Reinforcement Learning on Pieter Abbeel's Helicopter as a Gymnasium Environment

Sri Rama Bandi
sbandi@umass.edu

GitHub Repository: https://github.com/srirambandi/abbeel-heli-rl

1 Introduction

This project studies control and decision making for a simulated version of Pieter Abbeel's autonomous helicopter [1] [6], modeled as a continuous control Markov Decision Process (MDP) in Gymnasium [3]. The state is a 13-dimensional vector capturing position, velocities, and attitude, and the action space contains four continuous control inputs similar to those used in real RC helicopters.

My goal is to learn a policy that can stabilize the helicopter around a target 3D position. I implemented and compared four approaches: Behavior Cloning (BC), DAgger [7], PPO [8], and SAC [4]. BC and DAgger rely on expert demonstrations from a PID controller I designed, while PPO and SAC are warm-started from imitation weights to make RL feasible on this unstable system.

I evaluate all methods on both in-distribution (ID) and out-of-distribution (OOD) targets to understand their stability, generalization, and failure modes.

2 MDP Construction

I model the helicopter as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$. The thirteen dimensional state contains position, velocity, orientation (quaternion), and angular velocity. The four dimensional action space controls collective thrust and cyclic inputs, following the setup in [1].

The transition function uses a simplified but tuned dynamical model: drag terms, aerodynamic gains, and tilt limits were adjusted until the system behaved numerically stably and could hover without diverging. Small changes often caused immediate crashes, so this tuning was essential for learning.

The reward combines distance-to-goal shaping with penalties on tilt, velocity, and angular rates, plus terminal bonuses/penalties. Sparse rewards did not work for this environment; shaping was required for PPO and SAC to make progress.

3 Initial Approach

My first attempt was to train PPO from scratch. This failed: the helicopter quickly destabilizes, episodes are long, and the agent rarely experiences useful reward. Without surviving long enough to hover, PPO never discovered stable regions of the state space. This made it clear that I needed an expert policy to provide demonstrations and a good initialization before applying RL.

4 Expert Demonstrations

Since I did not have access to human pilot data like the original work [1], I implemented a PID controller that stabilizes the helicopter and tracks a target position. I used it to collect demonstrations across three regions: random targets, a hover region, and a curriculum around the evaluation goal.

This dataset provides diverse but structured trajectories and serves three roles: it directly trains BC, it bootstraps DAgger, and it warm-starts PPO and SAC, which cannot learn from scratch on this highly unstable system.

5 Algorithms

5.1 Behavior Cloning

Algorithm 1 Behavior Cloning

1: Load expert dataset $\mathcal{D} = \{(s_i, g_i, a_i^*)\}$
2: Initialize policy π_θ
3: **for** each training epoch **do**
4: Sample minibatch $(s, g, a^*) \sim \mathcal{D}$
5: $\hat{a} \leftarrow \pi_\theta(s, g)$
6: Compute loss $L = \|\hat{a} - a^*\|^2$
7: Update θ with Adam
8: **end for**

I implemented a simple goal-conditioned MLP policy that receives the current state and the desired goal. The policy predicts continuous actions using a tanh-squashed output head. Training followed standard supervised learning with MSE loss on expert actions.

BC serves mainly as a baseline and a warm start. It learns stable hovering behavior but struggles with long range generalization, which the later results confirm.

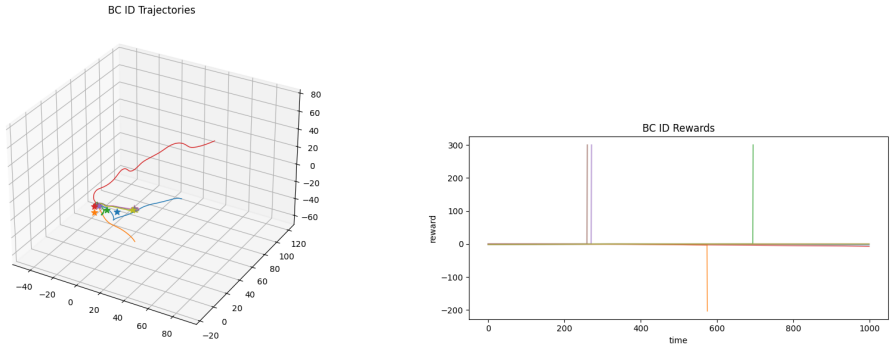

Figure 1: BC in-distribution performance: trajectories and rewards.

