

Curriculum Reinforcement Learning for Complex Reward Functions

Kilian Freitag¹, Kristian Ceder¹, Rita Laezza¹, Knut Åkesson¹, Morteza Haghir Chehreghani^{1,2}

¹Chalmers University of Technology

Gothenburg, Sweden

²University of Gothenburg

Gothenburg, Sweden

Email: {tamino, cederk, laezza, knut.akesson, morteza.chehreghani}@chalmers.se

Abstract—Reinforcement learning (RL) has emerged as a powerful tool for tackling control problems, but its practical application is often hindered by the complexity arising from intricate reward functions with multiple terms. The reward hypothesis posits that any objective can be encapsulated in a scalar reward function, yet balancing individual, potentially adversarial, reward terms without exploitation remains challenging. To overcome the limitations of traditional RL methods, which often require precise balancing of competing reward terms, we propose a two-stage reward curriculum that first maximizes a simple reward function and then transitions to the full, complex reward. We provide a method based on how well an actor fits a critic to automatically determine the transition point between the two stages. Additionally, we introduce a flexible replay buffer that enables efficient phase transfer by reusing samples from one stage in the next. We evaluate our method on the DeepMind control suite, modified to include an additional constraint term in the reward definitions. We further evaluate our method in a mobile robot scenario with even more competing reward terms. In both settings, our two-stage reward curriculum achieves a substantial improvement in performance compared to a baseline trained without curriculum. Instead of exploiting the constraint term in the reward, it is able to learn policies that balance task completion and constraint satisfaction. Our results demonstrate the potential of two-stage reward curricula for efficient and stable RL in environments with complex rewards, paving the way for more robust and adaptable robotic systems in real-world applications.

I. INTRODUCTION

Reinforcement Learning (RL) has emerged as a powerful paradigm in the field of robotic control, offering the promise of adaptable and efficient solutions to complex problems. RL has demonstrated its potential to learn optimal policies in a wide range of applications such as manipulation [11, 17, 20, 13] or mobile robotics [21, 38, 28, 37, 30]. However, as we transition from carefully curated benchmarks to realistic scenarios, a significant gap emerges, highlighting the challenges of applying RL in practical settings.

One of the primary challenges in realistic applications lies in the complexity of the environment and the multiplicity of objectives. While classical RL problems often focus on a single, well-defined goal, real-world scenarios typically involve multiple, sometimes conflicting objectives. For instance, a mobile robot might need to navigate to a goal location while simultaneously avoiding obstacles, maintaining a specific ve-

locity, and ensuring a smooth trajectory [36, 7]. This multi-objective nature of real-world problems poses a significant challenge to traditional RL approaches.

The reward hypothesis suggests that any learning objective can be expressed by a single scalar reward under certain assumptions [34, 5]. However, formulating an effective reward function for complex tasks is non-trivial and can often result in undesired behaviors [4, 19, 18]. Moreover, optimizing such complex rewards can be challenging due to the presence of local optima, where policies might satisfy only a subset of objectives (e.g., minimizing energy consumption by remaining stationary) without considering others. We refer to such potentially conflicting objectives as complex reward functions, where the complexity lies in the presence of strong local optima for undesired behaviors.

A line of work that has emerged to tackle challenging RL problems is curriculum learning [3, 23]. Inspired by how animals can be trained to learn new skills [33], learning proceeds from simple problems to gradually more complex ones. In the realm of RL, such curricula are often hand-designed and have been successfully employed in several applications in robotic control [31, 16, 20]. More recently several methods for automatic curriculum design have emerged that are able to learn curriculum policies [22] or that break down problems into smaller ones [1, 9] for more effective learning. Further, automatic curricula are used in sparse or no reward settings as intrinsically motivated exploration that encourages reaching diverse states [2, 25, 6, 32]. Nevertheless, such methods have been explored less for complex reward functions.

To address the challenge of learning with complex reward functions, we propose leveraging curriculum learning. We introduce a novel two-stage reward curriculum combined with a flexible replay buffer to effectively balance task success and constraint satisfaction. In the first phase, a subset of rewards is used for training to simplify the discovery of successful trajectories. When the policy has converged sufficiently, the second phase is initiated, optimizing the full reward. To automatically determine when to switch to the second phase, we track how well the actor optimizes the Q -function as a proxy for policy convergence. Additionally, our method allows for sample-efficient reuse of collected trajectories by incorporating

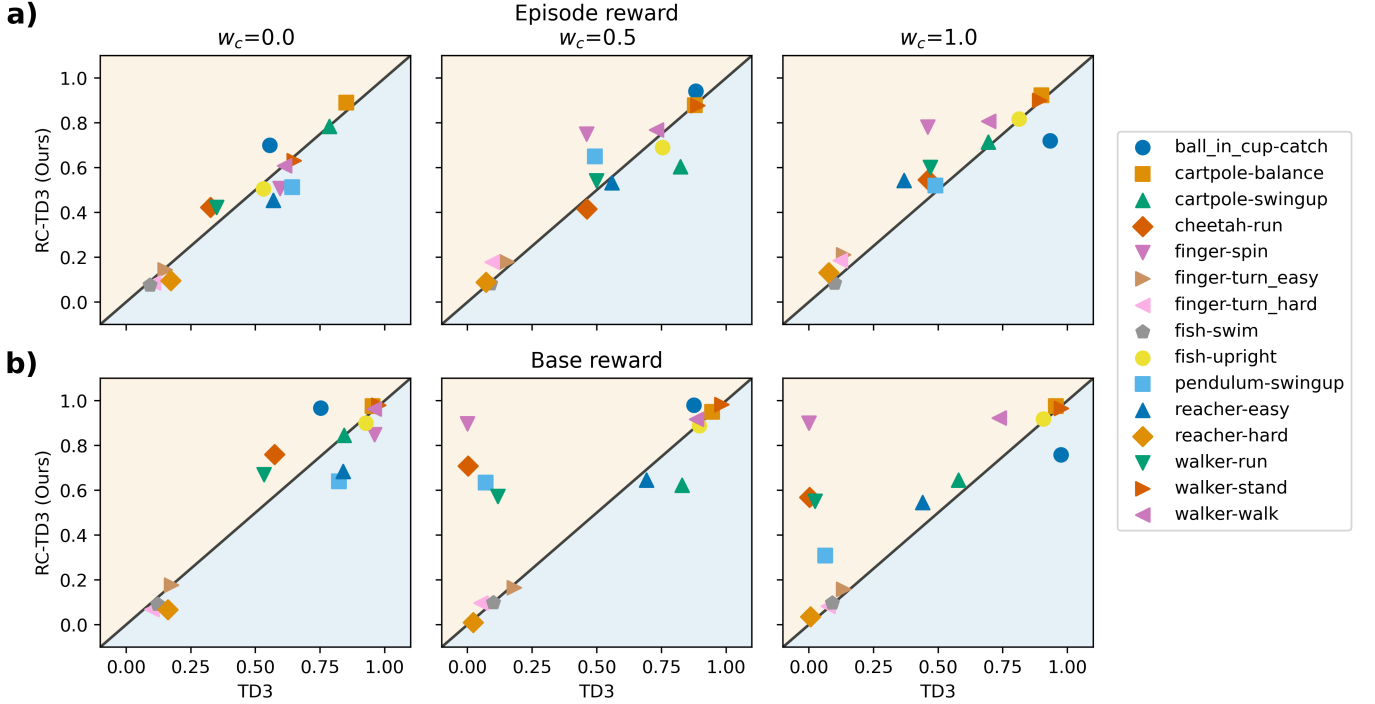


Fig. 1: Comparison of the standard TD3 algorithm with reward curriculum version (RC-TD3). Fig. a) shows the normalized mean episode reward visualized using constraint weight $w_c = 1.0$ for intuitive comparability between policies trained on different constraint weights w_c . As it can be seen, the curriculum becomes more effective with higher w_c . It seems to be most helpful for environments where the constraints are not automatically optimized by completing the task, i.e. the ones in the center of the left plot with $w_c = 0.0$. In environments where learning is unsuccessful in the first place or that are relatively simple even with constraints the effects of the curriculum are less pronounced. Fig. b) shows the mean base reward r_b without constraints. Importantly, employing a reward curriculum manages to keep or improve the base reward in almost all cases, especially for high w_c . This demonstrates its effectiveness in finding a better trade-off between task performance and constraint satisfaction, given that the baseline often gets stuck in the local minima of only optimizing constraints as in the case of finger spin.

two rewards in the replay buffer such that samples from the first phase can be reused for training with an updated reward in the second phase.

