

A Manipulation Planner implementation details

Implementation details and parameters for the manipulation planner are given in this section.

A.1 Rewards

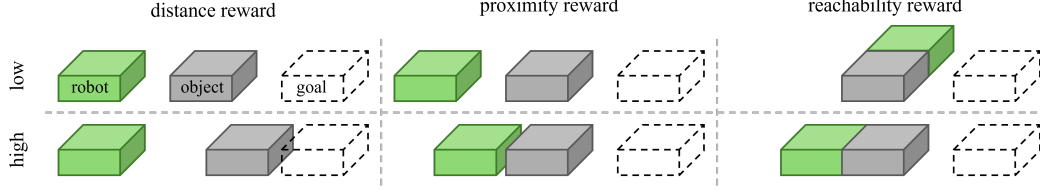


Figure 5: Visualization of low and high reward states for the three reward types of the manipulation planner.

Reachability reward clipping The reachability measure m defined in (8) is clipped to a lower bound value $m_l \in \mathbb{R}_+$ in our implementation to prevent overly large rewards,

$$\bar{m} = \max(m, m_{\min}). \quad (12)$$

The reward calculation (9) is updated to

$$r_m(\bar{m}) = -Q_m \log \left(\frac{\bar{m}}{m_{\min}} \right). \quad (13)$$

A.2 Node selection and extension horizon

Pareto distribution The continuous truncated Pareto distribution is defined as

$$p(x; n_n, \beta) = \frac{\beta x^{-(\beta+1)}}{1 - n_n^{-\beta}}, \quad (14)$$

for any $x \in [1, n_n]$ and β determines the width of the distribution. The probability for sampling a specific node index $i_n = i$ is

$$P(i_n = i; n_n, \beta) = \int_i^{i+1} p(x; n_n, \beta) dx = \frac{i^{-\beta} - (i+1)^{-\beta}}{1 - n_n^{-\beta}}. \quad (15)$$

Varying search parameters Instead of using fixed values for the Pareto distribution parameter β (greediness of the node selection) and the extension horizon (related to the task complexity), these values can be changed during the search. We update these values depending on whether the planner progresses in the search or not. The greediness of the node selection should be reduced when stuck in a local minima, and the extension horizon can be interpreted as a measure of task complexity. An intuitive example of the meaning of the extension horizon is the Rubik's cube, where one must execute a sequence of actions to improve the cube's configuration. Some states in this sequence are worse than the start state but are necessary to improve the overall configuration.

Algorithm 2 Node Extension Parameter Update

```

1: ...
2:  $i_n, n_e = \text{node\_selection}(\beta, n_e)$   $\triangleright$  Alg. 1, ln. 6
3: better_node = false
4: for  $i_e$  in  $1 : n_e$  do  $\triangleright$  Alg. 1, ln. 7
5:    $a_{n_n+1} = \text{action\_sampling}(p_a)$ 
6:    $s_{n_n+1} = \text{extension}(s_{n_n}, a_{n_n+1})$ 
7:   if is_best_node( $s_{n_n+1}$ ) then
8:      $\beta = \beta_u$ 
9:      $n_e = 0.95n_e + 0.05(i_e + 1)$ 
10:    better_node = true
11:    $n_n = n_n + 1$ 
12:    $i_n = n_n$ 
13: if better_node = false then
14:    $\beta = \max(0.99\beta, \beta_{\min})$ 
15:    $n_e = \min(0.95n_e + 0.05(n_e + 1), n_{e,u})$ 
16:    $n_n = n_n + 1$   $\triangleright$  Alg. 1, ln. 10
17: ...

```

We bound $\beta \in [\beta_l, \beta_u]$ with lower and upper bound $\beta_l, \beta_u \in \mathbb{R}_+$, respectively, and we bound $n_e \in [1, n_{e,u}]$ with upper bound $n_{e,u} \in \mathbb{N}_+$. In the inner-most for-loop of Alg. 1 (ln. 7), we check if the search progresses, i.e., if a new node with the overall highest reward is found. In this case, we update the parameters so that the node selection is greedier and the extension horizon approaches the number of extensions required for search progress. If no new best node is found, we update the parameters so that the node selection is less greedy and the extension horizon approaches the upper horizon bound. These updates to Alg. 1 are stated in Alg. 2.

A.3 Actions

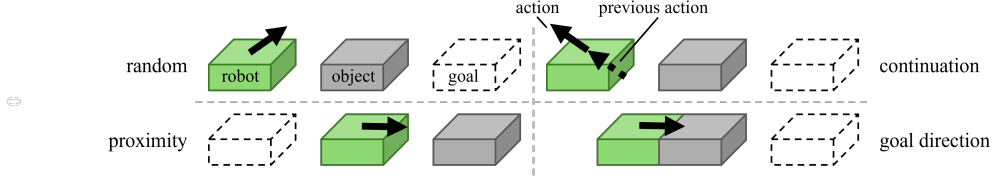


Figure 6: Visualization of the four action types of the manipulation planner.

Varying action step size It can be helpful to have actions with different time step sizes to allow both short, precise motions and long, far-reaching motions. We randomly sample the action step size as an integer multiple of the base step size Δt_a , i.e.,

$$\overline{\Delta t}_a = k \Delta t_a, \quad \text{with } k \sim \mathcal{U}\{1, k_u\}, \quad (16)$$

where $k_u \in \mathbb{N}$ is the upper bound for the step size scaling. After the planner’s search is finished, we split all actions to the base step size to obtain one common action step size for learning.

Closed-form solution for goal-directed action The closed-form solution for the optimal control problem (11) is

$$\Delta a_0 = - \left(B^\top Q B + R \right)^{-1} \left(B^\top Q (f_0^* - s_g) + R a_0^* \right). \quad (17)$$

A.4 Tree extension

Dynamics rollout The actions of the planner are position commands tracked by a low-level PD controller $u_t = -K_p(q_{r,t} - a_t) - K_d \dot{q}_{r,t}$ with controller gains K_p and K_d . A low-level control step size $\Delta t_c \leq \Delta t_a$ smaller than the action step size is used to generate a position reference

$$a_{\text{ref},t} = (q_{r,j_n} + a_{j_n}) \frac{\Delta t_a - t \Delta t_c}{\Delta t} + (q_{r,i_n} + a_{i_n}) \frac{t \Delta t_c}{\Delta t}. \quad (18)$$

The reference is a linear interpolation between the last commanded position for the parent node, $q_{r,j_n} + a_{j_n}$, and the last actual position plus the newly computed relative action, $q_{r,i_n} + a_{i_n}$, which creates a continuous control input. The same control scheme is used for actions created by the reinforcement learning policy.

