Revisiting Sparse Rewards for Goal-Reaching Reinforcement Learning

Gautham Vasan[†‡]
vasan@ualberta.ca

Yan Wang[†‡]
yan28@ualberta.ca

Fahim Shahriar[†‡]
fshahri1@ualberta.ca

James Bergstra[ζ]
james.bergstra@ocado.com

Martin Jagersand[‡]
jag@ualberta.ca

A. Rupam Mahmood[ι†‡]
armahmood@ualberta.ca

[†] Alberta Machine Intelligence Institute (Amii), Edmonton, Canada
[‡] Department of Computing Science, University of Alberta, Edmonton, Canada
[ι] Canada CIFAR AI Chair
[ζ] AI Platform, Ocado Technology, Toronto, Canada

Abstract

Many real-world robot learning problems, such as pick-and-place or arriving at a destination, can be seen as a problem of reaching a goal state as soon as possible. These problems, when formulated as episodic reinforcement learning tasks, can easily be specified to align well with our intended goal: -1 reward every time step with termination upon reaching the goal state, called *minimum-time* tasks. Despite this simplicity, such formulations are often overlooked in favor of dense rewards due to their perceived difficulty and lack of informativeness. Our studies contrast the two reward paradigms, revealing that the minimum-time task specification not only facilitates learning higher-quality policies but can also surpass dense-reward-based policies on their own performance metrics. Crucially, we also identify the goal-hit rate of the initial policy as a robust early indicator for learning success in such sparse feedback settings. Finally, using four distinct real-robotic platforms, we show that it is possible to learn pixel-based policies from scratch within two to three hours using constant negative rewards. Our video demo can be found here.[1]

1 Introduction

In Reinforcement Learning (RL), the task designer implicitly specifies the desired behavior using the reward signal. In order to guide the learning agent to a reasonable solution, task designers often rely on task-specific domain knowledge to hand-craft a dense reward function with state-to-state differences that facilitate faster learning. Since guiding rewards reflect the task designer's preferred behaviors, they could bias the solutions that the agent finds, potentially leading to sub-optimal outcomes (Riedmiller et al. 2018). For most goal-reaching problems, there is a simpler alternative that is easy to specify but still incorporates our intended goal accurately: a constant negative reward every time step, with termination upon reaching the goal state. We call this the *minimum-time* specification since maximizing the undiscounted sum of these rewards leads to reaching the goal state as soon as possible. This specification avoids biasing the final solution, focusing solely on recognizing task success. This approach is also suitable for specifying unforeseen goal-reaching tasks where a human instructor may be unable to provide domain knowledge beforehand.

---

[1]Video demo: https://youtu.be/a6zlVUuKzBc
Code for simulation experiments: https://github.com/gauthamvasan/rl_suite
Code for robot experiments: https://github.com/rlai-lab/relod

(a) Create-Reacher

(b) UR5-VisualReacher

(c) Vector-ChargerDetector
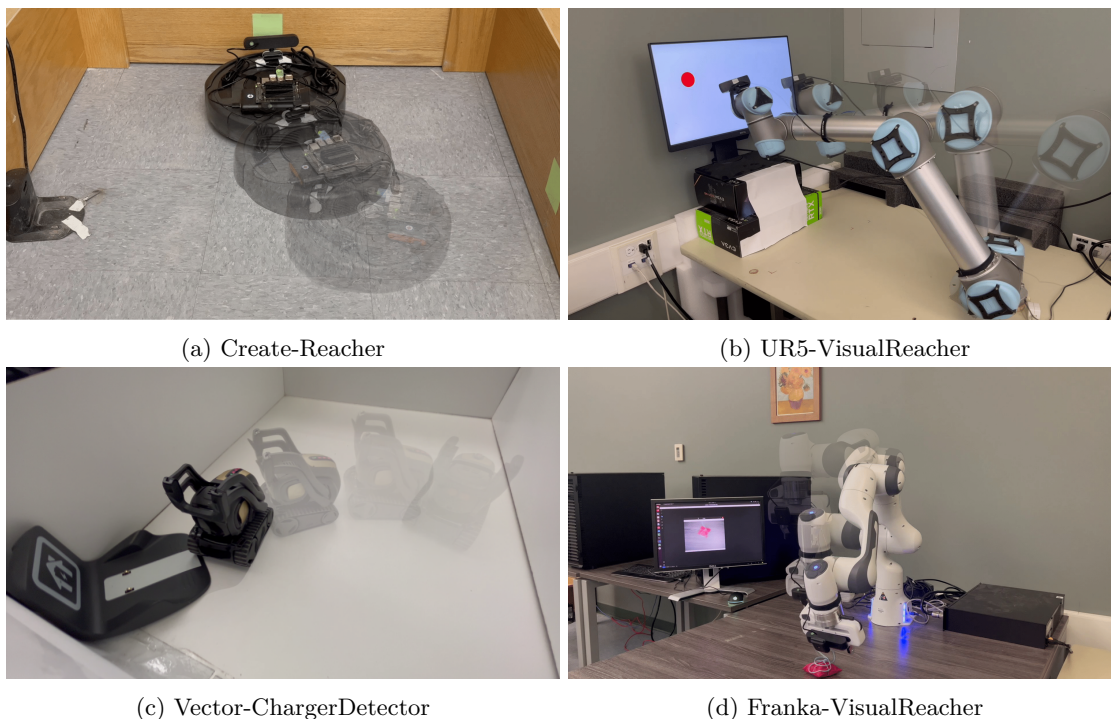
(d) Franka-VisualReacher

Figure 1: Real World Robot Tasks

An alternative and simple specification of rewards is to give a discounted positive reward upon "success" and zero everywhere else. Such a reward is often considered a *sparse reward*, as a differentiating reward signal is available only sparsely. The minimum-time reward can be seen as an extreme form of sparse reward, where the reward value never changes and the differentiating signal only comes from episodic accumulated rewards, called the *return*. However, minimum-time specification has the advantage of being simpler as it does not require discounting as part of the problem.

