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

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.

Keywords: Reinforcement Learning, Motion Planning, Robot Manipulation

1 Introduction

In recent years, deep reinforcement learning (RL) has shown promising results in continuous control problems [1, 2, 3, 4, 5]. Driven by rewards, robotic agents can learn tasks such as grasping [6, 7] and peg insertion [8]. 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.

Motion planning (MP) is an alternative for performing robot tasks in complex environments, and has been widely studied in the robotics literature [9, 10, 11, 12]. MP methods, such as RRT [10] and PRM [9], can find a collision-free path between two robot states in an 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 since 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.

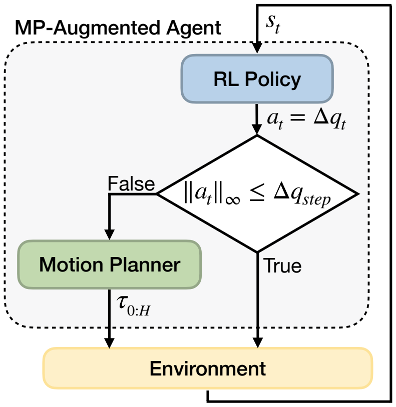


Figure 1: Our framework extends an RL policy with a motion planner. If the predicted action by the policy is above a threshold Δq_{step}, the motion planner is called; otherwise, it is directly executed.

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

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

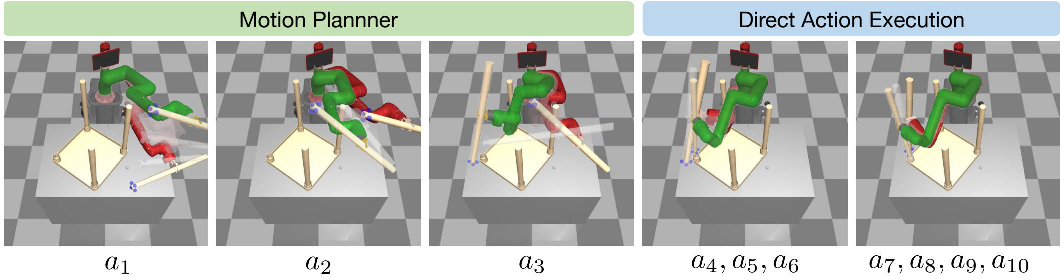


Figure 2: Learning manipulation skills in an obstructed environment is challenging due to frequent collisions and narrow passages amongst obstacles. For example, when a robot moves the table leg to assemble, it is likely to collide with other legs and get stuck between legs. Moreover, once the table leg is moved, the robot is tasked to insert the leg into the hole, which requires contact-rich interactions. 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.

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. Concretely, our approach trains a model-free RL agent that controls a robot by predicting state changes in joint space, where an action with a large joint displacement is realized using a motion planner while a small action is directly executed. By predicting a small action, the agent can perform sophisticated and contact-rich manipulation. On the other hand, a large action allows the agent to efficiently explore an obstructed environment with collision-free paths computed by MP.

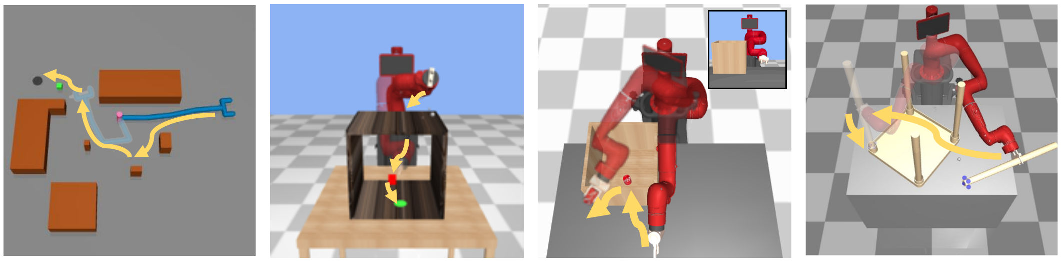
Our approach has three benefits: (1) MoPA-RL can add motion planning capabilities to *any* RL agent with joint space control as it does not require changes to the agent's architecture or training algorithm; (2) MoPA-RL allows an agent to freely switch between MP and direct action execution by controlling the scale of action; and (3) the agent naturally learns trajectories that avoid collisions by leveraging motion planning, allowing for safe execution even in obstructed environments.

The main contribution of this paper is a framework augmenting an RL agent with a motion planner, which enables effective and safe exploration in obstructed environments. In addition, we propose three challenging robotic manipulation tasks with the additional challenges of collision-avoidance and exploration in obstructed environments. We show that the proposed MoPA-RL learns to solve manipulation tasks in these obstructed environments while model-free RL agents suffer from local optima and difficult exploration.

