Motion Planner Augmented Reinforcement Learning for Robot Manipulation in Obstructed Environments

Jun Yamada[1]*, Youngwoon Lee[1]*, Gautam Salhotra[2], Karl Pertsch[1], Max Pflueger[2], Gaurav S. Sukhatme[2], Joseph J. Lim[1], Peter Englert[2]

[1] Cognitive Learning for Vision and Robotics Lab
[2] Robotic Embedded Systems Laboratory
Department of Computer Science
University of Southern California, Los Angeles, CA

Abstract

Deep reinforcement learning (RL) agents are able to learn contact-rich manipulation tasks by maximizing a reward signal, but require large amounts of experience, especially in environments with many obstacles that complicate exploration. In contrast, motion planners use explicit models of the agent and environment to plan collision-free paths to faraway goals, but suffer from inaccurate models in tasks that require contacts with the environment. To combine the benefits of both approaches, we propose motion planner augmented RL (MoPA-RL) which augments the action space of an RL agent with the long-horizon planning capabilities of motion planners. Based on the magnitude of the action, our approach smoothly transitions between directly executing the action and invoking a motion planner. We evaluate our approach on various simulated manipulation tasks and compare it to alternative action spaces in terms of learning efficiency and safety. The experiments demonstrate that MoPA-RL increases learning efficiency, leads to a faster exploration, and results in safer policies that avoid collisions with the environment. Videos and code are available at `https://clvrai.com/mopa-rl`.

1 Introduction

In recent years, deep reinforcement learning (RL) has shown promising results in continuous control problems [1, 2, 3, 4, 5, 6, 7]. Driven by rewards, robotic agents can learn tasks such as grasping [8, 9, 10] and peg insertion [10]. However, prior works mostly operated in controlled and uncluttered environments, whereas in real-world environments, it is common to have many objects unrelated to the task, which makes exploration challenging. This problem is exacerbated in situations where feedback is scarce and considerable exploration is required before a learning signal is received.

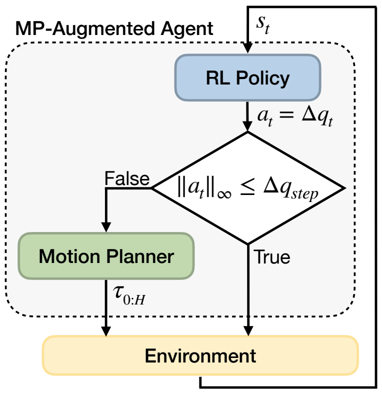


Figure 1: Our framework extends an RL policy with a motion planner. If the predicted action by the RL policy is above a threshold Δq_{step}, the motion planner is called. Otherwise, it is directly executed.

Motion planning (MP) is an alternative for performing robot tasks in complex environments, and has been widely studied in the robotics literature [11, 12, 13, 14, 15, 16, 17]. MP methods, such as RRT [12] and PRM [11], can find a collision-free path between two robot states in an

*Equal contribution. Correspondence to: `jy_597@usc.edu` and `lee504@usc.edu`

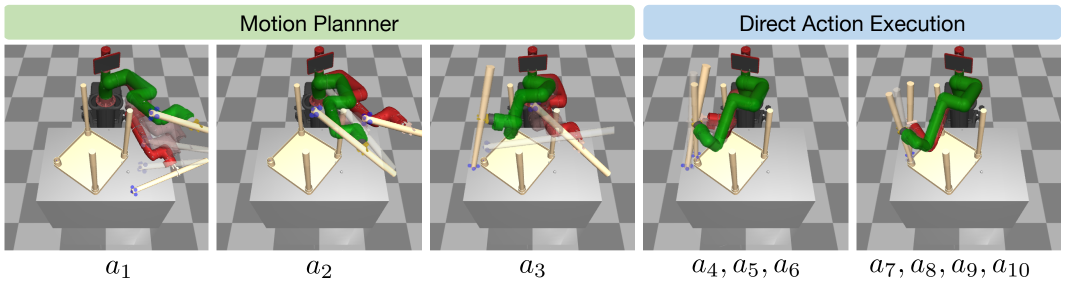


Figure 2: To learn both collision-avoidance and contact-rich skills, our method (MoPA-RL) combines motion planning and model-free RL. In the images, the green robot visualizes the target state provided by the policy. Initially, motion planning can be used to navigate to target states a_1, a_2, and a_3 while avoiding collision. Once the arm passes over other legs, a sequence of primitive actions $a_4 - a_{10}$ are directly executed to assemble the leg and tabletop.

obstructed environment using explicit models of the robot and the environment. However, MP struggles on tasks that involve rich interactions with objects or other agents, where it is challenging to obtain accurate contact models. Furthermore, MP methods cannot generate plans for complex manipulation tasks (e.g., object pushing) that cannot be simply specified by a single goal state.