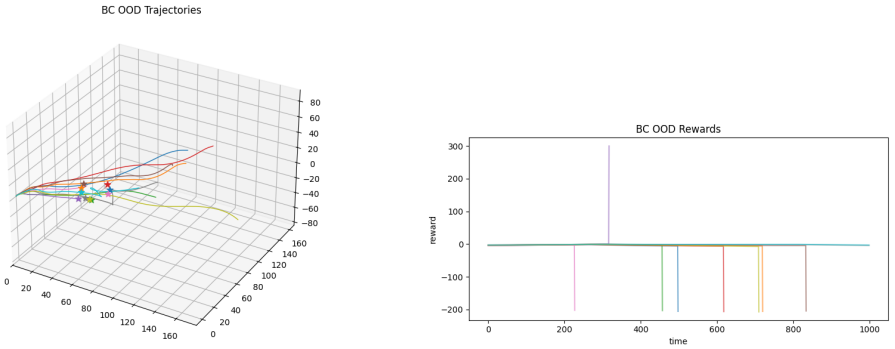

Figure 2: BC out-of-distribution performance.

5.2 DAgger

Algorithm 2 DAgger

Initialize learner π_θ (optionally warm-start from BC)
Initialize dataset $\mathcal{D}$ with expert demonstrations
for iteration $k = 1$ to K **do**
    Roll out learner π_θ in environment
    for each visited state s **do**
        Query expert for a^*
        Add (s, g, a^*) to $\mathcal{D}$
    end for
    Retrain π_θ on updated dataset $\mathcal{D}$
end for

DAgger repeatedly rolls out the current policy, queries the expert for corrective actions, aggregates these corrections, and retrains the model. This allowed the learner to encounter its own state distribution and avoid compounding BC errors.

DAgger performed extremely well: high success rate, low variance, and good generalization.

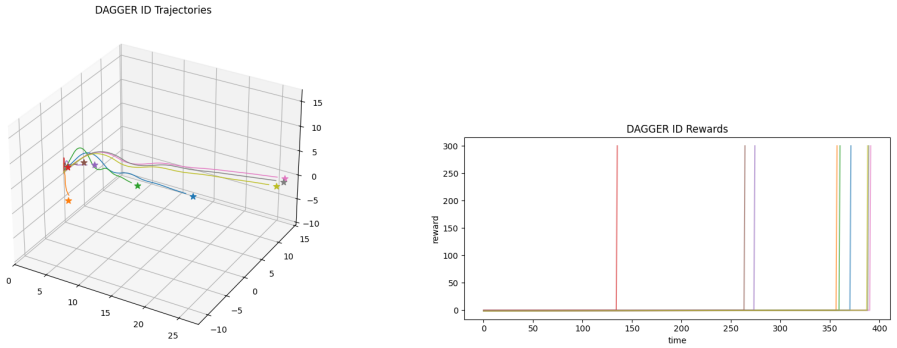

Figure 3: DAgger in-distribution performance.

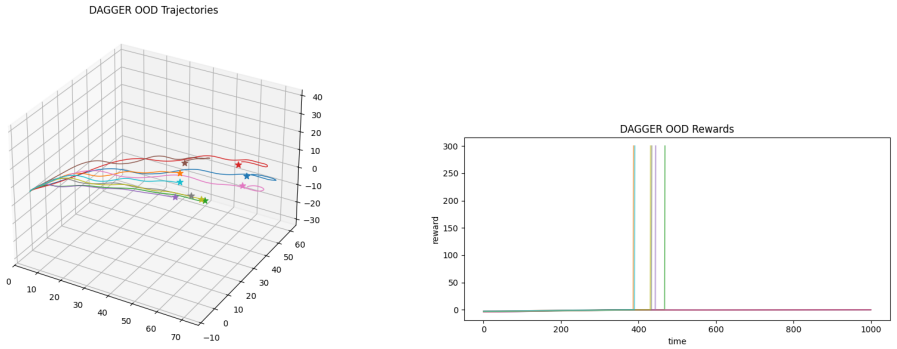

Figure 4: DAgger out-of-distribution performance.

5.3 PPO with Warm Start

Algorithm 3 Proximal Policy Optimization (Warm-Started)

Initialize actor π_θ, critic V_ϕ
if warm start **then**
    Load expert weights into actor
end if
for each epoch **do**
    Collect rollouts using π_θ
    Compute advantages $\hat{A}_t$ with GAE
    for each minibatch **do**
        Compute ratio $r_t = \frac{\pi_\theta(a_t|s_t,g_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t,g_t)}$
        Compute clipped objective
        Update actor and critic
    end for
end for

After BC training, I warm-started PPO by copying the expert's policy parameters. PPO then refined the actions using on-policy rollouts. Unlike the scratch PPO attempts, the warm-started version learned meaningful improvements but remained unstable at times, especially on OOD tasks.

