Curriculum Reinforcement Learning for Complex Reward Functions

Kilian Freitag[1] **Kristian Ceder**[1] **Rita Laezza**[1] **Knut Åkesson**[1] **Morteza Chehreghani**[2]
[1]Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden
[2]Department of Computer Science
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
{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 "reward hacking" 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 simpler subset of the 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 additional auxiliary terms in the reward definitions and a mobile robot scenario with several 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 auxiliary terms in the reward, it is able to learn policies that balance task completion and auxiliary objective satisfaction.

1 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 [Gu et al., 2016, Kalashnikov et al., 2018, Leyendecker et al., 2022, Han et al., 2023] or mobile robotics [Lillicrap, 2015, Zhu et al., 2017, Rodriguez-Ramos et al., 2018, Zhu and Zhang, 2021, Sang and Wang, 2022]. 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,

18th European Workshop on Reinforcement Learning (EWRL 2025).

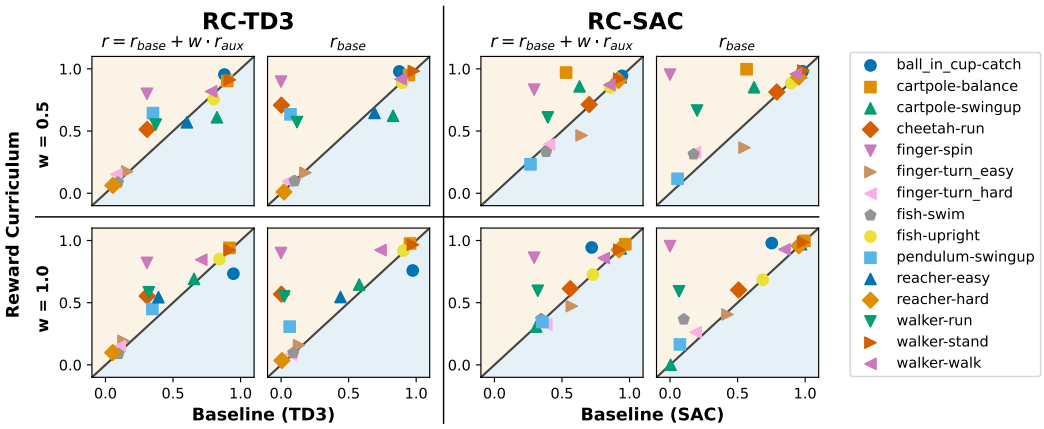


Figure 1: Comparison of the baseline TD3 and SAC algorithms with reward curriculum versions (RC-TD3 and RC-SAC). An icon above the diagonal means that the curriculum version outperforms the baseline. The first columns show the normalized mean episode reward visualized using auxiliary weight $w = 0.5$ and the second columns show the achieved base reward. The rows correspond to the w used during training.

maintaining a specific velocity, and ensuring a smooth trajectory [Zhang et al., 2023, Ceder et al., 2024]. 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 [Sutton, 2018, Bowling et al., 2023]. However, formulating an effective reward function for complex tasks is non-trivial and can often result in undesired behaviors [Booth et al., 2023, Knox et al., 2023, Knox and MacGlashan, 2024]. 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 learning the intended task, which is referred to as reward hacking. We call such potentially conflicting objectives complex reward functions, where the complexity lies in the presence of strong local optima for undesired behaviors. Notably, our setting focuses on a single reward function, distinct from multi-objective reinforcement learning (MORL) which typically considers a set of Pareto-optimal solutions for multiple objectives [**?**].

A line of work that has emerged to tackle challenging RL problems is curriculum learning [Bengio et al., 2009, Narvekar et al., 2020]. Inspired by how animals can be trained to learn new skills [Skinner, 1958], learning proceeds from simple problems to gradually more challenging ones. In the realm of RL, such curricula are often hand-designed and have been successfully employed in several applications in robotic control [Sanger, 1994, Hwangbo et al., 2019, Leyendecker et al., 2022]. More recently several methods for automatic curriculum design have emerged that are able to learn curriculum policies [Narvekar and Stone, 2019] or that break down problems into smaller ones [Andrychowicz et al., 2017, Fang et al., 2019] for more effective learning. Further, automatic curricula are used in sparse or no reward settings as intrinsically motivated exploration that encourages reaching diverse states [Bellemare et al., 2016, Pathak et al., 2017, Burda et al., 2018, Shyam et al., 2019]. Nevertheless, such methods have been explored less for 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 auxiliary objectives. 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 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 (DM) 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 auxiliary rewards, 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 auxiliary reward weights w. The complexity lies in finding a solution that maximally satisfies auxiliary terms 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 for higher weights of auxiliary terms. Especially for environments where the auxiliary terms encourage reward hacking, our method proves most useful and prevents getting stuck with undesired behaviors (see Fig. 1). In an ablation study we show that the "pretrained" network weights from the first phase help find better solutions and that the flexible replay buffer becomes important in the more challenging mobile robot environment. Furthermore, we 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 i) reusing past experiences with updated rewards (in order to improve sample-efficiency), and ii) 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 DM control suite environments and discuss in which environments a reward curriculum is most effective. Further we extensively evaluate our method in a mobile robot navigation problem with several conflicting reward terms.
3. In ablation studies, we investigate the impact of the key components of our method, including the automatic switching mechanism and flexible replay buffer, as well as the effect of switching between reward functions during training.

