Learning to Fly Using a Constant Reward Function

Anonymous authors
Paper under double-blind review

Abstract

Recently Reinforcement Learning (RL) has been applied to numerous domains including end-to-end quadrotor control. The training of useful policies using RL usually involves the time-consuming and often unprincipled, tuning-based design of a reward function. In this work, we present an alternative method to train end-to-end quadrotor control using a constant reward function. Since a constant reward function cannot carry information, we show that end-to-end policies can learn to fly solely through the feedback of the termination signal. Furthermore, we show how additional objectives can be encoded into the termination signal, leading to robust end-to-end policies that can be transferred to a real quadrotor and that even generalizes to a novel task like trajectory-tracking. Finally, we compare the trajectory-tracking performance of our policy to other classical and RL-based methods and find that our policies can achieve similar performance to other RL-based approaches.

Supplementary Material

Video (anonymous Google drive)

1 Introduction

Based on recent advancements in Deep Learning (DL), deep RL has become a promising method for solving decision-making problems and continuous control problems in particular. Continuous control problems arise in many fields, including robotics. Classically, robot control algorithms require strong assumptions which often even constrain the electric and mechatronic design of the hardware. The very general formulation of RL problems as Markov Decision Processs (MDPs) allows relaxing these constraints but introduces new challenges. One major challenge is the design of reward functions (Eschmann, 2021). The challenge of reward function design breaks down into two major components

- Accurately expressing the intent of the problem
- Allowing for stable and efficient learning of a control policy

These sub-goals often are in conflict, e.g., when the most accurate expression of the goal is a boolean reward that signals whether the task was accomplished or not. In this case, exploration can pose a major challenge to learning a policy purely through interaction. Furthermore, the stable approximation of the value function can become problematic if a long horizon (discount factor close to 1) is required to attain the goal.

Another example is the tracking of a trajectory or a target state. In this case, the goal of the problem is often best expressed in terms of a cost function (e.g. euclidean distance to the trajectory at every step in time). It is well-known that optimization problems can be transformed from minimizing a cost to maximizing a reward (and vice versa) by negating the objective. While theoretically sound, in practice the tension between the aforementioned subgoals arises in this case because we would like to truncate episodes that stray too far from the target trajectory. These cases should be truncated

not only because they "waste" environment interaction cycles in irrelevant/uninteresting portions of the state space but also because they can introduce numerical instabilities (unstable dynamical systems often diverge exponentially).

Recently Eschmann et al. (2024a) showed that the aforementioned challenges are particularly apparent in the problem of end-to-end quadrotor control. Eschmann et al. (2024a) introduce a RL-based training method that is highly efficient in terms of sample complexity and wall-clock training time (full, end-to-end training: 300 000 environment steps, 18 s training time on a consumer laptop). The proposed method includes a reward function and curriculum design. Prior to that RL has been applied to quadrotor control in multiple ways (Hwangbo et al., 2017; Molchanov et al., 2019; Gronauer et al., 2022) and even yielded impressive results in e.g. quadrotor drone racing (Kaufmann et al., 2023). The mentioned works all use different reward functions, showing that case-by-case reward function design is still a challenge for broader applicability. In Kaufmann et al. (2023) in particular, while achieving impressive performance, a highly complex reward function, composed of four main terms, each of which contains sub-terms with multiple parameters, is used.

With the goal of eliminating the complexity coming with reward function design, in this work, we present a method for training end-to-end quadrotor control with a constant reward function. In the following, we provide a theoretical explanation of our method and the findings, followed by an empirical evaluation including real-world experiments.

2 Method

We adopt the training method of Eschmann et al. (2024a) but replace the reward function:

$$\begin{aligned} r(\mathbf{s}, \mathbf{a}) = &- C_{rp}\|\mathbf{p}\|_2^2 - C_{rq}\left(1 - {q_w}^2\right) - C_{rv}\|\mathbf{v}\|_2^2 \\ &- C_{r\omega}\|\boldsymbol{\omega}\|_2^2 - C_{ra}\|\mathbf{a} - C_{rab}\|_2^2 + C_{rs}. \end{aligned}$$