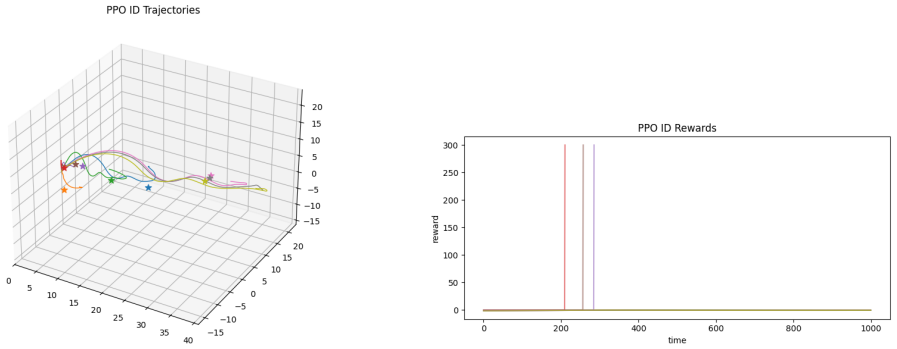

Figure 5: PPO in-distribution performance.

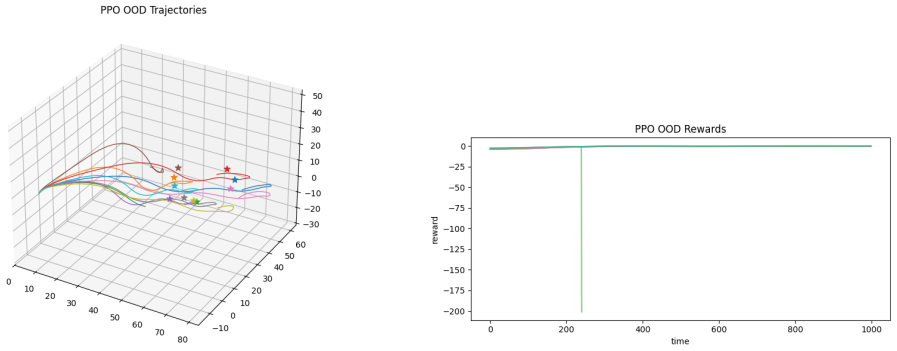

Figure 6: PPO out-of-distribution performance.

5.4 SAC with Warm Start

Algorithm 4 Soft Actor-Critic (Warm-Started)

Initialize actor, critics, and target critics
Initialize replay buffer $\mathcal{B}$
if warm start **then**
    Prefill $\mathcal{B}$ using expert policy
end if
for each training step **do**
    Sample action a_t from actor
    Execute a_t, store $(s_t, g_t, a_t, r_t, s_{t+1})$ in $\mathcal{B}$
    if $|\mathcal{B}|$ large enough **then**
        Update critics using entropy-regularized TD target
        Update actor to maximize $Q - \alpha \log \pi$
        Soft-update target critics
    end if
end for

I also warm-started SAC using BC parameters and prefilling its replay buffer with expert episodes. SAC achieves highly consistent but overly conservative behavior: it almost never crashes, but it also fails to reach far OOD goals. It stabilizes early and does not explore enough to reduce distance.

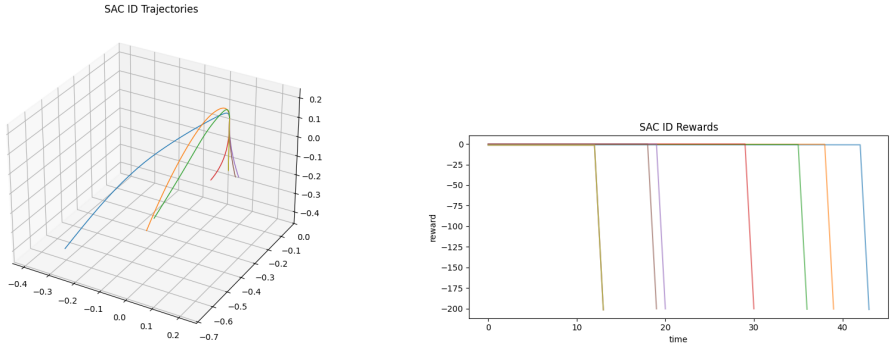

Figure 7: SAC in-distribution performance.