2 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, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function, $p_0 : \mathcal{S} \to [0, 1]$ 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 parameterized by weights ϕ, and denoted as $Q_\phi(s, a)$.

In our case, we consider any control problem with several reward terms that are available separately, where each can be categorized either as a base reward r_{base} if it helps learning the goal (for instance reward shaping terms), and auxiliary rewards r_{aux}, which specify the desired behavior. The reward is then given as the weighted sum with $w \in [0, 1]$

$$r = r_{\text{base}} + w \cdot r_{\text{aux}} \tag{1}$$

The goal is to develop a method that allows learning a policy $\pi_\theta(a|s)$ parametrized by weights θ that learns to complete tasks (encoded by r_{base}) while maximally satisfying the auxiliary objectives r_{aux} and being robust to different reward weights w. Intuitively, the challenge lies in finding policies that learn the task without reward hacking.

3 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) [Haarnoja et al., 2018] and the other based on Twin-Delayed DDPG (TD3) [Fujimoto et al., 2018], 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_{base}. In the second phase, we then directly optimize the full reward $r = r_{\text{base}} + w \cdot r_{\text{aux}}$. Details about RC-TD3 and RC-SAC can be found in Appendix A.

Formally, we define the curriculum reward as

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

where $\mathcal{CR}$ denotes the index of the current curriculum phase. Algorithm 1 in Appendix B describes the two-stage curriculum in detail.

3.1 Reuse of past experience

A key aspect of our method is leveraging past experiences after a phase switch. Specifically, we store tuples of the form $\{(s_t, a_t, r_{\text{base}}, r, s_{t+1})\}$ in the replay buffer. When transitioning to the second phase, we reuse experiences from the first phase by now utilizing reward r for training instead of r_{base}. This mechanism aims to stabilize training by populating the replay buffer with a diverse set of samples, thereby facilitating sample-efficient learning. Notably, this approach is only compatible with off-policy RL algorithms, as it relies on the ability to learn from stored experiences.

3.2 Automatic phase switch

Another key aspect of the reward curriculum is to find appropriate times to transition 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 the other terms 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 actor loss 5, for SAC it would be the actor loss 6 but without the additional entropy term. If the actor fits the critic well for m timesteps, i.e. $\forall i \in \{t-k+1, \ldots, t\} : J_{\pi,Q;i} < \Gamma_{\text{CR}}$ where Γ_{CR} is the threshold that defines a "good" fit, we switch to the second curriculum phase.

4 Environments

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

4.1 DM Control Suite

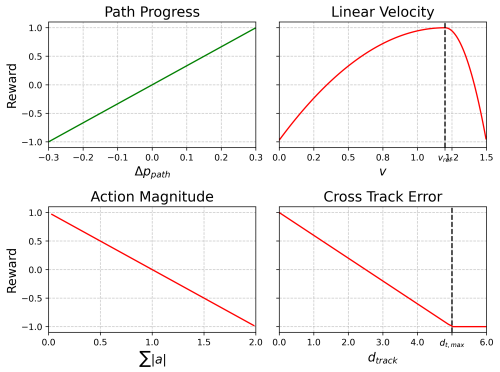
The DM Control Suite [Tassa et al., 2018] is a well-known collection of control environments in RL. We use the state-space observations and the environments have an action space $\mathcal{A} \in [-1, 1]^2$. The original rewards are in the range $[0, 1]$, which we take as base reward r_{base}. Furthermore, we introduce the following reward term to minimize action magnitudes, to find efficient solutions to the control problems

$$r_{\text{aux}} = -\sum_{i=1}^{d} |a_i| \tag{3}$$

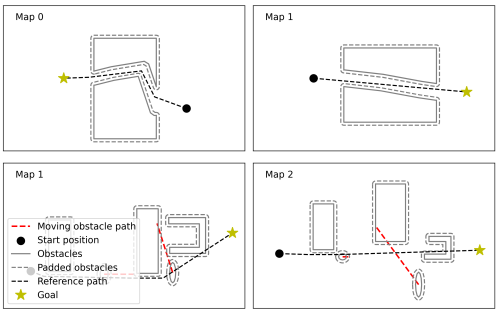
where d is the number of actions of each environment. This is used as an auxiliary term in the curriculum reward (see Equation 2).