A.5 Parameters

The planner’s parameters that are not task-specific are listed in Table 2.

| Parameter | Value | Parameter | Value |
|------------------------------------|-------|--|-------|
| goal bias b_g | 0.0 | Pareto parameter upper bound β_u | 1.2 |
| number of goals n_g | 30 | Pareto parameter lower bound β_l | 0.2 |
| number of extensions n_e | 30 | extension horizon bound $n_{e,u}$ | 10 |
| reachability reward bound m_l | 0.001 | base action time step Δt_a | 0.4s |
| action time step scale bound k_u | 3 | | |

Table 2: Planner parameters.

B Reinforcement learning implementation details

Implementation details and parameters for reinforcement learning are given in this section.

B.1 Pure DDPG

Our reinforcement learning setup largely follows the algorithm outlined in [50]. The policy network consists of four hidden layers with 256 neurons each. We use ReLU and tanh output activation functions to ensure the actions lie within $[-1, 1]$. The value function network also uses four hidden layers with 256 nodes each and ReLU activations but no output activation. We clip the value function’s objective to the minimum value possible for the rollout horizon to avoid the unbounded divergence of value estimates. To improve stability during training, we employ target networks with Polyak averaging updates for both the policy and the value function. A normalizer estimates the mean and standard deviation for each input and normalizes all input values to zero mean and unit variance. The exploration noise consists of actions chosen uniformly at random with a probability of η , and additive Gaussian noise for the actions chosen by the policy. A complete list of hyperparameters is given in Table 3 and the whole algorithm is stated in Alg. 3. We stop the training if the policy achieves a success rate of 1.0 or if the maximum number of epochs has been reached.

| Hyperparameter | Value |
|--|---------------|
| policy layers | 4 |
| value function layers | 4 |
| neurons per layer | 256 |
| policy learning rate | 0.001 |
| value function learning rate | 0.001 |
| Polyak averaging factor τ | 0.05 |
| random action chance η | 0.3 |
| discount factor γ | 0.98 |
| epochs n_{epochs} | task specific |
| cycles : n_{cycles} | 50 |
| rollouts n_{rollouts} | 2 |
| rollout horizon n_{steps} | task specific |
| training episodes n_{episode} | 40 |
| training batch size n_{batch} | 256 |
| replay buffer size | 10000N |
| evaluation runs per epoch | 10 |

Table 3: Training hyperparameters.

Algorithm 3 DDPG Algorithm

```

1: Initialize policy  $\pi_\theta$  and value function  $Q$  with random weights
2: Initialize targets  $Q'$  and  $\pi'$  with weights  $\phi' = \phi, \theta' = \theta$ 
3: Initialize replay buffer  $B$ 
4: for epoch in  $1 : n_{\text{epochs}}$  do
5:   for cycle in  $1 : n_{\text{cycles}}$  do
6:     for rollout in  $1 : n_{\text{rollouts}}$  do
7:       Sample start state  $s_0$  and goal state  $s_g$ 
8:       for  $t$  in  $1 : n_{\text{steps}}$  do ▷ policy rollout
9:         if  $\text{rand}() \leq \eta$  then
10:          Sample uniformly random action  $a_t$ 
11:         else
12:          Compute action  $a_t = \pi_\theta(s, s_g) + \mathcal{N}_t$ , with exploration noise  $\mathcal{N}_t$ 
13:          Take action  $a_t$  and obtain new state  $s_{t+1}$ 
14:          Store transition  $(s_t, a_t, s_{g,t}, r_t, s_{t+1})$  in  $B$ 
15:        Update normalizer with new samples ▷ normalizer update
16:      for episode in  $1 : n_{\text{episode}}$  do ▷ network training
17:        Sample minibatch of  $n_{\text{batch}}$  transitions  $(s_t, a_t, s_{g,t}, r_t, s_{t+1})$  from  $B$ 
18:        Compute value target  $y_i = r_i + \gamma Q'_{\phi'}(s_{i+1}, \pi'_{\theta'}(s_{i+1}, s_{g,i+1}), s_{g,i+1})$ 
19:        Compute value function gradient  $g_Q$  for  $l_Q = \frac{1}{n_{\text{batch}}} \sum_i (y_i - Q_\theta(s_i, a_i, s_{g,i}))^2$ 
20:        Compute policy gradient  $g_\pi$  for  $l_\pi = \frac{1}{n_{\text{batch}}} \sum_i Q_\theta(s_i, \pi_\phi(s_i, s_{g,i}), s_{g,i})$ 
21:        Apply the update to  $\pi_\phi$  and  $Q_\theta$ 
22:      Update the target networks:  $\phi' = \tau\phi + (1 - \tau)\phi', \theta' = \tau\theta + (1 - \tau)\theta'$ 
23:      Evaluate policy and stop if success rate = 1.0

```

B.2 Demonstrations in replay buffer

Instead of generating all data with an online policy rollout, it can also be taken from the planner’s search tree with a bias $b_p \in [0, 1]$. Accordingly, the rollout loop of Alg. 3 (ln. 6) is modified to use both the policy and the planner to generate trajectories, as stated in Alg. 4.

Trajectory lengths from the planner vary and can be shorter or longer than the rollout horizon n_{steps} . If the trajectory is too long, the beginning is cut so that only the last n_{steps} transitions remain, since these transitions are most likely to contain reward signals. If the trajectory is too short, the planner’s root state is repeated at the beginning until the trajectory length reaches n_{steps} . The action commands for the padded start states are the initial joint states of the robot.

B.3 Pre-training

A policy network $\pi_{\theta_{\text{IL}}}$ of the same size as π_{θ} can be trained with imitation learning from planner’s demonstrations. The policy is trained with input (s, s_g) and output (a) from the demonstrations. Similarly, the value function is trained, with input (s, s_g, a) and output (v) , where the discounted value v for an input tuple can be calculated trivially from the rewards (1) for all states in a demonstration trajectory. Algorithm 5 states the pre-training procedure. Note that the policy and the value function can be pre-trained separately.

The pre-trained policy and value function are used as described in [25]. In summary, during the policy rollout and network training phase, the better one of the two policies as evaluated by the value function is used. Note that we use the same value function for pre-training and online training.

C Experiment details

The evaluation systems are described in this section. Afterward, several ablations and comparisons for the manipulation planner and reinforcement learning are provided.