Despite its simplicity and ease of specification, minimum-time tasks are generally thought to be hard to solve (Andrychowicz et al. 2017). Guiding reward formulations offer state-by-state reward differences, which can be informative for quick policy improvement. This can provide early signs of learning, allowing us to determine whether learning can occur quickly. Sparse-reward tasks may take much longer to show any signs of learning, as informative signals are given only sparsely; a robot may only rarely stumble upon the goal via trial-and-error interactions (Kober et al. 2013).

In this paper, we conduct a series of carefully designed studies to compare dense-reward and minimum-time formulations in RL. Our research confirms the popular notion that dense-reward formulations facilitate faster learning in comparison to sparse-reward formulations. However, our findings also demonstrate that agents trained with minimum-time specification outperform those trained with dense rewards, even in terms of dense-reward performance metrics. In particular, we establish the superiority of the minimum-time formulation over the guiding reward formulation on tasks involving reaching with simulated and industrial robotic arms, both in terms of ease of specification and its ability to attain the desired final behavior. We identify that RL agents can quickly and reliably solve complex vision-based tasks in the minimum-time formulation *if* the agent can reach the goal often enough using its initial policy. Leveraging our insights, we successfully set up vision-based reaching tasks with sparse rewards on four distinct real robots and demonstrate their ability to learn a pixel-based control policy from scratch within two to three hours (see Fig. 1).

2 Related Work

Reward shaping Learning with sparse rewards can be challenging since the agent has to explore the environment extensively to discover the right set of actions that lead to the reward. Hence,

task designers often manually craft a reward function through trial-and-error such that it maximizes both the task performance metric and enables fast learning (Vasan and Pilarski 2017, Mahmood et al. 2018, Lee et al. 2019, Knox et al. 2023). Ma et al. (2023) proposed using a large language model to automate reward design in various tasks using task-specific prompts, reward templates and evolutionary search. Recent work by Rocamonde et al. (2023) and Wang et al. (2024) explores using large pre-trained vision-language models (VLMs) to design reward functions for RL tasks. These approaches, akin to neural architecture search, incur high computational costs. VLMs face challenges like sensitivity to visual realism, hallucinations, scalability issues, and mis-alignments between visual and language modalities. They often struggle to provide precise, task-relevant feedback necessary for effective RL, especially in tasks requiring detailed spatial understanding or abstract reasoning.

Ad hoc reward design often leads to *reward shaping*, where the reward function is used to communicate the underlying performance metric *and* to steer the agent's learning towards a desired policy (Ng et al. 1999). For example, Mataric (1994) proposed rewarding the agent for taking steps up the gradient rather than just for achieving the final goal to speed up learning. However, it is also widely recognized that ad hoc reward shaping is unsafe as it may alter the optimal solution for a particular RL task (Amodei et al. 2016, Knox et al. 2023).

Although guiding reward design seems like a sound approach, Booth et al. (2023) show that relying on trial-and-error to design guiding rewards can result in over-fitting, where reward functions are inadvertently excessively tailored for use with a specific algorithm. Rewards should only communicate to the agent *what* we want achieved rather than detailing *how* we want it achieved (Sutton and Barto 2018). In this work, we adhere to this principle and sidestep the complexities of reward function (mis)-design, thus focusing solely on making learning with sparse rewards tractable.

Learning from sparse rewards Riedmiller et al. (2018) introduced SAC-X, a method for learning from sparse rewards, which learns policies for auxiliary tasks concurrently with the main task to explore the observation space efficiently. Hertweck et al. (2020) extended this by proposing agent-internal auxiliary tasks to enhance exploration in sparse reward settings, particularly for tasks like Ball-in-Cup, using only raw sensor data. Andrychowicz et al. (2017) presented Hindsight Experience Replay (HER) for learning from failed episodes by treating seen states as pseudo-goals, applicable to environments with multiple goal states but limited in vision-based tasks. Nair et al. (2018) expanded HER for vision-based tasks like Reaching and Pushing but with constraints on camera setup and applicability to 2D plane environments. Korenkevych et al. (2019) use autoregressive processes for smoother exploration compared to Gaussian policies. While these approaches use novel strategies to improve exploration in sparse reward scenarios, our work differs by predicting whether a minimum-time task can be effectively learned from scratch based on initial policy performance.

Minimum time problems in optimal control aim to transfer a system from an initial to a final state in the shortest time, with the cost function being the time taken between steps (Chui and Chen 2012, Chapter 9). For example, Penicka et al. (2022) combines classical path planning with model-free deep RL to optimize a neural network policy for minimum-time flight of a quadrotor through a sequence of waypoints with obstacles. Control benchmarks in RL like MountainCar (Sutton and Barto 2018) and DotReacher (Garg et al. 2022) also use the minimum-time formulation with a reward of -1 each step until termination.

3 How to Specify Goal-Reaching Tasks in RL?

In this section, we explore various formulations of goal-reaching tasks as discussed in the literature. We use the classic Reacher problem as an illustrative example (Fig. 2). The reaching task using a two-link robot arm aims to move the fingertip of a planar arm with two degrees of freedom to a random spherical target on a 2D plane (Tassa et al. 2018). In this scenario, *we hold an abstract concept wherein the objective is for the robot arm to swiftly reach a designated target state and remain there.* The difficulty of the Reacher task is conditional on the size of the target.