4.2 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 other objectives. The environment maps are randomized and contain both permanent (e.g., walls and corridors) and temporary (e.g., dynamic or static) obstacles. A



(a) 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 other penalizing terms are colored in red.



(b) 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.

Figure 2: Dense reward functions and training environments

subset of exemplary environment maps can be found in Figure 2b. Furthermore, a reference path that considers only permanent obstacles is computed using A* [Hart et al., 1968]. 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 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. The objectives include reaching a goal position while driving at a reference velocity, staying close to the reference path, and creating smooth trajectories. Thus, 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 can be expressed as dense terms evaluated at each step. To enable intuitive weighting, we normalize each term between $[-1, 1]$. We chose this range over $[-1, 0]$ to discourage the policy from learning to crash immediately.

To achieve smooth trajectories we encourage minimizing accelerations. As this corresponds to our action space, we penalize high action values, similar to the DM control experiments, as

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

Further, to drive 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:

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

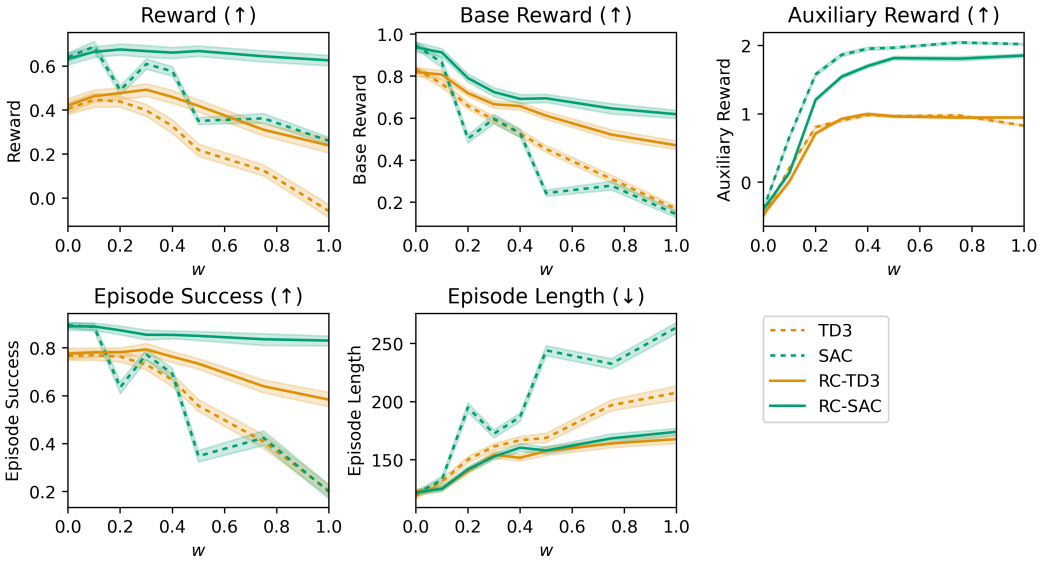


Figure 3: Performance of TD3, SAC, RC-TD3 and RC-SAC for the mobile robot environment. Results are averaged over the last $50,000$ training timesteps and 4 seeds. Both RC-TD3 and RC-SAC consistently outperform TD3 and SAC across all metrics, indicating the effectiveness of a reward curriculum. Interestingly, RC-SAC and SAC manage to better optimize the auxiliary rewards compared to RC-TD3 and TD3. Overall, RC-SAC performs the best and manages to retain high success rates across all w.

where $\kappa \in [0, 1]$ 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 Ng et al. [1999], by encouraging progress along the reference path, through

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

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 Ng et al. [1999]. Figure 2a gives an overview of the dense reward terms used.

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

$$r = r_g + w_p r_p + w(r_v + r_a + r_x) \tag{4}$$

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

While selecting which reward terms should belong to r_{base} and r_{aux} was straight-forward in the case of the modified DM control suite, it becomes less clear when having more than two objectives such as in the case of the mobile robot. However, we found that simply using the terms that encourage the agent to only learn the task as r_{base} (See Appendix D for more details). Specifically, we chose the base reward as $r_{\text{base}} = r_g + w_p r_p$, and all other reward terms become part of the auxiliary reward, thus $r_{\text{aux}} = w(r_a + r_x + r_v)$.

