Synergistic Task and Motion Planning with Reinforcement Learning Non-Prehensile Actions

Gaoyuan Liu[1,2], Joris de Winter[1,3], Denis Steckelmacher[4], Roshan Kumar Hota[1,3]
Ann Nowe[4], and Bram Vanderborght[1,2]

***Abstract*— Robotic manipulation in cluttered environments requires synergistic planning among prehensile and non-prehensile actions. Previous work 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* to learn pushing skills.**

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].

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 method 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 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 needs 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 make no further manipulations possible, or the objects can be pushed against the bin wall and be damaged.

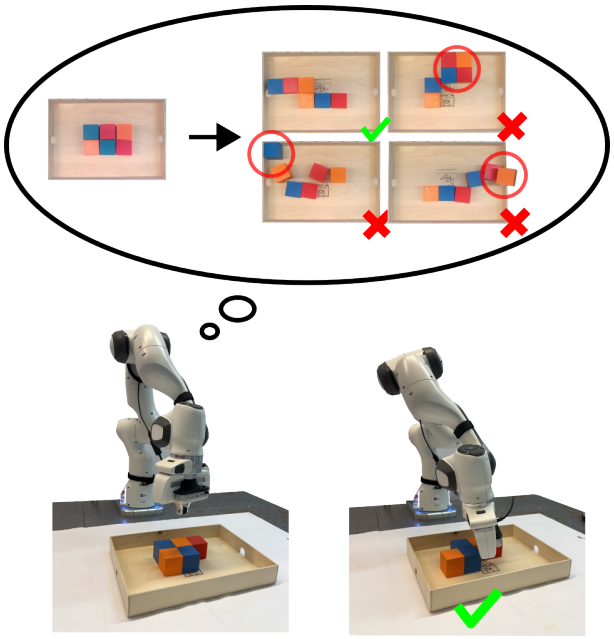


Fig. 1. For a cluttered bin pick-and-place task, the robot needs to push the objects into a situation where they can be grasped with certain task sequence and grasp position, and the TAMP solution can solve the problem by planning a rational task sequence and grasping pose. As for the pushing actions,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

Previous work leverages the adaptability of RL to learn such non-prehensile actions. Synergy learning algorithms, i.e. learning 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

This work was funded by the *Flemish* Government under the program *Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen*, China Scholarship Council (CSC) and EU-project euROBIN (No.101070596).

[1] Authors are with Brubotics, Vrije Universiteit Brussel, Brussels, Belgium. gaoyuan.liu@vub.be

[2] Authors are affiliated to imec, Belgium

[3] Authors are affiliated to Flanders Make, Belgium

[4] Ann Nowe and Denis Steckelmacher are with the Artificial Intelligence (AI) Lab, Vrije Universiteit Brussel, Brussels, Belgium.

[4], [5]. On one hand, for deterministic actions such as picking, sampling-based methods are generally considered to be more efficient and reliable than RL-based methods since even grasping with small errors can cause damages on 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 the small deviation when the purpose of pushing actions is just to make grasping feasible.

Considering different merits from 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 synergistic solution 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 provides a brief background on PDDLStream. Section IV details the methods and concepts in our framework. In Section V, we analyze the learning performance of the presented framework and demonstrate the obtained results. Section VI 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 task [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 leveraging 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 congested area for a mobile robot can lead to a sub-optimal planning solution. 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 TAMP process is to train a neural network with data set which contains pre-planned policies, the trained neural network therefore can make decisions quickly without planning from scratch [18]. Moreover, a feasibility checking neural network is trained to avoid unfeasible motion planning process [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 synergies between pushing and grasping [4]. With RGB-D signal as input, the value function which scores different pushing and grasping motions can be predicted. The motion with the best value will be chosen to be executed. Following work focus 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 search process [24]. This module evaluates pushing actions by predicting grasping feasibility with a separate network, instead, PDDLStream can evaluate grasping feasibility by sampling all the possible grasping poses. Moreover, previous work assumes that the cluttered objects are on a flat holding surface, thus no collision or dead-end situations are considered.