In this work, we propose motion planner augmented RL (MoPA-RL) which combines the strengths of both MP and RL by augmenting the action space of an RL agent with the capabilities of a motion planner. Our approach has three benefits: (1) MoPA-RL can add motion planning capabilities to *any* RL agent with joint space control as it does not require changes to the agent's architecture or training algorithm; (2) MoPA-RL allows an agent to freely switch between MP and direct action execution by controlling the scale of action; and (3) the agent naturally learns trajectories that avoid collisions by leveraging motion planning, allowing for safe policy execution even in obstructed environments. We show that MoPA-RL learns to solve manipulation tasks in these obstructed environments while model-free RL agents suffer from local optima and difficult exploration.

2 Method

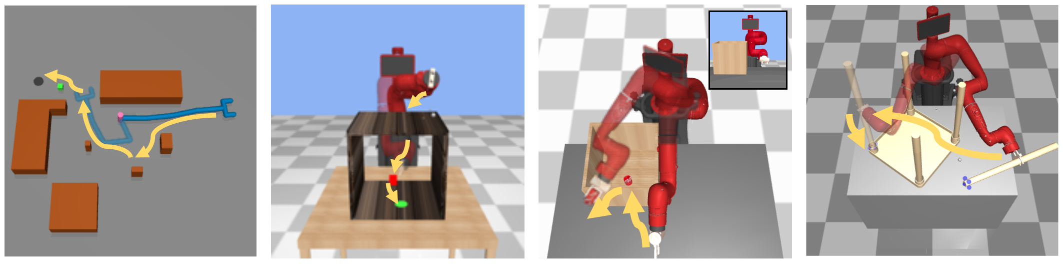
2.1 Motion Planner Augmented Reinforcement Learning

We formulate the problem as a Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho, \gamma)$ consisting of states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition function $P(s' \in \mathcal{S}|s, a)$, reward $R(s, a)$, initial state distribution ρ, and discount factor $\gamma \in [0, 1]$. The agent's action distribution at time step t is represented by a policy $\pi_\phi(a_t|s_t)$ with state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$, where ϕ denotes the parameters of the policy. Once the agent executes the action a_t, it receives a reward $r_t = R(s_t, a_t)$. The performance of the agent is evaluated using the discounted sum of rewards $\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$, where T denotes the episode horizon.

In RL for continuous control, the action space can be defined as the joint displacement $a_t = \Delta q_t$, where q_t represents robot joint angles. To prevent collision and reduce control errors, the action space is constrained to be small, $\mathcal{A} = [-\Delta q_{\text{step}}, \Delta q_{\text{step}}]^d$, where Δq_{step} represents the maximum joint displacement for a direct action execution [18] and d denotes the dimensionality of the action space.

On the other hand, a kinematic motion planner computes a collision-free path from a start joint state q_t to a goal joint state g_t. We denote the motion planner as MP(q_t, g_t) and the collision-free path as $\tau_{0:H} = (q_t, q_{t+1}, \ldots, q_{t+H})$, where H is the number of states in the path and $q_{t+H} = g_t$. The sequence of actions $a_{t:t+H-1}$ that realize the path $\tau_{0:H}$ can be obtained by computing the displacement between consecutive joint states, $\Delta\tau_{0:H} = (\Delta q_t, \ldots, \Delta q_{t+H-1})$.

To efficiently learn a manipulation task in an obstructed environment, we propose motion planner augmented reinforcement learning (MoPA-RL). Our method harnesses a motion planner for controlling a robot toward a faraway goal without colliding with obstacles, while directly executing small actions for sophisticated manipulation. By utilizing MP, the robot can effectively explore the environment avoiding obstacles and passing through narrow passages. For contact-rich tasks, where MP often fails due to an inaccurate contact model, model-free RL can be used instead of the motion planner.



(a) 2D Push (b) Sawyer Push (c) Sawyer Lift (d) Sawyer Assembly

Figure 3: Manipulation tasks in obstructed environments. (a) *2D Push*: The 2D reacher agent has to push the green ball to the goal (black circle). (b) *Sawyer Push*: Sawyer arm should push the red box toward the goal (green circle). (c) *Sawyer Lift*: Sawyer arm takes out the can from the long box. (d) *Sawyer Assembly*: Sawyer arm moves and inserts the table leg into the hole in the table top. The environment is built upon the IKEA furniture assembly environment [19].

As illustrated in Figure 1, our framework consists of two components: an RL policy $\pi_\phi(a|s)$ and a motion planner MP(q, g). In our framework, the motion planner is integrated into the RL policy by enlarging its action space, $\tilde{\mathcal{A}} = [-\Delta q_{\text{MP}}, \Delta q_{\text{MP}}]^d$. The agent directly executes an action if it is in the original action space. If an action is sampled from outside of the original action space, which requires a large movement of the agent, the motion planner is called and computes a path to realize the large joint displacement.