5 Results & Discussion

Results are computed over four random seeds and to facilitate comparison across different training configurations, we report the rewards with fixed weights $w = 0.5$ for DM Control and $w = 0.1$ for the mobile robot environment. Note that these weights are used solely for evaluation purposes and differ from the reward weights used during training. We report the normalized episode reward, where 0 corresponds to the worst performance and 1 corresponds to the best performance. Details can be found in Appendix C.

5.1 DM Control

We test our method for $w \in \{0.5, 1.0\}$ on the DM 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. For the automatic switch we use $\Gamma_{\text{CR}} = -50$ with $m = 20$. We train for $2,000,000$ steps and use the average over the last $50,000$ steps as final result. In Fig. 1 we show the mean episode reward obtained with RC-TD3 and RC-SAC on the Y-axis versus the episode reward with TD3 and SAC on the X-Axis. A table with mean values and standard deviations can be found in the Appendix E. A symbol above the diagonal indicates that the curriculum improves performance, and below implies the opposite. It shows that the reward curriculum is equal or better in almost all cases. This is especially the case for environments that do not automatically optimize the auxiliary rewards 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 auxiliary term seems to hinder learning the task, leading to suboptimal policies with the baselines. In those cases, where other terms 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 at all. In this case, the curriculum does not have any effect and learning remains unsuccessful. Furthermore, when examining the base reward r_{base} in Fig. 1 (i.e. the reward that only encodes task learning) it can be seen that employing a reward curriculum is even more beneficial in terms of r_{base} than when looking at r, indicating that it indeed encourages learning the task and prevents reward hacking. In contrast, the baseline does not learn the problem at all in some cases, such as for finger spin, walker run or cheetah run. In these instances, the majority of the episode reward of the baseline solely comes from exploiting auxiliary terms.

5.2 Mobile Robot

For the mobile robot we evaluate our method for the auxiliary weights $w = \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.0\}$. We choose these values as we expect more refined changes for smaller w. We train for $T = 1,000,000$ steps and use the average over the last $50,000$ training steps as final metrics. The automatic switch parameters used are $\Gamma_{\text{CR}} = -10$ for RC-TD3 and $\Gamma_{\text{CR}} = -20$ for RC-SAC with $m = 20$. 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 as a function of w in Fig. 3. To ensure comparability, the reward is calculated using a fixed $w = 0.1$, while the other terms are independent of w. Consistent with previous observations, the reward curriculum exhibits increased effectiveness with growing w. While the auxiliary rewards r_{aux} are relatively similar between the reward curriculum versions and their baselines, the curriculum versions more effectively learn the task, indicated by an increased base reward r_{base}. Interestingly, RC-SAC achieves a substantially higher success rate, particularly for large w, surpassing RC-TD3 and yielding the best results in this environment. Furthermore, RC-SAC on average switched phases at 477 ± 96 while RC-TD3 switched at 246 ± 25. Importantly, the results demonstrated that optimizing the auxiliary objectives indeed conflicts with task performance, as they exhibit opposing trends. Overall, the value $w = 0.2$ 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 auxiliary objectives. In Appendix E, we show how the success rates of the different versions progress over training, which corroborates the finding that the baselines get stuck early on.

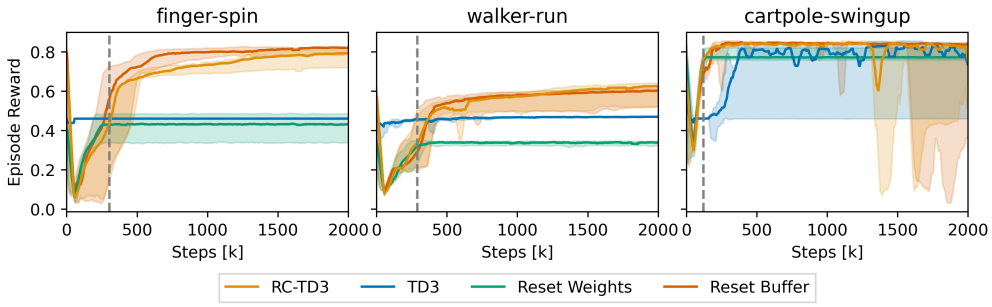


Figure 4: Comparison of the median episode reward of RC-TD3, resetting the network weights and resetting the replay buffer when changing curriculum phases. The values are smoothed by taking the running average with window size 50 [k].