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. BACKGROUND

In this section we provide a brief background on the PDDLStream, an existing open-source sampling-based TAMP framework, which will be employed throughout this paper.

TAMP is defined as the problem of planning for a robot that operates in environments containing a large number of objects, changing the state of the objects while taking actions to move itself through the world [27]. The solution of a TAMP problem needs to integrate task planning and motion planning as well as their interconnections and constraints. The robotic task planning can be abstracted as a logical planning problem. That is, given the logical relations among the states and the actions, an initial state and a goal state, the planner attempts to find a scheme of actions following which the robot can steer the world from the initial state to the goal state. PDDL provides a standard encoding language for planning tasks [7]. In PDDL, a *problem* $(\mathcal{A}, \mathcal{I}, \mathcal{G})$ is described by a set of actions $\mathcal{A}$, an initial state $\mathcal{I}$, and a goal set of literals $\mathcal{G}$. The action set $\mathcal{A}$ only contains actions with deterministic effect, which is different than the RL action space A. A plan π is a finite sequence of action instances.

The states in PDDL formulation are discrete, which is not directly applicable for a robotic planning problem, where the configuration space is continuous. To find an applicable solution for a task planning problem, the continuous values needs to be sampled as an instance for the discrete PDDL planner. The *stream* therefore is integrated in the process. A *stream* $s(\bar{x})$ is a conditional generator endowed with a declarative specification of any facts its inputs and outputs always satisfy. On top of the classical PDDL *problem* $(\mathcal{A}, \mathcal{I}, \mathcal{G})$, a PDDLStream *problem* $(\mathcal{A}, \mathcal{S}, \mathcal{I}, \mathcal{G})$ adds a set of streams $\mathcal{S}$. The set of streams $\mathcal{S}$ can be seen as augments of the initial state $\mathcal{I}$, recursively sampling an instance from the potentially infinite set of facts $\mathcal{I}^*$ that hold initially and cannot be changed. A more detailed introduction of PDDLStream can be found in [6].

IV. METHODOLOGY

The learned policy provides a promising alternative to a handcrafted 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 a 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.

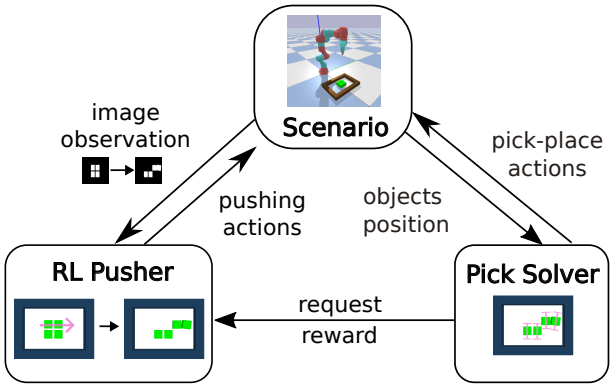


Fig. 2. The hybrid TAMP planning method structure. The scenario is the cluttered bin-picking environment in simulation or in real world. During training, the pick solver sends requests when there is no solvable objects, and evaluates the pusher's behaviour by giving rewards. Therefore, the pusher can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

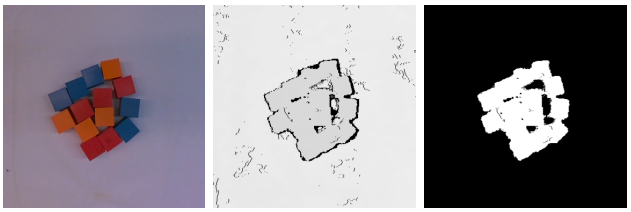
A. Markov Decision Process

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

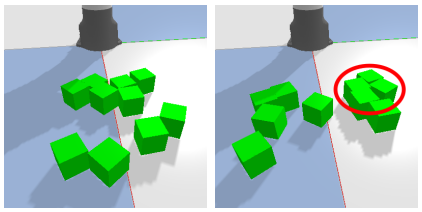
The pusher is defined as a RL agent which observes the current state of objects and provides a pushing action to separate the jammed objects while avoiding irreversible failures. We explain the action space, observation space, reward function and how we generate the training data and launch the training 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 $\boldsymbol{p}_1 = [s_x,\ s_y]$ represent the grid position whose center point is the start point and $\boldsymbol{p}_2 = [e_x,\ e_y]$ represent the grid position whose center point is the end point. After getting the start and end points, we use the constrained sampling-based motion planner [28] to plan a straight line, which is the trajectory followed by the end-effector.