C.1 Evaluation systems

Table 4 provides the task-specific parameters for the evaluation systems. Note that we only list the distance reward scaling $Q_{d,o,q}$ for the object’s position states since all object velocity states have a scaling $Q_{d,o,q} = 0.1I$ and the robot states reward scaling is zero: $Q_{d,r} = 0I$. A description of each system is provided below.

Box push The box push task is a one-dimensional pushing task where one actuated pusher box (robot) must push a second unactuated box (object) to a goal position on a line. Due to its simplicity,

Algorithm 4 Planner Rollout

```

1: ...
2: for rollout in  $1 : n_{\text{rollouts}}$  do           ▷ Alg. 3, ln. 6
3:   if rand()  $\leq b_p$  then
4:     Sample goal state  $s_g$ 
5:     Find node closest to  $s_g$ 
6:     Find path from root node to closest node
7:     Clip or pad trajectory if necessary
8:     Store all transitions in  $B$ 
9:   else
10:    Sample  $s_0$  and  $s_g$            ▷ Alg. 3, ln. 7
11:    for  $t$  in  $1 : N$  do           ▷ Alg. 3, ln. 8
12:  ...

```

Algorithm 5 Pre-training

```

1: for demo in  $n_{\text{demos}}$  do           ▷ demo generation
2:   Sample goal state  $s_g$ 
3:   Find node closest to  $s_g$ 
4:   Find path from root node to closest node
5:   Store all transitions in  $B$  if goal reached
6:   Compute value for all transition
7: for episode in  $1 : n_{\text{episode}}$  do   ▷ network training
8:   Sample minibatch of  $n_{\text{batch}}$  tuples
   ( $s_t, a_t, s_{g,t}, v_t$ ) from  $B$ 
9:   Compute value function gradient  $g_Q$  for  $l_Q =$ 
    $\frac{1}{n_{\text{batch}}} \sum_i (v_i - Q_{\theta}(s_i, a_i, s_{g,i}))^2$ 
10:  Compute policy gradient  $g_{\pi}$  for  $l_{\pi} =$ 
    $\frac{1}{n_{\text{batch}}} \sum_i \|a_i - \pi_{\phi}(s_i, s_{g,i})\|^2$ 
11:  Apply the update to  $\pi_{\phi}$  and  $Q_{\theta}$ 

```

| System | n_{epochs} | n_{steps} | p_a | $Q_{d,o,q}$ | Q_p | Q_m |
|---------------|---------------------|--------------------|-----------|-------------------------------|----------|-------|
| Box push | 150 | 75 | [1 1 2 2] | diag(1) | I | 1 |
| Box push 2D | 150 | 100 | [6 2 2 1] | diag(1, 1) | $0.001I$ | 0.001 |
| Planar hand | 150 | 80 | [1 1 2 2] | diag(1, 1, $\pi/2$) | $0.01I$ | 0.01 |
| Floating hand | 150 | 100 | [1 1 1 1] | diag(1, 2, $\pi/2$) | $0.01I$ | 0.01 |
| Allegro hand | 300 | 50 | [1 3 2 2] | diag(100, 100, 100, 1, 1, 10) | $0.01I$ | 0.01 |
| Box grasp | 300 | 50 | [0 1 1 1] | diag(1, 1, 10, 0, 0, 0) | $0.01I$ | 0.01 |
| Ball turn | 300 | 75 | [2 1 3 3] | diag(1, 1, 1, 2, 2, 0) | $0.1I$ | 0.1 |
| Stool lift | 300 | 75 | [1 1 1 1] | diag(1, 1, 1, 2, 2, 0) | I | 0.1 |

Table 4: Task parameters.

541 this task can give valuable insight into the general properties of planning and learning. Moreover,
542 only pushing but no pulling is possible, which creates an interesting manipulation challenge.

543 **Box push 2D** The two-dimensional box push task is the direct extension of the one-dimensional
544 box push task. Just as the one-dimensional box push task, it is an illustrative example. The com-
545 plexity of the task is increased by placing the pusher box between the object box and the goal. This
546 setup requires the pusher to move around the obstacle before pushing it towards the goal.

547 **Planar hand** The planar hand is a simplified representation of in-hand manipulation in a two-
548 dimensional plane. A hand with two two-link fingers is used to rotate and lift a box. Even with the
549 restriction to a plane, this task requires dexterous manipulation and gives insight into the planner’s
550 ability to solve in-hand manipulation tasks.

551 **Floating hand** The floating hand has the same kinematic as the planar hand and is also limited to
552 a two-dimensional plane. However, instead of a fixed wrist, the floating hand can move in the plane.
553 The task is to reorient a box and move it in space. This task combines dexterous manipulation as
554 well as picking and placing an object, two commonly required skills.

555 **Allegro hand** The Allegro hand is a robotic hand by Wonik Robotics with four fingers, sixteen
556 actuated finger joints, and 2 actuated wrist joints. The task is to reorient a cube by turning it around
557 the upright z-axis. In-hand manipulation is a commonly investigated robotic skill due to its high
558 dimensionality and required dexterity.

559 **Box grasp** The box grasp task uses a stationary quadrupedal robot with a mounted arm to grasp
560 and lift a box with a handle. It is a classic pick-and-place task, where finding a proper grasp is the
561 main challenge. This task demonstrates our method’s ability to solve pick-and-place scenarios on
562 real systems.

563 **Ball turn** The ball turn task involves a quadrupedal robot on its back that is using its legs to reorient
564 a ball. This setup resembles in-hand manipulation tasks where multiple multi-joint fingers (legs) are
565 used to reorient an object. We use this setup to demonstrate our method’s ability to perform in-hand
566 manipulation on real systems.

567 **Stool lift** The stool lift task comprises two stationary quadrupedal robots with mounted arms that
568 lift a bar stool into an upright configuration. The robots are deliberately prevented from grasping
569 the stool by fixing their grippers in a closed state. In this manner, whole-body and bimanual manip-
570 ulation of the stool is required, with contact occurring on multiple parts of the robot arms. We use
571 this task as an exemplary real-world cleaning scenario, where robots can help put moved or fallen
572 furniture back into desired configurations. By forcing whole-body contact, we also demonstrate our
573 method’s ability to perform whole-body manipulation on real systems.

574 C.2 Manipulation Planner

575 We present additional search results for two tasks and several studies on the planner’s components
576 and parameters.

577 C.2.1 Additional tasks

578 **Box push and box push 2D results** Extending the results presented in Fig. 2, we show the search
579 progress for the two additional tasks, box push and box push in 2D. The results are in line with the
reported results for the other tasks.

