolvable situation. The objects in the red circle form an interlock, which requires pushing actions to solve the situation. During the data generation process, we discard situations in (a) and save situations in (b).

the quality of the experience the agent will gather. Randomizing uniformly the block positions in the workspace will not necessarily generate an unsolvable (jammed) initial situation. The agent can not learn useful pushing skills when no pushing action is needed. To solve this issue, we add a data generating process before training. In the data-generating process, we first randomly choose object positions in the workspace, then pass the situation to the pick solver to evaluate whether the situation is solvable. Only unsolvable situations will be added in the training data set. The instances of different situations are shown in Fig. 4.

5) Training: Proximal Policy Optimization (PPO) [28] is a state-of-the-art policy gradient RL algorithm. By clipping the objective function, PPO avoids the severe performance drop caused by a noisy reward. We leverage this property for the solving-time uncertainty of the pick solver. For the policy, we use the same Convolutional Neural Network (CNN) structure as in [29], the CNN contains three convolutional layers and one flatten layer. The policy neural network is shown in Fig. 5.

C. Sampling-Based Pick Solver

The pick solver is a PDDLStream implementation of manipulation problem (domain) in a bin-picking environment. The details of PDDLStream and its solutions can be found in [6]. The goal in this domain is to pick and place all the objects from a bin to a plate. To formulate the PDDLStream problem, we use the following parameters: `?b` is the name of a block; `?r` is the region where the plate is; `?p` is 6 DOF block pose; `?g`

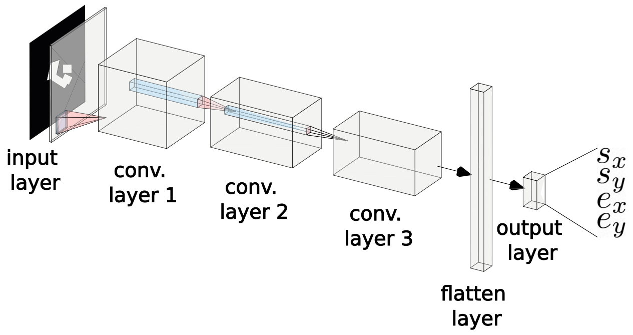

Fig. 5. CNN policy structure. Our policy takes the image of the current situation as input and contains three convolutional layers and one flatten layer. The output is four numbers which represent a 2D pushing motion.

is a 6 DOF grasp transform relative to the robot's end-effector; `q` is a 7 DOF joint-space configuration; and `?t` is a trajectory composed of a finite sequence of joint-space configurations. The static predicates include `Block`, `Conf`, `Pose`, `Grasp`, `Kin`, `Motion`, `Contain`, `CFree` which are the constant facts. The static predicates declare the types of parameters, for example, `Block` declares that `?b` is a block. The grounded predicates have to satisfy the constraints, for example, `Motion` declares that `?q1`and `?q2` are the start and end configurations for a trajectory `?t`, it also indicates that `?t` respects joint limits and collisions constraints. The fluent predicates include: `AtConf`, `AtPose`, `Holding`, `Empty` which are the facts that can be changed by the actions. Two actions are defined: `pick` and `place`.

To reduce the time lag caused by stream sampling, we only use necessary streams in the pick solver during training: (1) `sample-grasp` samples the collision-free poses for each grasp; (2) `test-cfree-approach-pose` checks if the approach of each grasping is collision-free; (3) `inverse-kinematics` samples the robot configurations and trajectories, which makes sure the task plan is inverse-kinematic accessible.

D. Action Validity Check

In each training step, the pushing action provided by the agent can be invalid. Invalid pushing actions should not be evaluated by the pick solver. For example, after a pushing action that did not touch any objects, the solvability for the current state stays the same as in the last step. Also, we assume the bin is fixed on the table, therefore, a pushing action that causes the collision between object and bin walls should be considered as failure and not be evaluated by the pick solver even if the state becomes solvable. Evaluating invalid actions will cause waste of training time and undesirable colliding behaviours.