Background A Markov Decision Process (MDP) is used to model the agent-environment interaction, where an agent interacts with its environment at discrete timesteps. The agent takes an action $A_t \in \mathcal{A}$ at state $S_t \in \mathcal{S}$ every timestep t using a probability distribution π called a *policy*: $A_t \sim \pi(\cdot|S_t)$. After executing the action, the agent receives the next state S_{t+1} at the subsequent timestep $t+1$ and a reward R_{t+1} according to a transition probability density function $S_{t+1}, R_{t+1} \sim p(\cdot, \cdot|S_t, A_t)$. We use continuous state space $\mathcal{S}$ and action space $\mathcal{A}$ for all our tasks.

The *observation space* consists of the position and velocity of the fingertip, and the vector from the fingertip to the target. The *action space* is the torques to be applied to the two joints. We now introduce three specifications based on the choice of reward function and termination.

Guiding Reward Formulation	**Contact Reward Formulation**	**Minimum-Time Formulation**
$R_t = \begin{cases} 1 & \text{if in target,} \\ -\|x_{goal} - x_{pos}\| & \text{otherwise.} \end{cases}$	$R_t = \begin{cases} 1 & \text{if in target,} \\ 0 & \text{otherwise.} \end{cases}$	$R_t = -1$ (until reaching the target)
Fixed length episodes ($T = 1000$)	Fixed length episodes ($T = 1000$)	Varying length episodes; Termination upon reaching the goal with near zero velocity

Table 1: Task Formulations: Choices for reward function and termination conditions

Guiding reward formulation The reward function is inspired by Brockman et al. (2016), where reacher is incentivized to get as close to the target as possible. We also include a commonly used precision reward term previously used on a real robot reacher task using the UR5 industrial arm (Lan et al. 2022, Farrahi and Mahmood 2023, Che et al. 2023, Elsayed et al. 2024). The reward function has two components: (i) penalty term calculated as the negative of the Euclidean distance between the fingertip and the target, and (ii) precision term where the agent receives $+1$ reward when the fingertip is inside the target (see Table 1). Each episode has a time limit of 1000. The time limit is set long enough to incentivize the agent to remain at the goal once reached.
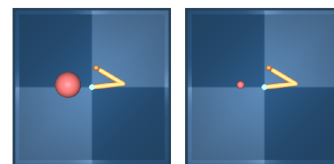

Figure 2: Reacher-Easy (left) and Reacher-Hard (right) from DeepMind Control Suite

Contact Reward Formulation The agent will receive a reward of $+1$ whenever it is within the target sphere and 0 otherwise (see Table 1). Each episode has a time limit of 1000. This choice of reward and timeout follows the *dm_control* suite (Tassa et al. 2018).

Minimum-Time formulation Shorter episodes are encouraged by imposing a reward of -1 for each step until termination. The episode terminates when the fingertip of the arm reaches the target with near-zero velocity. An episode is completed only when the agent reaches the goal state. If the agent exceeds the time limit, we reset the environment such that the goal state remains the same, but the robot arm moves to a different starting state. Note that, unlike the previous formulations, we get rid of the fixed time limit horizon, where all episodes have a uniform episode length.

Since resetting the agent has an associated time cost in the real world, we also penalize the agent when it fails to reach the goal state within the time limit (e.g., $-K$ where K timesteps is required to reset the agent). The episodic returns and lengths are adjusted appropriately. For example, consider a task where the reward is -1 each timestep, the time limit is 100 steps, and reset penalty $= -20$. If an agent times out thrice consecutively and finally reaches the goal in 25 steps since the last timeout, then the return of the episode, in this case, is $-100 - 20 - 100 - 20 - 100 - 20 - 25 = -385$, and the length of the episode is 385. Our learning curves use undiscounted returns estimated in this manner.

4 Which Formulation Achieves the Desired Reaching Behavior?

We use a state-of-the-art RL method— Soft Actor-Critic (SAC) to to solve each formulation of the Reacher task (Haarnoja et al. 2018). We use the same hyper-parameters (listed in A.1) to train all variants. From Fig. 3a, it is clear that both Reacher-Easy and Reacher-Hard agents learn fast with the guiding reward formulation. While the speed of learning is not as rapid, it is still fast with the

(a) Guiding reward formulation (b) Minimum-time formulation (c) Contact reward formulation
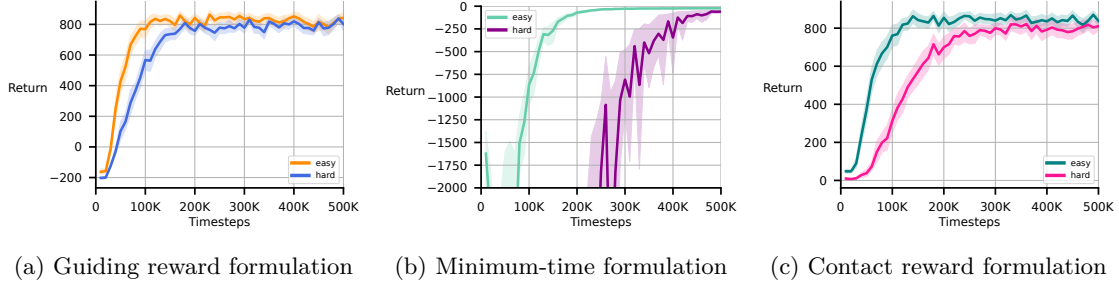
Figure 3: Each solid learning curve is an average of 30 independent runs. The shaded regions represent a 95% confidence interval

contact reward formulation (see Fig. 3c). In contrast, when utilizing the minimum-time formulation (see Figure 3b), the learning curves exhibit a slower progression and more pronounced variability.

