Augmented Task and Motion Planning with Reinforcement Learning Non-Prehensile Motion

Gaoyuan Liu[1,2], Joris de Winter[1,3], Denis Steckelmacher[4],
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, namely *pusher*, and introduce a hybrid planning method. The procedure contains pushing motions which can eliminate interlocked situations and make the problem solvable. Also, the sampling-based algorithm evaluates the pushing actions by providing reward in the training process, thus the *pusher* can learn to avoid situations containing irreversible failures. The proposed hybrid planning method is validated 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 two planning levels gives more comprehensive solutions which consider both logical and geometric constraints. The 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 algorithms can give task sequences and motion plans based on the logical relations among tasks and the geometric constraints such as physical collisions. The sampling-based algorithms can provide solutions when the situation is solvable, i.e. there is at least one collision free task and motion plan can be found. However, cluttered bin picking problems are likely remain unsolvable because of the object proximity to the bin walls or to other objects

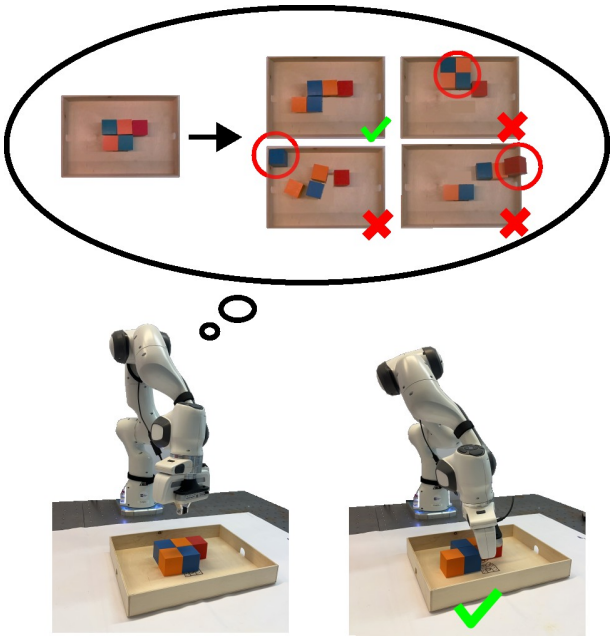


Fig. 1. For a cluttered bin pick-and-place task, the robot first needs to push the objects into a situation where they can be grasped with certain task sequence and grasp position, and then the TAMP solution can solve the problem by planning a rational task sequence and grasping pose. As for the pushing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

[2]. Such situations require non-prehensile actions such as pushing to change the position or orientation of objects, yet the non-prehensile actions are hard to sample because of the stochastic effects. Previous work leverages the adaptability of RL to learn such non-prehensile actions. Synergy learning algorithms can effectively solve the problem by separating the cluttered objects by pushing them until the goal object is feasible [1]. However, pushing objects in a bin can cause irreversible failures during manipulation, for instance, the objects can be pushed to corners of the bin and makes no further manipulations possible, or the objects can be pushed against the bin wall and be damaged.

To solve the issues, we combine the previous work on both sampling-based TAMP and vision-based RL synergy learning and introduce a hybrid planning method which is comprised of a RL *pusher* and a sampling-based *solver*. The sampling-based TAMP 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

This work was supported by the *Flemish* Government under the program *Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen* and China Scholarship Council (CSC).

[1] Authors are with the Department of Mechanical Engineering, 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.

work interactively, the pusher helps to provide a solvable situation for the solver while the solver evaluate 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 agent first plans pushing actions which separate the jammed objects and make the situation solvable. Afterwards the sampling-based method plans a task sequence and motion trajectories without extra training process. The main contributions of this work are as follows: (1) We coordinate the abilities of learning-based and sampling-based methods and give a hybrid planning method; (2) We provide a novel reward shaping strategy for robotics RL; (3) We extend the ability of the previous TAMP 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. In Section IV, we analyze the learning performance of the presented framework and demonstrate the obtained results. 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 [3]. Such algorithms are proven to be probabilistically complete and computationally efficient [1]. The TAMP problem is described and solved with the Planning Domain Definition Language (PPDL) [4] with a sampling preprocess, i.e. *stream* [3]. By doing so, off-the-shelf artificial intelligence planning algorithms, such as FastDownward [5], can be deployed seamlessly. Toki et al. extend the ability of the TAMP algorithm to the dynamic environment by sampling in the object-centered frames [6]. By minimizing the task cost function, the strategies of TAMP can also be optimized [7].