2.2 Action Space Rescaling

The proposed motion planner augmented action space $\tilde{\mathcal{A}} = [-\Delta q_{\text{MP}}, \Delta q_{\text{MP}}]^d$ extends the typical action space for model-free RL, $\mathcal{A} = [-\Delta q_{\text{step}}, \Delta q_{\text{step}}]^d$. An action $\tilde{a}$ from the original action space $\mathcal{A}$ is directly executed with a feedback controller. On the other hand, an action from outside of $\mathcal{A}$ is handled by the motion planner. However, in practice, Δq_{MP} is much larger than Δq_{step}, which results in a drastic difference between the proportions of the action spaces for direct action execution and motion planning. Especially with high-dimensional action spaces, this leads to very low probability $(\Delta q_{\text{step}} / \Delta q_{\text{MP}})^d$ of selecting direct action execution during exploration. Hence, this naive action space partitioning biases using motion planning over direct action execution and leads to failures of learning contact-rich manipulation tasks.

To circumvent this issue, we balance the ratio of sampling actions for direct action execution $a \in \mathcal{A}$ and motion plan actions $\tilde{a} \in \tilde{\mathcal{A}} \setminus \mathcal{A}$ by rescaling the action space. To increase the portion of direct action execution, we apply a piecewise linear function f to the policy output $u \in [-1, 1]^d$. From the policy output u, the action (joint displacement) of the i-th joint can be computed by

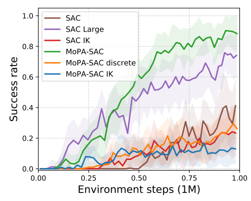
$$a_i = f(u_i) = \begin{cases} \frac{\Delta q_{\text{step}}}{\omega} u_i & |u_i| \leq \omega \\ \text{sign}(u_i) \left[\Delta q_{\text{step}} + (\Delta q_{\text{MP}} - \Delta q_{\text{step}}) \left(\frac{|u_i| - \omega}{1 - \omega} \right) \right] & \text{otherwise} \end{cases}, \tag{1}$$

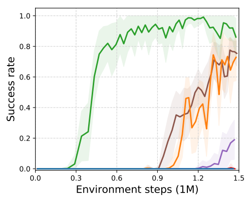
where $\omega \in [0, 1]$ determines the desired ratio between the sizes of the two action spaces.

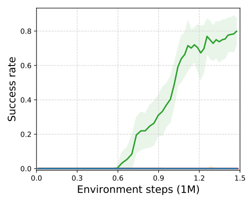
3 Experiments

To verify efficiency and safety of our method, we conduct experiments on the following hard-exploration tasks in obstructed settings: *2D Push*, *Sawyer Push*, *Sawyer Lift*, and *Sawyer Assembly*. We train *2D Push*, *Sawyer Push* and *Sawyer Assembly* using sparse rewards. For *Sawyer Lift*, we use a shaped reward function, similar to Fan et al. [10]. In all tasks, the agent receives a sparse completion reward upon solving the tasks. Further details about the environments and reward functions can be found in the supplementary material.

We compare the performance of our method, MoPA-SAC, against vanilla Soft Actor-Critic (SAC, [3]), SAC with the larger action space $\tilde{\mathcal{A}}$ (SAC Large), and SAC with IK controller (SAC IK). We also test MoPA-SAC with explicitly choosing between MP and RL (MoPA-SAC Discrete) and our method on the Cartesian action space instead of joint pose (MoPA-SAC IK).


(a) 2D Push


(b) Sawyer Push


(c) Sawyer Lift

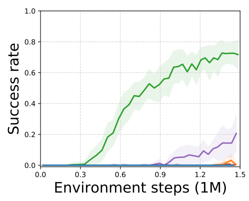

(d) Sawyer Assembly

Figure 4: Success rates of our MoPA-SAC (green) and several baselines averaged over 4 seeds. Our approach can leverage the motion planner to converge with fewer environment steps than the baseline. Both SAC and ours are trained for the same number of environment steps.

We compare the learning performance of all approaches on four tasks in Figure 4. Only our MoPA-SAC approach is able to learn all four tasks, while other methods converge more slowly or struggle to obtain any rewards. While conventional model-free RL agents struggle to learn complex motions from scratch, our approach can leverage the capabilities of the motion planner to successfully learn to produce collision-free movements.

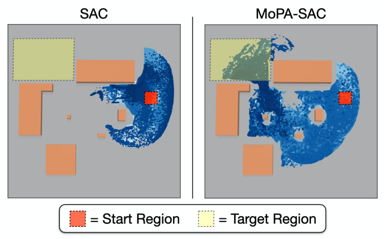


Figure 5: End-effector positions of SAC (**left**) and MoPA-SAC (**right**) after the first 100k training environment steps in *2D Push*. The usage of the motion planner allows the agent to fast exploration.