Evaluating the final learned behaviours
For a fair comparison of the three task specifications, we evaluate the final learned policies using three metrics (see Table. 4): 1) *Ease of specification*: Do you require any domain knowledge to craft a reward function? 2) *Fast learning*: Is learning quick and robust during training? 3) *Superior Final Performance*: Once the training phase concludes, which version demonstrates the most desirable final learned behavior, as intended by the task designer?

	Guiding Reward Formulation	Contact Reward Formulation	Minimum-Time Formulation
Ease of specification	×	✓	✓
Fast Learning	✓	∼	×
Superior Final Performance	×	×	✓

Figure 4: Evaluating each task specification

To answer which formulation achieves superior final performance, we measure: 1) the number of time steps needed to reach the target (termed "steps *to* goal"), and 2) the duration the agent remains at the target once reached (termed "steps *on* goal"). We evaluate all 30 learned policies from each formulation (90 runs in total) on 500 episodes, lasting $5K$ timesteps each ($2.5M$ steps in total).

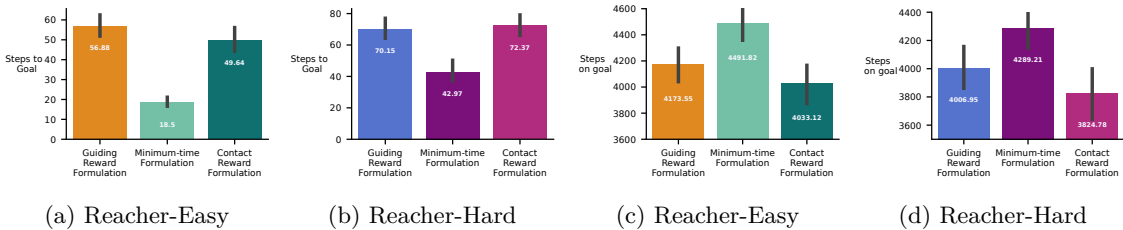(a) Reacher-Easy (b) Reacher-Hard (c) Reacher-Easy (d) Reacher-Hard

Figure 5: Comparison of the three formulations of the reaching task using the number of steps it requires to reach the goal (5a, 5b) and the number of steps it stays within the target sphere upon reaching it (5c, 5d). The standard error is estimated over 30 independent runs. In bar charts 5a & 5b, a lower value signifies better performance. In 5c & 5d, a higher value signifies better performance.

While the guiding reward formulation achieves faster learning compared to guiding reward and minimum-time formulation (Fig. 3), it is not easy to specify. Despite slower learning and less visually appealing learning curves, the final learned performance shows surprisingly superior quality and robustness when using the minimum-time formulation (Fig. 5): $\sim 3X$ faster on Reacher-Easy and $\sim 2X$ faster on Reacher-Hard (Fig. 5a & 5b). The minimum-time policy also stays inside the target area longer, resulting in over 500 higher accrued rewards (Fig. 5c & 5d).

Do the learned policies achieve superior performance in their own formulations? We evaluate policies trained using contact reward and minimum-time formulations on the guiding reward task by assessing policies trained on other formulations every 10K timesteps. The minimum-time

formulation performs comparably to the guiding reward formulation on Reacher-Easy (Fig. 6a) but surpasses it on Reacher-Hard (Fig. 6d). While a distinct hierarchy exists in terms of optimality of minimum-time policies, the same is not true for policies trained with guiding rewards.

We perform a similar evaluation on other formulations. While the contact reward policy performs on par or slightly better than the guiding reward policy, it still lags behind the minimum-time policy (Fig. 6b and 6e). In the minimum-time task, both the guiding reward and contact reward formulations perform poorly (Fig. 6c and 6f). Notably, the agents trained by the guiding reward and contact reward formulations struggle to achieve near-zero velocity at the goal state, which is necessary for completing episodes in the minimum-time formulation.

(a) Guiding Reward task   (b) Contact Reward task   (c) Minimum-Time task

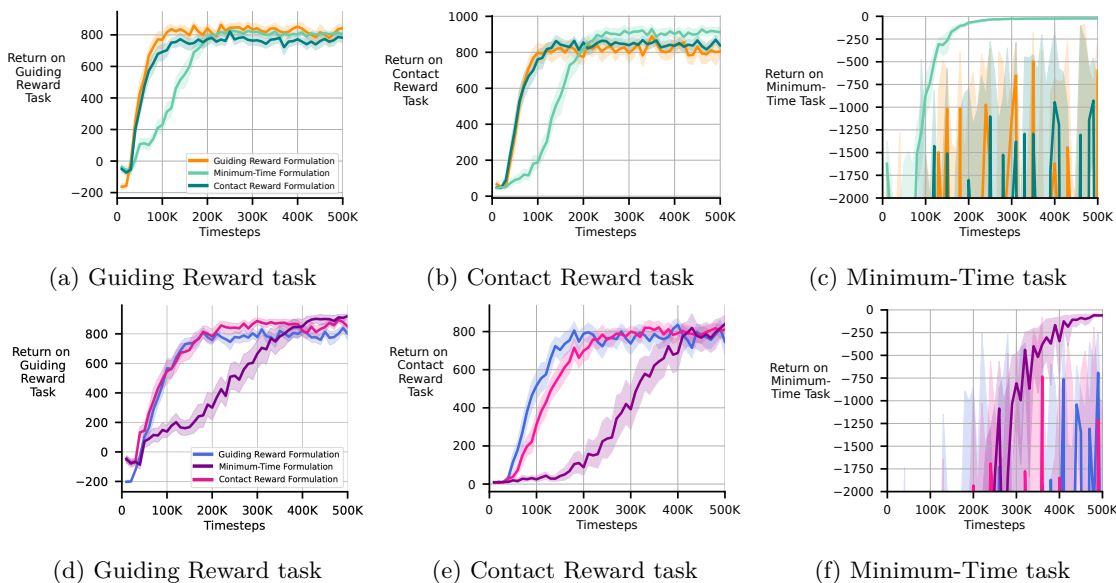(d) Guiding Reward task   (e) Contact Reward task   (f) Minimum-Time task