To improve the efficiency of the sampling process and leveraging past experience, Beomjoon 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 [8]. Rohan et al. formulate the local search as a Markove decision process (MDP), and use RL to guide the sampling in the motion level [9]. Similarly, a neural network trained on prior planning experience is integrate to score the relevance of streams [10]. 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 [11]. Previous TAMP algorithms requires a solvable initial situation, that is, there exists at least one collision-free task and motion plan. However, such assumption will not always be true in a cluttered pick-and-place problem.

Recently the vision-based RL has been utilized on different manipulation tasks [12] [13]. The end-to-end RL algorithms requires enormous amount of data to show effects. Andy et al. narrow down the problem to learning the synergies between pushing and grasping [14]. 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 [15]. They assumed 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 [16], the cornered objects can be released by a sliding action, thus no dead-end situation is considered in their work. Michael et al. use pushing motions to increase the bin-picking success ratio, similarly, suction grasps are used to reach the cornered objects. In our work, we try to solve the similar problem without using alternative motion primitives or end-effectors, and the collision issues will also be considered.

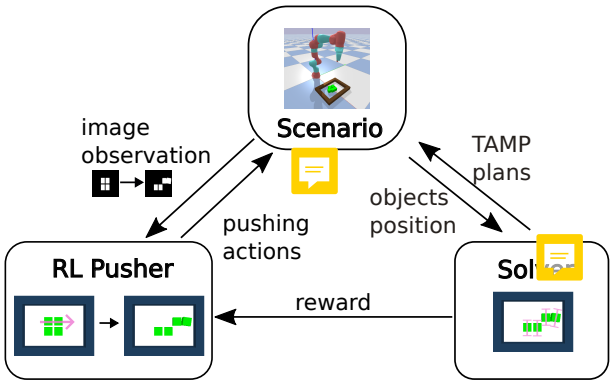


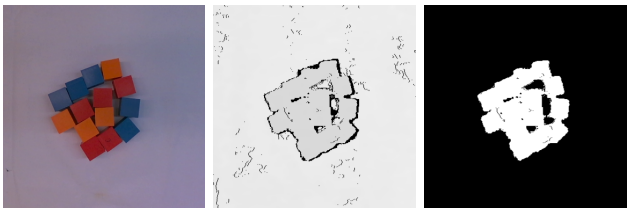
Fig. 2. The hybrid TAMP planning method structure. The scenario is the cluttered environment in simulation or in real world. During training, the solver evaluates the pusher's behaviour by giving rewards during training, therefore, the pusher can effectively learn to create a solvable situation for the solver. After training, the hybrid planning method has the ability to solve the pick-and-place problems with unsolvable initial states.

III. METHODOLOGY

The essence of our method is that the agent should only learn when necessary. In our case, actions with deterministic effects, such as picking with known object positions, can be solved by the sampling-based solver, which avoids the lengthy training process. For actions with stochastic effects such as pushing, sampling-based solver tends to fail since no precise consequence of the motions can be sampled, therefore the RL-based algorithm is used to learn such uncertainty. The structure of our hybrid planner is shown in Fig.2.

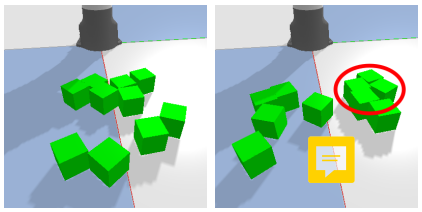
A. Problem Formulation

We formulate the problem of pushing augmented TAMP as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, A, \mathcal{R}, \mathcal{T}, \gamma)$, 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. Our method focus on reward shaping by sampling-based solvability. This will relax the accuracy



(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; (b) an unsolvable situation. The objects in the red circle form an interlock which requires pushing actions to solve the situation.

requirement of the pushing actions, and boost the learning process. 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 an 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. We relax the accuracy requirements 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 grids. 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 Rapidly-exploring Random Tree (RRT) algorithm to plan a straight line which is the trajectory followed by the end-effector.

2) Observation Space: We use visual representation as the observation space, which is the binary images generated from an RGB-D camera. In the training procedure, we use the image generated by the simulation. In this work we consider one layer of cluttered objects, which means no stacking situations. Therefore, to reduce the size of the observation space, we generate binary images from raw depth images, where pixels containing objects are encoded as 1, and empty space pixels are encoded as 0. An example of observation is shown in Fig.3.