(a) (b) (c)

Fig. 3. The observation of the RL agent. (a) The cluttered objects we observe. (b) The gray scale depth image received from the RGB-D camera. (c) The 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.



(a) (b)

Fig. 4. Training data generation. When we randomly distribute the objects on the workspace, it can generate a solvable or an unsolvable situation. (a) 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) an 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).

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 are 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 a negative 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 is explained in Section IV-C.

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

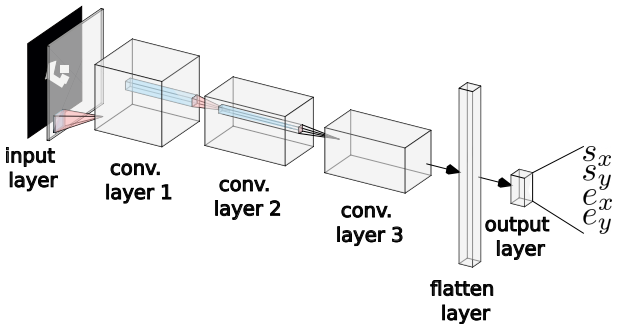


Fig. 5. The CNN policy structure. Our policy takes the image of the current situation as input and contains three convolutional layers and one flatten layer. The output is four numbers which represent a 2D pushing motion.

5) Training: Proximal Policy Optimization (PPO) [29] 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 [30], 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 goal in this domain is to pick and place all the objects in 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` 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 training time wasting 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 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 $\ell^2 -$ norm of the error between two images:

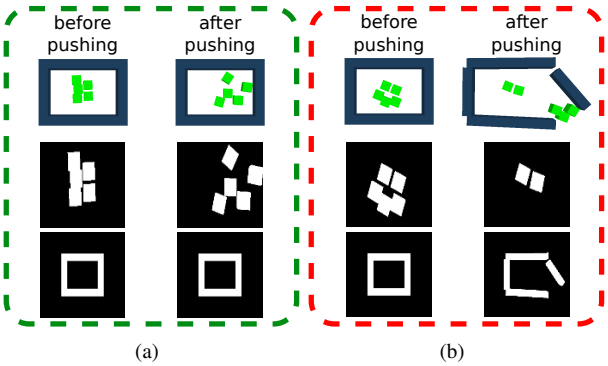


Fig. 6. The example of pushing validation. (a) The green block is an example of valid pushing, where the objects observation images (the second row) before and after the pushing action are different while the bin observation images (the third row) stay the same. (b) The red block is an example of invalid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

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

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 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.

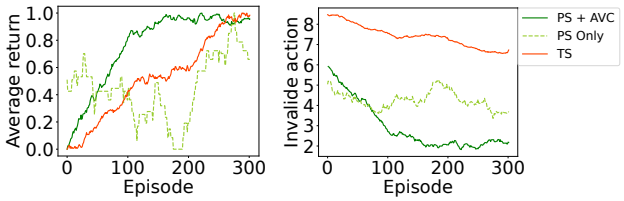
$$Valid_c(s,a) = \begin{cases} \texttt{True} & ||o_{bin}(s) - o'_{bin}(s)||_2 <= \epsilon \\ \texttt{False} & \text{otherwise} \end{cases} \tag{3}$$

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 a 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



(a) (b)

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.

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.

V. 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 the interlock situation where we can better verify our method. All objects are regarded 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 [31], while the communication and control of the robot is achieved using ROS (Robot Operating System). MoveIt [32] is used as the motion planner for the pushing actions.