Figure 6: Evaluating the performance of all formulations across the different specifications of Reacher-Easy (Fig. 6a, 6b, 6c) and Reacher-Hard (Fig. 6d, 6e, 6f). Each solid learning is an average of 30 independent runs. The shaded regions represent a 95% confidence interval.

5 Does Choice of Time Limit Impact Success in Minimum-Time Tasks?

While specifying tasks in the minimum-time formulation is straightforward, achieving successful learning can be challenging with complex real-world problems. One factor that impacts the difficulty of a minimum-time task is the exploration strategy. An implicit way of controlling exploration in RL is to invoke a environment reset after a certain time limit. Many benchmark tasks in RL, such as OpenAI Gym (Brockman et al. 2016) and DeepMind Control Suite (Tassa et al. 2018), including existing minimum-time tasks, such as Mountain Car (Sutton and Barto 2018), treat time limits as an intrinsic property of their task formulations. They often use fixed time limits to separate episodes and improve exploration efficiency. Intuitively, a large time limit increases the chances of reaching the target since the agent has more time to explore the environment. However, a large time limit can hurt exploration if the agent gets stuck in uninformative states due to a sub-optimal policy.

To investigate the effect of the time limit on the frequency of reaching the goal state, we recorded the number of *target hits*, that is, times the agent reaches the goal state using an initial gaussian policy $\mathcal{N}(0, 1)$ within $20K$ timesteps. We plot the number of target hits versus the choice of time limits for a minimum-time variant of *Ball-in-Cup* (Tassa et al. 2018) and Reacher in Fig. 7, revealing that time limits indeed affect an agent's exploration substantially.

We used the Non-Visual and Visual Ball-in-Cup, Non-Visual Reacher-Easy, and Non-Visual Reacher-Hard tasks to test whether the number of target hits affects training performance. The learning
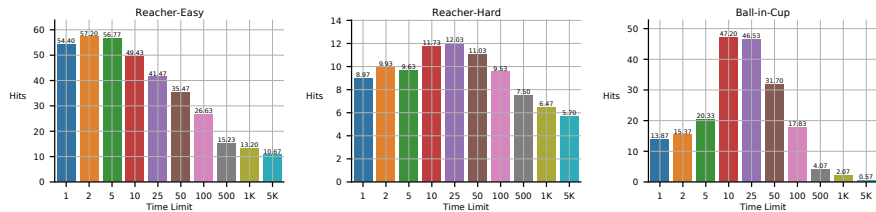
Figure 7: A histogram plot of the choice of time limits versus the number of target hits, that is, the number of times the agent reaches the goal state using an initial policy within 20K timesteps.
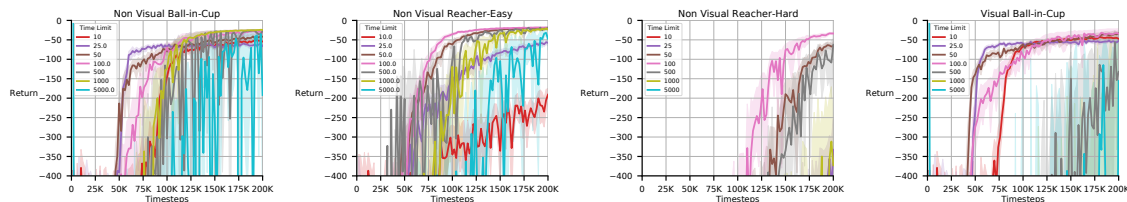

Figure 8: Learning performance of SAC on three non-visual and one visual task in simulation for multiple choices of timeout. Each solid curve is averaged over 30 independent runs. The shaded regions represent a 95% confidence interval.

curves of each task across multiple time limits shown in Fig. 8 establish a clear relationship between the learning performance and the number of target hits. See Appendix A.2 for training details.

Timeout is a solution parameter While timeouts can indeed help exploration in RL, depending on the specific context and implementation, it is oft-ignored as a parameter in task specification. Our results *confirm* that the choice of the time limit can substantially affect overall learning performance, as there is a direct correlation between the frequency of reaching the goal state and the final learning performance. We suggest that *time limits should be treated as a tunable solution parameter that a learning system can tweak* rather than a fixed parameter inherent to the problem. Since our definition of an episode is independent of the choice of the time limit in the minimum-time formulation, we can safely modify the time limit of a task without altering the problem.

6   Learning Vision-Based Minimum-Time Tasks on Real Robots

As demonstrated in simulation, we aim to show that the initial policy's performance can help predict if a minimum-time task can be learned swiftly and consistently with real robots as well. We employ a carefully tuned setup to showcase learning using the minimum-time formulation on four held-out robot tasks. After analyzing our simulation results, we consider that achieving an average of 10 target hits per $20K$ steps is sufficient for successful learning. With the hyper-parameters we have selected, including a mini-batch size of 256 and a replay buffer capacity of $100K$, we estimate that approximately one out of every 8 mini-batch updates would involve a transition sample featuring the goal state. The replay buffer would contain at least 50+ diverse target hits.