A valid action in our case should satisfy three requirements: (1) The pushing action should be executable in the sense of inverse-kinematics; (2) the pushing action should change object positions; (3) the pushing action should not cause collisions between objects and obstacles, i.e., the bin. Therefore, we evaluate the validity of actions from three perspectives: inverse-kinematics, effect and collision. For the inverse-kinematics criterion, if the motion planner can calculate a collision-free

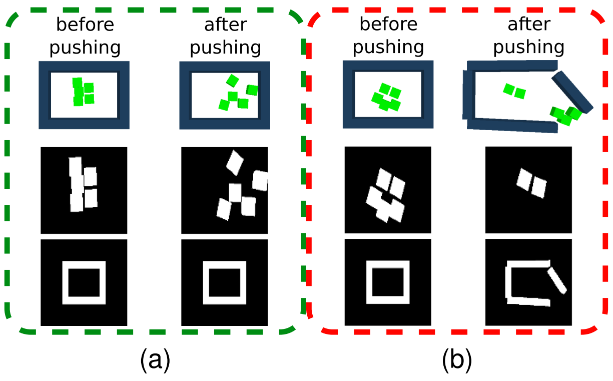

Fig. 6. Example of pushing validation. (a) Green block is an example of valid pushing, where the objects observation images (the second row) before and after the pushing action are different while the bin observation images (the third row) stay the same. (b) Red block is an example of invalid pushing, the changes of the bin observation images indicate the collision between objects and the bin.

trajectory for the pushing motion, then the criterion is satisfied. Here, the collision-free criterion considers the robot itself. For example, the robot's start position of a pushing action should not be colliding with objects. A colliding start position will make the motion planning fail.

$$Valid_i(s,a) = \begin{cases} \texttt{False} & MotionPlan = \texttt{None} \\ \texttt{True} & \text{otherwise} \end{cases} \tag{1}$$

The effect criterion requires that the pushing motion changes object positions. Changes on object positions are evaluated as the difference in object positions before and after the pushing action. To evaluate the difference, we compare the agent's observations on objects before and after the pushing action. In our work, the observation space is a 2D image, therefore, we calculate the ℓ^2-norm of the error between two images:

$$Valid_e(s,a) = \begin{cases} \texttt{False} & ||o(s) - o'(s)||_2 <= \epsilon \\ \texttt{True} & \text{otherwise} \end{cases} \tag{2}$$

Where o is the observed image after the pushing action, o' is the observed image before the pushing action, ϵ is the threshold which evaluate whether the pushing motion makes difference on the object clutter. An example of the observation is shown in Fig. 6.

The collision criterion requires that the pushing motion will not cause collisions between objects and the bin. To detect collisions between object and bin, we regard the surrounding walls of the bin as four movable objects, and detect the difference of the observation images before and after the pushing action. Image changes indicate collisions between an object and the bin. The image-based collision checking provides a unified solution that can be easily extended to more obstacles without extra calculation.

$$Valid_c(s,a) = \begin{cases} \texttt{True} & ||o_{bin}(s) - o'_{bin}(s)||_2 <= \epsilon \\ \texttt{False} & \text{otherwise} \end{cases} \tag{3}$$

Therefore, the total validation function can be given as:

$$Valid(s,a) = Valid_i(s,a) \wedge Valid_e(s,a) \wedge Valid_c(s,a) \tag{4}$$

E. Reward Shaping

After the validity check, we use solvability of the current state to evaluate the pushing action. Here, we define the solvability as an integer number which describes how many objects can be solved in the current state. In other words, the RL pusher attempts to increase the picking solvability of the state, while the pick solver helps shape the reward for the training procedure. Intuitively, in the cluttered manipulation domain, the more separate the objects are, the easier the pick solver will find a plan, since the *stream* can easily sample collision-free grasping poses. Given the maximum solving time, the reward for the last pushing action is given by:

$$r_s(s,a) = \begin{cases} 5n & Valid(s,a) = \texttt{True} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Where $n = Solvability(\tau_s, s, a)$ is evaluated by the pick solver, that is, how many objects can be picked after the pushing action a. The tolerant solving time τ_s can be introduced as a hyperparameter in the reward function. With this constraint we encourage the RL pusher to create a situation which is not only solvable in arbitrary horizon, but also able to be solved in the fixed time limit.