To analyze the efficacy of our method, we first evaluate it on several modified environments from the DeepMind control suite, where the agent in addition to the original reward gets penalized for taking large actions. Furthermore, we evaluate it on a mobile robot that is tasked to reach a goal location while avoiding collisions. Simultaneously, it must fulfill several constraints, including maintaining a reference velocity, staying close to a planned path when possible, and generating smooth trajectories. We formulate the reward in an intuitive way, to enable simple testing of different constraint weights w_c . The complexity again does not lie in finding any valid solution, but in finding a solution that maximally satisfies constraints while still reaching the goal, exacerbated by the plurality of the objective.

In our experiments, we compare our method to a baseline that always trains on the full reward. We show that our two-stage reward curriculum becomes more effective the higher the constraints are weighted in the overall reward. Especially for

environments where the constraints hinder solving the task, our method is able to successfully learn the task while satisfying constraints where the baseline fails and gets stuck in solely optimizing constraints (see Fig. 1). In an ablation study where we either reset the network weights or the replay buffer when changing curriculum phases, we show that the main factor for the success of the method lies in the “pretrained” network weights from the first phase. Resetting the replay buffer does not greatly change the outcome. Furthermore, we examine how our automatic curriculum switch performs in comparison to a static switch after a predetermined number of steps. Our results demonstrate that the automatic switch effectively identifies suitable switching times, leading to faster convergence than the static approach. Our contributions can be summarized as follows:

- 1) We introduce a novel two-stage reward curriculum to effectively learn complex rewards by reusing past experiences with updated rewards and an automatic mechanism to switch phases. The curriculum is integrated in two RL methods, one based on SAC and the other based on TD3.

- 2) We extensively evaluate our method on several modified DeepMind control suite environments and discuss in which environments a reward curriculum is most effective.
- 3) In ablation studies we demonstrate that the key ingredients for our method are the network weights obtained in the first phase and the automatic switching mechanism.
- 4) To understand the impact of our method on environments with several conflicting reward terms, we extensively evaluate it on a mobile robot navigation problem.

We will make the code used for our experiments openly available.

II. PROBLEM FORMULATION

We formulate the problem as a Markov Decision Process (MDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, p_0, \gamma \rangle$. \mathcal{A} represents the action space, \mathcal{S} the state space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the state transition function, $p_0 : \mathcal{S} \rightarrow [0, 1]$ the initial state distribution and $\gamma \in [0, 1)$ a discount factor. The reward is denoted by r ; we omit its explicit dependence on state and action $r(s, a)$ for a less clustered notation.

The objective of RL is to maximize the expected return, $\mathbb{E}[G_n]$, where $G_n = \sum_{k=n}^N \gamma^{k-n} r_k$ is the cumulative discounted reward, with n being the current step and N the maximum steps per episode. A key concept for achieving this goal is the action-value function, commonly referred to as the Q -function, which represents the expected return when taking action a in state s . The Q -function is typically parameterized by ϕ , and denoted $Q_\phi(s, a)$.

In our case, we consider any control problem with several reward terms, where each can be categorized either as a base reward r_b if it helps learning the goal (for instance reward shaping terms), and constraint rewards r_c , which specify the desired behavior. The reward is then given as the weighted sum

$$r = r_b + w_c r_c \quad (1)$$

Given this problem formulation, the goal is to develop methods that allow learning a policy $\pi(a|s)$ that learns to complete tasks while maximally satisfying the constraints r_c and being robust to different constraint weights w_c . Intuitively, the challenge lies in finding policies that learn the task without exploiting rewards that relate to the constraints.

III. REWARD CURRICULUM

We propose a novel two-stage reward curriculum, to effectively learn complex reward functions in a sample-efficient manner. In principle the reward curriculum can be combined with any off-policy RL algorithm, though in this work we focus on two versions of our method: one based on Soft-Actor Critic (SAC) [12] and the other based on Twin-Delayed DDPG (TD3) [10], which we denote as RC-SAC and RC-TD3 respectively. In the first phase of the curriculum, we consider only a subset of reward terms for training, denoted as r_b . In the second phase, we then directly optimize the full reward

Algorithm 1 Off-policy Reward Curriculum

```

Initialize network parameter  $\phi_1, \phi_2, \bar{\phi}_1, \bar{\phi}_2, \theta, \bar{\theta}$ 
 $t \leftarrow 0$ 
 $\mathcal{CR} = 0$  ▷ Initial curriculum phase
for each iteration do
  for each environment steps do
    if using SAC then
       $a_t \sim \pi_\theta(a_t|s_t)$ 
    else
       $a_t = \pi_\theta(s_t) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma)$ 
    end if
     $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ 
     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_b, r, s_{t+1})\}$ 
     $t \leftarrow t + 1$ 
  end for
  if  $\forall i \in \{t - m + 1, \dots, t\} : J_{\pi, Q; i} < J_{\pi, Q; \mathcal{CR}}$  then
     $\mathcal{CR} = 1$ 
  end if
  for each gradient steps do
     $B = \{(s, a, r_b, r, s')\} \sim \mathcal{D}$ 
    if  $\mathcal{CR} == 0$  then ▷ Choose reward
       $r_{cr} \leftarrow r_b$ 
    else
       $r_{cr} \leftarrow r$ 
    end if
     $B \leftarrow \{(s, a, r_{cr}, s')\}$ 
    for  $i \in \{1, 2\}$  do
       $\phi_i \leftarrow \phi_i - \lambda_Q \nabla_{\phi_i} \frac{1}{|B|} \sum_{(s, a, r_{cr}, s') \in B} J_Q(\phi_i)$ 
    end for
     $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta \frac{1}{|B|} \sum_{s \in B} J_\pi(\theta)$  ▷ Delayed in TD3
    if using SAC then
       $\alpha \leftarrow \lambda_\alpha \nabla_\alpha J(\alpha)$ 
    end if
     $\bar{\phi}_i \leftarrow \tau_{\text{targ}} \bar{\phi}_i + (1 - \tau_{\text{targ}}) \phi_i$  for  $i \in \{1, 2\}$ 
    if using TD3 then
       $\bar{\theta} \leftarrow \tau_{\text{targ}} \bar{\theta} + (1 - \tau_{\text{targ}}) \theta$ 
    end if
  end for
end for

```

$r = r_b + w_c r_c$. Notably, we store the rewards from both phases in the replay buffer, resulting in the tuple $\{(s_t, a_t, r_b, r, s_{t+1})\}$. This allows us to reuse past experiences when transitioning to the second phase, where we switch the reward during training to ensure stable and sample-efficient learning.

Formally, we define the curriculum reward as

$$r_{cr} = \begin{cases} r_b & \text{if } \mathcal{CR} = 0 \\ r & \text{otherwise} \end{cases} \quad (2)$$

where \mathcal{CR} is the current curriculum phase. Algorithm 1 describes the two-stage curriculum in detail.