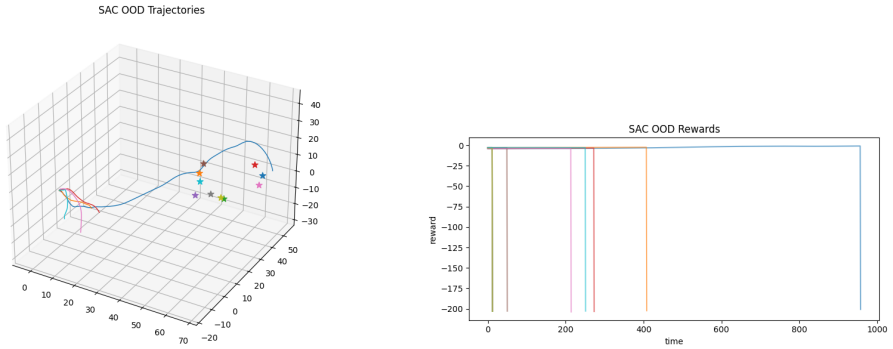

Figure 8: SAC out-of-distribution performance.

6 Comparative Analyses

Reward Statistics

Algo	ID Mean	ID Std	OOD Mean	OOD Std
BC	-500.21	778.11	-1571.62	670.21
DAgger	151.43	109.98	-566.23	301.27
PPO	-169.33	331.45	-881.27	150.75
SAC	-218.74	12.97	-816.02	697.54

Table 1: Return distribution across 20 rollouts per algorithm.

Outcome Statistics

Algo	ID S/C/H	ID FinalDist	ID Steps	OOD S/C/H	OOD FinalDist	OOD Steps
BC	3/1/5	26.64	756.1	1/7/2	73.02	638.9
DAgger	9/0/0	0.99	326.4	6/0/4	1.35	656.0
PPO	3/0/6	4.94	750.6	0/1/9	9.06	924.1
SAC	0/9/0	16.20	26.1	0/10/0	55.35	221.1

Table 2: Success (S), crash (C), hover (H), distance to goal, and steps.

From these results, DAgger clearly stands out. It achieves the highest success rate, lowest variance, and the best robustness to OOD targets. This matches theory: DAgger prevents compounding errors by repeatedly correcting the learner on its own state distribution, which is exactly the failure mode of BC.

PPO improves over BC but remains inconsistent. Its on-policy updates and limited exploration horizon make it sensitive to long episodes. When PPO drifts into states with poor value estimates, it struggles to recover, which explains the mix of successes and hovering outcomes.

SAC behaves very differently. In theory, SAC's entropy regularization should encourage broader exploration. In practice, on this helicopter environment, the stochastic policy combined with a highly sensitive continuous dynamics system leads it into unstable flight regimes. Because SAC learns off-policy, a few bad states in the replay buffer can dominate learning early, producing the high crash rate observed. SAC becomes over-exploratory rather than safe or conservative here, unlike in simpler locomotion tasks where entropy helps. This explains why SAC almost never succeeds, even though it takes fewer steps before termination, the crashes happen quickly.

7 Future Work

A natural direction is to explore model-based reinforcement learning methods such as Dreamer [5] or PETS [2]. Both approaches learn latent or ensemble-based dynamics models and plan ahead before taking actions. This fits the helicopter problem well: long-horizon credit assignment, unstable physics, and sparse reward without shaping are exactly the settings where model-based RL is known to outperform model-free methods.

My environment also includes v2 and v3 extensions with static and moving obstacles. These versions introduce risk-sensitive navigation and partial observability, which are strong use-cases for MPC-guided planning or hierarchical RL architectures. Exploring these directions would likely produce safer and more robust helicopter controllers.

8 Contributions of each each student

I implemented the helicopter environment, designed the reward shaping, built the PID expert, collected demonstrations, implemented BC, DAgger, PPO, and SAC, and created the comparative evaluation pipeline and visualizations.

References

[1] Abbeel, P., Coates, A., and Ng, A. Y. (2010). Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639.

[2] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*.

[3] Foundation, F. (2023). Gymnasium: A standard api for reinforcement learning environments.

[4] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proceedings of the International Conference on Machine Learning (ICML)*.

[5] Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Dream to control: Learning behaviors by latent imagination. *International Conference on Learning Representations (ICLR)*.

[6] Laboratory, S. (2010). Stanford autonomous helicopter project.

[7] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 627–635.

[8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.