TABLE I

Hyperparameters

Name	Value
`policy`	CnnPolicy
`learning_rate`	0.0001
`n_steps`	100
`batch_size`	10
`tolerant_solving_time` τ_s	40s

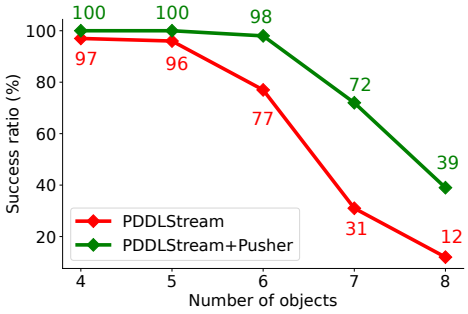


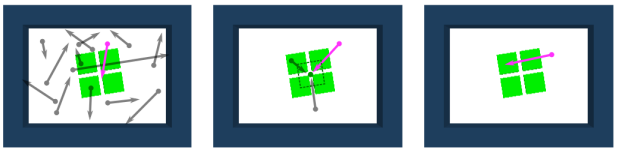
Fig. 8. Success ratio improvement. In different scenarios, the hybrid method (PDDL+Pusher) achieves higher success ratios, which indicates the RL pusher can effectively improve the capability of PDDLStream.

C. Training Results

To illustrate the training process, we draw the learning curves of two important indices: average return and 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 (ST) 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 episode (4 hours) while we set 300 episodes (6 hours) in total. It is worth noticing 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, and both methods have to solve the cluttered bin picking problem in 60 seconds. Each scenario has a different number of objects. We generate 100 object distributions in each scenario. The solving 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



(a) (b) (c)

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.

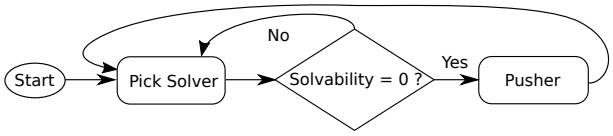
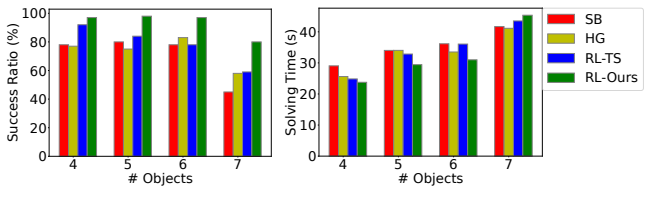


Fig. 10. The method workflow. To compare our method with the baselines, we keep the workflow between two modules as the same, only the pusher is changed among the baseline pushers and our RL pusher.

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 the pusher we use here is trained in a 5-object scenario, but it also shows effect 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. In this experiment, we use the validation data generated in Section IV-B.4 and run each method 100 times. The total tolerant solving time is set as



(a) (b)

Fig. 11. Baseline pusher comparison. We compare (a) the planning success ratio (in 60s) 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.

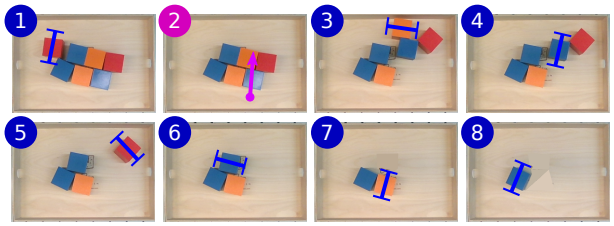


Fig. 12. Real-world experiment. By interactively activating pick solver and RL pusher, the cluttered bin picking problem can be solved. The blue segments indicate a pick action and the red arrow indicates a pushing action.

(a) (b) (c)

Fig. 13. Extension experiment. In order to extend the ability of our method. We extract the geometrical property of objects as bounding boxes. (a) The real scenario; (b) Bonding box detected by the camera; (c) Scenario reconstruction in simulation for pick solver collision checking.