3) Reward: The reward which evaluates the pushing actions are given by the sampling-based solver. It follows a simple rule: if the pushing action creates a situation which is solvable by the solver, then positive reward will be given, otherwise negative reward (punishment) will be given. The details of the sampling-based solver is explained in Section III-C.

4) Data Generation: The 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.

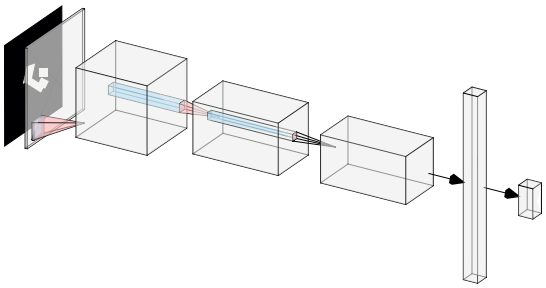


Fig. 5. The CNN policy structure. Our policy takes the image of the current situation as input and contains three convolutional layer and one flatten layer. The output is four numbers which represent a 2D pushing motion.

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 to the solver before training. In the data generating process, we first randomly choose object positions in the workspace, then pass the situation to the solver to evaluate weather 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) [17] is a bench-marking policy gradient RL algorithm. By clipping the objective function, PPO avoids the severe performance drop caused by the noisy reward. We leverage this property for the solving-time uncertainty of the sampling-based TAMP solver. We use a Convolutional Neural Network (CNN) structure for the policy, containing three convolutional layers and one flatten layer. The policy neural network is shown in Fig.5.

C. Sampling-Based TAMP Solver

1) PDDLStream: 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.

In this work, we use the Planning Domain Description

Language (PDDL) as the expression of planning problems. In PDDL, a *domain* file defines all object types, constants, variables, predicates and actions. A *predicate* p is a Boolean function which evaluates a fact to be true or negated. A *fact* $p(\bar{x})$ is a true predicate. A *literal* is a fact or a negated fact. On top of these, a *state* is described with a set of literals. An *action* contains a parameter tuple $\bar{X} = \langle X_1, ..., X_k \rangle$, a precondition literal set $pre(a)$ and a effect literal set $eff(a)$, which describes the action execution condition and the change the action will make in the environment. To instantiate an action $a(\bar{x})$, we need to replace the parameters $\bar{X}$ with objects $\bar{x}$. 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 $\pi = [a_1(\bar{x}_1), ..., a_k(\bar{x}_1)]$ is a finite sequence of k 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 introduced to integrate task planning and motion planning [3]. A *stream* $s(\bar{X})$ is a conditional generator endowed with a declarative specification of any facts its inputs and outputs always satisfy. A stream is defined with two parts: a python generator and a PDDL format declaration. The stream generator, coded in python, enumerates the output $\bar{y}_i$ from the input $\bar{x}$ and continuous conditions. For example, the `inverse_kinematic` generator numerates the configurations q for the robot joints, given an input as the pose of the end-effector $\bar{\mathbf{x}} = p_{ee}$. The stream declaration in PDDL format contains the domain constraints $s.domain = \{p \mid \forall \bar{x} \in \bar{X}.p(\bar{x})\}$ (here p indicates predicate) which specify the facts that the inputs $s.input$ satisfy, and the certified constrains $s.certified$ which is a set of predicate that asserts any facts that the input $s.input$ and output $s.output$ pairs $\langle \bar{x}, \bar{y} \rangle$ 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.

2) Robotic Manipulation Domain: PDDLStream is applied to a 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 an 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 constrains, 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`. Worth to note that here only the actions with deterministic effect can be used, thus pushing actions with stochastic effects cannot be directly integrated here. [can you put here the PDDL code? or is that too long? The same way they did it in the paper you sent me]JD

D. Reward Shaping

The reward we use to evaluate the current pushing action is composed of three parts: an action-validity reward r_v, and a solvability reward r_s.

$$r(s, a) = r_v(s, a) + r_s(s, a) \tag{1}$$

The action-validity reward encourages the agent choosing pushing actions which are valid. 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. The solvability reward encourages the robot to push objects to solvable positions, which means there is no interlocks or dead-end situations.