IV. Experiment and Results

A. Domain Setting

We apply the proposed method to a cluttered bin-picking problem in which objects are jammed together in a narrow bin. The robot has to deliver all of them on a plate by pushing and pick-and-place actions. Pushing actions always follow straight line segments, and pick-and-place actions only consider top-down overhead grasps. All objects are simplified as cubes, since the right angles of cubes can easily generate an interlock situation where we can better verify our method. All objects are regarded as homogeneous, their geometric information can be approximated by a bonding box. Based on the conclusions in PDDLStream planning algorithms comparison [6], we use the `adaptive` algorithm in the sampling based pick solver.

B. Experiment Setting

We train the RL agent in simulation and validate the trained agent on a real robot. We use a Franka Emika Panda robot. A Realsense depth camera (D455) is used to observe the objects. A NVIDIA Quadro RTX 3000 GPU is used to train the RL model. The simulation and training environment is built in PyBullet [30], while the communication and control of the robot is achieved using ROS (Robot Operating System). MoveIt [31] is used as the motion planner for the pushing actions.

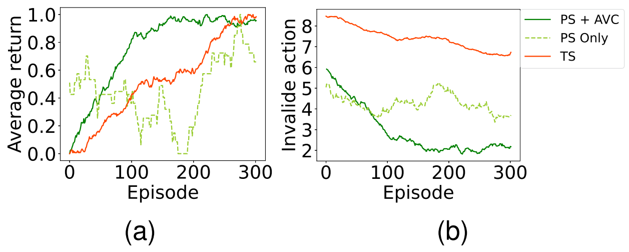

Fig. 7. Learning curves. We choose two indices to illustrate the learning process: (a) Average return of episodes and (b) invalid action per episode. We compare three different reward shaping methods including pick solver with action validity checking (PS+AVC) (our method), pick solver without action validity checking (PS Only), and total singulation (TS) reward.

TABLE I
HYPERPARAMETERS

Name	Value
`policy`	CnnPolicy
`learning_rate`	0.0001
`n_steps`	100
`batch_size`	10
`tolerant_solving_time` τ_s	60 s

C. Training Results

To illustrate the training process, we draw the learning curves of two important indices: the average return and the number of invalid actions per episode. We compare the learning process with different reward shaping methods which include (1) pick solver reward with action validity checking (PS+AVC); (2) pick solver reward without the action validity checking (PS Only); (3) total singulation (TS) reward designed in [25], where a positive reward is given when a pushing action increases the average distance among objects. As shown in Fig. 7, by obtaining higher return (accumulated reward), our reward shaping method learns to reduce the number of invalid actions efficiently, while the others struggle to reduce invalid actions or require more training episodes. The learning process converges after around 200 episodes (4 hours) while we set 300 episodes (6 hours) in total. It is worth noting that without the effect criterion in the action validity checking, the same training process spends 11 hours, since actions with no effect were also evaluated by the pick solver. The training hyperparameters are given in Table I.

D. Comparison Experiment

1) Ablation Comparison: To validate the trained model, we did an ablation comparison between the previous PDDLStream without pusher and our pusher-integrated hybrid method. In this experiment, objects are randomly distributed in the bin. The total tolerant solving time is set as $\tau_s = 60$ s. Each scenario has a different number of objects. We generate 100 object distributions in each scenario. The success ratios of each method related to the object numbers are shown in Fig. 8. The result shows that both methods suffer capability decline in scenarios with more objects since the objects are more likely to jam together or in a dead-end position. However, the pusher can improve the success ratio of the PDDLStream in different scenarios. It is worth noting that

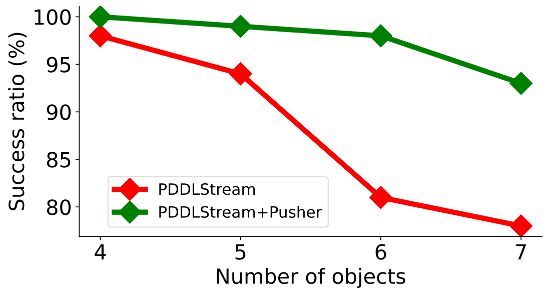

Fig. 8. Success ratio improvement. In different scenarios, the hybrid method (PDDL+Pusher) achieves higher success ratios, which indicates the RL pusher can effectively improve the capability of PDDLStream.