2 Related Work

Controlling robots to solve complex manipulation tasks using deep RL [13, 14, 15, 3] has been an active research area. Specifically, model-free RL [13, 14, 3] has been well studied for robotic manipulation tasks, such as picking and placing objects [16], in-hand dexterous manipulation [17, 18], and peg insertion [19]. To tackle long-horizon tasks, hierarchical RL (HRL) approaches have extended RL algorithms with temporal abstractions, such as options [20, 21], modular networks [22, 4, 5], and goal-conditioned low-level policies [23]. However, these approaches are typically tested on controlled, uncluttered environments and often require a large number of samples.

On the other hand, motion planning [9, 24, 25, 26, 27, 28], a cornerstone of robot motion generation, can generate a collision-free path in cluttered environments using explicit models of the robot and environment. Probabilistic roadmaps (PRMs) [9, 24, 25] and rapidly-exploring random trees (RRTs) [26, 27, 28] are two common sampling-based motion planning techniques. These planning approaches can effectively compute a collision-free path in static environments; however, these methods have difficulties handling dynamic environments and contact-rich interactions, which frequently occur in object manipulation tasks.



(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 box 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.

There are several works that combine motion planning and reinforcement learning to get the advantages of both approaches. A typical approach is to decompose the problem into the parts that can and cannot be solved with MP. Then, RL is used to learn the part that the planner cannot handle [29, 30, 31, 32, 33]. However, this separation often relies on task-specific heuristics that are only valid for a limited task range. MP can be incorporated with RL in the form of a modular framework with a task-specific module switching rule [34] and HRL with pre-specified goals for the motion planner [35]. Xia et al. [33] uses a motion planner as a low-level controller and learn an RL policy in a high-level action (subgoal) space, which limits the capability of learning contact-rich skills. Instead, we propose to *learn* how to balance the use of the motion planner and primitive action (direct action execution) using model-free RL with minimal task-specific knowledge.

3 Motion Planner Augmented Reinforcement Learning

In this paper, we address the problem of solving manipulation tasks in the presence of obstacles. Exploration by deep reinforcement learning (RL) approaches for robotic control mostly relies on small perturbations in the action space. However, RL agents struggle to find a path to the goal in obstructed environments due to collisions and narrow passages. Therefore, we propose to harness motion planning (MP) techniques for RL agents by augmenting the action space with a motion planner. In Section 3.2, we describe our framework, MoPA-RL, in detail. Afterwards, we elaborate on the motion planner implementation and RL agent training.

3.1 Preliminaries

We formulate the problem as a Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$ 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_0, 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 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 [13] and d denotes the dimensionality of the action space.

On the other hand, a kinematic motion planner $\text{MP}(q_t, g_t)$ computes a collision-free path $\tau_{0:H} = (q_t, q_{t+1}, \dots, q_{t+H} = g_t)$ from a start joint state q_t to a goal joint state g_t, where H is the number of states in the path. 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, $\Delta\tau_{0:H} = (\Delta q_t, \dots, \Delta q_{t+H-1})$.

Algorithm 1 Motion Planner Augmented RL (MoPA-RL)

```
Require: Motion planner MP, augmented MDP $\tilde{\mathcal{M}}(\mathcal{S}, \tilde{\mathcal{A}}, \tilde{P}, \tilde{R}, \rho_0, \gamma)$, action limits $\Delta q_{\text{step}}, \Delta q_{\text{MP}}$
    number of reused trajectories $M$
Initialize policy $\pi_\phi$ and replay buffer $\mathcal{D}$
for $i = 1, 2, \dots$ do
    Initialize episode $s_0 \sim \rho_0, \tilde{t} \leftarrow 0, t \leftarrow 0$
    while episode not terminated do
        $\tilde{a}_{\tilde{t}} \sim \pi_\phi(\tilde{a}_{\tilde{t}} | s_t)$
        if $||\tilde{a}_{\tilde{t}}||_\infty > \Delta q_{\text{step}}$ then
            $H_{\tilde{t}}, \tau_{0:H_{\tilde{t}}} \leftarrow \text{MP}(q_t, q_t + \tilde{a}_{\tilde{t}})$    ▷ Motion planner execution
            $s_{t+H_{\tilde{t}}}, \tilde{r}_{\tilde{t}} \leftarrow \tilde{P}(s_t, \Delta\tau_{0:H_{\tilde{t}}}), \tilde{R}(s_t, \Delta\tau_{0:H_{\tilde{t}}})$
            for $j = 1, \dots, M$ do
                Sample intermediate transitions $\tau_{n:m}$ from $\tau_{0:H_{\tilde{t}}}$
                $\tilde{a}, \tilde{r} \leftarrow \Delta\tau_{n:m}, \tilde{R}(s_{t+n}, \Delta\tau_{n:m})$
                $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_{t+n}, \tilde{a}, \tilde{r}, s_{t+m}, m - n)\}$    ▷ Reuse motion plan trajectories
            end for
        else
            $H_{\tilde{t}} \leftarrow 1$
            $s_{t+H_{\tilde{t}}}, \tilde{r}_{\tilde{t}} \leftarrow \tilde{P}(s_t, \tilde{a}_{\tilde{t}}), \tilde{R}(s_t, \tilde{a}_{\tilde{t}})$    ▷ Direct action execution
        end if
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, \tilde{a}_{\tilde{t}}, \tilde{r}_{\tilde{t}}, s_{t+H_{\tilde{t}}}, H_{\tilde{t}})\}$
        $t \leftarrow t + H_{\tilde{t}}, \tilde{t} \leftarrow \tilde{t} + 1$
        Update $\pi_\phi$ using model-free RL
    end while
end for
```