A. RC-SAC

In the following, we show how our method integrates with SAC. Instead of solely optimizing the expected return, SAC

makes use of the maximum entropy objective [39] such that a policy additionally tries to maximize its entropy \mathcal{H} at each state which is defined as

$$\pi_{\theta}^* = \arg \max_{\pi_{\theta}} \sum_t \mathbb{E}_{(s_t, a_t) \sim \rho_{\pi_{\theta}}} [r_{\text{cr}}(s_t, a_t) + \alpha \mathcal{H}(\pi_{\theta}(\cdot|s_t))]$$

where α is a parameter to control the policy temperature, i.e. the relative importance of the entropy term. SAC makes use of two Q-functions parameterized by ϕ_1 and ϕ_2 which are updated as

$$J_Q(\phi_i) = (Q_{\phi_i}(s, a) - y(r_{\text{cr}}, s'))^2 \quad (3)$$

where i is the Q-function index and the targets are computed as

$$y(r_{\text{cr}}, s') = r_{\text{cr}} + \gamma \left[\min_{i=1,2} Q_{\bar{\phi}_i}(s', \tilde{a}') - \alpha \log \pi_{\theta}(\tilde{a}'|s') \right]$$

Importantly, $\tilde{a}' \sim \pi_{\theta}(\cdot|s')$ is newly sampled during training and $Q_{\text{tar},i}(s, a)$ is the i th target Q-function. Note that instead of optimizing for a general reward r , we use r_{cr} in this step which changes depending on the phase. The policy update is given by

$$J_{\pi}(\theta) = \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_{\theta}(s)) - \alpha \log \pi_{\theta}(\tilde{a}_{\theta}(s)|s)$$

where $\tilde{a}_{\theta}(s)$ is sampled from $\pi_{\theta}(\cdot|s)$ via the reparameterization trick [29]. We make use of the entropy-constraint variant of SAC as described in [12], where α is updated as

$$J(\alpha) = \mathbb{E}_{a \sim \pi_{\theta}} [-\alpha \log \pi_{\theta}(a|s) - \alpha \mathcal{H}(\pi_{\theta}(\cdot|s))]$$

An important parameter in SAC is the initial value for α , which we denote as α_{init} . It determines the importance of the entropy term and, thus how much the agent explores different states. When α converges to low values, the agent focuses mainly on the objective instead and becomes more deterministic. For our experiments, we set $\alpha_{\text{init}} = 1.0$ and clip it to a minimum of $\alpha_{\text{min}} = 0.0001$ to increase stability.

B. RC-TD3

Furthermore, we show how the reward curriculum integrates with Twin-Delayed DDPG (TD3) [10]. It extends the Deep Deterministic Policy Gradient (DDPG) [21] algorithm to enhance its stability by making use of two Q-functions, delay the policy updates, and add noise to target actions to avoid exploitation of Q-function errors.

It uses a deterministic policy, though for sampling an action Gaussian noise is added for improved exploration

$$a_t = \pi_{\theta}(s_t) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

A similar procedure but with clipped noise is used when calculating the next action to update the estimated Q function to avoid exploitation of overestimations as

$$\tilde{a} = \pi_{\bar{\theta}}(s') + \epsilon \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$$

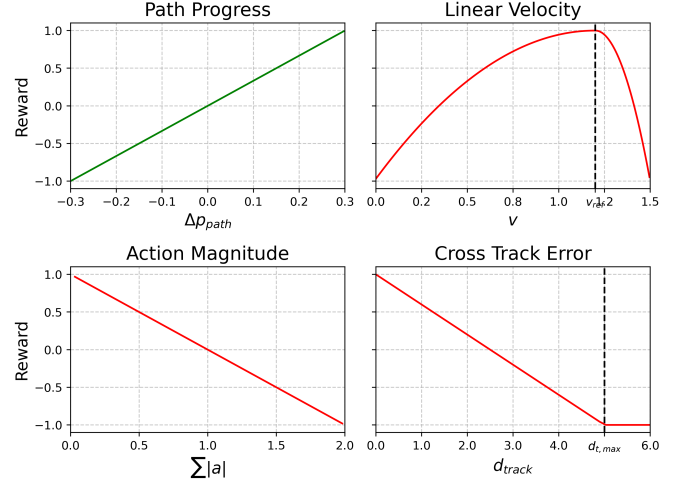


Fig. 2: Functions for dense reward terms with $\kappa = 0.942$, $v_{\text{ref}} = 1.2$ and $d_{\text{track,max}} = 5$. The range for each term is normalized to $[-1, 1]$. Green shows the reward shaping term that enables finding the goal faster. The soft constraints are colored in red.

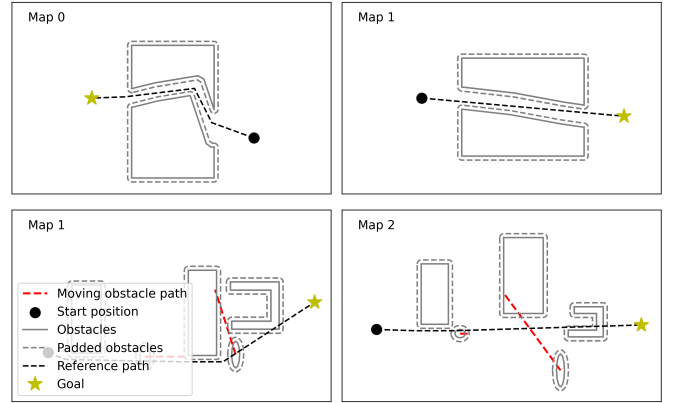


Fig. 3: Exemplary environment maps used for training. Obstacle positions, paths, initial states, and goal positions are randomized. While maps 0 and 2 contain dynamic obstacles, maps 1 and 3 only contain static ones.

where $\pi_{\bar{\theta}}$ is a target policy. The Q targets are then computed as

$$y(r_{\text{cr}}, s') = r_{\text{cr}} + \gamma \min_{i=1,2} Q_{\bar{\phi}_i}(s', \tilde{a}')$$

and the Q function is updated as in Equation III-A. The policy is updated using the first Q network as

$$J_{\pi}(\theta) = Q_{\phi_1}(s, \pi_{\theta}(s)) \quad (4)$$

Importantly, the policy is not updated in every iteration. Often it is delayed to update only in every second iteration, which we also employ in this work. In all experiments we set $\tilde{\sigma} = 0.2$, $c = 0.5$ and $\sigma = 0.1$. For more details about the parameters and specific design choices, we refer to the original paper.

C. Automatic phase switch

One key aspect of the reward curriculum is when to switch from the initial phase to training with the full reward. We hypothesize that the actor has to sufficiently learn the task using the base reward before constraints should be added. As a proxy for learning, we consider how well the actor optimizes the critic. In the case of TD3 that would simply be the normal actor loss III-B, for SAC it would be the actor loss III-A but without the additional entropy term. If the actor fits the critic well for m timesteps, i.e. $\forall i \in \{t-k+1, \dots, t\} : J_{\pi, Q; i} < \Gamma_{CR}$ where Γ_{CR} is the threshold that defines a "good" fit, we switch to the second curriculum phase.

We also considered other metrics to find when to best initiate the second phase such as policy entropy, but found that it is not straightforward to relate entropy to policy convergence, and in addition this would limit the algorithm to stochastic policies. Further, we hypothesize that the Q fit would be a more stable and accurate measure compared to simply tracking the obtained reward.

IV. ENVIRONMENTS

We first evaluate the methods RC-SAC and RC-TD3 on several different environments from the DeepMind control suite and then on a mobile robot environment with several reward terms. In the following, we describe the details of the environments.

A. DeepMind Control Suite

The DeepMind Control Suite [35] is a well-known collection of control environments in RL. We use the state-space observations and the environments have an action space $\mathcal{A} \in [-1, 1]^2$. The original rewards are always in $r \in [0, 1]$, which we take as base reward r_b . Furthermore, we introduce the following reward term to minimize action magnitudes, to find efficient solutions to the control problems

$$r_c = -\frac{1}{d} \|a\|_1 \quad (5)$$

which is used as the constraint term in the curriculum reward II.

B. Mobile Robot

To evaluate our method in a more realistic and complex setting (in terms of rewards), we consider a mobile robot navigation problem where the robot is tasked to reach a goal position while avoiding obstacles and satisfying various constraints. The environment maps are randomized and contain both permanent (e.g., walls and corridors) and temporary (e.g., dynamic or static) obstacles. A subset of exemplary environment maps can be found in Figure 3. Furthermore, a reference path is computed using A* [14], considering only permanent obstacles. This reference path should simulate a setting where an optimal path is pre-computed but the robot might not be able to naively follow it due to temporary obstacles. The constraints include driving at a reference velocity, staying close to the reference path, and creating smooth trajectories.

The state space $\mathcal{S} \in \mathbb{R}^{178}$ consists of the latest two lidar observations, the robot's position, the current speed, the reference path, and the goal position. The action space $\mathcal{A} \in [-1, 1]^2$ represents the translational and angular acceleration.