With a constant reward function

$$r(\mathbf{s}, \mathbf{a}) = 1. \tag{1}$$

Now one might wonder how the MDP can encode any information about the problem and how an RL method, which usually requires a total ordering over policies, can find suitable policies for the problem. To understand the effect of a constant reward function, we can write the value function for a particular policy π as the expectation over the sum of discounted rewards:

$$V(\mathbf{s}) = \mathbb{E}\left[\sum_{t=0}^{N-1} \gamma^t R_t\right],\ R_t = r(\mathbf{s}_t, \mathbf{a}_t),\ \mathbf{s}_t \sim p(\mathbf{s}_0)\prod_{i=1}^{t} p(\mathbf{s}_i|\mathbf{s}_{i-1}, \mathbf{a}_{i-1})\pi(\mathbf{a}_{i-1}|\mathbf{s}_{i-1}),\ \mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t) \tag{2}$$

$$= \mathbb{E}\left[\sum_{t=0}^{N-1} \gamma^t 1\right] = \mathbb{E}\left[\frac{1-\gamma^N}{1-\gamma}\right] = \sum_{N=1}^{\infty} \frac{1-\gamma^N}{1-\gamma} p(N) \tag{3}$$

Figure 1 shows the value of a policy that deterministically achieves N length episodes under the constant reward function from Equation (1). We can see that the value of a policy solely depends on the distribution over episode lengths $p(N)$ that it achieves. The distribution over episode lengths $p(N)$ (for a particular policy) is not generally tractable but can be sampled from, through interaction, as usual in RL.

Alternatively, we can understand the information transfer through the termination signal using the recursive formulation of the *Bellman equation*:

$$V(\mathbf{s}_t) = \begin{cases} r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V(\mathbf{s}_{t+1}) & \text{if } \mathbf{s}_{t+1} \notin \text{terminal} \\ r(\mathbf{s}_t, \mathbf{a}_t) + \underbrace{\gamma \cancel{V(\mathbf{s}_{t+1})}}_{=0} & \text{if } \mathbf{s}_{t+1} \in \text{terminal} \end{cases} \tag{4}$$

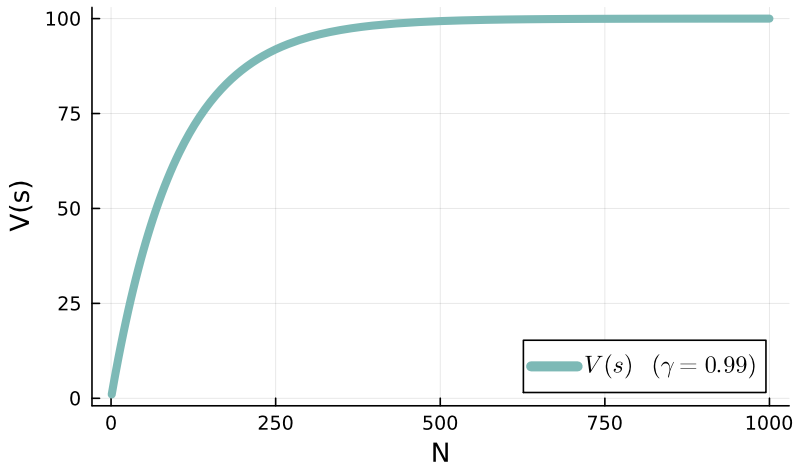


Figure 1: This plot displays the value $V(\mathbf{s})$ for a particular episode length N according to Equation (3). Note that for this illustration $p(N) = 1$ is assumed which means that the policy achieves the same episode length for all initial states deterministically.

Note that this viewpoint is expressed in terms of the Monte-Carlo approximation of the Bellman equation, where $\mathbf{s}_{t+1}$ is sampled according to Equation (2).

For training value function approximations in deep RL, Equation (4) is usually transformed into a quadratic loss function called the Mean-Squared Bellman Error (MSBE). We can see that if $\mathbf{s}_{t+1}$ is terminal, the loss is actually "grounded" because $V(\mathbf{s}_t)$ only needs to approximate $r(\mathbf{s}_t, \mathbf{a}_t)$. Hence, in the terminal case, the training of the value function approximator is not subject to the bias introduced by bootstrapping (i.e. using the approximation $V(\mathbf{s}_{t+1})$ in the target).

Anecdotally we observe that at the beginning of the training, the value function approximation has the lowest MSBE close to the terminal states. The values close to the terminal states are less of a moving target because they are constant given a stationary policy. In contrast, the other values, e.g. around the initial states, are predicting a moving target as the value function approximation is improving (even for a stationary policy).

Even in our simple setup, using the constant reward function from Equation (1), the value of initial states is determined by the distribution over episode lengths which cannot be regressed directly and has to be learned through bootstrapping.