3.2 Motion Planner Augmented Reinforcement Learning

To efficiently learn a manipulation task in an obstructed environment, we propose motion planner augmented RL (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, actions can be directly executed instead of calling a planner.

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. 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.

To integrate the motion planner with an MDP $\mathcal{M}$, we first define an augmented MDP $\tilde{\mathcal{M}}(\mathcal{S}, \tilde{\mathcal{A}}, \tilde{P}, \tilde{R}, \rho_0, \gamma)$, where $\tilde{\mathcal{A}} = [-\Delta q_{\text{MP}}, \Delta q_{\text{MP}}]^d$ is an enlarged action space with $\Delta q_{\text{MP}} > \Delta q_{\text{step}}$, $\tilde{P}(s'|s, \tilde{a})$ denotes the augmented transition function, and $\tilde{R}(s, \tilde{a})$ is the augmented reward function. Since one motion planner call can execute a sequence of actions in the original MDP $\mathcal{M}$, the augmented MDP $\tilde{\mathcal{M}}$ can be considered as a semi-MDP [20], where an option $\tilde{a}$ executes an action sequence $\Delta\tau_{0:H}$ computed by the motion planner. For simplicity of notation, we use $\tilde{a}$ and $\Delta\tau_{0:H}$ interchangeably. The augmented transition function $\tilde{P}(s'|s, \tilde{a}) = \tilde{P}(s'|s, \Delta\tau_{0:H})$ is the state distribution after taking a sequence of actions and the augmented reward function $\tilde{R}(s, \tilde{a}) = \tilde{R}(s, \Delta\tau_{0:H})$ is the discounted sum of rewards along the path.

On the augmented MDP $\tilde{\mathcal{M}}$, the policy $\pi_\phi(\tilde{a}|s)$ chooses an action $\tilde{a}$, which represents a change in the joint state Δq. The decision whether to call the motion planner or directly execute the predicted action is based on its maximum magnitude $||\tilde{a}||_\infty$, i.e., the maximum size of the predicted displacement, as illustrated in Figure 1. If the joint displacement is larger than an action threshold Δq_{step} for any joint (i.e., $||\tilde{a}||_\infty > \Delta q_{\text{step}}$), which is likely to lead to collisions, the motion planner is used to compute a collision-free path $\tau_{0:H}$ towards the goal $g = q + \tilde{a}$. To follow the path, the agent executes the action sequence $\Delta\tau_{0:H}$ over H time steps. Otherwise, i.e., $||\tilde{a}||_\infty \leq \Delta q_{\text{step}}$, the action is directly executed using a feedback controller for a single time step, as is common practice in model-free RL. This procedure is repeated until the episode is terminated. Then, the policy π_ϕ is trained to maximize the expected returns $\mathbb{E}_{\pi_\phi}\left[\sum_{\tilde{t}=0}^{\tilde{T}} \gamma^t \tilde{R}(s_{\tilde{t}}, \tilde{a}_{\tilde{t}})\right]$, where $\tilde{T}$ is the episode horizon on $\tilde{\mathcal{M}}$ and $t = \sum_{i=0}^{\tilde{t}-1} H_i$ is the number of primitive actions executed before time step $\tilde{t}$. The complete RL training with the motion-planner augmented agent is described in Algorithm 1.