For the action-validity reward, we consider the rationality of the pushing action:

$$r_v(s, a) = \begin{cases} 0 & Valid(s, a) = \texttt{True} \\ -2 & \text{otherwise} \end{cases} \tag{2}$$

Where $Valid$ is a compound function which are comprised of multiple criterion. The detailed implementation in our case can be found in Section IV.

We use solvability of the current state to evaluate the previous pushing action. In other words, the RL pusher helps the solver to create a solvable situation, while the solver helps shape the reward for the training procedure. Intuitively, in the cluttered manipulation domain, the more separate the objects are, the easier the 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:

$$r_s(s, a) = \begin{cases} 10 & Solvable(\tau_s, s, a) = \texttt{True} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Where the $Solvable(\tau_s)$ is evaluated by the sampling-based solver. 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 implement the proposed method in a cluttered bin picking problem in which objects are jammed together in a narrow bin. The robot has to first separate them by pushing actions and then pick and place them on the plate. All the 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.

Based on the conclusions in PDDLStream planning algorithms comparison [3], we use the `adaptive` algorithm in the sampling based 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, while the communication and control of the robot is achieved by ROS (Robot Operating System). MoveIt is used as the motion planner for the pushing actions [18].

C. Reward Implementation

In this section we first give the details about how we implement the reward functions and the training procedure, then we show the training results.

In our case, 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 trajectory for the pushing motion, then the criterion is satisfied.

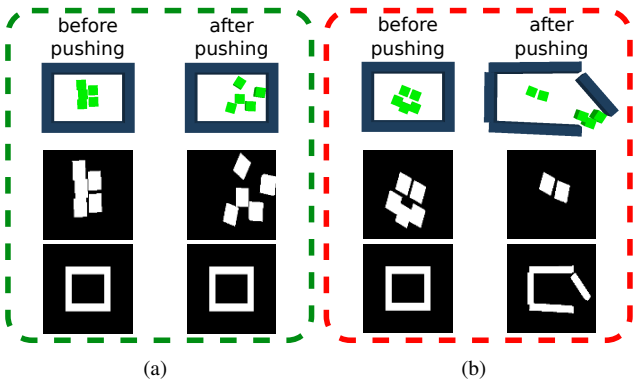
$$Valid_i(s,a) = \begin{cases} \texttt{False} & MotionPlan = \texttt{None} \\ \texttt{True} & \text{otherwise} \end{cases} \tag{4}$$

The effect criterion requires the pushing motion makes differences on the object positions. 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:

$$Valid_e(s,a) = \begin{cases} \texttt{False} & ||o(s) - o'(s)||_2 <= \epsilon \\ \texttt{True} & \text{otherwise} \end{cases} \tag{5}$$

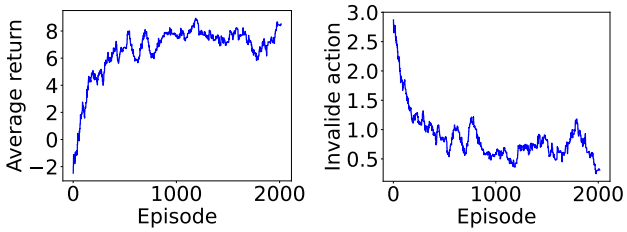
Where o is the observed image after the pushing action, o' is the observed image before the pushing action, ϵ is the threshold which evaluate whether the pushing motion makes difference on the object cluttered situation. An example of the observation is shown in Fig.6.

The collision criterion requires the pushing motion will not cause collision 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.



(a) (b)

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.



(a) (b)

Fig. 7. Learning curves. We choose two indices to illustrate the learning process: (a) The average return of episodes, the agent can efficiently learn to obtain more return (accumulated reward) and the optimal return converges after 1000 episodes; (b) The invalid action. The number of invalid pushing actions chosen by the agent in every episode. The agent will learn to avoid invalid pushing actions after training, which indicates the effect of the the reward function.

$$Valid_c(s,a) = \begin{cases} \texttt{True} & ||o_{bin}(s) - o'_{bin}(s)||_2 <= \epsilon \\ \texttt{False} & \text{otherwise} \end{cases} \tag{6}$$

Therefore, the validation function in equation(2) can be given as:

$$Valid(s,a) = Valid_i(s,a) \wedge Valid_e(s,a) \wedge Valid_c(s,a) \tag{7}$$

D. Training Results

We use scenario with 5 objects to train the pusher, and validate the trained model in scenarios with different numbers (4-8) of objects. We show the graphs of training and validation processes in this section.