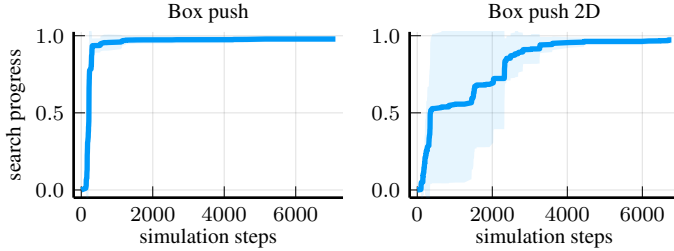


Figure 7: The performance of the Manipulation Planner on two additional tasks. Relative closeness
580 to the goal for the best node in the tree so far vs. the total number of nodes (simulation steps).

581 C.2.2 Ablations

582 **Extension horizon** The extension horizon specifies how many sequential actions are taken once
583 a node has been selected to extend the tree. We assume that tasks of different complexity require
584 different extension horizons. We evaluate different but fixed extension horizons for all tasks to
585 measure the influence on the planner’s search progress in Fig. 8. We also evaluate the automatically
586 varying horizon that we use in our implementation as described in Appendix A.2. We measure
587 the average closeness to the goal over the full search duration, which is larger for quicker search

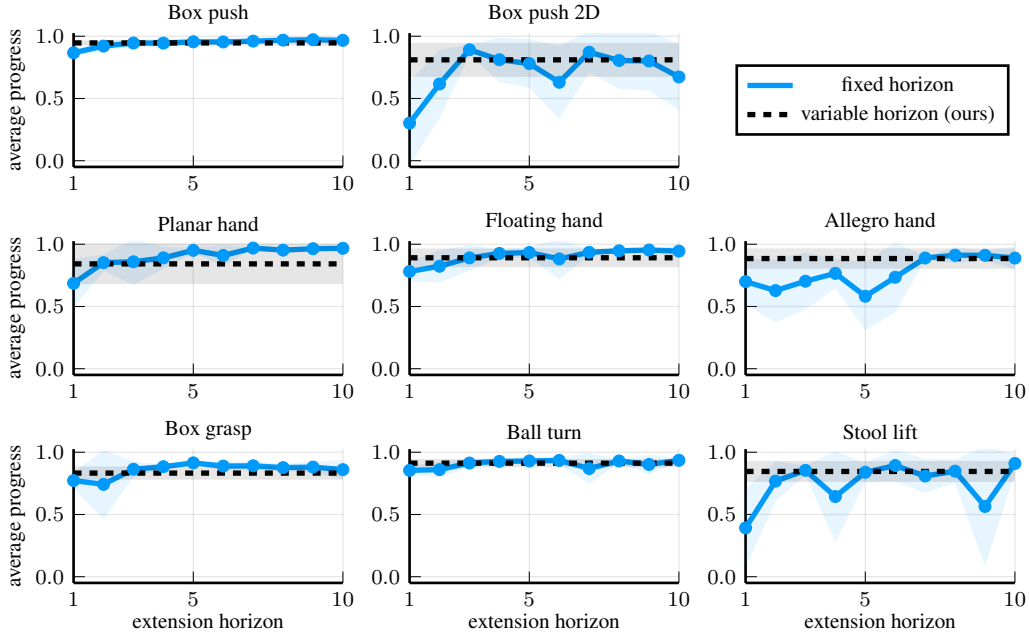


Figure 8: The performance of the Manipulation Planner for different extension horizons measured
as the average closeness to the goal during the search. Comparison of different fixed horizons (blue)
and variable horizon (black).

588 progress and higher final closeness. While a longer extension horizon tends to improve performance
 589 overall, there is no strong correlation. At the same time, the varying horizon performs well in all
 590 cases with reduced variance compared to the fixed horizons.

591 **Pareto distribution exponent** The Pareto distribution parameter β modifies how greedily nodes
 592 with high rewards are extended. We evaluate different but fixed Pareto parameters for all tasks to
 593 investigate how the parameter influences the planner’s search progress in Fig. 9. We also evaluate
 594 the automatically varying parameter that we use in our implementation as described in Appendix
 595 A.2. For intuition, a parameter $\beta = 0.05$ means that the currently best node will be selected with
 596 a probability of roughly $P = 10\%$ whereas $\beta = 2.0$ means a probability of $P = 75\%$. There
 597 is a consistent trend that the planner’s performance decreases as the node selection becomes more
 598 greedy (the parameter increases). This effect could be caused by the node selection becoming so
 599 greedy that the search gets stuck in local minima by repeatedly selecting the same few nodes. Using
 600 an automatically varying range for the parameter does not lead to an improvement over always
 601 choosing a low parameter. Apparently, the Pareto distribution with a low β already has a good
 tradeoff between exploitation and exploration.

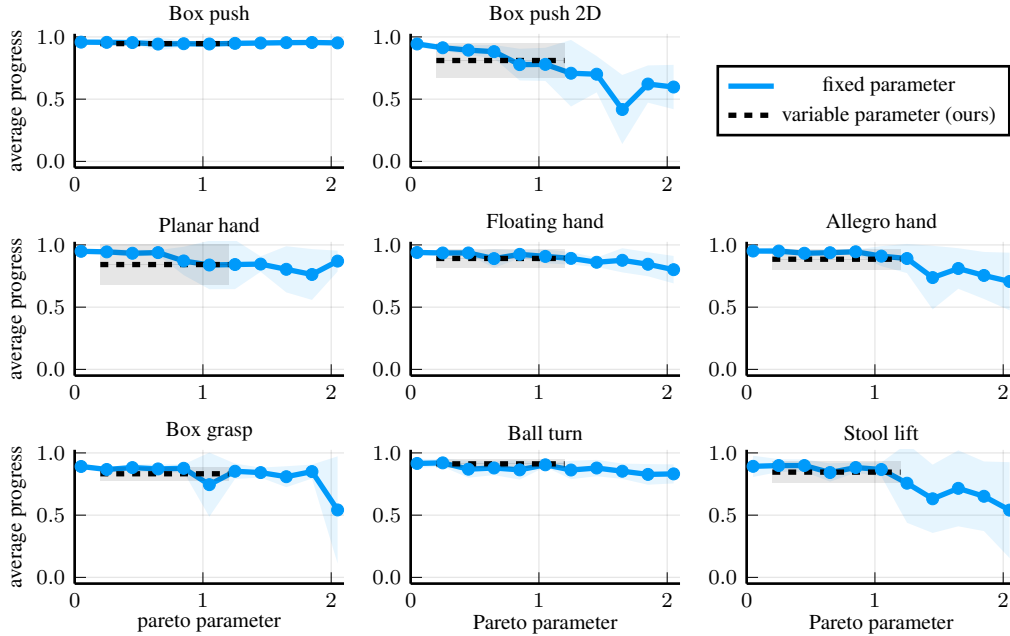


Figure 9: The performance of the Manipulation Planner for different Pareto parameters measured as the average closeness to the goal during the search. Comparison of different fixed parameters (blue) and variable parameter (black).