The proposed method has three advantages. First, our method gives the policy the freedom to choose whether to call the motion planner or directly execute actions by predicting large or small actions, respectively. Second, the agent naturally produces trajectories that avoid collisions by leveraging MP, allowing for safe policy execution even in obstructed environments. The third advantage is that MoPA-RL can add motion planning capabilities to *any* RL algorithm with joint space control as it does not require changes to the agent's architecture or training procedure.

3.3 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.

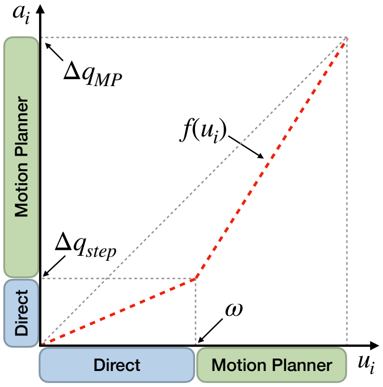


Figure 4: To balance the chance of choosing direct action execution and motion planning during exploration, we increase the action space for direct action execution $\mathcal{A}$.

To circumvent this issue, we balance the ratio of sampling actions for direct action execution $\tilde{a} \in \mathcal{A}$ and motion plan actions $\tilde{a} \in \tilde{\mathcal{A}} \setminus \mathcal{A}$ by rescaling the action space. Figure 4 illustrates the distribution of direct action and motion plan action in $\tilde{\mathcal{A}}$ (y-axis) and the desired distribution (x-axis). To increase the portion of direct action execution, we apply a piecewise linear function f to the policy output $u \in [-1, 1]^d$ and get joint displacements Δq, as shown by the red line in Figure 4. From the policy output u, the action (joint displacement) of the i-th joint can be computed by

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

where $\omega \in [0, 1]$ determines the desired ratio between the sizes of the two action spaces and sign$(\cdot)$ is the sign function.

3.4 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 the policy output u in $[-1, 1]$, we apply $\tanh$ activation to the policy output. Before executing the action in the environment, we transform the policy output with the action rescaling function $f(u)$ described in Equation 1. The policy is trained using a model-free RL method, Soft Actor-Critic [3].

To improve sample efficiency, we randomly sample M intermediate transitions of a path from the motion planner, and store the sampled transitions 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 (see Section A.1). For hyperparameters and more details about training, please refer to Section C.

3.5 Motion Planner Implementation

Our method seamlessly integrates model-free RL and MP through the augmented action space. Our method is agnostic to the choice of MP algorithm. Specifically, we use RRT-Connect [36] from the open motion planning library (OMPL) [37] due to its fast computation time. After the motion planning, the resulting path is smoothed using a shortcutting algorithm [38]. For collision checking, we use the collision checking function provided by the MuJoCo physics engine [39].

The expensive computation performed by the motion planner can be a major bottleneck for training. Thus, we design an efficient MP procedure with several features. First, we reduce the number of costly MP executions by using a simpler motion planner that attempts to linearly interpolate between the initial and goal states instead of the sampling-based motion planner. If the interpolated path is collision-free, our method uses this path for execution and skips calling the expensive MP algorithm. If the path has collision, then RRT-Connect is used to find a collision-free path amongst obstacles.

Moreover, the RL policy can predict a goal joint state that is in collision or not reachable. A simple way to resolve it 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. Thus, our method finds an alternative collision-free goal joint state by iteratively reducing the action magnitude and checking collision, similar to Zhang and Manocha [40]. 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.

4 Experiments

We design our experimental evaluation to answer the following questions: (1) Can MoPA-RL solve complex manipulation tasks in obstructed environments more efficiently than conventional RL algorithms? (2) Is MoPA-RL better able to explore the environment? (3) Does MoPA-RL learn policies that are safer to execute?

4.1 Environments

To answer these questions, we conduct experiments on the following hard-exploration tasks in obstructed settings, simulated using the MuJoCo physics engine [39] (see Figure 3 for visualizations):