Note that it is especially important to consider the difference between truncation (e.g. time-limit) and termination ("agent dead") (Pardo et al., 2018) in the case of a constant reward function because all the information is carried through the termination signal, where the state-value is grounded at zero. Additionally, sample efficiency is a key requirement for fast training, hence we would like the collected interaction data to be as expressive about our problem as possible. In practice, this is achieved by termination when the agent strays too far off the target and by truncation after it e.g. converges to the target state.

We use the end-to-end training pipeline and parameters proposed in Eschmann et al. (2024a) as a base to implement our proposed method. Like in Eschmann et al. (2024a), the truncation happens after 500 steps (5 s), and the value of the final, truncated state is bootstrapped. Using this training pipeline the policy directly maps observations (position, orientation, linear & angular velocity, and the action history) to control outputs (Revolutions Per Minute (RPM) through Pulse-Width Modulation (PWM)). The main change we apply is replacing the reward function with a constant as defined in Equation (1). This simplifies the method considerably because it also removes the requirement for the curriculum (which we deactivate). Furthermore, the number of parameters that

have to be tuned is reduced considerably because the ~ 20 parameters of the curriculum and reward function are not required anymore. Moreover, we replace the dynamics parameters of the Crazyflie model in the simulator with the dynamics parameters recently estimated in Eschmann et al. (2024b).

2.1 Encoding Additional Objectives in the Termination Signal

As shown in Figure 2 ("Constant Reward"), using a constant reward function induces a kind of null-space, where the value-function is flat. In the case of quadrotors with box-based termination conditions ($\pm 0.6\,\mathrm{m}$ in our case) this means the policy can freely roam around this area. In combination with the inherent stochasticity and instability of RL this gives rise to interesting patterns like the circular motions that can be seen in 2. These patterns vary considerably across different initial seeds.

To constrain the space of (near) optimal policies more strongly, we propose an integral position (and orientation) error term which becomes part of the state and can be used in the termination function:

$$\begin{aligned} \dot{p}_{ei} &= \|\mathbf{p}\|_2^2 \\ \dot{q}_{ei} &= |2\cos^{-1} q_w| \\ \mathbf{s} \in \text{terminal}_2 \quad &\text{iff. } (p_{ei} > T_{p_{ei}}) \text{ or } (q_{ei} > T_{q_{ei}}) \text{ or } (\mathbf{s} \in \text{terminal})\,. \end{aligned} \tag{5}$$

Where terminal_2 is the expanded set of terminal states represented by the new termination function that takes into account the position and orientation integral error (p_{ei} and q_{ei} respectively). The position and orientation error integral thresholds are set to $T_{p_{ei}} := 1$ and $T_{q_{ei}} := 50$ respectively. Since Eschmann et al. (2024a) includes an asymmetric actor-critic scheme (Pinto et al., 2018) we only include the error integral terms in the observations of the critic, the observations of the actor remain unchanged.

3 Results

We find that our method can reliably learn to fly, even when just using the constant reward function, without encoding additional objectives in the termination signal. We train the policy for $1\,000\,000$ environment steps which only takes about $88\,\mathrm{s}$ on a 2023 MacBook Pro due to RLtools (Eschmann et al., 2023). In Figure 2 the resulting trajectories of 100 rollouts of 10 different policies (based on different random seeds) are shown. We can see that without the error integral terms, the policies roam around the allowed space because the value function is mostly flat around the origin. From the state-visitation densities in Figure 2 we can see that it still learns to avoid being close to the termination boundaries where the state-values are zero as described in Section 2.

3.1 Encoding Additional Objectives in the Termination Signal

When using the method described in Section 2.1 to encode additional objectives in the termination signal, we find that the standard deviation is about half and the mean is much closer to the target (origin). Figure 2 only shows the positions but we also find that using the error integral termination greatly reduces the linear velocities and stead-state error making it plausible to transfer the policy to the real world.