603 **Sub-goal ratio** To assess the usefulness of having sub-goals during the planner’s search, as in
 604 rapidly exploring random trees, we evaluate different ratios of sub-goals to node extensions in Fig.
 605 10. A goals/extensions ratio of 1/900 means we choose one goal and do 900 extensions for this
 606 single goal, whereas a ratio of 900/1 means we choose 900 goals and do one extension for each
 607 of these goals. Note that in this evaluation, we always do a total of 900 extensions. There is
 608 no consistent trend indicating that more or less sub-goals improve search performance. Multiple
 609 explanations exist for this results: Our tasks could have very few local minima. However, since the
 610 Pareto parameter does influence search performance (cf. Fig. 9), there may be many local minima,
 611 but our node selection strategy is already good enough to avoid these. Yet another possibility is that
 612 the difference in sub-goals is too small to provide a benefit over searching with just a single goal.

613 **Action types** We investigate the effectiveness of different combinations of action types with just
 614 random actions as a baseline in Fig. 11. The fact that purely random actions already lead to good

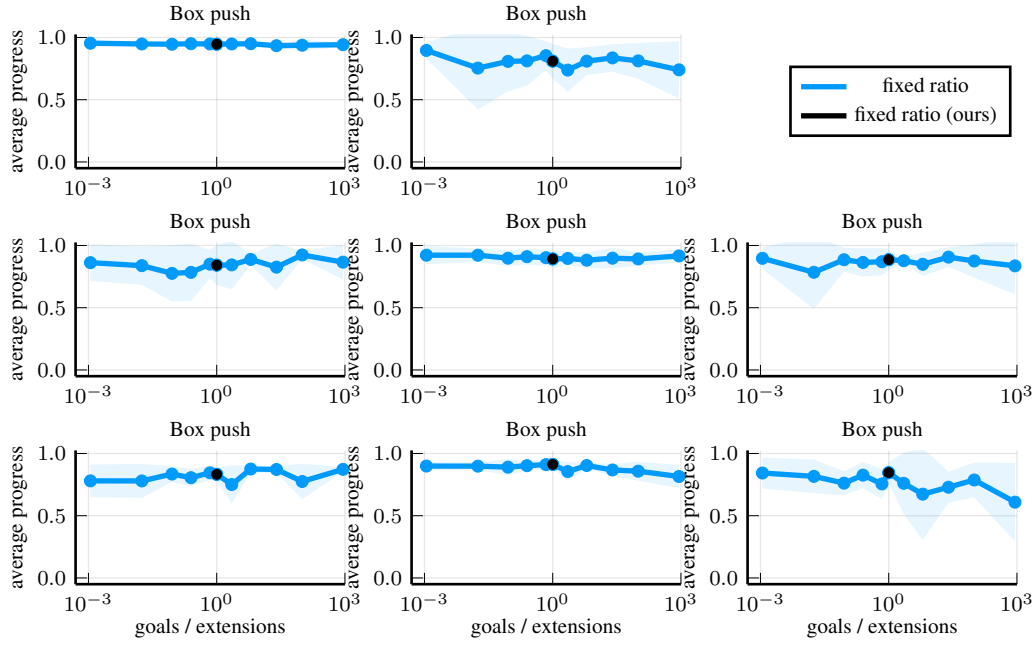


Figure 10: The performance of the Manipulation Planner for different ratios of goals/extensions measured as the average closeness to the goal during the search.

615 results in many tasks indicates that the other components in the planner, such as the heuristic node
 616 selection and specific rewards, lead to meaningful search progress. Still, proximity and goal-directed
 actions besides random actions in the search further improve the performance for some tasks.

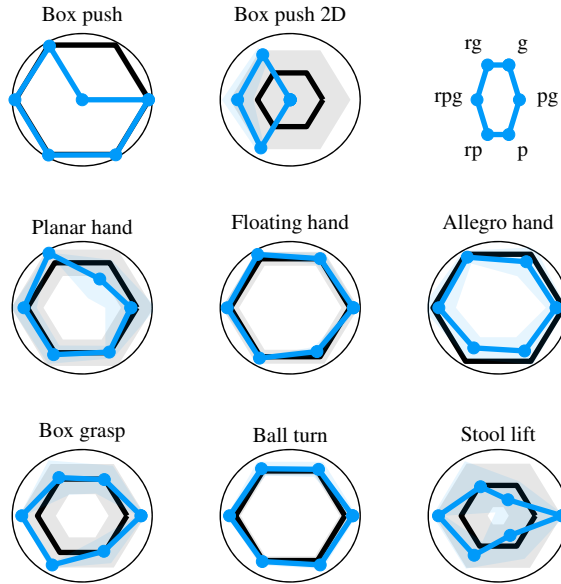


Figure 11: The performance of the Manipulation Planner for different combinations of action types measured as the average closeness to the goal during the search. Comparison of the following action types (blue): gradient (g), random+gradient (rg), random+proximity+gradient (rpg), random+proximity (rp), proximity (p), proximity+gradient (pg), and random (black).

618 **Proximity and reachability reward scaling** The effect of different proximity and reachability
 619 reward scalings is shown in Fig. 12. For most tasks, the search fails once the proximity reward

620 becomes too high, potentially because the necessary distance between the robot and the object for
 621 reconfiguration is punished too severely. The reachability reward scaling shows a similar although
 622 weaker trend. Especially for the more complex tasks involving the Allegro hand or robot arms, it
 623 is difficult to clearly state how to scale the rewards, and a parameter sweep is necessary to find the
 624 optimal value. It can be assumed that the same effect would occur for dense-reward reinforcement
 625 learning, highlighting one of the benefits of the planner, as parameter sweeps can be executed much
 faster with the planner than with learning.