To further analyze the exploration efficiency, we compare the exploration behavior in the first 100k training steps of our MoPA-RL agent and the SAC agent on the *2D Push* task in Figure 5. The SAC agent initially explores only in close proximity to its starting position as it struggles to find valid trajectories between the obstacles. In contrast, the motion-planner augmented agent explores a wider range of target positions by using the motion planner to find collision-free trajectories to faraway goal states. This allows the agent to more quickly learn the task, especially in the presence of many obstacles.

The ability to execute safe collision-free trajectories, even in environments with many obstacles, is important for the application of RL agents in the real world. We hypothesize that the MoPA-RL agents can leverage the motion planner to learn trajectories that avoid unnecessary collisions. To validate this, we report the average contact force of all robot joints on successful rollouts from the trained policies in Figure 6. The MoPA-RL agents show low average contact forces that are mainly the result of the necessary contacts with the objects that need to be pushed or lifted. Crucially, these agents are able to perform the manipulations *safely* while avoiding collisions with obstacles. In contrast, conventional RL agents are unable to effectively avoid collisions in the obstructed environments, leading to high average contact forces.

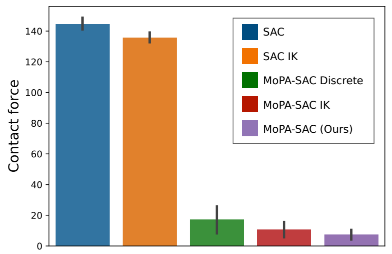


Figure 6: Averaged contact force in an episode over 7 executions in *2D Push*. MoPA-RL naturally learns collision-safe trajectories with motion planning.

4 Conclusion

In this work, we propose a flexible framework that combines the benefits of both motion planning and reinforcement learning for sample-efficient learning of continuous robot control in obstructed environments. Specifically, we augment a model-free RL policy with a sampling-based motion planner with minimal task-specific knowledge, the policy can learn when to use the motion planner and when to take a single-step action directly through reward maximization. The experimental results show that our approach improves the training efficiency over conventional model-free RL baselines, especially in environments that require object manipulations in the presence of many obstacles.

Acknowledgments and Disclosure of Funding

We thank our colleagues from the CLVR lab and RESL for the valuable discussions that considerably assisted the research.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

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

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1856–1865, 2018.

[4] Y. Lee, S.-H. Sun, S. Somasundaram, E. Hu, and J. J. Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2019.

[5] Y. Lee, J. Yang, and J. J. Lim. Learning to coordinate manipulation skills via skill behavior diversification. In *International Conference on Learning Representations*, 2020.

[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.

[7] S. Levine and V. Koltun. Guided policy search. In *International Conference on Machine Learning*, 2013.

[8] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. In *International Symposium on Experimental Robotics*, 2016.

[9] 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*, pages 651–673, 2018.

[10] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, pages 767–782, 2018.

[11] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE International Conference on Robotics and Automation*, 1996.

[12] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, 1998.

[13] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[14] M. Elbanhawi and M. Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2014.

[15] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration for fast path planning. In *IEEE International Conference on Robotics and Automation*, 1994.

[16] M. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Department of Computer Science, Utrecht University, 1992.

[17] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Department, Iowa State University, 1998.

[18] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation*, 2017.

[19] Y. Lee, E. S. Hu, Z. Yang, A. Yin, and J. J. Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019.

[20] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 1582–1591. PMLR, 2018.

[21] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query paoverth planning. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001. IEEE, 2000.

[22] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[23] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26(8):845–863, 2007.

[24] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[25] L. Zhang and D. Manocha. An efficient retraction-based rrt planner. In *IEEE International Conference on Robotics and Automation*, pages 3743–3750, 2008.

[26] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

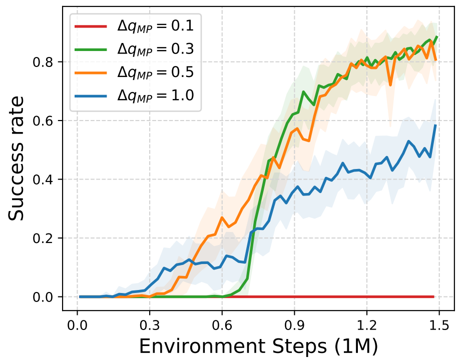
A Ablation Studies

A.1 Action range

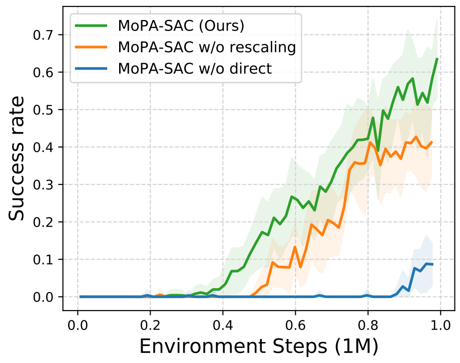
We analyze the influence of the action range Δq_{MP} on task performance in Figure 7a. We find that for too small action ranges the policy cannot efficiently explore the environment and does not learn the task. Yet, for too large action ranges the number of possible actions the agent needs to explore is large, leading to slow convergence. In between, our approach is robust to the choice of action range and able to learn the task efficiently.