3.2 Real World Test

We transfer a policy trained using the error integral termination to a real Crazyflie quadrotor and test its performance in a trajectory-tracking task. Figure 3 shows the tracking performance of the learned, end-to-end policy at two different cycle times of a figure-eight trajectory. We find that even though the policy has only been trained with a constant reward function and the objective to fly

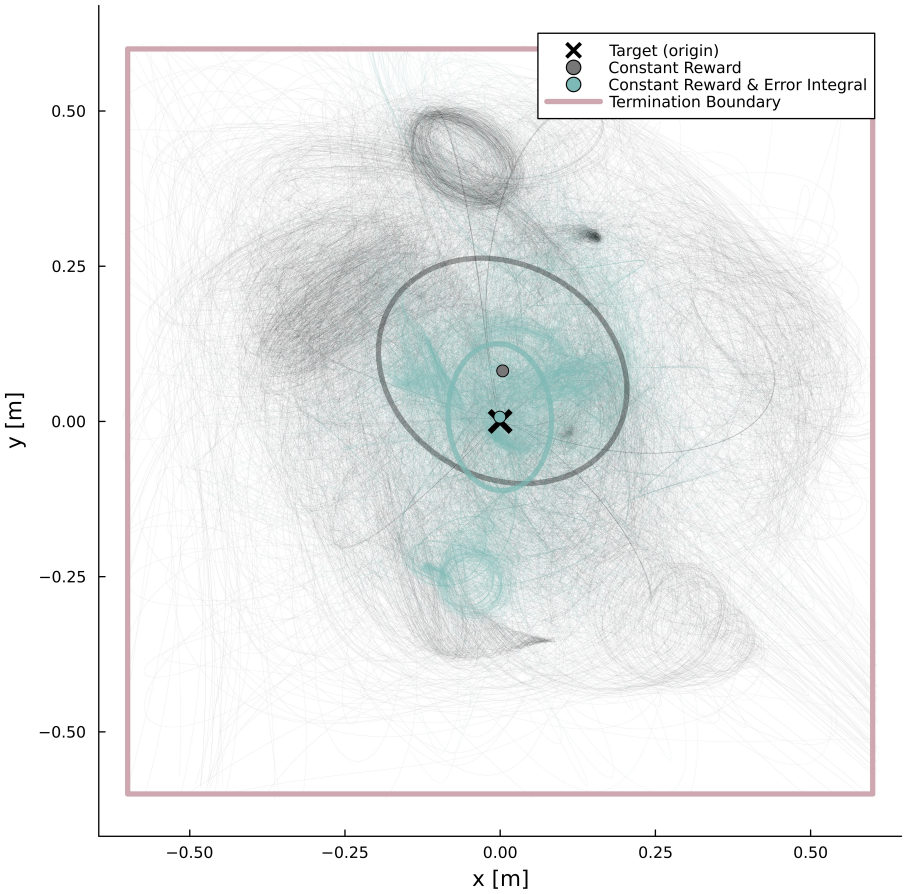
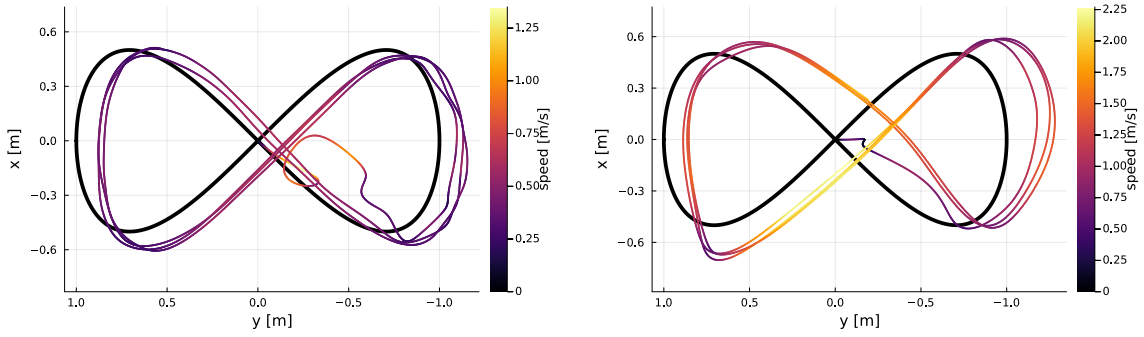


Figure 2: Rollouts of 10 different policies trained with different random seeds (100 episodes with random initial states each). The positions are projected into the xy-plane and the mean and standard deviation are shown by dots and ellipses respectively.

to the origin, the trained policy can generalize to downstream tasks like trajectory tracking. Like Eschmann et al. (2024a) this generalization is made possible by offsetting the position and velocity observations based on the desired trajectory.



(a) Slow ($T = 15\,\mathrm{s}$)

(b) Normal ($T = 5.5\,\mathrm{s}$)

Figure 3: Real-world tracking of a Lissajous trajectory with different cycle times (reference trajectory in black).