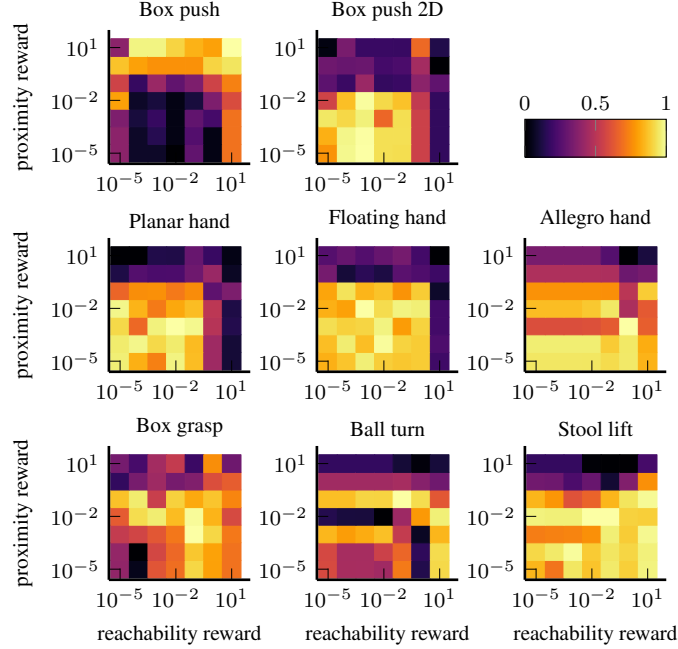


Figure 12: The performance of the Manipulation Planner for different combinations of action types measured as the average closeness to the goal during the search. Comparison of different proximity and reachability reward scalings. Purple is bad, yellow is good.

627 C.3 Reinforcement learning

628 We present additional learning results for two tasks and several comparisons for different learning
 629 methods and demonstration usage.

630 C.3.1 Additional tasks

631 **Box push and box push 2D results** Extending the results presented in Fig. 3, we show the training
 632 progress for the two additional tasks, box push and box push in 2D. The results are in line with the
 reported results for the other tasks.

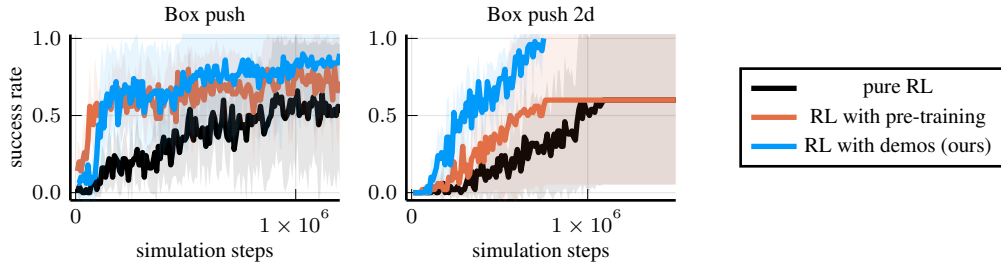


Figure 13: The performance of reinforcement learning for manipulation on two additional tasks. Comparison of our approach of adding demonstrations to the replay buffer (blue), RL with an additional imitation policy pre-training (red), and pure RL (black).

C.3.2 Ablations

Demonstration ratio Adding demonstrations to the RL replay buffer improves performance compared to pure online learning, but pure offline learning with the demonstration data does not yield good results. Accordingly, there must be a tradeoff between online exploration and offline data. We investigate this tradeoff by varying the ratio of demonstrations in the replay buffer in Fig. 14. We measure the average reward over the entire training duration, which is larger for quicker learning progress and a higher final success rate. It appears that a small amount of demonstrations gives enough reward signals to bootstrap RL and enable exploration from a sub-optimal solution. Already having a low ratio of roughly 10% yields satisfactory results, and we use 25% as it gives the most robust performance for all tasks.

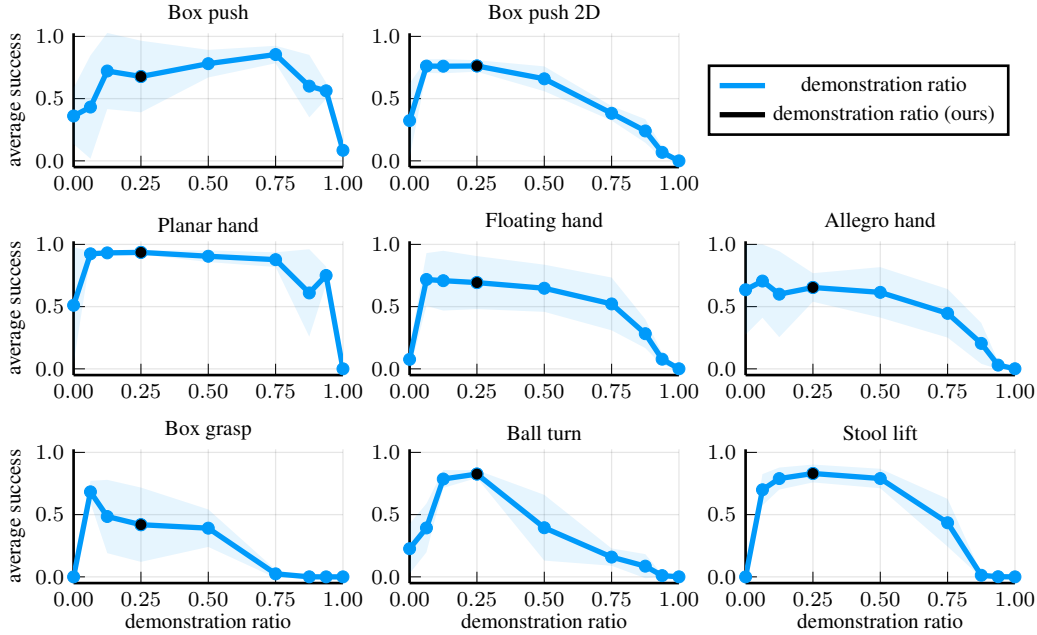


Figure 14: The performance of reinforcement learning (RL) for manipulation measured as the average success during training on eight tasks. Comparison of different demonstration to online exploration ratios in the replay buffer. A ratio of 0.0 is pure online RL, 1.0 is pure offline RL.

Demonstration inclusion comparison Three different methods for adding demonstrations to the replay buffer are investigated. (1) Use a fixed ratio of demonstrations to online exploration. (2) Use a variable ratio, where the ratio decreases as the learning success rate increases. (3) Add a fixed number of demonstrations from the planner’s tree only initially. As can be seen in Fig. 15, the specific method of introducing demonstrations into the replay buffer does not make a large difference. However, just adding demonstrations in the beginning is sometimes not enough to bootstrap learning successfully.