For the reward design, we will focus on easily interpretable formulations. There are three possible outcomes of an episode: (1) reach the goal, (2) timeout, i.e. reach maximum steps, or (3) collide with an obstacle. Empirical tests have shown that penalizing collisions mainly hinders exploration and does not lead to better final policies. Therefore, the only outcome-based reward included is:

$$r_g = \begin{cases} 100 & \text{if reached goal} \\ 0 & \text{otherwise} \end{cases}$$

The other objectives in the mobile robot problem can be expressed as soft constraints evaluated at each step, which provide dense reward signals. To enable intuitive weighting, we normalize each term such that all soft constraint rewards are between $[-1, 1]$. We chose this range over $[0, 1]$ to discourage the robot from learning to immediately crash.

We consider minimizing accelerations to achieve smooth trajectories. As this corresponds to our action space, we penalize high action values, similar to the other experiments, as

$$r_a = 1 - \sum_{i=1}^2 |a_i|, \quad r_a \in [-1, 1]$$

To encourage driving at a desired reference velocity v_{ref} , we model a velocity reward as a piece-wise quadratic function centered on v_{ref} , and scaled to our desired range. This choice is motivated by the intuition that slowing down is less severe than going too fast.

$$r_v = 1 - \frac{l_2^\kappa(v_t - v_{ref}) \cdot 2}{\max(l_2^\kappa(-v_{ref}), l_2^\kappa(v_{max} - v_{ref}))}, \quad r_v \in [-1, 1]$$

with the piece-wise quadratic function l_2^κ being defined as:

$$l_2^\kappa(x) = \begin{cases} \kappa x^2 & \text{if } x > 0 \\ (1 - \kappa)x^2 & \text{otherwise} \end{cases}$$

where κ controls the slope of the negative and positive regions and $v_{max} = 1.5$ is the maximum velocity.

Further, we linearly penalize deviating from the reference trajectory with the following reward term

$$r_x = \text{clip}\left(\frac{|d_{track}|}{d_{track, \max}}, -1, 1\right), \quad r_x \in [-1, 1]$$

where d_{track} is the distance between the agent and the reference trajectory and $d_{track, \max}$ is a tuning parameter that sets the maximum distance. This is done to make the robot's behavior predictable, such that it only deviates from a planned path if necessary.

To enable effective learning we additionally make use of potential-based reward shaping [24], by encouraging progress along the reference path, through

$$r_p = \frac{p_{\text{path}}(s') - p_{\text{path}}(s)}{v_{\max} \cdot dt}, \quad r_p \in [-1, 1]$$

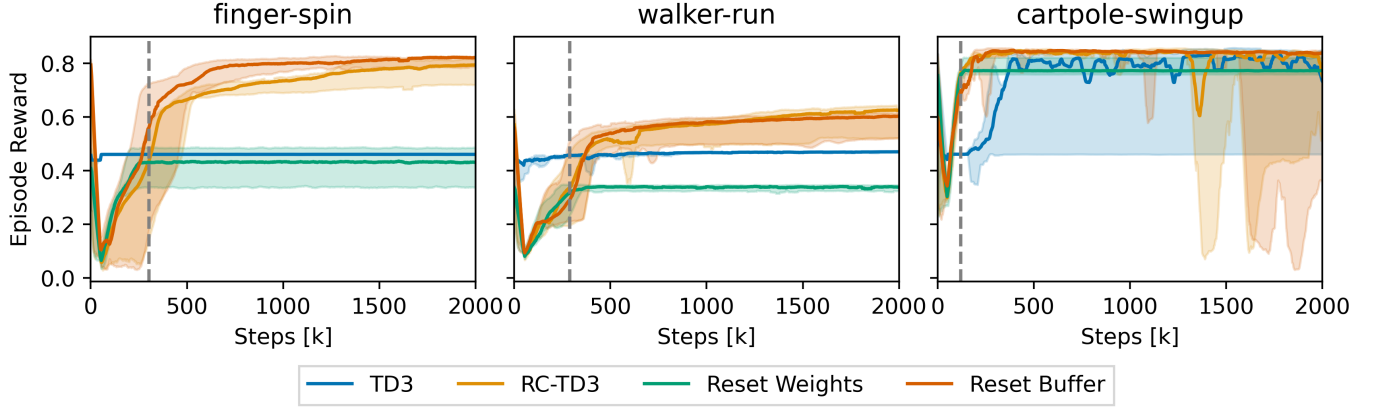


Fig. 4: Investigation of the median episode reward of RC-TD3 to resetting the network weights and resetting the networks when changing curriculum phases. The values are smoothed by taking the running average with window size 50 [k]. As it can be seen, resetting the network deteriorates performances while resetting the replay buffer has relatively little influence on the final outcomes. We conclude that the main benefit of a reward curriculum in these environments comes from improved exploration given by a “pretrained” network. The gray dashed line indicated the mean time when the curriculum phase was switched in RC-TD3.

where $p_{\text{path}}(s)$ is the position on a path in state s and the denominator is the maximum distance traversed in one step. As it is potential-based, it does not alter the ordering over policies [24]. Figure 2 gives an overview of the dense reward terms used.

Assuming that all constraints are equally important we assign equal constraint weights w_c . The reward is then given by

$$r = r_g + w_p r_p + w_c (r_v + r_a + r_x) \quad (6)$$

where we set the reward-shaping weight $w_p = 0.25$ throughout all experiments.

C. Experimental setup

As neural network architecture, we use two fully connected layers, each with 256 hidden units and ReLU activation. We use a replay ratio of 1, where we first sample 1000 environment steps and then train for the same number of gradient steps. Our replay buffer has a capacity of 1,000,000 samples and we set $\lambda_Q = \lambda_\pi = \lambda_\alpha = 3.0 \times 10^{-4}$, $\tau_{\text{targ}} = 0.995$, and the batch size to 128. In the DeepMind control suite experiments we set $\Gamma_{\text{CR}} = -50$ with $m = 20$ (in [k]) for both RC-TD3 and RC-SAC, while we use $\gamma = 0.999$ for RC-TD3 and $\gamma = 0.99$ for RC-SAC with a maximum of 1000 steps per episode. For the mobile robot we use $\gamma = 0.99$, $m = 20$, $\Gamma_{\text{CR}} = -6$ for RC-TD3 and $\Gamma_{\text{CR}} = -20$ for RC-SAC with a maximum of 300 steps per episode. We utilized Nvidia T4 and A40 GPUs as computational resources. All results are computed using 4 random seeds. Further, we report the rewards (unless stated otherwise) with the fixed values $w_c = 1.0$ for DM Control and $w_c = 0.1$ for the mobile robot case, even though w_c might have different values during training. We do this in order to make runs with different w_c comparable. We report

the normalized episode reward unless stated otherwise, such that 0 is the worst and 1 is the best.

V. RESULTS & DISCUSSION

A. DM Control

We test our method for $w_c \in \{0.0, 0.5, 1.0\}$ on the DeepMind Control Suite environments pendulum swingup, fish upright, fish swim, reacher hard, reacher easy, finger turn hard, finger turn easy, finger spin, ball in cup catch, cartpole balance, cheetah run, cartpole swingup, walker stand, walker walk and walker run. We train for 2,000,000 steps. In Fig. 1 a) we show the mean episode reward (calculated with $w_c = 1.0$ for comparability) obtained with RC-TD3 on the Y-axis versus the episode reward with TD3 on the X-axis. A table with mean values and standard deviations can be found in the Appendix III. An X above the diagonal indicates that the curriculum improves performance, and below implies the opposite. The results are averaged over the last 50,000 steps and 4 random seeds. As expected, both methods perform similarly when $w_c = 0.0$. However, when w_c increases, the curriculum becomes more effective. This is especially the case for environments that do not automatically optimize the constraint by completing the task. For instance, in cartpole balance it is beneficial for the task to predict small actions, thus a curriculum does not have a sizable effect. However, for tasks like finger spin, adding the constraints seems to hinder learning the task, leading to suboptimal policies with TD3 as w_c increases. In those cases, where the task can be learned but constraints substantially impact learnability, our curriculum method proves to be most effective (i.e. environments with episode reward around the range from -100 to 500). Furthermore, there is one group in the bottom left, where the policy does not successfully learn the problem without constraints. In this case, the curriculum does not have any