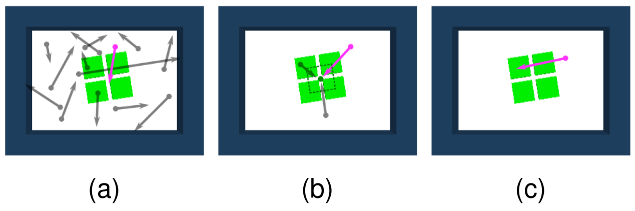

Fig. 9. Pushing strategies. (a) Sampling-based pusher randomly chooses start and end points of the pushing action in the workspace, simulates the consequence and checks validity of each sample, the first valid pushing action will be chosen. (b) Heuristic-guided pusher works similarly as the sampling-based pusher except pushing actions always end at the geometric center of objects. (c) RL pushers choose a pushing action by the RL agent. The grey arrows indicate invalid pushing action samples, and the pink arrows indicate valid pushing actions.

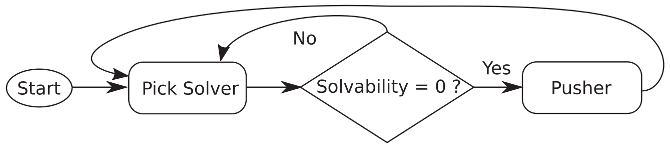

Fig. 10. Method workflow. To compare our method with the baselines, we keep the workflow between two modules as the same, only the pusher is changed among the baseline pushers and our RL pusher.

the pusher we use here is trained in a 5-object scenario, but it is also effective in scenarios with 4–7 objects.

2) Baseline Comparison: We compared our method with baseline methods: sampling-based pusher, heuristic-guided pusher, and a RL pusher trained with TS reward. Sampling-based method samples pushing actions by conducting forward simulation runs. Heuristic-guided method works similarly as sampling-based method, except the pusher is aware of the position of objects and always pushes to their geometric center. The different pushing strategies are illustrated in Fig. 9.

In the experiment, the workflow between pick solver and pusher is kept as in Fig. 10, only the pusher is changed among sampling-based pusher, heuristic-guided pusher and two kinds of RL-based pusher. Instead of using a random distribution, we use the object distribution generated in Section III-B4 in order to ensure the initial state is not solvable. We run each method 100 times. The total tolerant solving time is set as $\tau_s = 60$ s. Fig. 11 shows that our RL pusher can achieve higher success ratios than baseline methods, while the baseline methods suffer from long

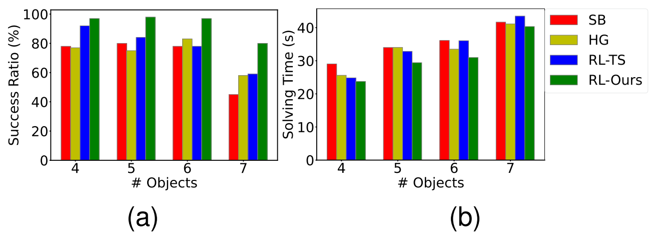

Fig. 11. Baseline pusher comparison. We compare (a) the planning success ratio (in 60 s) and (b) the solving time among planners with different pushing strategies, including sampling-based (SB), heuristic-guided (HG), RL with total-singulation reward (RL-TS) and our method.