5.3 Ablation Studies

To better understand the impact of our method's components, we perform the following ablation studies. They are conducted on a subset of the environments which should capture the main trends, namely on finger spin, walker run, cartpole swingup, and the mobile robot environment.

What is the importance of the flexible replay buffer? In the first experiment, we compare resetting the network weights or resetting the replay buffer (i.e. deleting all samples collected so far) after switching curriculum phases to our method which reuses past experience via the flexible replay buffer. The results for the DM control environments are shown in Fig. 4. As can be seen, resetting the weights substantially decreases performance while resetting the buffer has little influence. This indicates that for those environments, the main contribution of the reward curriculum is the pre-trained policy. However, when resetting the buffer in the mobile robot environment (see Fig. 6) we can see that the performance deteriorates substantially compared to using the flexible replay buffer. We conclude that the buffer becomes more important to stabilize training when the environment becomes more challenging. The high standard deviations in cartpole swingup are due to training instabilities later in training, which do not seem to be specific to our method.

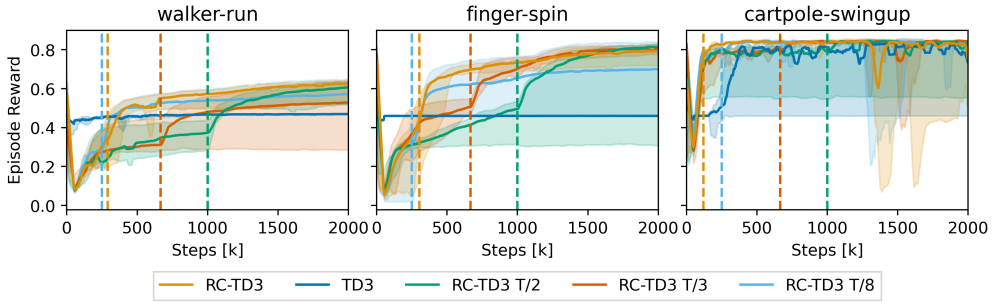


Figure 5: Comparison of the median episode reward of RC-TD3 with automatic curriculum switch as described in Section 3.2 to switching at times T/8, T/3, and T/2 where T are the total number of timesteps. The values are smoothed by taking the running average with window size 50 [k] and the dashed lines indicate the time each curriculum switched phases.

How does the automatic switch compare to switching after a fixed amount of timesteps? The second ablation study focuses on testing the effectiveness of the automatic curriculum switch. We make use of the same DM control 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. It shows that 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 3 in the Appendix 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

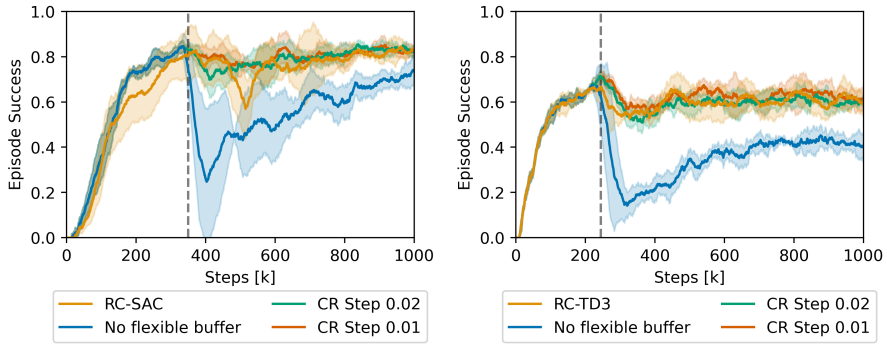


Figure 6: Comparison of the median success rate in the mobile robot environment of RC-TD3 and RC-SAC to resetting the buffer at the curriculum switch and linearly transitioning between r_{base} and r after the curriculum switch with step sizes 0.01 and 0.02.

that the times where the curriculum does not switch phases correspond to an agent not learning the task. Similar training instabilities for cartpole swingup as before also occur in this experiment.

Does the sudden curriculum step deteriorate performance? Lastly, we want to further investigate the sudden switch between training on r_{base} and r. In order to do so we compare our training to linearly annealing r_{base} towards r over a certain amount of steps in the mobile robot environment. The results are presented in Fig. 6 for the two different step sizes 0.01 and 0.02 which transition for 100,000 and 50,000 steps respectively. As it can be seen, linear annealing does not substantially alter the performance. One hypothesis for why the sudden step does not deteriorate performance is that $r_{\text{base}} \subset r$, thus reward signals and therefore losses in phase 2 should have similar trends as in phase 1.