Using this heuristic, we aim to choose a time limit for the robot task such that the number of target hits exceeds 10 within $20K$ timesteps using the initial policy. Typically, an action cycle time of $10 - 125Hz$ is commonly utilized with robot control. In this scenario, collecting $20K$ samples would take utmost 35 minutes, which is feasible for real-world experimentation. Once an appropriate time limit is chosen, we proceed with training the task using an asynchronous implementation of the SAC algorithm and the hyper-parameters specified in Appendix A.1 Note that as per the minimum-time formulation, a reward of -1 is assigned at every step for all robot tasks until the episode ends.

Create-Reacher We modified the guiding reward task introduced by Wang et al. (2023) into a minimum-time task, as illustrated in Fig. 1a. To facilitate image-based tasks and onboard inference,

we equipped a Depstech 4K camera and a Jetson Nano 4GB computer. The objective of this task is to swiftly reach one of the green sticky notes. Each episode terminates when the green sticky note occupies at least 20% of the camera image.

UR5-VisualReacher This task is a minimum-time adaptation of the guiding reward task proposed by Yuan and Mahmood (2022) . The objective is to move the robot's fingertip to the target (red blob) on the screen as fast as possible (Fig. 1b). The episode terminates once the target occupies more than 1.5% pixels of the current image. The target remains unchanged until the arm reaches it. Upon termination, we reset the arm to a predefined posture and generate new random targets.

Vector-ChargerDetector We propose a novel vision-based goal-reaching task involving a low-cost mobile robot called Anki Vector (Fig. 1c). The agent sends actions in the form of velocity commands to the wheels of the robot every $100ms$ over WiFi. The episode terminates when the charger symbol is centred and occupies roughly 25% of the image. During a reset, Vector first moves backwards and reorients itself in a random direction.

Franka-VisualReacher We also introduce another novel task involving a 7-DOF robot arm, Franka Emika Panda. The task is to move the wrist-mounted camera close to a randomly placed bean bag on the table (see Fig. 1d). Every episode, we place the arm and the bean bag in random positions. The agent controls the arm by sending velocity commands for each joint at $25Hz$. The episode ends when the bean bag covers more than 12% of the image captured by the camera. We use an adaptation of the code from Karimi et al. (2023) to set up this task.

Task	Time Limit	Average Target Hits	Training (Steps)	Control Frequency (Hz)	Initial Hits Estimation Time (mins)	Robot Experience Time (hours)
Franka VisualReacher	75 (3s) 150 (6s) 750 (30s)	12.6 ± 1.0 13.0 ± 2.08 7.8 ± 0.87	60k	25	13.3	$1.1\bar{1}$
Create Reacher	333 (15s)	18.8 ± 3.92	100k	$22.2\bar{2}$	15	1.25
UR5 VisualReacher	150 (6s)	14.4 ± 2.5	100k	25	13.3	$1.1\bar{1}$
Vector ChargerDetector	300 (30s)	11 ± 1.1	160k	10	$33.3\bar{3}$	$4.4\bar{4}$

Table 2: Robot experiment setup. The average target hits are calculated over five independent runs using the initial Gaussian policy $\mathcal{N}(0,1)$ spanning $20K$ timesteps for each task. Note that robot experience time does not include time required to reset the environment, charging the batteries, etc.
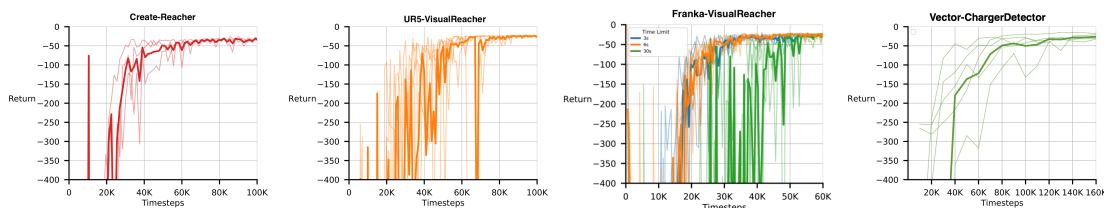

Figure 9: Learning performance of SAC on four vision-based policy learning tasks with robots. Each thin learning curve here represents an independent run. The bold learning curves of the same color are the average of all independent runs.

While seemingly simple, these tasks pose significant complexity for model-free learning algorithms aiming to learn vision-based policies from scratch within a reasonable timeframe. The initial random policy performs poorly, warranting its exclusion from Fig. 9. Real robot task data collection is slow and costly, leading us to test only three time limits for Franka-VisualReacher and select one heuristically for other tasks (Table 2). Five runs were conducted for each task. Utilizing the *ReLoD* system (Wang et al. 2023) is essential for effective real-time learning. Franka-VisualReacher and Vector-ChargerDetector achieve sufficient target hits, while Create-Reacher and UR5-VisualReacher surpass this threshold. We anticipate SAC agents can effectively learn policies for all tasks, which is

supported by the successful learning curves depicted in Fig. 9. This validates our observation that the initial policy's performance is indicative of successful learning on minimum-time tasks.