To illustrate the training process, we draw the learning curves of two important indices: average return and invalid action. As shown in Fig.7, by obtain higher return (accumulated reward), the agent learns to reduce the number of invalid actions chosen in one episode. The learning process

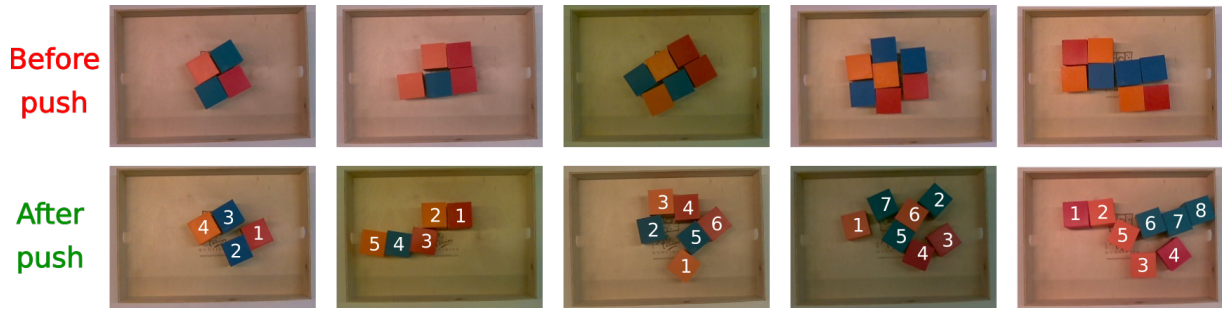


Fig. 8. Real world experiments. We implement the hybrid method on the real robot, and validate the effect of the pusher in scenarios with different number of objects. The first row shows the initial situations where the objects jam together and no possible PDDLStream solution can be found. The second row shows that after pushing actions, the situations become solvable by PDDLStream and the number marked on objects are the picking sequence planned by PDDLStream. With such rational plan, all the objects can be picked.

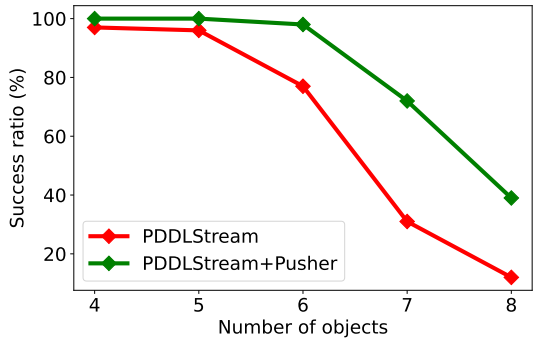


Fig. 9. Success ratio improvement. In different scenarios, the hybrid method (PDDL+Pusher) achieves higher success ratios, which indicates the RL pusher can efficiently improve the capability of PDDLStream.

converges after around 1000 episode (4 hours) while we set 2000 episodes (15 hours) in total.

To validate the trained model, we compare the success ratios of two different planning methods: previous PDDLStream without pusher and pusher-integrated hybrid planning method. The objects are randomly distributed in the center region of the bin, and both methods have to solve the cluttered bin picking problem in 20 seconds. Each scenario has different number of objects, we generate 100 cluttered situations in each scenario, and the solving ratios of each method related to the object numbers are shown in Fig.9. The results show that the planners suffers 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 to note that the pusher we use here is trained in a 5-objects scenario, but it also shows robustness in scenarios with 4-7 objects.

We implement the hybrid method on the real robot, and validate the pusher's effect in different scenarios. As shown in Fig.8, the pusher can effectively make the state solvable while avoiding dead-end situations and collisions. After pushing, the solver can quickly provide a sequence by which all the objects can be picked. More experiment can be found here: [TODO: make the video.][GL].

V. CONCLUSIONS

In this paper, we provide a hybrid planning method which combines the strength of sampling-based TAMP algorithm, i.e. 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 planer can separate the jammed objects and make the situation solvable for PDDLStream, while avoiding dead-end situation or collisions. We validate 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 by 30 percent. The future work will be extending the method to a more uncertain environment such as with unknown-shape objects and increasing the robustness of the system.

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] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.