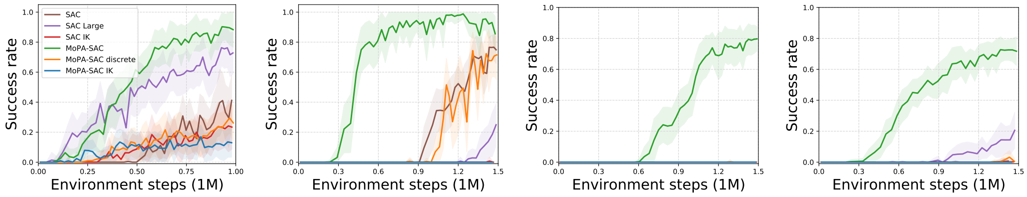
- **2D Push:** A 4-joint 2D reacher needs to push an object into a goal location in the presence of multiple obstacles.
- **Sawyer Push:** A Rethink Sawyer robot arm with 7 DoF needs to push an object into a goal position, both of which are inside a box.
- **Sawyer Lift:** The Sawyer robot arm needs to grasp and lift a block out of a deep box.
- **Sawyer Assembly:** The Sawyer arm needs to move a leg attached to its gripper towards a mounting location while avoiding other legs, and assemble the table by inserting the leg into the hole. The environment is built upon the IKEA furniture assembly environment [41].

We train *2D Push*, *Sawyer Push* and *Sawyer Assembly* using sparse rewards: when close to the object or goal the agent receives a reward proportional to the distance between end-effector and object or object and goal; otherwise there is no reward signal. For *Sawyer Lift*, we use a shaped reward function, similar to Fan et al. [8]. In all tasks, the agent receives a sparse completion reward upon solving the tasks. Further details about the environments can be found in Section B.

4.2 Baselines

We compare the performance of our method with the following approaches:

- **SAC:** A policy trained to predict displacements in the robot's joint angles using Soft Actor-Critic (SAC, [3]), a state-of-the-art model-free RL algorithm.



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

Figure 5: Success rates of our MoPA-SAC (green) and baselines averaged over 4 seeds. All methods are trained for the same number of environment steps. MoPA-SAC can solve all four tasks by leveraging the motion planner and learn faster than the baselines.

- **SAC Large:** A variant of SAC predicts joint displacements in the extended action space $\tilde{\mathcal{A}}$. To realize a large joint displacement, a large action $\tilde{a}$ is linearly interpolated into a sequence of small actions $\{a_1, \ldots a_m\}$ such that $\tilde{a} = \sum_{i=1}^{m} a_i$ and $||a_i||_\infty \leq \Delta q_{\text{step}}$. This baseline tests the importance of collision-free MP for a large action space.
- **SAC IK:** A policy trained with SAC which outputs displacement in Cartesian space instead of the joint space. For 3D manipulation tasks, the policy also outputs the displacement in end-effector orientation as a quaternion. Given the policy output, a target *joint* state is computed using inverse kinematics and the joint displacement is applied to the robot.
- **MoPA-SAC (Ours):** Our method predicts a joint displacement in the extended action space.
- **MoPA-SAC Discrete:** Our method with an additional discrete output that explicitly chooses between MP and RL, which replaces the need for a threshold ω.
- **MoPA-SAC IK:** Our method with end-effector space control instead of joint space displacements. Again, inverse kinematics is used to obtain a target joint state, which is either directly executed or planned towards with the motion planner.

4.3 Efficient RL with Motion Planner Augmented Action Spaces

We compare the learning performance of all approaches on four tasks in Figure 5. Only our MoPA-SAC is able to learn all four tasks, while other methods converge more slowly or struggle to obtain any rewards. The difference is especially large in the *Sawyer Lift* and *Sawyer Assembly* tasks. This is because precise movements are required to maneuver the robot arm to reach inside the box that surrounds the objects or avoid the other table legs while moving the leg in the gripper to the hole. While conventional model-free RL agents struggle to learn such complex motions from scratch, our approach can leverage the capabilities of the motion planner to successfully learn to produce collision-free movements.

To further analyze *why* augmenting RL agents with MP capability improves learning performance, we compare the exploration behavior in the first 100k training steps of our MoPA-SAC agent and the conventional SAC agent on the *2D Push* task in Figure 6. 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 states by using MP to find collision-free trajectories to faraway goal states. This allows the agent to quickly learn the task, especially in the presence of many obstacles. Efficient exploration is even more challenging in the obstructed 3D environments. Therefore, only the MoPA-SAC agent that leverages the motion planner for efficient exploration is able to learn the manipulation tasks.