6 Related Work

Reward shaping is a well-established technique to facilitate sample-efficient learning, with a history spanning decades [Dorigo and Colombetti, 1994, Randløv and Alstrøm, 1998]. The most prevalent approach is potential-based reward shaping, which preserves the policy ordering [Ng et al., 1999]. 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. [2020] 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 [Bellemare et al., 2016, Pathak et al., 2017, Burda et al., 2018, Shyam et al., 2019]. 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. [2019] 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. [2022] 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. [2024] employ a three-stage reward curriculum, incrementally increasing complexity and realism by incorporating additional terms. However, they find reward curriculum learning largely ineffective. While these works employ reward curricula, they focus solely on single environments without discussing broader implications. To the best of our knowledge, we are the first to systematically investigate two-stage curricula, examining how sample-efficient experience transfer between phases can mitigate reward exploitation.

7 Conclusion

In this work, we present a two-stage reward curriculum, where we first train an RL agent on an easier subset of rewards and switch to training using the full reward. 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 determine when to change curriculum 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 auxiliary weights and environments where the other objectives 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

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.

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.

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.

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.

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.

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.

Kristian Ceder, Ze Zhang, Adam Burman, Ilya Kuangaliyev, Krister Mattsson, Gabriel Nyman, Arvid Petersén, Lukas Wisell, and Knut Akesson. Birds-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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

Lars Leyendecker, Markus Schmitz, Hans Aoyang Zhou, Vladimir Samsonov, Marius Rittstieg, and Daniel Lütticke. Deep Reinforcement Learning for Robotic Control in High-Dexterity Assembly TasksA Reward Curriculum Approach. *International Journal of Semantic Computing*, 16(03): 381–402, 2022. URL https://www.worldscientific.com/doi/abs/10.1142/S1793351X22430024.

TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. URL https://arxiv.org/abs/1509.02971.

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 2533, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099. URL https://arxiv.org/abs/1812.00285.

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.

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.

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.

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.

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.

Alejandro Rodriguez-Ramos, Carlos Sampedro, Hriday Bavle, 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.

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.

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.

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.

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.

Burrhus F Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958. URL https://psycnet.apa.org/record/1959-07839-001.

Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

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.

Ze Zhang, Yao Cai, Kristian Ceder, Arvid Enliden, Ossian 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.

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.

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.

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.

A RL Algorithms

For both RC-TD3 and RC-SAC, we use target networks that are updated using Polyak averaging with $\tau = 0.005$, where the target parameters are updated as $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$ for policy parameters and $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$ for Q-value parameters.

A.1 RC-TD3

In the following, we show how the reward curriculum integrates with Twin-Delayed DDPG (TD3) [Fujimoto et al., 2018]. It extends the Deep Deterministic Policy Gradient (DDPG) [Lillicrap, 2015] 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 $\tilde{a}'$ to update the estimated Q function to avoid exploitation of overestimations as

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

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

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

and the Q function is updated by minimizing the Smooth L1 loss **?** to the target

$$J_Q(\phi_i) = \text{SmoothL1}(Q_{\phi_i}(s,a) - y(r_{cr}, s'))$$

in order to stabilize training. The policy is updated using the first Q network as

$$J_\pi(\theta) = Q_{\phi_1}(s, \pi_\theta(s)) \tag{5}$$

Importantly, the policy is not updated in every iteration. In our case we follow the standard implementation and only update the actor in every second iteration. 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.

A.2 RC-SAC

Furthermore, we show how our method integrates with SAC. Instead of solely optimizing the expected return, SAC makes use of the maximum entropy objective [Ziebart, 2010] such that a policy additionally tries to maximize its entropy $\mathcal{H}$ at each state which is defined as

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

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

$$J_Q(\phi_i) = \text{SmoothL1}(Q_{\phi_i}(s,a) - y(r_{cr}, s'))$$

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

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

Importantly, $\tilde{a}' \sim \pi_\theta(\cdot|s')$ is newly sampled during training and $Q_{\bar{\phi}_i}$ is the ith 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

$$J_\pi(\theta) = \min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) \tag{6}$$

where $\tilde{a}_\theta(s)$ is sampled from $\pi_\theta(\cdot|s)$ via the reparameterization trick [Ruiz et al., 2016]. We make use of the entropy-constraint variant of SAC as described in [Haarnoja et al., 2018], where α is updated as

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

where $\tilde{\mathcal{H}}$ is the target entropy. 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 Reward Curriculum Algorithm

Here weights with bar indicate target networks and m is the window length for how many consecutive policy losses J_π needs to be below the threshold $J_{\pi,\text{CR}}$ in order to move to the second curriculum phase.