A.2 Action rescaling and direct action execution

In Figure 7b we ablate the action space rescaling introduced in Section 2.2. We find that action space rescaling improves learning performance by encouraging balanced exploration of both single-step and motion planner action spaces. More crucial is however our hybrid action space formulation with direct and MP action execution: MoPA-SAC trained *without* direct action execution struggles on contact-rich tasks, since it is challenging to use the motion planner for solving contact-rich object manipulations.



(a) Action range Δq_{MP}



(b) Action space rescaling

Figure 7: (a) Averaged contact force in an episode over 7 executions in *Push 2D*. Leveraging a motion planner, all variants of our method naturally learn collision-safe trajectories. (b) Comparison of our model with different action range values Δq_{MP} on *Sawyer Lift*. (c) Comparison of our model w/ and w/o action rescaling or w/o direct action execution on *Sawyer Lift*.

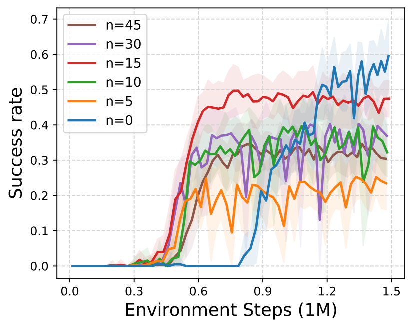
A.3 Reuse of Motion Planning Trajectories

As mentioned in Section D, to improve sample efficiency of motion plan actions, we sample n sub-trajectories of the motion plan trajectory $\tau_{0:H} = (q_t, q_{t+1}, \ldots, q_{t+H})$ and augment the replay buffer with sub-sampled motion plan transitions $(s_{t+a_i}, \Delta\tau_{a_i:b_i}, s_{t+b_i}, \tilde{R}(s_{t+a_i}, \Delta\tau_{a_i:b_i}))$, where $a_i < b_i \in [0, H]$ and $i \in [1, n]$.

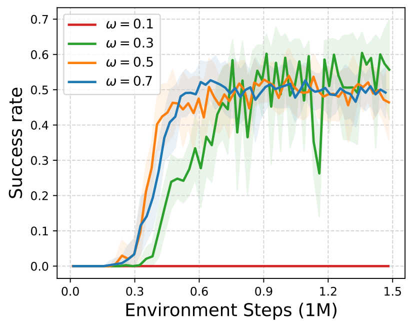
Figure 8a shows the success rates of our model for different n, the number of samples reused per motion plan trajectory. Reusing trajectory of motion planner in this way improves the sample efficiency as the success rate starts increasing earlier than the one without reusing motion plan trajectories ($n = 0$). However, augmenting too many samples ($n = 30, 45$) degrades the performance since it biases the distribution of the replay buffer to motion plan actions and reduces the sample efficiency of the direct action executions, which results in slow learning of contact-rich skills. This biased distribution of transitions leads to convergence towards sub-optimal solutions while the model without bias $n = 0$ eventually finds a better solution.

A.4 Further Study on Action Space Rescaling

In Section 2.2, we proposed action space rescaling to balance the sampling ratio between direct action execution and motion planning. As illustrated in Figure 8b, our method without action space rescaling ($\omega = 0.1$) fails to solve *Sawyer Assembly* while the policy with action space rescaling learns



(a) Number of reused sample n



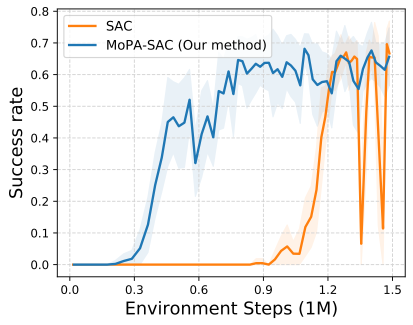
(b) Ratio in action space ω

Figure 8: Learning curves of ablated models on *Sawyer Assembly*. (a) Comparison of our model with different number of samples reused from trajectories of the motion planner. (b) Comparison of our model with different action space rescaling parameter ω.

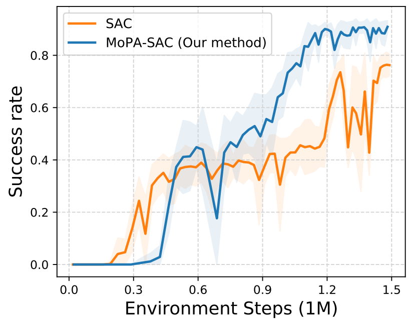
to solve the task. This failure is mainly because direct action execution is crucial for inserting the table leg and the policy without action space rescaling rarely explores the direct action execution space, which makes the agent struggle to solve the task. We also find that the ω value is not sensitive in *Sawyer Assembly*, as different ω values achieve similar success rates.