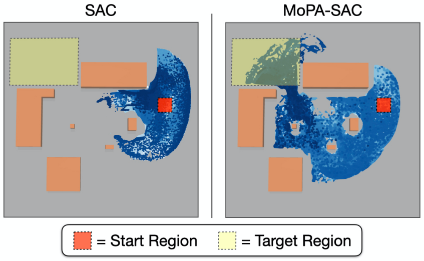
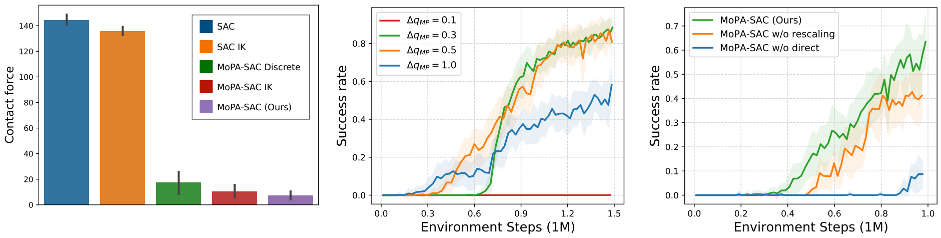


Figure 6: End-effector positions of SAC (**left**) and MoPA-SAC (**right**) after the first 100k training environment steps in *2D Push* are plotted in blue dots. The use of motion planning allows the agent to explore the environment more widely early on in training.

The comparison between different action spaces for our method in Figure 5 shows that directly predicting joint angles and using the motion planner based on action magnitude leads to the best



(a) Contact force

(b) Action range Δq_{MP}

(c) Action space rescaling

Figure 7: (a) Averaged contact force in an episode over 7 executions in *2D Push*. 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*.

learning performance (*MoPA-SAC (Ours)*). In contrast, computing the target joint angles for the motion planner using inverse kinematics (*MoPA-SAC IK*) often produces configurations that are in collision with the environment, especially when manipulations need to be performed in narrow spaces. *MoPA-SAC Discrete* needs to jointly learn how and *when* to use MP by predicting a discrete switching variable. We find that this approach rarely uses MP, leading to worse performance.

4.4 Safe Policy Execution

The ability to execute safe collision-free trajectories in obstructed environments is important for the application of RL in the real world. We hypothesize that the MoPA-RL agents can leverage MP 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 7a. 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.

4.5 Ablation Studies

Action range: We analyze the influence of the action range Δq_{MP} on task performance in Figure 7b. 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.

Action rescaling and direct action execution: In Figure 7c we ablate the action space rescaling introduced in Section 3.3. 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.

5 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 with a sampling-based motion planner with minimal task-specific knowledge. The RL 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 RL methods, especially on manipulation tasks in the presence of many obstacles. These results are promising and motivate future work on using more advanced motion planning techniques in the action space of reinforcement learning. Another interesting direction is the transfer of our framework to real robot systems.

Acknowledgments

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*, 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, 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.

[7] 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.

[8] 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.

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

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

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

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

[13] 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.

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

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

[16] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and N. Heess. Reinforcement and imitation learning for diverse visuomotor skills. In *Robotics: Science and Systems*, 2018.

[17] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[18] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2018.

[19] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *IEEE International Conference on Robotics and Automation*, pages 8973–8979, 2019.

[20] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[21] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Association for the Advancement of Artificial Intelligence*, 2017.

[22] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175, 2017.

[23] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *Neural Information Processing Systems*, pages 3303–3313, 2018.

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

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

[26] 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.

[27] S. M. LaValle, J. J. Kuffner, B. Donald, et al. *Algorithmic and computational robotics: new directions*, chapter Rapidly-exploring random trees: Progress and prospects, pages 293–308. AK Peters, 2001.

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

[29] P. Englert and M. Toussaint. Learning manipulation skills from a single demonstration. *International Journal of Robotics Research*, 37(1):137–154, 2018.

[30] R. Vuga, B. Nemec, and A. Ude. Enhanced Policy Adaptation Through Directed Explorative Learning. *International Journal of Humanoid Robotics*, 12(3), 2015.

[31] S. P. Singh, A. G. Barto, R. Grupen, and C. Connolly. Robust reinforcement learning in motion planning. In *Advances in Neural Information Processing Systems*, pages 655–662, 1994.

[32] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[33] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese. Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation. *arXiv preprint arXiv:2008.07792*, 2020.

[34] M. A. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, F. Ramos, and D. Fox. Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning. *IEEE International Conference on Robotics and Automation*, 2020.

[35] D. Angelov, Y. Hristov, M. Burke, and S. Ramamoorthy. Composing diverse policies for temporally extended tasks. *IEEE Robotics and Automation Letters*, 5(2):2658–2665, 2020.

[36] 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.

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

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

[39] 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.

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

[41] 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.