Using a curriculum to train a robust policy Using a hand-crafted curriculum in RL involves designing a structured sequence of tasks to accelerate an agent's learning. Beginning with simpler tasks and gradually introducing more challenging ones accelerates skill development and knowledge transfer. It's possible to seamlessly integrate a curriculum with minimum-time tasks, where the acceptable threshold for reaching a goal state can be progressively reduced over time. We present a single demonstration featuring the UR5-VisualReacher and Franka-VisualReacher tasks, which were trained with a curriculum over $200K$ timesteps, and we showcase their performance in our video[2]

(a) Steps to goal plot for UR5-VisualReacher

(b) Steps to goal plot for Franka-VisualReacher

(c) Steps on goal plot for UR5-VisualReacher
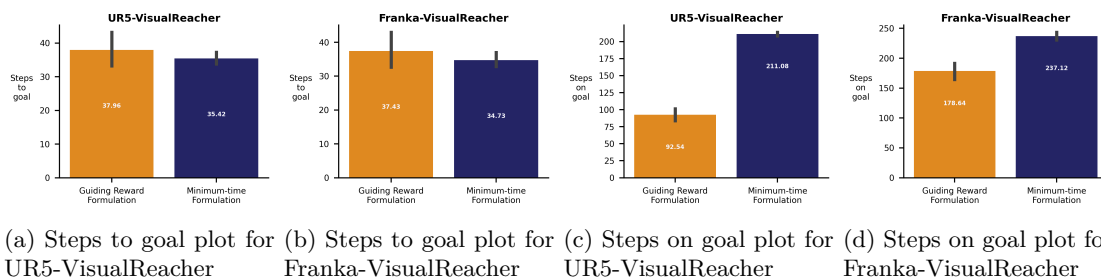
(d) Steps on goal plot for Franka-VisualReacher

Figure 10: Comparison of the guiding reward and minimum-time formulations of the reaching task using the number of steps it requires to reach the goal (10a, 10b) and the number of steps it stays on target upon reaching it (10c, 10d). In bar charts 10a & 10b, a lower value signifies better performance. In 10c & 10d, a higher value signifies better performance.

Evaluation of final learned behaviors on the robot arms We evaluate UR5-VisualReacher and Franka-VisualReacher policies under minimum-time and guiding reward formulations, akin to Section 4, using steps to goal and steps on goal metrics. The guiding reward task has been previously used in Yuan and Mahmood (2022) and Grooten et al. (2024) and is detailed in Appendix A.3. We conduct 2500 episodes lasting 500 timesteps each ($1.25M$ steps total) for UR5-VisualReacher and 1000 episodes lasting 1000 timesteps each ($1M$ steps total) for Franka-VisualReacher. Analysis of Figure 10 shows that minimum-time policies are marginally faster and maintain target contact longer compared to guiding reward policies, consistent with simulation results (Fig. 5), highlighting the effectiveness of the minimum-time formulation.

7 Conclusions

Our research advocates for re-evaluating the use of constant negative rewards in real robot learning as a cost-effective approach to reduce the need for extensive reward engineering and achieve superior final performance. We showed that the minimum-time approach not only simplifies task specification but can also surprisingly lead to better policies, even when assessed using metrics designed for the guiding reward approach, as demonstrated through empirical evidence in simulation and real robots. Through an empirical investigation of the performance of SAC on multiple complex vision-based simulated and real robotic control tasks in the minimum-time formulation, we identified that an agent could achieve successful learning performance if it can reach the goal often enough using its initial policy. Contrary to popular belief, we showed that it is possible to have robust, and reliable learning from scratch on complex vision-based robotic control tasks within a reasonable timeframe using only constant negative rewards. We established that the time limit should be a tunable solution parameter instead of a problem parameter in the minimum-time formulation. In total, we conducted multiple independent runs in our experiments, which took nearly 200 hours of usage on real robots. Our model-free reinforcement learning system achieved real-time learning of pixel-based control of several different kinds of physical robots from scratch.

---

[2] https://drive.google.com/file/d/1O8D3oCWq5xf2hi1JOlMBbs6W1ClrvUFb/view?usp=sharing

Acknowledgments

References

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, 30.

Booth, S., Knox, W. B., Shah, J., Niekum, S., Stone, P., and Allievi, A. (2023). 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.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Che, F., Vasan, G., and Mahmood, A. R. (2023). Correcting discount-factor mismatch in on-policy policy gradient methods. In *International Conference on Machine Learning*, pages 4218–4240. PMLR.

Chui, C. K. and Chen, G. (2012). *Linear systems and optimal control*, volume 18. Springer Science & Business Media.

Elsayed, M., Farrahi, H., Dangel, F., and Mahmood, A. R. (2024). Revisiting scalable hessian diagonal approximations for applications in reinforcement learning. In *International conference on machine learning*.

Farrahi, H. and Mahmood, A. R. (2023). Reducing the cost of cycle-time tuning for real-world policy optimization. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Garg, S., Tosatto, S., Pan, Y., White, M., and Mahmood, A. R. (2022). An alternate policy gradient estimator for softmax policies. In *International Conference on Artificial Intelligence and Statistics*.

Grooten, B., Tomilin, T., Vasan, G., Taylor, M. E., Mahmood, R. A., Fang, M., Pechenizkiy, M., and Mocanu, D. C. (2024). Madi: Learning to mask distractions for generalization in visual deep reinforcement learning. In *AAMAS'24: 2024 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems . . . .

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Hertweck, T., Riedmiller, M., Bloesch, M., Springenberg, J. T., Siegel, N., Wulfmeier, M., Hafner, R., and Heess, N. (2020). Simple sensor intentions for exploration. *arXiv preprint arXiv:2005.07541*.

Karimi, A., Jin, J., Luo, J., Mahmood, A. R., Jagersand, M., and Tosatto, S. (2023). Dynamic decision frequency with continuous options. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7545–7552. IEEE.

Knox, W. B., Allievi, A., Banzhaf, H., Schmitt, F., and Stone, P. (2023). Reward (mis) design for autonomous driving. *Artificial Intelligence*, 316:103829.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Korenkevych, D., Mahmood, A. R., Vasan, G., and Bergstra, J. (2019). Autoregressive policies for continuous control deep reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2754–2762.