Algorithm 1 Off-policy Reward Curriculum (RC-TD3)

Initialize networks $\phi_1, \phi_2, \bar{\phi}_1, \bar{\phi}_2, \theta, \bar{\theta}$
Initialize replay buffer $\mathcal{D}$, timestep $t \leftarrow 0$
Initialize curriculum phase $\mathcal{CR} \leftarrow 0$ ▷ Start with base reward only
for each iteration **do**
  for each environment step **do**
    $a_t \leftarrow \pi_\theta(s_t) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma)$
    Execute a_t, observe s_{t+1}, $r_{\text{base},t}$, and $r_{\text{aux},t}$
    Store $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_{\text{base},t}, r_{\text{aux},t}, s_{t+1})\}$
    $t \leftarrow t + 1$
  end for
  /* Curriculum Update: Phase 0 → Phase 1 */
  if policy loss plateaus over m steps: $\forall i \in \{t-m+1, \ldots, t\} : J_{\pi;i} < J_{\pi;\mathcal{CR}}$ **then**
    $\mathcal{CR} \leftarrow 1$ ▷ Change to phase 2
  end if
  for each gradient step **do**
    Sample batch $B = \{(s, a, r_{\text{base}}, r_{\text{aux}}, s')\} \sim \mathcal{D}$
    /* Select Curriculum Reward */
    $r_{\text{cr}} \leftarrow \begin{cases} r_{\text{base}} & \text{if } \mathcal{CR} = 0 \\ r_{\text{base}} + w \cdot r_{\text{aux}} & \text{if } \mathcal{CR} = 1 \end{cases}$
    $B_{\text{CR}} \leftarrow \{(s, a, r_{\text{cr}}, s')\}$
    for $i \in \{1, 2\}$ **do**
      $\phi_i \leftarrow \phi_i - \lambda_Q \nabla_{\phi_i} J_Q(\phi_i; B_{\text{CR}})$
    end for
    $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta; B_{\text{CR}})$ ▷ Delayed in TD3
    Update target networks with rate τ_{targ}
  end for
end for

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 $1_0 00\,000$ 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 DM 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 $\gamma = 0.99$, $m = 20$, $\Gamma_{\text{CR}} = -10$ 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.

D 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 DM control suite it was relatively clear which initial reward to use, it might be that some subset of initial rewards r_{base} 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:

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

where

$$\begin{aligned} r_{\text{gp}} &= r_g + r_p, \\ r_{\text{gpv}} &= r_g + r_p + r_v, \\ r_{\text{gpa}} &= r_g + r_p + r_a, \\ r_{\text{gpx}} &= r_g + r_p + r_x. \end{aligned}$$

We train for $T = 300\,000$ environment steps on the randomized maps in Figure 2b over 4 seeds for each subset with $w = 0.5$. Fig. 7 shows the task success rate for each subset over the training steps. It can be concluded that non of the other terms seem 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.

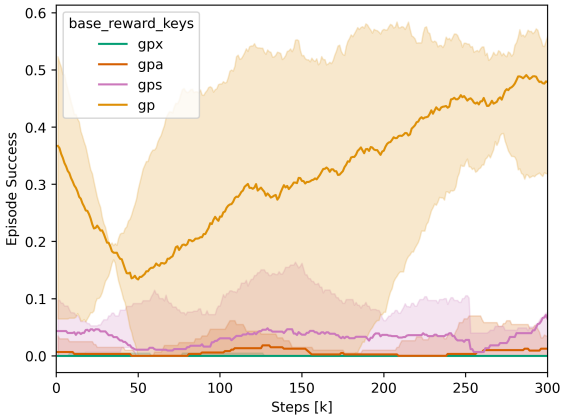


Figure 7: Success rate for the mobile robot during training using TD3 with different initial reward subsets for $w = 0.5$.

E Additional Results

In Fig. 8, we show the success rate of the mobile robot during training. We can see that both RC versions manage to increase the success rate throughout training while the baselines get stuck early on.

The episode rewards obtained using SAC and RC-SAC are presented in detail in Table 1. Similarly, the episode rewards for TD3 and RC-TD3 are reported in Table 2. The iteration at which the curriculum switched for various tasks using the reward curriculum algorithms is summarized in Table 3. Notably, instances where the curriculum failed to progress to the second phase were associated with subpar performance in the initial phase.