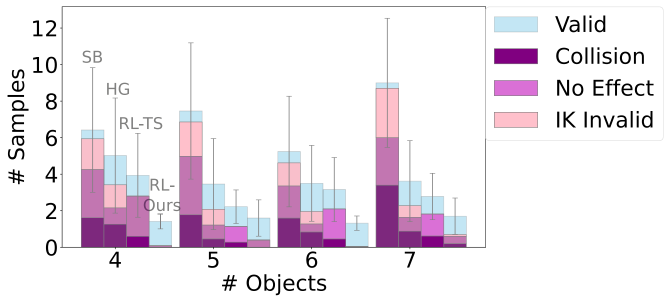

Fig. 12. Pushing action samples. We compare the number of samples made by different pushers. The result illustrates that our RL pusher can solve the problem with fewer total samples, that is, it can find valid pushing actions more efficiently. Each cluster of bars contains 4 pushers as we indicated. Different colors indicate valid and various invalid samples.

computational time. To better illustrate the sampling efficiency, we compare the pushing samples made by each pusher. As shown in Fig. 12, our RL pusher needs less samples to solve the problem since it can find valid actions efficiently which leads to situations with better solvability. The valid actions defined here does not necessarily improve the solvability but satisfy the three types of validity criterion. For example, an action that pushes an object to the bin corner can still pass the validity check, even though such an action will lead to low solvability or make the task fail. Therefore, the sampling number is not rigidly inversely proportional to the success ratio, but in general avoiding sampling invalid actions can speed up the solving process and benefit the success ratio.

E. Real-World Experiments

Finally, we implemented the proposed method on the real robot. We performed 40 experiments each with 4-7 objects respectively. Objects are manually located in the bin with unsolvable initial states. With the same tolerant solving time $\tau_s = 60$ s, the results show that the planner with RL pusher achieves success ratios of 92.5%, 87.5%, 92.5% and 77.5%. A planning example is shown in Fig. 13. The corresponding success ratios in the simulations are 97.5%, 95%, 97.5%, 80%. The success ratio declines in the real world can be caused by physics difference and the noise of real camera observations. In Fig. 14, we show the potential to extend our algorithm to objects with other shapes. The code of the proposed method and more experiment videos

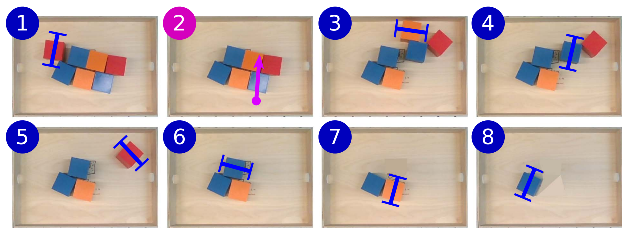

Fig. 13. Real-world experiment. By interactively activating pick solver and RL pusher, the cluttered bin-picking problem can be solved. The blue segments indicate a pick action and the red arrow indicates a pushing action.

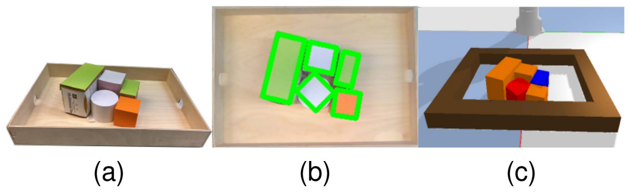

Fig. 14. Extension experiment. In order to extend the ability of our method. We extract the geometrical property of objects as bounding boxes. (a) Real scenario; (b) Bonding box detected by the camera; (c) Scenario reconstruction in simulation for pick solver collision checking.

can be found on GitHub: https://github.com/Gaoyuan-Liu/Non-prehensile-Augmented-TAMP.

F. Discussion

Here, we discuss limits of the current work and the potential for the future research: (1) Our method specifies some properties of the actions. For example, only top-down overhead grasping and straight-line pushing are considered. Such setting can be changed in the pick solver configuration files, therefore, this assumption will not severely reduce the generality of the method; (2) we customize a RL environment for the specific cluttered bin-picking domain. Such a domain shares the features of a lot of other real-world problems where a robot has to plan with non-prehensile actions with an unknown effect. However, RL environments need to be tuned and retraining processes are required on new domains with current method. A more general presentation on domains with more diverse non-prehensile actions needs to be studied.