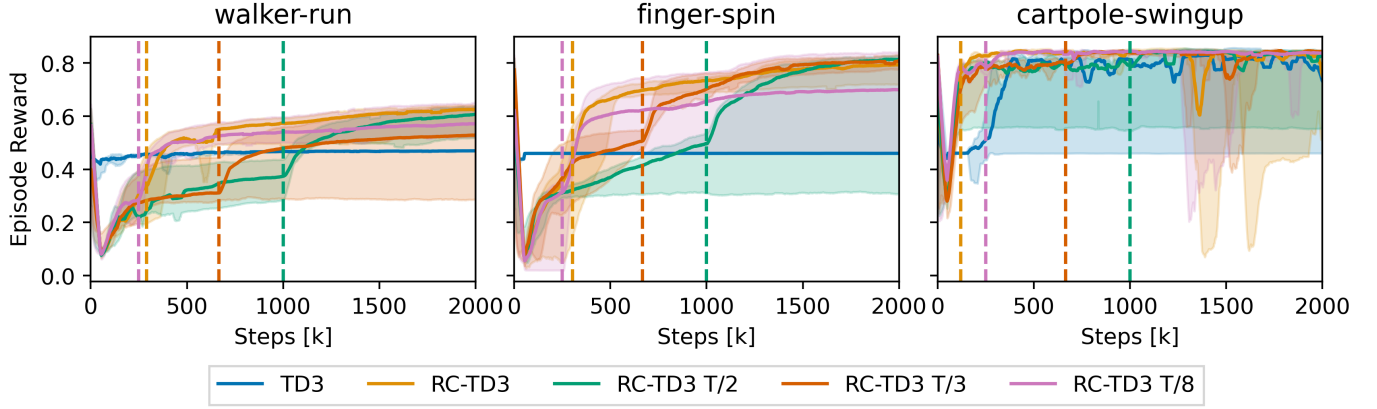


Fig. 5: Comparison of the median episode reward of RC-TD3 with automatic curriculum switch as described in Section III-C to switching at times $T/8$, $T/3$, and $T/2$ where T are the total number of timesteps. The values are smoothed by taking the running average with window size 50 [k] and the dashed lines indicate the time each curriculum switched phases. The results indicate that for some environments such as cartpole swingup, the exact time does not seem to matter while the automatic switch manages to pick a more beneficial time in the case of walker run. Importantly, all reward curriculum versions work better than not employing a curriculum and our method, RC-TD3, yields one of the best results in all cases.

effect and learning remains unsuccessful. In Fig. 1 b) we show the base reward r_b , thus the reward that only encodes the task without any constraints for both TD3 and RC-TD3. As can be seen, the reward curriculum becomes more beneficial as the constraint weight w_c increases. Importantly, it can be seen that in the case of $w_c = 1.0$ in Fig. 1 b) the reward curriculum performs equal or better for nearly all cases, indicating that it indeed manages to learn the task even with strong constraints. In contrast, the baseline does not learn the problem at all in many cases, such as for finger spin, walker run or cheetah run. Therefore, the majority of the episode reward for the cases where the baseline does not learn the task comes from only optimizing the constraints. Overall, the results confirm that the reward curriculum effectively prevents getting stuck in the local minima of only optimizing the constraints, and continues to enable effective task learning. Consistent with this analysis, comparisons of RC-SAC against SAC exhibit similar trends (see Appendix 8), thereby substantiating the efficacy of our proposed approach

To better understand the impact of each component of our method, we perform the following ablation studies. There are two ways in which the reward curriculum can improve performance: i) the agent learned the task in the first phase, which serves as a helpful prior in the second phase, or, ii) the replay buffer with updated rewards helps to stabilize training and to find high reward regions. In order to test the hypotheses we perform experiments on environments where the curriculum was effective, namely on finger spin, walker run, and cartpole swingup. In the first experiment, we compare either resetting the networks or resetting the replay buffer after switching curriculum phases to our method where we keep both. The results are shown in Fig. 4. As can be seen, resetting the weights substantially decreases performance while resetting the buffer has little influence. Therefore, we

conclude that the main benefit of our method for those environments is to effectively “pretrain” the networks which help find high-reward regions. Note that the high standard deviations in the cartpole swingup environment are due to training instabilities, where the agent temporarily unlearns the task. These instabilities occurred later in training, after the curriculum phase had changed, indicating that they are not specific to our method.

The second ablation study focuses on testing the effectiveness of the automatic curriculum switch. We make use of the same environment subset as before, finger spin, walker run, and cartpole swingup. The proposed automatic switch to change the curriculum phase is compared to statically switching at times at $T/2$, $T/3$, and $T/8$. Fig. 5 illustrates the results. As can be seen, our automatic switch leads to equal or better results than switching at a fixed time while converging to better solutions earlier, thus being more sample-efficient. Table I shows the average switch steps (in [k]) of our method, showing that in most cases phases are switched relatively early, except when learning is challenging. For instance, in finger turn hard it either switches phases late or not at all in the given timesteps, indicating that it did not successfully learn the problem. We observe that the times where the curriculum does not switch phases correspond to an agent not learning the task. Similar training instabilities in some cases as observed before for cartpole swingup also occur in this experiment.

B. Mobile Robot

While there might be cases in which a non-intuitive reward subset to take as base reward r_b is best, we found that simply using the terms that encourage the agent to only learn the task works best (See Appendix A for more details). Specifically, we chose the base reward as $r_b = r_g + w_p r_p$, and all other reward terms become part of the constraint reward, thus