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

Table 1: Mean and standard deviation of the episode reward for SAC and RC-SAC for the DM 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.

	$w = 0.5$		$w = 1.0$	
	TD3	RC-TD3	TD3	RC-TD3
ball in cup-catch	766 ± 306	884 ± 22	867 ± 34	439 ± 760
cartpole-balance	758 ± 292	756 ± 306	798 ± 226	847 ± 73
cartpole-swingup	646 ± 84	209 ± 636	388 ± 332	429 ± 357
cheetah-run	-77 ± 1	-169 ± 276	-77 ± 1	92 ± 227
finger-spin	-80 ± 1	$\mathbf{501 \pm 75}$	-80 ± 1	$\mathbf{561 \pm 74}$
finger-turn easy	-697 ± 384	-640 ± 362	-732 ± 335	-579 ± 361
finger-turn hard	-810 ± 234	-642 ± 355	-758 ± 298	-630 ± 396
fish-swim	-824 ± 142	-837 ± 142	-801 ± 111	-830 ± 135
fish-upright	508 ± 253	381 ± 249	626 ± 176	634 ± 174
pendulum-swingup	-16 ± 232	299 ± 443	-22 ± 219	41 ± 475
reacher-easy	116 ± 600	65 ± 634	-264 ± 689	86 ± 749
reacher-hard	-857 ± 167	-823 ± 179	-845 ± 155	-738 ± 302
walker-run	-3 ± 24	$\mathbf{84 \pm 57}$	-61 ± 5	$\mathbf{202 \pm 126}$
walker-stand	780 ± 41	753 ± 66	783 ± 128	799 ± 105
walker-walk	462 ± 321	538 ± 123	391 ± 447	613 ± 98

Table 2: Mean and standard deviation of the episode reward for TD3 and RC-TD3 for the DM 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.

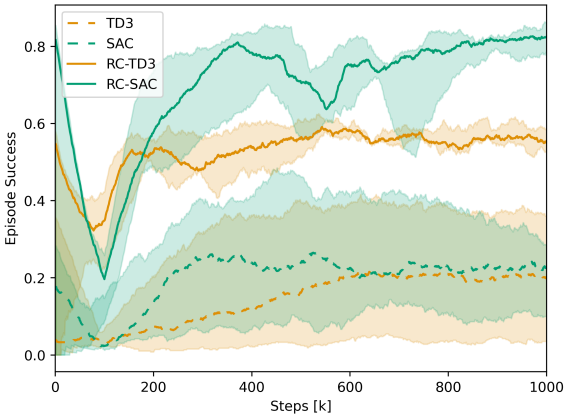


Figure 8: Success rate for the mobile robot during training with $w = 1.0$.

	RC-TD3		RC-SAC	
	$w = 0.5$	$w = 1.0$	$w = 0.5$	$w = 1.0$
fish-upright	101 ± 14	100 ± 7	90 ± 7	142 ± 85
finger-turn hard	735 ± 571	1306 ± 609 *	1057 ± 0	nan $\pm$ nan
cheetah-run	305 ± 50	285 ± 32	451 ± 20	486 ± 28
fish-swim	273 ± 78	270 ± 66	nan $\pm$ nan	nan $\pm$ nan
walker-walk	190 ± 47	186 ± 30	141 ± 15	154 ± 32
walker-stand	128 ± 10	122 ± 11	94 ± 8	91 ± 4
walker-run	279 ± 67	290 ± 48	506 ± 215	462 ± 65
pendulum-swingup	313 ± 100 *	488 ± 54 *	504 ± 184	504 ± 96
reacher-hard	300 ± 171	316 ± 146	206 ± 16	174 ± 17
reacher-easy	124 ± 73	93 ± 39	104 ± 19	101 ± 13
finger-spin	255 ± 71	303 ± 95	245 ± 67	302 ± 28
ball in cup-catch	194 ± 80	227 ± 99 *	64 ± 17	66 ± 17
cartpole-balance	84 ± 1	82 ± 2	51 ± 2	51 ± 1
cartpole-swingup	122 ± 8	120 ± 8	97 ± 11	84 ± 7
finger-turn easy	359 ± 50	331 ± 73	824 ± 286	1601 ± 370

Table 3: Number of steps [k] when the curriculum phase switches for RC-TD3 and RC-SAC. In certain cases, the second phase was never initiated which is shown by * if $\frac{3}{4}$, is $\frac{2}{4}$ and means $\frac{1}{4}$ runs switched to the second phase. Nan indicates that no run switched to the second phase. The few times when the curriculum did not switch phases corresponded to unsuccessful initial training.