A.5 Performance in Uncluttered Environments

We further verify whether our method does not degrade the performance of model-free RL in uncluttered environments. Therefore, we remove obstacles, such as a box on a table in *Sawyer Lift* and three other table legs in *Sawyer Assembly*. Figure 9a and Figure 9b show that our method is as sample efficient as the baseline SAC and it is even better in *Sawyer Lift w/o box* because our method does not need to learn how to control an arm for the reaching skill.



(a) Sawyer Lift w/o box

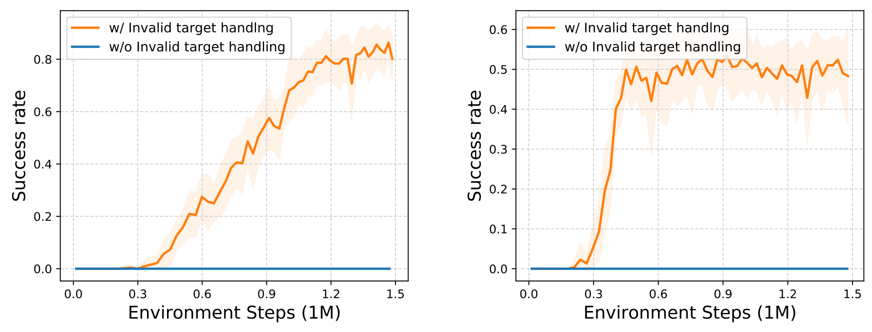


(b) Sawyer Assembly w/o legs

Figure 9: Success rate on (a) *Sawyer Lift w/o box* and (b) *Sawyer Assembly w/o legs*.

A.6 Handling of Invalid Target Joint States for Motion Planning

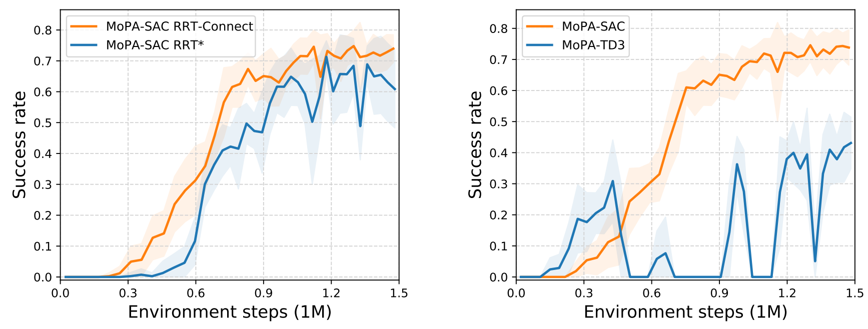
When a predicted target joint state $g_t = q_t + \tilde{a}_t$ for motion planning is in collision with obstacles, instead of penalizing or using the invalid action $\tilde{a}_t$, we search for a valid action by iteratively moving the target joint state towards the current joint state and executing the new valid action, described in Section B. We investigate the importance of handling the invalid actions for motion planning, by comparing to a naive approach for handling invalid actions in which the robot does not execute any action and a transition (s_t, a_t, r_t, s_{t+1}) is added into a replay buffer, where $s_t = s_{t+1}$ and r_t is the reward of being at the current state. Figure 10a and Figure 10b show that MoPA-SAC with naive handling of invalid states cannot learn to solve the tasks, which implies that our proposed handling of invalid target state is very crucial to train MoPA-SAC agents. A reason behind this behavior is that the agent can explore the state space even though the invalid target joint state is given.



(a) Sawyer Lift

(b) Sawyer Assembly

Figure 10: Ablation of effect from invalid target handling on (a) *Sawyer Lift* and (b) *Sawyer Assembly*.



(a) Different motion planner

(b) Different RL algorithm

Figure 11: Learning curves of ablated models on *Sawyer Assembly*. (a) Comparison of our model with different motion planner algorithms. (b) Comparison of our model with different RL algorithms.

A.7 Ablation of Motion Planning algorithm

We also test whether our framework is compatible with different motion planning algorithms. Figure Figure 11a shows the comparison of our method using RRT-connect and RRT* [13], and MoPA-SAC with RRT* learns to solve tasks less efficiently than MoPA-SAC with RRT-connect, since, in our experiments, RRT-Connect found better paths than RRT* within the limited time given to both planners.

A.8 Ablation of Model-free RL algorithm

In order to verify the compatibility of our method with different RL algorithms, we replaced SAC with TD3 [20] and compare those results. As illustrated in Figure 11b, MoPA-TD3 shows unstable training, though the best performing seed can achieve around 1.0 success rate.

B Motion Planner Details

Our method seamlessly integrates model-free RL and motion planning through the augmented action space. Our method is agnostic to the choice of motion planning algorithm. Specifically, we use RRT-Connect [21] from the open motion planning library (OMPL) [22] due to its fast computation time. After the motion planning, a shortcutting algorithm [23] is used to smooth the path. For collision checking, we use the collision checking function provided by the MuJoCo physics engine [24].