Pre-training comparison Directly using the pre-trained policy only works for the most simple box push task, as shown in Table 5, where we evaluated ten runs for each task. Apparently, the data generated by the plan-

| | Success rate |
|---------------|-----------------|
| Box push | 0.10 ± 0.13 |
| Box push 2D | 0.00 |
| Planar hand | 0.00 |
| Floating hand | 0.00 |
| Allegro hand | 0.00 |
| Box grasp | 0.00 |
| Ball turn | 0.00 |
| Stool lift | 0.00 |

Table 5: Success rate for directly deploying the pre-trained policy.

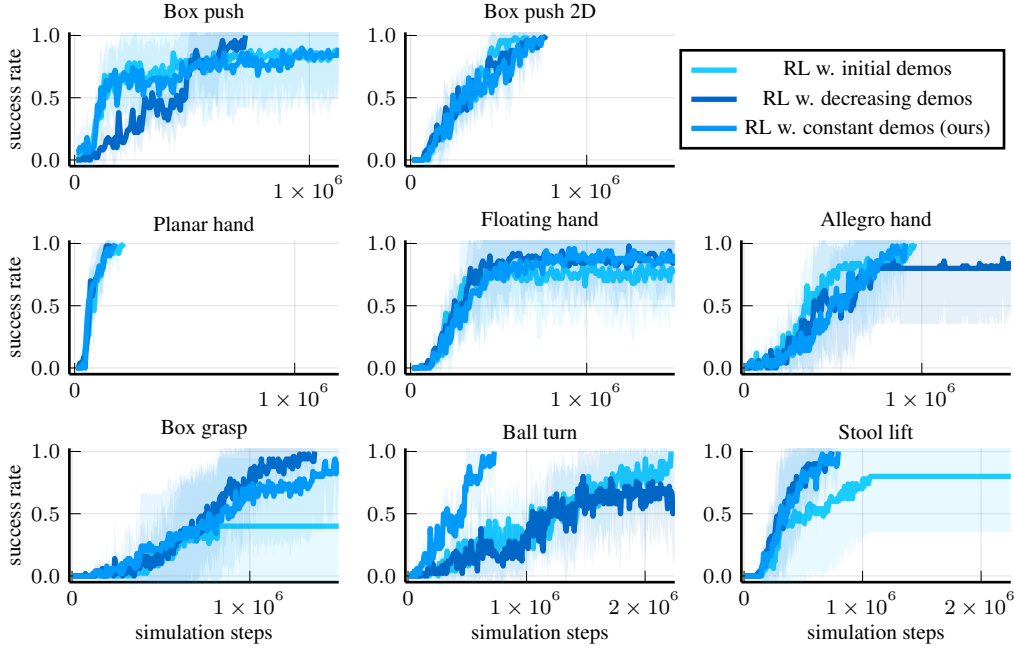


Figure 15: The performance of reinforcement learning for manipulation with demonstrations on eight tasks. Comparison of adding a constant ratio of demonstrations to the replay buffer (blue), adding fewer demonstrations with increasing success rate (dark blue), and adding demonstrations only at the beginning (purple).

ner is not good enough to directly perform imitation learning with it, and online exploration with RL is required to find successful policies.

Accordingly, we compare three different pre-training settings in Fig. 16. (1) Only pre-training a policy π_{IL} and using it alongside the online RL policy. (2) Pre-training a policy and the value function. (3) Pre-training the policy and value function as well as adding a fixed ratio of demonstrations to the replay buffer. While adding a pre-trained policy to RL improves performance over pure DDPG, it does not significantly and consistently perform much better. Only once demonstrations are added to the replay buffer does the performance with a pre-trained policy improve significantly. Since the performance of adding demonstrations alongside pre-training is not better than just adding demonstrations, we attribute this improvement to using demonstrations and not to the pre-trained policy.

HER comparison For reference, we provide a comparison to hindsight experience replay (HER) in RL, since this method can improve learning performance in sparse-reward settings. The results are shown in Fig. 17. While using HER improves the learning performance for some tasks compared to pure DDPG, it does not lead to any improvement for others, and almost always performs worse than adding demonstrations to the replay buffer.

C.4 Hardware evaluation

Setup We use the quadrupedal Spot robot by Boston Dynamics with an arm mounted for certain tasks. Control and communication with the robot are achieved via the bosdyn software development kit. Accurate state estimation is achieved using the OptiTrack motion capture system and placing markers on the objects. Communication between the Optitrack System and the robots was handled via ROS2, while gRPC was used to command policy actions with the Spot robots.

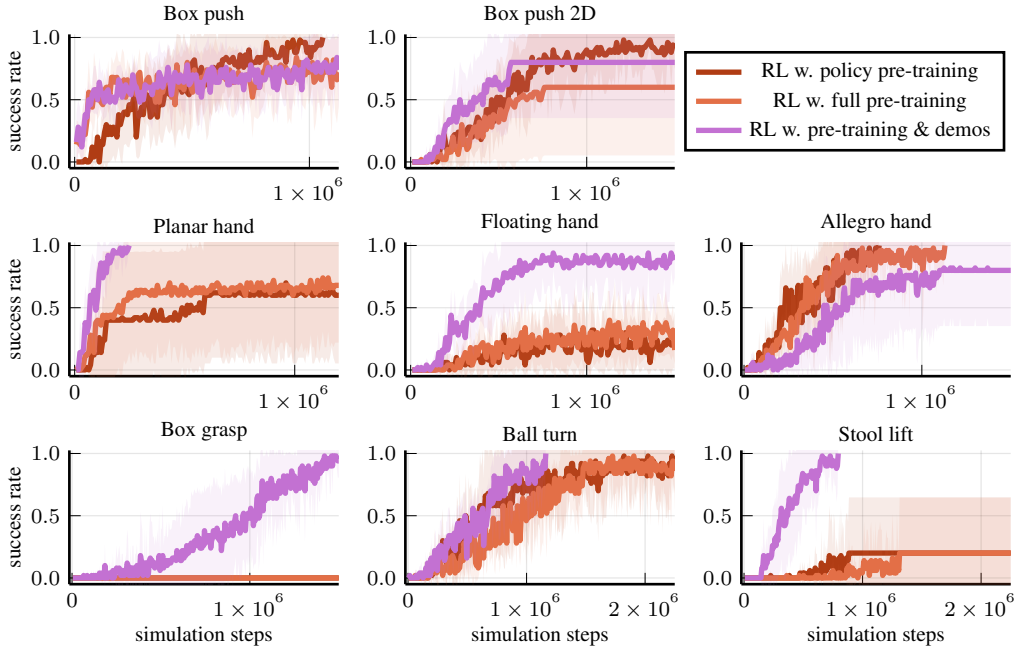


Figure 16: The performance of reinforcement learning for manipulation with pre-training on eight tasks. Comparison of pre-training just the policy (light red), pre-training the policy and value function (red), and pre-training the policy and value function as well as adding demonstrations (dark red).