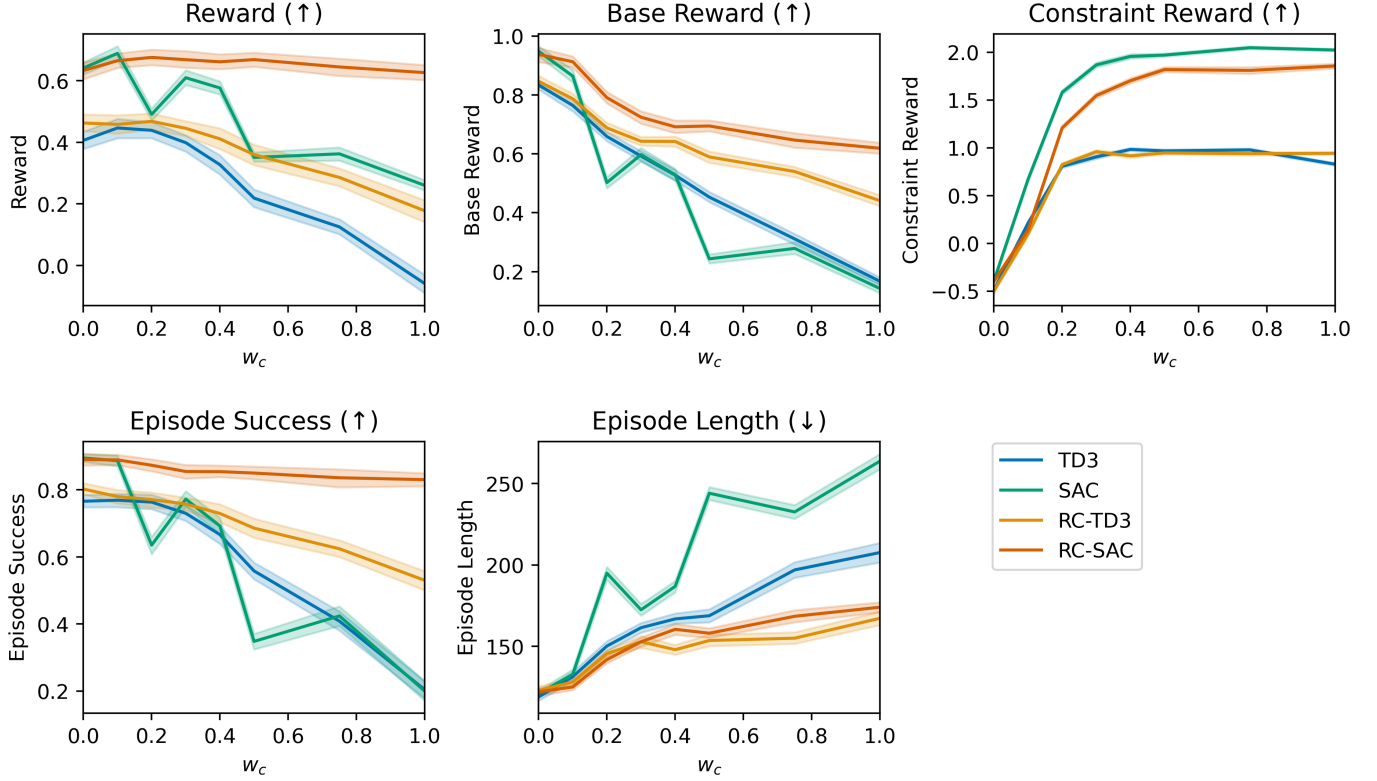


Fig. 6: Performance of TD3, SAC, RC-TD3 and RC-SAC for the mobile robot environment. Results are averaged over the last 50,000 training timesteps and 4 seeds. Both RC-TD3 and RC-SAC consistently outperform TD3 and SAC across all metrics, indicating the effectiveness of a reward curriculum for constraint environments. Interestingly, RC-SAC and SAC manage to better optimize the constraints compared to the TD3 versions. Overall, RC-SAC performs the best and manages to retain high success rates across all constraint weights.

Environment	$w_c = 0.0$	$w_c = 0.5$	$w_c = 1.0$
fish-upright	118 ± 17	101 ± 14	102 ± 7 *
finger-turn_hard	1224 ± 580	735 ± 571	1220 ± 730 †
cheetah-run	270 ± 29	305 ± 50	293 ± 33 *
fish-swim	252 ± 60	273 ± 78	265 ± 75 *
walker-walk	192 ± 18	190 ± 47	186 ± 30
walker-stand	122 ± 4 †	128 ± 10	122 ± 11
walker-run	321 ± 64	279 ± 67	290 ± 48
pendulum-swingup	331 ± 76 *	313 ± 100 *	488 ± 54 *
reacher-hard	314 ± 144	300 ± 171	316 ± 146
reacher-easy	104 ± 44	124 ± 73	93 ± 39
finger-spin	669 ± 670	255 ± 71	303 ± 95
ball_in_cup-catch	396 ± 368	194 ± 80	227 ± 99 *
cartpole-balance	83 ± 3	84 ± 1	82 ± 2
cartpole-swingup	124 ± 15	122 ± 8	120 ± 8
finger-turn_easy	406 ± 70	359 ± 50	331 ± 73

TABLE I: Number of steps [k] when the curriculum phase switches. In certain cases, the second phase was never initiated which is shown by * if $\frac{3}{4}$ and † is $\frac{2}{4}$ runs switched to the second phase. The few times when the curriculum did not switch phases corresponded to unsuccessful initial training.

$r_c = w_c(r_a + r_x + r_v)$. The experiments are repeated for the constraint weights $w_c = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.0\}$ and train for $T = 1,000,000$ steps. We choose these values

as we expect more refined changes for smaller w_c . In the case of TD3, we start with $\sigma = 0.9$ and anneal it to $\sigma = 0.1$ over the first T/2 frames for additional exploration.

We present the results of TD3, SAC, RC-TD3, and RC-SAC, averaged over the last 50,000 training steps, as a function of the selected constraint weights in Fig. 6. To ensure comparability, the reward is calculated using a fixed $w_c = 0.1$, while the other terms are independent of w_c . Consistent with observations in the DeepMind Control Suite, the reward curriculum exhibits increased effectiveness with growing w_c . While the constraint rewards r_c are relatively similar between the reward curriculum versions and their baseline, the curriculum versions more effectively learn the task, indicated by an increased base reward r_b . Moreover, SAC and RC-SAC outperform TD3 and RC-TD3 in terms of initial success rate and constraint optimization. Interestingly, RC-SAC achieves a substantially higher success rate, particularly for large w_c , surpassing RC-TD3 and yielding the best results in this environment. One reason for this could be that RC-SAC on average switched phases at 477 ± 96 while RC-TD3 switched at 170 ± 44 , indicating that switching later might be beneficial. In addition, the results demonstrated that optimizing the constraints indeed conflicts with task performance, as

they exhibit opposing trends. Overall, the value $w_c = 0.2$ seems to lead to the highest reward for both RC-TD3 and RC-SAC. These findings corroborate our earlier results and demonstrate the efficacy of our approach in achieving higher rewards compared to the baselines, especially in the presence of strong constraints.

VI. RELATED WORK

Reward shaping is a well-established technique to facilitate sample-efficient learning, with a history spanning decades [8, 27]. The most prevalent approach is potential-based reward shaping, which preserves the policy ordering [24]. Building on this foundation, recent work has explored the concept of automatic reward shaping, where the shaping process is learned and adapted during training. For instance, Hu et al. [15] formulate reward shaping as a bi-level optimization problem, where the high-level policy optimizes the true reward and adaptively selects reward shaping weights for the lower-level policy. This approach has been successfully applied to various tasks with sparse rewards, effectively creating an automatic reward-shaping curriculum. Another line of research focuses on building automatic reward curricula through intrinsic motivation, encouraging exploration of unexpected states [2, 25, 6, 32]. However, these methods primarily modify the shaping terms while keeping the ground-truth rewards fixed.

Although less prevalent, several works have employed reward curricula in reinforcement learning (RL) using subsets of the true reward. For instance, Hwangbo et al. [16] utilize an exponential reward curriculum, initially focusing on locomotion and gradually increasing the weight of other cost terms to refine the behavior. Similarly, Leyendecker et al. [20] observe that RL agents optimized on complex reward functions with multiple constraints are highly sensitive to individual reward weights, leading to local optima when directly optimizing the full reward. They successfully learn a policy by gradually increasing constraint weights based on task success. Furthermore, Pathare et al. [26] employ a three-stage reward curriculum, incrementally increasing complexity and realism by incorporating additional terms. However, they find reward curriculum learning largely ineffective. To the best of our knowledge, we are the first to systematically investigate a two-stage curriculum for complex rewards, leveraging sample-efficient experience transfer between phases to mitigate reward exploitation.

VII. LIMITATIONS

While the proposed reward curriculum shows promising results, there are still remaining open questions. For instance, in some environments it happens that a successful policy in the first phase deteriorates in performance in the second one. Future work could investigate more stable phase switches while retaining enough flexibility to effectively optimize constraints. Another issue is that currently our method depends on the specific value selected as lower threshold for when an actor sufficiently fits a Q -function. This greatly impacts when and if phases are switched, and thus also the final performance.

Furthermore, we do not evaluate our method in experiments using real robots. While the essence of the problem should be comparable, it might bring in new aspects that are challenging to anticipate in simulation.

VIII. CONCLUSION

In this work, we present a two-stage reward curriculum, where we first train an RL agent with on an easier subset of rewards and switch to training using the full reward after a certain time. We introduce a flexible replay buffer that adaptively changes rewards to reuse samples from one phase in the next. Further, we provide a mechanism to automatically change phases. In extensive experiments we show that our method outperforms the baseline only trained on the full reward in almost all cases. It is especially successful for high constraint weights and environments where the constraints substantially hinder learning the task. We believe that this work contributes towards developing more stable RL methods to effectively learn challenging objectives. For future work, methods to automatically detect a feasible reward subset for the first phase could be developed and further evaluations in real-world experiments are required.

ACKNOWLEDGMENTS

The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. This work is supported by the Vinnova project AIHURO (Intelligent human-robot collaboration) and the ITEA project ArtWork.