The expensive computations performed by the motion planner can be a major bottleneck for RL training. Thus, we design the motion planning procedure with several features to improve training efficiency. First, we reduce the number of costly motion planner executions by using a simpler motion planner that attempts to linearly interpolate between the initial and goal state before executing the sampling-based motion planner. If the interpolated path is collision-free, our method uses this path for execution and skips calling the expensive motion planning algorithm. If the path is not collision-free, then RRT-Connect is used to find a collision-free path amongst obstacles.

Second, the RL policy can predict a goal joint state that is in collision or not reachable by the robot. A simple way to resolve this action is to ignore the current action and sample a new action. However, it slows down training because the policy can repeatedly output invalid actions, especially in an environment with many obstacles. Instead, we find an alternative collision-free goal joint state by iteratively reducing the action magnitude and checking collision, similar to [25]. This strategy prevents the policy from being stuck or wasting samples, which results in improved training efficiency. Finally, we allow the motion planner to derive plans while grasping an object by considering the object as a part of the robot once it holds the object.

C Environment Details

All of our environments are simulated in the MuJoCo physics engine [24]. The positions of the end-effector, object, and goal are defined as p_{eef}, p_{obj}, and p_{goal}, respectively. T is the maximum episode horizon.

Table 1: Environment specific parameters for MoPA-SAC

Environment	Action dimension	Reward scale	Δq_{step}	Δq_{MP}	ω	n	T
2D Push	4	0.2	0.1	1.0	0.7	30	400
Sawyer Push	8	1.0	0.05	0.5	0.7	15	250
Sawyer Lift	8	0.5	0.05	0.5	0.5	15	250
Sawyer Assembly	7	1.0	0.05	0.5	0.5	15	250

C.1 2D Push

A reacher agent with 4 joints needs to first reach an object while avoiding obstacles and then push the object to the goal region.

Success criteria: $||p_{\text{goal}} - p_{\text{obj}}||_2 \leq 0.05$.

Initialization: The goal and box position are randomly sampled from regions shown in Figure 5. Moreover, random noise is added to the agent's initial pose.

Observation: The observation consists of $(\sin\theta, \cos\theta)$ for each joint angle θ, velocity of each joint, the box position $(x_{\text{obj}}, y_{\text{obj}})$, the box velocity, and end-effector position $(x_{\text{eef}}, y_{\text{eef}})$.

Rewards: Instead of defining a dense reward over all states which causes sub-optimal solutions, we define the reward function such that the agent receives a signal only when the end-effector is close to the object (i.e., $||p_{\text{eef}} - p_{\text{obj}}||_2 \leq 0.1$). The reward function consists of rewards for reaching the box and pushing the box to the goal region.

$$\begin{aligned} R_{\text{push}} = \; & 0.1 \cdot \mathbb{1}_{||p_{\text{eef}} - p_{\text{obj}}||_2 \leq 0.1}(1 - \tanh(5||p_{\text{eef}} - p_{\text{obj}}||_2)) \\ & + 0.3 \cdot \mathbb{1}_{||p_{\text{obj}} - p_{\text{goal}}||_2 \leq 0.1}(1 - \tanh(5||p_{\text{obj}} - p_{\text{goal}}||_2)) + 150 \cdot \mathbb{1}_{\text{success}} \end{aligned} \tag{2}$$

C.2 Sawyer Lift

In *Sawyer Lift* the agent has to pick up an object inside a box. To lift the object, the Sawyer arm first needs to get into the box, grasp the object, and lift the object above the box.

Success criteria: The goal criteria is to lift the object above the box height.

Initialization: Random noise sampled from $\mathcal{N}(0, 0.02)$ is added to the initial position of a sawyer arm. The target position is always above the height of the box.

Observation: The observation consists of each joint state $(\sin\theta, \cos\theta)$, velocity of each joint, the goal position, the object position and quaternion, end-effector coordinates, the relative distance between the end-effector and object.

Rewards: This task can be decomposed into three stages; reach, grasp, and lift. For each of the stages, we define the reward function, and the agent receives the maximum reward over three values. The success of grasp is detected when both of two fingers of the end-effector touch the object.

$$\begin{aligned} R_{\text{lift}} = \max \big(& \underbrace{0.1 \cdot (1 - \tanh(10||p_{\text{eef}} - p_{\text{obj}}||_2))}_{\text{reach}}, \underbrace{0.35 \cdot \mathbb{1}_{\text{grasp}}}_{\text{grasp}}, \\ & \underbrace{0.35 \cdot \mathbb{1}_{\text{grasp}} + 0.15 \cdot (1 - \tanh(15 \cdot \max(p^z_{\text{goal}} - p^z_{\text{obj}}, 0)))}_{\text{lift}} \big) \end{aligned} \quad (3)$$

C.3 Sawyer Push

Sawyer Push requires the agent to reach an object in a box and push the object toward a goal region.

Success criteria: $||p_{\text{goal}} - p_{\text{obj}}||_2 \leq 0.05$.