V. Conclusion

In order to solve the TAMP problem in cluttered environments, we provide a hybrid planning method which combines the strength of PDDLStream and RL algorithms. The planning method is used to solve a cluttered bin-picking problem which contains multiple constraints such as jammed cluttering, dead-end situation and collisions. By augmenting PDDLStream with an RL pusher, the planner can separate the jammed objects and make the situation solvable for PDDLStream, while avoiding dead-end situations and collisions. We implement the method in both simulation and real world. The results show that the RL pusher can increase the success ratio of the PDDLStream in cluttered bin-picking problems.

REFERENCES

[1] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Sampling-based methods for factored task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 13–14, pp. 1796–1825, 2018.

[2] M. Danielczuk, J. Mahler, C. Correa, and K. Goldberg, "Linear push policies to increase grasp access for robot bin picking," in *Proc. IEEE Conf. Automat. Sci. Eng.*, 2018, pp. 1249–1256.

[3] M. A. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2018, pp. 6231–6235.

[4] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 4238–4245.

[5] K. Xu, H. Yu, Q. Lai, Y. Wang, and R. Xiong, "Efficient learning of goal-oriented push-grasping synergy in clutter," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 6337–6344, Oct. 2021.

[6] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2020, pp. 440–448.

[7] C. Aeronautiques et al., "PDDL–The planning domain definition language," Tech. Rep., 1998.

[8] M. Helmert, "The fast downward planning system," *J. Artif. Intell. Res.*, vol. 26, pp. 191–246, 2006.

[9] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 844–851, Apr. 2020.

[10] C. Zhang and J. A. Shah, "Co-optimizing task and motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 4750–4756.

[11] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 13, 2021, pp. 11962–11971.

[12] R. Chitnis, T. Silver, B. Kim, L. Kaelbling, and T. Lozano-Perez, "CAMPs: Learning context-specific abstractions for efficient planning in factored MDPs," in *Proc. Conf. Robot Learn.*, 2021, pp. 64–79.

[13] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *Int. J. Robot. Res.*, vol. 38, no. 7, pp. 793–812, 2019.

[14] R. Chitnis et al., "Guided search for task and motion plans using learned heuristics," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 447–454.

[15] M. Khodeir, B. Agro, and F. Shkurti, "Learning to search in task and motion planning with streams," *IEEE Robot. Automat. Lett.*, vol. 8, no. 4, pp. 1983–1990, Apr. 2023.

[16] Y. Jiang, F. Yang, S. Zhang, and P. Stone, "Task-motion planning with reinforcement learning for adaptable mobile service robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 7529–7534.

[17] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett, "Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 1940–1946.

[18] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," in *Proc. Robot.: Sci. Syst.*, 2020.

[19] L. Xu, T. Ren, G. Chalvatzaki, and J. Peters, "Accelerating integrated task and motion planning with neural feasibility checking," 2022, *arXiv:2203.10568*.

[20] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *Int. J. Robot. Res.*, vol. 40, no. 6-7, pp. 866–894, 2021.

[21] J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer, "Search-based task planning with learned skill effect models for lifelong robotic manipulation," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 6351–6357.

[22] D. Kalashnikov et al., "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. Conf. Robot Learn.*, 2018, pp. 651–673.

[23] A. Hundt et al., ""Good Robot!": Efficient reinforcement learning for multi-step visual tasks with SIM to real transfer," *IEEE Robot. Automat. Lett.*, vol. 5, no. 4, pp. 6724–6731, Oct. 2020.

[24] B. Huang, T. Guo, A. Boularias, and J. Yu, "Interleaving Monte Carlo tree search and self-supervised learning for object retrieval in clutter," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 625–632.

[25] I. Sarantopoulos, M. Kiatos, Z. Doulgeri, and S. Malassiotis, "Total singulation with modular reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 4117–4124, Apr. 2021.

[26] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Proc. Robot. Res.: 18th Int. Symp.*, 2020, pp. 405–419.

[27] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 1, no. 1, pp. 159–185, 2018.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[29] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[30] E. Coumans and Y. Bai, "Pybullet, a Python module for physics simulation for games, robotics and machine learning," 2016.

[31] M. Görner, R. Haschke, H. Ritter, and J. Zhang, "Moveit! task constructor for task-level motion planning," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2019, pp. 190–196.