[42] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

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

A Additional Ablation Studies

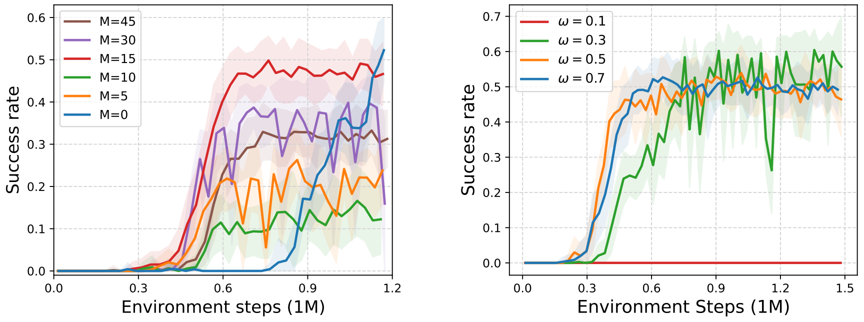
We provide further analysis: (1) the effect of reusing motion planning trajectories to augment training data (Section A.1); (2) the ablation on the action space rescaling (Section A.2); (3) the performance of our approach in uncluttered environments compared to baselines (Section A.3); (4) the ablation of invalid target joint state handling (Section A.4); (5) the ablation of motion planner algorithms (Section A.5); and (6) the ablation of RL algorithms (Section A.6).

A.1 Reuse of Motion Plan Trajectories

As mentioned in Section 3.4, to improve sample efficiency of motion plan actions, we sample M intermediate trajectories of the motion plan trajectory $\tau_{0:H} = (q_t, q_{t+1}, \dots, q_{t+H})$ 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 $i \in [1, M]$ (see Algorithm 1). Figure 8a shows the success rates of our model with different M, the number of sub-sampled motion plan transitions per motion plan action. Reusing trajectory of MP in this way improves the sample efficiency as the success rate starts increasing earlier than the one without reusing motion plan trajectories ($M = 0$). However, augmenting too many samples ($M = 30, 45$) 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 $M = 0$ eventually finds a better solution.

A.2 Further Study on Action Space Rescaling

In Section 3.3, we propose 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 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 contact-rich task. We also find that the ω value is not sensitive in *Sawyer Assembly*, as different ω values achieve similar success rates.



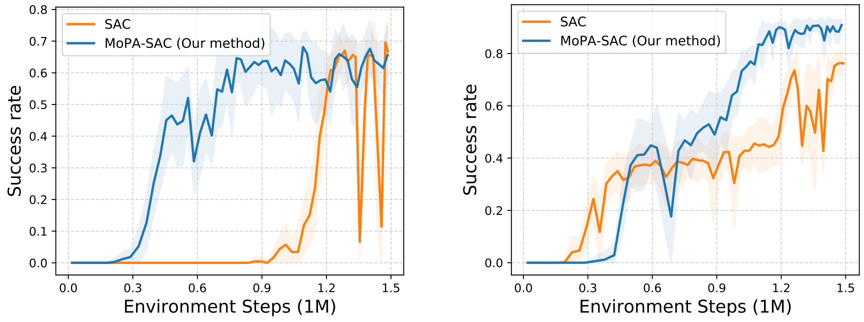
(a) Number of sub-sampled motion plan transitions M

(b) Action space rescaling ratio ω

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

A.3 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

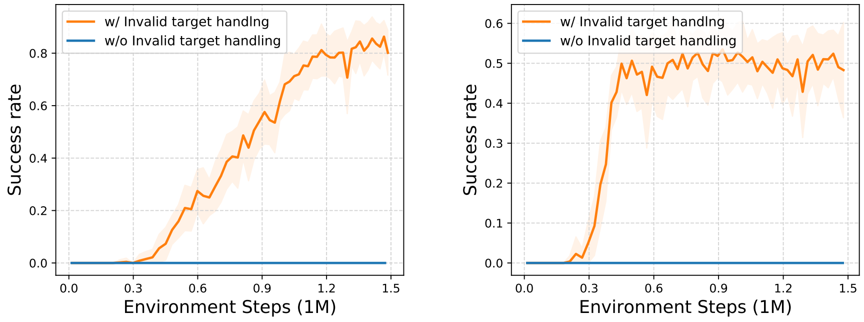
(b) Sawyer Assembly w/o legs

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

A.4 Handling of Invalid Target Joint States for Motion Planning

When a predicted target joint state $g = q + \tilde{a}$ for motion planning is in collision with obstacles, instead of penalizing or using the invalid action $\tilde{a}$, we search for a valid action by iteratively moving the target joint state towards the current joint state and executing the new valid action, as described in Section 3.5.