679

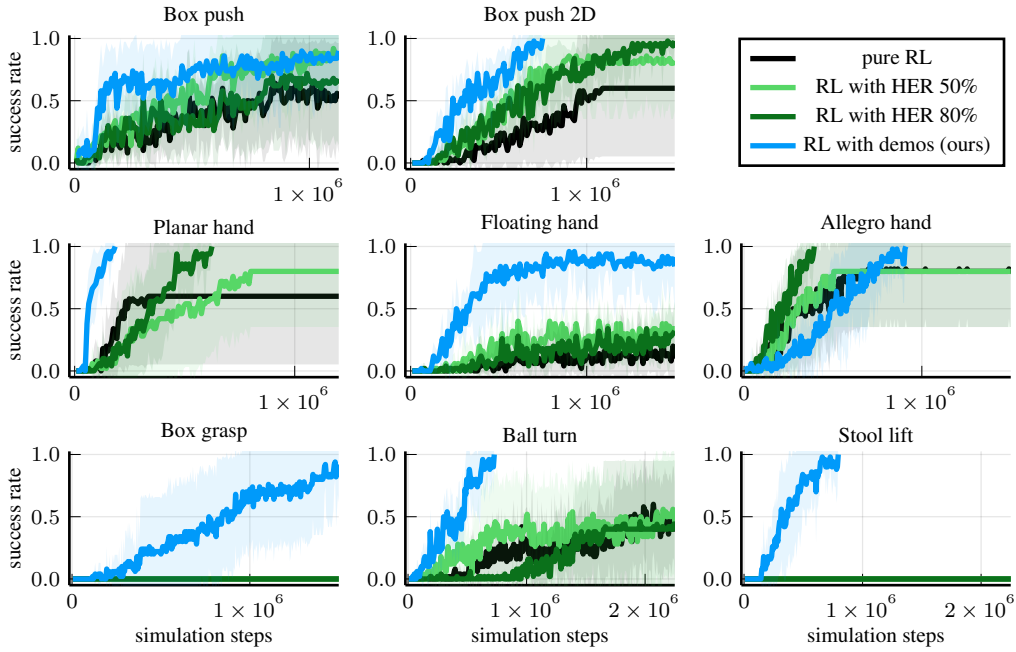


Figure 17: The performance of reinforcement learning for manipulation with hindsight experience replay on eight tasks. Comparison of our approach of adding demonstrations to the replay buffer (blue), RL with 50% HER relabeling (light green), RL with 80% HER relabeling (dark green), and pure RL (black).

Box grasp The box grasp task evaluates if the robot is able to grasp and lift the box. A trial is considered successful if the box is lifted off the ground. We evaluate 27 different starting locations by building a grid of start positions and orientations for the box:

$$x \in \{-0.1, 0.0, 0.1\} \times y \in \{-0.25, 0.00, 0.25\} \times \theta_z \in \{-45^\circ, 0^\circ, 45^\circ\}. \quad (19)$$

Fig. 18 shows the task progression for the box-grasp task. The challenge in this task is to learn that in order to lift the box, the handle must be grasped. Our trained policy is able to consistently grasp the handle and performs well.

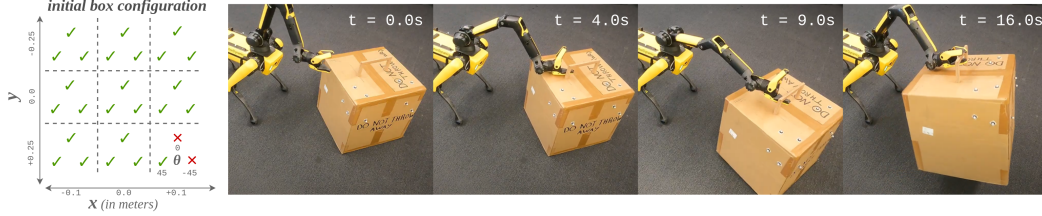


Figure 18: **Left:** the successful and failed initial configuration for the box-grasp tasks. **Right:** deployment of the trained policy for one Spot grasping and lifting the box.

Ball turn The ball-turn task is used as an in-hand-manipulation-equivalent example. A trial is considered successful if the box is rotated such that the angle deviation from the upright z-axis is below 40° . We evaluate 20 different starting locations by testing a progression of start rotations around the y-axis for the ball with increments of 15° :

$$\theta_z \in \{\pm 180^\circ, \pm 165^\circ, \dots, \pm 60^\circ, \pm 45^\circ\}. \quad (20)$$

Fig. 19 shows the task progression for the ball-turn task. The robot can successfully perform some of the desired orientations. We attribute failures to the sim2real gap, with a rigid ball in simulation training and a soft yoga ball in the real-system setting.

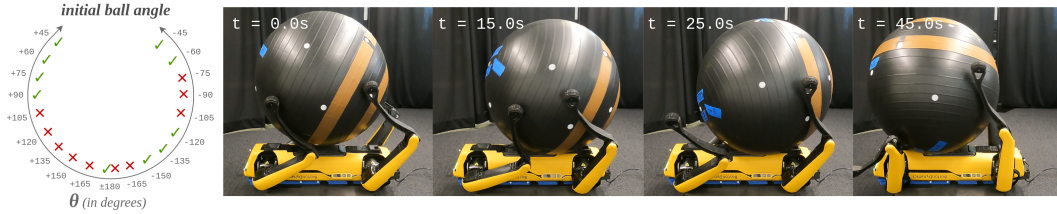


Figure 19: **Left:** the successful and failed initial orientations for the ball-turn tasks. **Right:** deployment of the trained policy for one Spot turning the ball to the upright position.

Stool lift The stool-lift task requires whole-body manipulation and is a bimanual task representative of a real-world cleaning task. A trial is considered successful if the robots maneuver the stool in the upright position. We evaluate 25 different starting locations by building a grid of start positions for the stool:

$$x \in \{-0.2, -0.1, 0.0, 0.1, 0.2\} \times y \in \{-0.2, -0.1, 0.0, 0.1, 0.2\}. \quad (21)$$

Fig. 4 shows the task progression for the stool-lift task. Despite the challenging whole-body and bimanual setup, the robots frequently manage to lift the stool in the upright configuration. Failures can be attributed to a lack of measuring the robots' body positions. When contact forces occur, the robots' bodies are displaced (which we do not measure), leading to position offsets for the robot arms. In such cases, the policy has false state measurements and produces unsuccessful actions.