We observe some violent behavior initially, after activating the trained policy to track the trajectory as can be seen in Figure 3. Since, after settling in, the policy is smoothly tracking the trajectory we believe this is caused by the handling of the initialization of the action history directly after switching to the policy and not an inherent artifact of our method.

Table 1 shows a comparison of the tracking performance of our trained policy with other classic and learning-based approaches for quadrotor control. Even though our proposed method only uses a constant reward function, we find that our approach achieves similar tracking performance to other RL-base methods.

Interval	Slow (15 s)		Normal (5.5 s)	
Controller	$\overline{e}$	$\overline{e}_{xy}$	$\overline{e}$	$\overline{e}_{xy}$
PID	0.23	0.22	0.72	0.72
Geometric Mellinger & Kumar (2011)	**0.06**	**0.04**	**0.16**	0.16
Nonlinear Brescianini et al. (2013)	0.29	0.11	0.38	0.32
INDI Smeur et al. (2016)	0.21	0.21	1.13	1.13
RL Molchanov et al. (2019)	0.15	0.13	0.25	0.24
RL Gronauer et al. (2022)	**0.06**	0.05	0.23	0.21
RL Eschmann et al. (2024a)	0.08	0.08	0.17	**0.15**
Ours	0.17	0.14	0.26	0.25

Table 1: Real-world trajectory tracking error $\overline{e}$ (RMSE including z in meter) and $\overline{e}_{xy}$ (RMSE excluding z in meter) when tracking the Lissajous trajectory in Fig. 3 using different controllers. Note: This table is based on Table III in Eschmann et al. (2024a).

4 Conclusion

In this work, we propose a novel method for training end-to-end quadrotor control policies solely based on a constant reward function. We find that our method enables training of surprisingly robust policies (especially when incorporating the error integral termination) that are even transferable to a real quadrotor and generalize to new tasks like trajectory tracking. Our method removes the need to design a reward function and hence removes the burden of tuning its parameters. While the error integral termination introduces two new parameters, we find that they do not require much tuning (we only tried 2-3 values for each of them) to yield useful policies. On the other hand, we believe that the performance could still be significantly improved by tuning the other hyperparameters (termination conditions on the remaining parts of the state, initial state distribution, discount factor, etc.) which we did not tune. In conclusion, we believe that our method yields surprising results that are interesting to researchers and practitioners not only in the field of quadrotor control but in RL for continuous control in general.

References

Dario Brescianini, Markus Hehn, and Raffaello D'Andrea. Nonlinear quadrocopter attitude control: Technical report. Technical report, ETH Zurich, 2013.

Jonas Eschmann. Reward function design in reinforcement learning. *Reinforcement Learning Algorithms: Analysis and Applications*, pp. 25–33, 2021.

Jonas Eschmann, Dario Albani, and Giuseppe Loianno. RLtools: A Fast, Portable Deep Reinforcement Learning Library for Continuous Control, 2023.

Jonas Eschmann, Dario Albani, and Giuseppe Loianno. Learning to fly in seconds. *IEEE Robotics and Automation Letters*, pp. 1–8, 2024a. doi: 10.1109/LRA.2024.3396025.

Jonas Eschmann, Dario Albani, and Giuseppe Loianno. Data-driven system identification of quadrotors subject to motor delays. *arXiv preprint arXiv:2404.07837*, 2024b.

Sven Gronauer, Matthias Kissel, Luca Sacchetto, Mathias Korte, and Klaus Diepold. Using simulation optimization to improve zero-shot policy transfer of quadrotors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10170–10176, 2022. doi: 10.1109/IROS47612.2022.9981229.

Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a Quadrotor with Reinforcement Learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2017.2720851.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620 (7976):982–987, 2023.

Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2520–2525, 2011. ISBN 978-1-61284-386-5. doi: 10.1109/ICRA.2011.5980409.

Artem Molchanov, Tao Chen, Wolfgang Hönig, James A. Preiss, Nora Ayanian, and Gaurav S. Sukhatme. Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 59–66, 2019. doi: 10.1109/IROS40897.2019.8967695.

Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4045–4054. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/pardo18a.html.

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning. In *Robotics: Science and Systems XIV*, 2018. doi: 10.15607/RSS.2018.XIV.008.

Ewoud J. J. Smeur, Qiping Chu, and Guido C. H. E. de Croon. Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles. *Journal of Guidance, Control, and Dynamics*, 39(3):450–461, 2016. doi: 10.2514/1.G001490.