We investigate the importance of handling the invalid actions for motion planning by comparing to a naive approach for handling invalid actions that 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

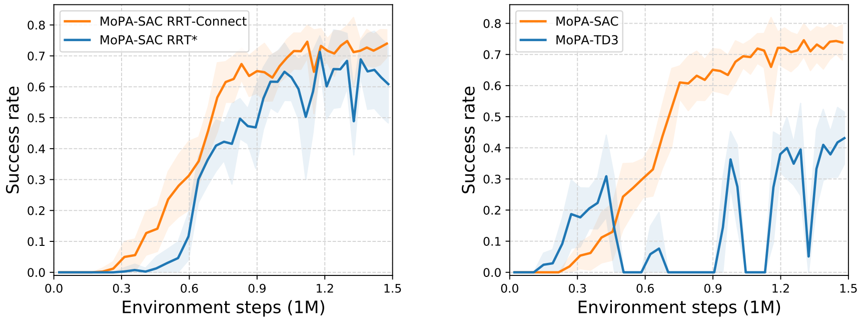
Figure 10: Ablation of invalid target handling on (a) *Sawyer Lift* and (b) *Sawyer Assembly*.

A.5 Ablation of Motion Planning Algorithms

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

A.6 Ablation of Model-free RL Algorithms

To verify the compatibility of our method with different RL algorithms, we replaced SAC with TD3 [42] and compare the learning performance. As illustrated in Figure 11b, MoPA-TD3 shows unstable training, though the best performing seed can achieve around 1.0 success rate.



(a) Different motion planners

(b) Different RL algorithms

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.

B Environment Details

All of our environments are simulated in the MuJoCo physics engine [39]. 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	M	T
2D Push	4	0.2	0.1	1.0	0.7	30	400
Sawyer Push	7	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	0.7	15	250

B.1 2D Push

A 2D-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 x and y position of goal and box are randomly sampled from $\mathcal{U}(-0.35, -0.24)$ and $\mathcal{U}(0.13, 0.2)$ respectively. Moreover, the random noise sampled from $\mathcal{U}(-0.02, 0.02)$ is added to the agent's initial pose.

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

Rewards: Instead of defining a dense reward over all states which can cause 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.

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

B.2 Sawyer Push

The *Sawyer Push* task 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 $(\sin\theta, \cos\theta)$, angular joint velocity, the goal position p_{goal}, the object position and quaternion, end-effector coordinates p_{eef}, the distance between the end-effector and object, and the distance between the object and target.

Rewards:

$$\begin{aligned} R_{\text{push}} = \; & 0.1 \cdot \mathbb{1}_{||p_{\text{eef}} - p_{\text{obj}}||_2 \leq 0.1}(1 - \tanh(5 \cdot ||p_{\text{eef}} - p_{\text{obj}}||_2)) \\ & + 0.3 \cdot \mathbb{1}_{||p_{\text{obj}} - p_{\text{goal}}||_2 \leq 0.1}(1 - \tanh(5 \cdot ||p_{\text{obj}} - p_{\text{goal}}||_2)) + 150 \cdot \mathbb{1}_{\text{success}} \end{aligned} \tag{3}$$

B.3 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 $(\sin\theta, \cos\theta)$, angular joint velocity, the goal position, the object position and quaternion, end-effector coordinates, the 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 the two fingers touch the object.

$$\begin{aligned} R_{\text{lift}} = & \max\big(\underbrace{0.1 \cdot (1 - \tanh(10 \cdot ||p_{\text{eef}} - p_{\text{obj}}||_2))}_{\text{reach}}, \underbrace{0.35 \cdot \mathbb{1}_{\text{grasp}}}_{\text{grasp}}, \\ & \underbrace{0.35 \cdot \mathbb{1}_{\text{grasp}} + 0.15 \cdot (1 - \tanh(15 \cdot \max(p^z_{\text{goal}} - p^z_{\text{obj}}, 0)))}_{\text{lift}}\big) + 150 \cdot \mathbb{1}_{\text{success}} \end{aligned} \tag{4}$$

B.4 Sawyer Assembly

The *Sawyer Assembly* task is to assemble the last table leg to the table top where other three legs are already assembled. The Sawyer arm needs to avoid the other table legs while moving the leg in its gripper to the hole since collision with other table legs can move the table. 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.

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

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

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

Rewards:

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

C Training Details

For reward scale in our baseline, we use 10 for all environments. In our method, each reward can be much larger than the one in baseline because it uses a cumulative reward along a motion plan trajectory 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 [43] for categorical reparameterization with temperature of 1.0.

Table 2: SAC 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	$-\dim(\mathcal{A})$

C.1 Wall-clock Time

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.

Table 3: Comparison of the wall-clock training time in hours

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