Initialization: The random noise sampled from $\mathcal{N}(0, 0.02)$ is added to the goal position and the initial pose of the Sawyer arm.

Observation: The observation consists of each joint state $(\sin\theta, \cos\theta)$, velocity of each joint, the goal position p_{goal}, the object position and quaternion, end-effector coordinates p_{eef}, the relative distance between the end-effector and object, and the relative distance between the object and target.

Rewards:

$$\begin{aligned} R_{\text{push}} = \ & 0.1 \cdot \mathbb{1}_{||p_{\text{eef}} - p_{\text{obj}}||_2 \leq 0.1} (1 - \tanh(5||p_{\text{eef}} - p_{\text{obj}}||_2)) \\ & + 0.5 \cdot \mathbb{1}_{||p_{\text{obj}} - p_{\text{goal}}||_2 \leq 0.1} (1 - \tanh(5||p_{\text{obj}} - p_{\text{goal}}||_2)) + 150 \cdot \mathbb{1}_{\text{success}} \end{aligned} \quad (4)$$

C.4 Sawyer Assembly

The agent needs to avoid the other table legs while moving the leg in a gripper to a hole. Note that the table leg that the agent manipulates is attached to the gripper; therefore, it does not need to learn how to grasp the leg. In this task, if the robot hits the table legs, the table position moves. This environment is based on the IKEA furniture assembly environment [19].

Success criteria: The goal criteria is when the table leg is pegged into a hole. The goal position is at the bottom of the hole, and its goal criteria is represented by $||p_{\text{goal}} - p_{\text{head}}||_2 \leq 0.05$, where p_{head} is position of head of the pole.

Initialization: Random noise sampled from $\mathcal{N}(0, 0.02)$ is added to the initial position of the Sawyer arm. The target and obstacle position are fixed.

Observation: The observation consists of each joint state $(\sin\theta, \cos\theta)$, velocity of each joint, the hole position p_{goal}, positions of two ends of the leg in hand $p_{\text{head}}, p_{\text{tail}}$, the object quaternion.

Rewards:

$$R_{\text{assembly}} = 0.4 \cdot \mathbb{1}_{||p_{\text{head}} - p_{\text{hole}}||_2 \leq 0.3} (1 - \tanh(15||p_{\text{head}} - p_{\text{hole}}||_2)) + 150 \cdot \mathbb{1}_{\text{success}} \quad (5)$$

D Training Details

We model the policy π_ϕ as a neural network. The policy and critic networks consist of 3 fully connected layers of 256 hidden units with ReLU nonlinearities. The policy outputs the mean and standard deviation of a Gaussian distribution over an action space. To bound actions in $[-1, 1]$, we apply $\tanh$ activation to the policy output. Before executing the action in the environment, we transform the action with the action rescaling function f described in Section 2.2. The policy is trained using a model-free RL method, Soft Actor-Critic [3].

To improve sample efficiency, we randomly sample the intermediate transitions of a path from the motion planner, and store the sampled transition in the replay buffer. By making use of these additional transitions, the agent experience can cover a wider region in the state space during training. For hyperparameters and more details about training, please refer to the supplementary material.

Table 2: SAC Shared Hyperparameter

Parameter	Value
Optimizer	Adam
Learning rate	3e-4
Discount factor (γ)	0.99
Replay buffer size	10^6
Number of hidden layers for all networks	2
Number of hidden units for all networks	256
Minibatch size	256
Nonlinearity	ReLU
Target smoothing coefficient (τ)	0.005
Target update interval	1
Network update per environment step	1
Target entropy	$-\dim(\mathcal{A})$

For reward scale in our baseline, we use 10 for all environment. In our method, each reward can be much larger than the one in baseline, because it is a cumulative reward when the motion planner is called. Therefore, larger reward scale in our method degrades the performance, and using small reward scale $0.1 \sim 0.5$ enables the agent to solve tasks. Moreover, α in SAC, which is a coefficient of entropy, is automatically tuned. To train a policy over discrete actions with SAC, we use Gumbel-Softmax distribution [26] for categorical reparameterization with temperature of 1.0.

D.1 Wall-clock Time

Table 3: Comparison of the wall-clock time ($\sim$ hours)

	Sawyer Push	Sawyer Lift	Sawyer Assembly
MoPA-SAC	15	17	14
SAC	24	24	24

The wall-clock time of our method depends on various factors, such as the computation time of an MP path and the number of policy updates. As Table 3 shows, MoPA-RL learns quicker in wall-clock time compared to SAC for 1.5M environment steps. This is because SAC updates the policy once for every taken action, and our method requires fewer policy actions for completing an episode. As a result, our method performs fewer costly policy updates. Moreover, while a single call to the motion planner can be computationally expensive (0.3 seconds in our case), we need to invoke it less frequently since it produces a multi-step plan (40 steps on average in our experiments). We further increased the efficiency of our method by introducing a simplified interpolation planner.