REFERENCES

- [1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017. URL <https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html>.
- [2] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016. URL <https://proceedings.neurips.cc/paper/2016/hash/afda332245e2af431fb7b672a68b659d-Abstract.html>.
- [3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. URL <https://dl.acm.org/doi/abs/10.1145/1553374.1553380>.
- [4] Serena Booth, W Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications. In *Proceedings*

- of the AAAI Conference on Artificial Intelligence, volume 37, pages 5920–5929, 2023. URL <https://ojs.aaai.org/index.php/AAAI/article/view/25733>.
- [5] Michael Bowling, John D Martin, David Abel, and Will Dabney. Settling the reward hypothesis. In *International Conference on Machine Learning*, pages 3003–3020. PMLR, 2023. URL <https://proceedings.mlr.press/v202/bowling23a.html>.
 - [6] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018. URL <https://arxiv.org/abs/1810.12894>.
 - [7] Kristian Ceder, Ze Zhang, Adam Burman, Ilya Kungaliyev, Krister Mattsson, Gabriel Nyman, Arvid Petersén, Lukas Wisell, and Knut Akesson. Bird’s-Eye-View Trajectory Planning of Multiple Robots using Continuous Deep Reinforcement Learning and Model Predictive Control. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024. URL <https://ieeexplore.ieee.org/abstract/document/10801434>.
 - [8] Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370, 1994. URL <https://www.sciencedirect.com/science/article/pii/0004370294900477>.
 - [9] Meng Fang, Tianyi Zhou, Yali Du, Lei Han, and Zhengyou Zhang. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems*, 32, 2019. URL <https://proceedings.neurips.cc/paper/2019/hash/83715fd4755b33f9c3958e1a9ee221e1-Abstract.html>.
 - [10] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. URL <https://proceedings.mlr.press/v80/fujimoto18a.html>.
 - [11] Shixiang Gu, Ethan Holly, Timothy P Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 1:1, 2016. URL <https://arxiv.org/abs/1610.00633>.
 - [12] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018. URL <https://arxiv.org/abs/1812.05905>.
 - [13] Dong Han, Beni Mulyana, Vladimir Stankovic, and Samuel Cheng. A survey on deep reinforcement learning algorithms for robotic manipulation. *Sensors*, 23(7): 3762, 2023. URL <https://www.mdpi.com/1424-8220/23/7/3762>.
 - [14] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/tssc.1968.300136. URL <https://doi.org/10.1109/tssc.1968.300136>.
 - [15] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33:15931–15941, 2020. URL <https://proceedings.neurips.cc/paper/2020/hash/b710915795b9e9c02cf10d6d2bdb688c-Abstract.html>.
 - [16] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019. URL <https://www.science.org/doi/abs/10.1126/scirobotics.aau5872>.
 - [17] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pages 651–673. PMLR, 2018. URL <https://proceedings.mlr.press/v87/kalashnikov18a>.
 - [18] W Bradley Knox and James MacGlashan. How to Specify Reinforcement Learning Objectives. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*, 2024. URL <https://openreview.net/forum?id=2MGEQNrmDN>.
 - [19] W Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis) design for autonomous driving. *Artificial Intelligence*, 316:103829, 2023. URL <https://www.sciencedirect.com/science/article/pii/S0004370222001692>.
 - [20] Lars Leyendecker, Markus Schmitz, Hans Aoyang Zhou, Vladimir Samsonov, Marius Rittstiege, and Daniel Lütticke. Deep Reinforcement Learning for Robotic Control in High-Dexterity Assembly Tasks—A Reward Curriculum Approach. *International Journal of Semantic Computing*, 16(03):381–402, 2022. URL <https://www.worldscientific.com/doi/abs/10.1142/S1793351X22430024>.
 - [21] TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. URL <https://arxiv.org/abs/1509.02971>.
 - [22] Sanmit Narvekar and Peter Stone. Learning Curriculum Policies for Reinforcement Learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’19*, page 25–33, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099. URL <https://arxiv.org/abs/1812.00285>.
 - [23] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50, 2020. URL <https://www.jmlr.org/papers/v21/20-212.html>.
 - [24] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations:

- Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999. URL https://www.teach.cs.toronto.edu/~csc2542h/fall/material/csc2542f16_reward_shaping.pdf.
- [25] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017. URL <https://proceedings.mlr.press/v70/pathak17a.html?ref=https://githubhelp.com>.
- [26] Deepthi Pathare, Leo Laine, and Morteza Haghir Chehreghani. Tactical Decision Making for Autonomous Trucks by Deep Reinforcement Learning with Total Cost of Operation Based Reward. *arXiv preprint arXiv:2403.06524*, 2024. URL <https://arxiv.org/abs/2403.06524>.
- [27] Jette Randløv and Preben Alstrøm. Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *ICML*, volume 98, pages 463–471, 1998. URL <https://dl.acm.org/doi/10.5555/645527.757766>.
- [28] Alejandro Rodriguez-Ramos, Carlos Sampedro, Hriday Bave, Ignacio Gil Moreno, and Pascual Campoy. A deep reinforcement learning technique for vision-based autonomous multirotor landing on a moving platform. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1010–1017. IEEE, 2018. URL <https://ieeexplore.ieee.org/abstract/document/8594472>.
- [29] Francisco R Ruiz, Michalis K Titsias, and David M Blei. The generalized reparameterization gradient. *Advances in neural information processing systems*, 29, 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/hash/f718499c1c8cef6730f9fd03c8125cab-Abstract.html.
- [30] Hanying Sang and Shuquan Wang. Motion planning of space robot obstacle avoidance based on DDPG algorithm. In *2022 International Conference on Service Robotics (ICoSR)*, pages 175–181. IEEE, 2022. URL <https://ieeexplore.ieee.org/abstract/document/10136963>.
- [31] Terence D Sanger. Neural network learning control of robot manipulators using gradually increasing task difficulty. *IEEE transactions on Robotics and Automation*, 10(3):323–333, 1994. URL <https://ieeexplore.ieee.org/abstract/document/294207>.
- [32] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR, 2019. URL <https://proceedings.mlr.press/v97/shyam19a.html>.
- [33] Burrhus F Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958. URL <https://psycnet.apa.org/record/1959-07839-001>.
- [34] Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.
- [35] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018. URL <https://arxiv.org/abs/1801.00690>.
- [36] Ze Zhang, Yao Cai, Kristian Ceder, Arvid Enliden, Osian Eriksson, Soleil Kylander, Rajath Sridhara, and Knut Åkesson. Collision-Free Trajectory Planning of Mobile Robots by Integrating Deep Reinforcement Learning and Model Predictive Control. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–7. IEEE, 2023. URL <https://ieeexplore.ieee.org/abstract/document/10260515>.
- [37] Kai Zhu and Tao Zhang. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, 26(5):674–691, 2021. URL <https://ieeexplore.ieee.org/abstract/document/9409758>.
- [38] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017. URL <https://ieeexplore.ieee.org/abstract/document/7989381>.
- [39] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010. URL <http://proxy.lib.chalmers.se/login?url=https://www.proquest.com/dissertations-theses/modeling-purposeful-adaptive-behavior-with/docview/845728212/se-2?accountid=10041>.

APPENDIX

A. Mobile robot base reward

An important assumption underlying our approach is that training on a subset of rewards is easier than training on the full reward function. However, while in the case of the DeepMind control suite it was relatively clear which initial reward to use, it might be that some subset of initial rewards r_b converges faster than others. To empirically validate this, we perform experiments on different subsets of the shaped reward function. Specifically, we consider the following subsets:

$$r_b \in \{r_{\text{full}}, r_{\text{gp}}, r_{\text{gpc}}, r_{\text{gpv}}, r_{\text{gpa}}, r_{\text{gpx}}\}$$

where

$$r_{\text{gp}} = r_g + r_p,$$

$$r_{\text{gpv}} = r_g + r_p + r_v,$$

$$r_{\text{gpa}} = r_g + r_p + r_a,$$

$$r_{\text{gpx}} = r_g + r_p + r_x.$$

We train for $T = 300\,000$ environment steps on the randomized maps in Figure 3 over 4 seeds for each subset with $w_c = 0.5$. Fig. 7 shows the task success rate for each subset over the training steps. It can be concluded that non of the constraints seems to help learning the task and in fact make learning more challenging. Therefore, this confirms that only training on the reward terms that clearly contribute to learning the task is the most beneficial subset for this environment.