$\tau_s = 60s$. Fig. 11 shows that our RL pusher can achieve higher success ratios than baseline methods, while the baseline methods suffer long computational time. It is worth noticing that (1) heuristic-guided method is not always better than sampling-based method because pushing deviation from the geometric center sometimes provides a better solution. And, (2) scenarios with less objects are not necessarily easier for the sampling-based and heuristic-guided method because of the trade-off between effect criterion and collision criterion.

E. Real-World Experiments

Finally, we implemented the proposed method on the real robot. We performed 40 experiments with 4-7 objects. Each time we randomly choose the number of objects and rearrange the distribution of the objects with interlocks. The results show that the planner with RL pusher achieves 85% task success ratio. A planning example is shown in Fig. 12. The success ratio decline in the real world can be caused by physics difference and the noise of real camera observations. The extension experiment with unknown objects requires (1) training the RL pusher in scenarios with different types of object and (2) more advanced computer vision aid for the pick solver. As shown in Fig. 13, we provide one solution in which we detect bonding boxes of objects and reconstruct the scenario in the simulation. The code of the proposed method and more experiment videos can be found on GitHub: `https://github.com/Gaoyuan-Liu/Non-prehensile-Augmented-TAMP`.

F. Discussion

The proposed method provides a solution for the cluttered bin picking problem and shows improvement compared to the baseline methods. However, from the experiments we

also found some limits of the current work: (1) Our method currently only considers objects with primitive shapes. With more sophisticated computer vision aid, the method can be extended to objects with complex shapes; (2) we customize a RL environment for the cluttered bin-picking domain, a more general presentation for domains with different constraints needs to be studied.

VI. CONCLUSIONS

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. Garrett, T. Lozano-Pérez, and L. Kaelbling, "Sample-based methods for factored task and motion planning," 2017.

[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. on Automation Science and Engineering (CASE)*, pp. 1249–1256, IEEE, 2018.

[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," 2018.

[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 *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4238–4245, IEEE, 2018.

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

[6] 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, pp. 440–448, 2020.

[7] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. SRI, A. Barrett, D. Christianson, *et al.*, "Pddl— the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

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

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

[10] C. Zhang and J. A. Shah, "Co-optimizing task and motion planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4750–4756, IEEE, 2016.

[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 *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 11962–11971, 2021.

[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 *Conference on Robot Learning*, pp. 64–79, PMLR, 2021.

[13] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *The International Journal of Robotics Research*, vol. 38, no. 7, pp. 793–812, 2019.

[14] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 447–454, IEEE, 2016.

[15] M. Khodeir, B. Agro, and F. Shkurti, "Learning to search in task and motion planning with streams," *arXiv preprint arXiv:2111.13144*, 2021.

[16] Y. Jiang, F. Yang, S. Zhang, and P. Stone, "Task-motion planning with reinforcement learning for adaptable mobile service robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7529–7534, IEEE, 2019.

[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 *2022 International Conference on Robotics and Automation (ICRA)*, pp. 1940–1946, IEEE, 2022.

[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," *arXiv preprint arXiv:2006.05398*, 2020.

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

[20] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Learning compositional models of robot skills for task and motion planning," *The International Journal of Robotics Research*, 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 *2022 International Conference on Robotics and Automation (ICRA)*, pp. 6351–6357, IEEE, 2022.

[22] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*, pp. 651–673, PMLR, 2018.

[23] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager, ""good robot!": Efficient reinforcement learning for multi-step visual tasks with sim to real transfer," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6724–6731, 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 *2022 International Conference on Robotics and Automation (ICRA)*, pp. 625–632, IEEE, 2022.

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

[26] A. Eitel, N. Hauff, and W. Burgard, "Learning to singulate objects using a push proposal network," in *Robotics research*, pp. 405–419, Springer, 2020.

[27] 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.

[28] Z. Kingston, M. Moll, and L. E. Kavraki, "Sampling-based methods for motion planning with constraints," *Annual review of control, robotics, and autonomous systems*, vol. 1, no. 1, pp. 159–185, 2018.

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

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

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