Lan, Q., Tosatto, S., Farrahi, H., and Mahmood, R. (2022). Model-free policy learning with reward gradients. In *International Conference on Artificial Intelligence and Statistics*, pages 4217–4234. PMLR.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. (2020). Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33:19884–19895.

Lee, M. A., Zhu, Y., Srinivasan, K., Shah, P., Savarese, S., Fei-Fei, L., Garg, A., and Bohg, J. (2019). Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. (2023). Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on robot learning*, pages 561–591. PMLR.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Machine learning proceedings 1994*, pages 181–189. Elsevier.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. (2018). Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer.

Penicka, R., Song, Y., Kaufmann, E., and Scaramuzza, D. (2022). Learning minimum-time flight in cluttered environments. *IEEE Robotics and Automation Letters*, 7(3):7209–7216.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pages 4344–4353. PMLR.

Rocamonde, J., Montesinos, V., Nava, E., Perez, E., and Lindner, D. (2023). Vision-language models are zero-shot reward models for reinforcement learning. *arXiv preprint arXiv:2310.12921*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690.*

Vasan, G. and Pilarski, P. M. (2017). Learning from demonstration: Teaching a myoelectric prosthesis with an intact limb via reinforcement learning. In *2017 International Conference on Rehabilitation Robotics (ICORR)*, pages 1457–1464. IEEE.

Wang, Y., Sun, Z., Zhang, J., Xian, Z., Biyik, E., Held, D., and Erickson, Z. (2024). Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. *arXiv preprint arXiv:2402.03681.*

Wang, Y., Vasan, G., and Mahmood, A. R. (2023). Real-time reinforcement learning for vision-based robotics utilizing local and remote computers. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9435–9441. IEEE.

Yuan, Y. and Mahmood, A. R. (2022). Asynchronous reinforcement learning for real-time control of physical robots. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 5546–5552. IEEE.

A   Appendix

A.1   Choice of Hyper-parameters

Hyper-parameter	Values
Replay buffer	$100K$
Actor step size	3e-4
Critic step size	3e-4
Entropy coefficient step size	3e-4
Batch size	256
Discount factor (γ)	0.99
Update every k steps	2
Num. update epochs every k^{th} step	1
Actor MLP hidden sizes	[512 512]
Critic MLP hidden sizes	[512 512]
Warm-up time steps	1000
Adam optimizer betas	[0.9, 0.999]
Initial temperature	0.1
Neural network activation	ReLU

Table 3

In most configurations, effective learning is achieved without tuning the hyper-parameters, which indicates the reliability of the algorithm, our learning architecture, and our task setup. Note that for experiments in Sec. 5, the warm-up time steps is set to 20000.

A.2   Neural Network Architecture

Our convolutional neural network (CNN) architecture comprises four convolutional layers, followed by a combination of a Spatial Softmax layer and proprioception information. The convolutional layers have 32 output channels and 3x3 kernels, with stride of two for the first three layers and one for the last layer. After these convolutional layers, tuse spatial-softmax (Levine et al. 2016) to convert the encoding vector into soft coordinates to track the target more precisely. Additionally, proprioception information is concatenated with the spatial softmax features. The exact number of parameters depends on the input data size and task-specific requirements. The two MLP layers have 512 hidden units each. All the layers except the final output layer use ReLU activation. We use a squashed Gaussian policy, which means that samples are obtained according to:
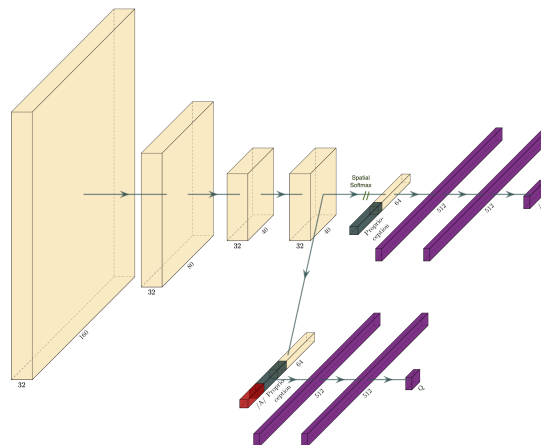

Figure 11: Our CNN architecture combines vision and proprioception information. The yellow blocks represent convolutional layers. The teal block represents proprioception vector. The magenta blocks represent fully connected layers. The red block represents the previous action.

$$a_\theta(s, \varepsilon) = tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon), \varepsilon \sim \mathcal{N}(0, I)$$

We also apply random cropping to augment images in mini-batches to learn more robust representations given our limited amount of observations (Laskin et al. 2020). Note that we train the encoder only using the critic loss. The actor does not propagate any gradients to the shared encoder.

A.3 Guiding Reward Specification for the UR5 and Franka Reaching Tasks

The reward function is defined as follows (Yuan and Mahmood 2022):

$$r_t = \frac{c}{hw} M_t \odot W$$

Here, c represents a scaling coefficient, while h and w denote the height and width of the image in pixels, respectively. M_t is a binary mask of shape $h \times w$, identifying whether each pixel currently displays the color red. Additionally, W refers to a weighting matrix of shape $h \times w$, characterized by values decreasing from 1 at its center to 0 towards the edges. These quantities are combined element-wise through the Hadamard product $\odot$.

The reward function encourages the robot to maneuver its camera closer to the target and ensure the target remains centered within the frame. For consistency across experiments, we set $c = 800$ and bound the rewards between 0 and 4.

Each episode spans 150 timesteps, with each timestep lasting 40 ms, totaling 6 seconds. The agent sends an action every timestep, which the SenseAct system repeats five times every 8 ms.