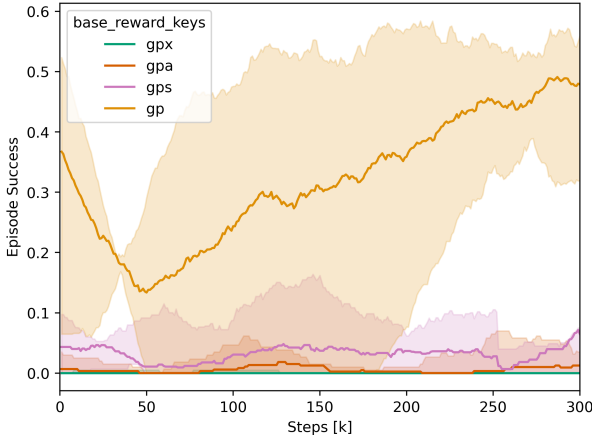


Fig. 7: Success rate for the mobile robot during training using TD3 with different initial reward subsets for $w_c = 0.5$. As it can be seen, adding any constraint lead to worse performance than only taking the reward terms that intuitively contribute to learning the task.

B. Additional Results

Furthermore, we show results obtained with RC-SAC compared to the baseline SAC. A comparison of episode rewards can be seen in Fig. 8. Detailed values can be found in Table II. The trends are similar to RC-TD3, improvements of RC-SAC

	$w_c = 0.5$		$w_c = 1.0$	
	SAC	RC-SAC	SAC	RC-SAC
walker-walk	622 \pm 63	655 \pm 38	578 \pm 99	648 \pm 96
walker-stand	785 \pm 16	777 \pm 45	787 \pm 16	792 \pm 15
walker-run	-15 \pm 51	165 \pm 35	-103 \pm 42	196 \pm 45
fish-upright	680 \pm 146	666 \pm 194	500 \pm 387	492 \pm 391
pendulum-swingup	<u>-255 \pm 478</u>	-419 \pm 561	<u>-8 \pm 254</u>	-131 \pm 467
fish-swim	<u>-26 \pm 179</u>	-314 \pm 286	<u>-65 \pm 119</u>	-262 \pm 296
reacher-hard	<u>804 \pm 142</u>	785 \pm 192	814 \pm 135	<u>817 \pm 104</u>
reacher-easy	<u>834 \pm 101</u>	831 \pm 97	834 \pm 151	<u>839 \pm 105</u>
finger-turn_hard	<u>27 \pm 369</u>	-152 \pm 610	<u>-39 \pm 419</u>	-288 \pm 468
ball_in_cup-catch	851 \pm 19	851 \pm 19	407 \pm 781	853 \pm 19
finger-turn_easy	<u>384 \pm 492</u>	28 \pm 539	<u>293 \pm 488</u>	11 \pm 576
finger-spin	-118 \pm 2	545 \pm 82	-118 \pm 2	630 \pm 55
cartpole-balance	25 \pm 891	<u>914 \pm 2</u>	911 \pm 10	<u>914 \pm 2</u>
cartpole-swingup	268 \pm 690	<u>725 \pm 2</u>	<u>-76 \pm 4</u>	-79 \pm 2
cheetah-run	315 \pm 64	<u>325 \pm 81</u>	177 \pm 79	<u>236 \pm 42</u>

TABLE II: Mean and standard deviation of the episode reward for SAC and RC-SAC for the DeepMind control suite over 4 random seeds. Bold values indicate that one method was clearly better (i.e. the standard deviations did not overlap) and underlines indicate that the mean value was higher.

over SAC become more pronounced for higher w_c and are more relevant for some environments than others. Importantly, the environments where RC-SAC is effective correspond to the ones of RC-TD3. Detailed values of the mean and standard deviations can be seen in Table III.

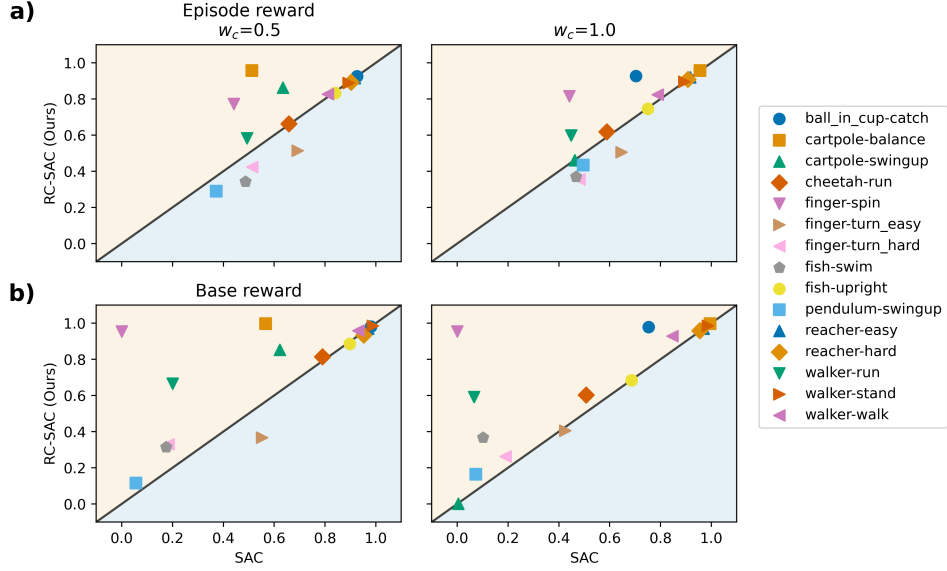


Fig. 8: a) Episode reward averaged over 4 random seeds for RC-SAC on the x axis and SAC on the y axis. b) Base reward for the same setup as in before, which corresponds to task performance without considering constraints. Overall, the outcomes are very similar to the ones observed for RC-TD3 and TD3.

	$w_c = 0.0$		$w_c = 0.5$		$w_c = 1.0$	
	TD3	RC-TD3	TD3	RC-TD3	TD3	RC-TD3
fish-upright	<u>60 ± 95</u>	12 ± 122	<u>508 ± 253</u>	381 ± 249	626 ± 176	<u>634 ± 174</u>
finger-turn_hard	<u>-788 ± 319</u>	-828 ± 260	-810 ± 234	<u>-642 ± 355</u>	-758 ± 298	<u>-630 ± 396</u>
finger-turn_easy	<u>-700 ± 405</u>	-713 ± 391	-697 ± 384	<u>-640 ± 362</u>	-732 ± 335	<u>-579 ± 361</u>
cheetah-run	-348 ± 342	<u>-155 ± 164</u>	<u>-77 ± 1</u>	-169 ± 276	-77 ± 1	<u>92 ± 227</u>
fish-swim	-817 ± 170	-850 ± 123	<u>-824 ± 142</u>	-837 ± 142	<u>-801 ± 111</u>	-830 ± 135
walker-walk	<u>226 ± 82</u>	218 ± 86	462 ± 321	<u>538 ± 123</u>	391 ± 447	613 ± 98
walker-stand	<u>300 ± 83</u>	263 ± 102	<u>780 ± 41</u>	753 ± 66	783 ± 128	<u>799 ± 105</u>
walker-run	-301 ± 72	-154 ± 67	-3 ± 24	84 ± 57	-61 ± 5	202 ± 126
pendulum-swingup	<u>283 ± 277</u>	26 ± 455	-16 ± 232	<u>299 ± 443</u>	-22 ± 219	<u>41 ± 475</u>
reacher-hard	<u>-656 ± 397</u>	-809 ± 267	-857 ± 167	<u>-823 ± 179</u>	-845 ± 155	<u>-738 ± 302</u>
reacher-easy	<u>137 ± 401</u>	-91 ± 537	<u>116 ± 600</u>	65 ± 634	-264 ± 689	<u>86 ± 749</u>
finger-spin	<u>191 ± 34</u>	14 ± 198	-80 ± 1	501 ± 75	-80 ± 1	561 ± 74
ball_in_cup-catch	110 ± 612	<u>400 ± 228</u>	766 ± 306	<u>884 ± 22</u>	<u>867 ± 34</u>	439 ± 760
cartpole-balance	704 ± 186	<u>779 ± 152</u>	<u>758 ± 292</u>	756 ± 306	798 ± 226	<u>847 ± 73</u>
cartpole-swingup	<u>571 ± 144</u>	569 ± 94	<u>646 ± 84</u>	209 ± 636	388 ± 332	<u>429 ± 357</u>

TABLE III: Mean and standard deviation of the episode reward for TD3 and RC-TD3 for the DeepMind control suite over 4 random seeds. Bold values indicate that one method was clearly better (i.e. the standard deviations did not overlap) and underlines indicate that the mean value was higher.