Reinforcement and Imitation Learning Applied to Autonomous Aerial Robot Control

Gabriel Moraes Barros[1] and Esther L. Colombini[1][0000−0003−0467−3133]

Laboratory of Robotics and Cognitive Systems, Institute of Computing
University of Campinas, Campinas, São Paulo, Brazil
gabrielmoraesbarros@gmail.com,esther@ic.unicamp.br
http://larocs.ic.unicamp.br

Abstract. In robotics, the ultimate goal of reinforcement learning is to endow robots with the ability to learn, improve, adapt, and reproduce tasks with dynamically changing constraints based on exploration and autonomous learning. Reinforcement Learning (RL) aims at addressing this problem by enabling a robot to learn behaviors through trial-and-error. With RL, a Neural Network can be trained as a function approximator to directly map states to actuator commands making any predefined control structure not-needed for training. However, the knowledge required to converge these methods is usually built from scratch. Learning may take a long time, not to mention that RL algorithms need a stated reward function. Sometimes, it is not trivial to define one. Often it is easier for a teacher, human or intelligent agent, do demonstrate the desired behavior or how to accomplish a given task. Humans and other animals have a natural ability to learn skills from observation, often from merely seeing these skills' effects: without direct knowledge of the underlying actions. The same principle exists in Imitation Learning, a practical approach for autonomous systems to acquire control policies when an explicit reward function is unavailable, using supervision provided as demonstrations from an expert, typically a human operator. In this scenario, this work's primary objective is to design an agent that can successfully imitate a prior acquired control policy using Imitation Learning. The chosen algorithm is GAIL since we consider that it is the proper algorithm to tackle this problem by utilizing expert (state, action) trajectories. As reference expert trajectories, we implement state-of-the-art on and off-policy methods PPO and SAC. Results show that the learned policies for all three methods can solve the task of low-level control of a quadrotor and that all can account for generalization on the original tasks.

Keywords: Reinforcement Learning · Imitation Learning · Aerial Robots

M.Sc. Disseration submitted to CTDR 2020 - Defense date: 22/4/2020
Supervisor: Prof. Dr. Esther Luna Colombini

1 Introduction

While Robotics has shaped the way that the world manufactures its goods nowadays, it was mainly due to Optimum Control, with restricting tolerances and, predominantly, careful tuning of an expert. Although robotic applications in industry are suitable for high-volume production of a few different items (like automobiles), it cannot change rapidly for newer tasks, especially by a group of non-roboticists and a massive part of the universe of tasks that could be handled by robots is neglected due to a restrictive starting costs.

If quicker modeling is wanted, it would be more beneficial to learn control policies. Model-free RL [1] is an area of research that tries to achieve such results by learning from experience. RL aims to solve the same problem as optimal control. Still, since the state transitions dynamics is not available to the agent, the consequences of its actions have to be learned by itself while interacting with the environment, by trial-and-error. More recently, RL methods were coupled to Neural Networks forming the Deep Reinforcement Learning (DRL) reserach field.

Although recent advances in DRL have shown tremendous and promissory results, one of the most significant caveats of this solution is that the nature of the learning by gradient descent requires a considerable training time and a lot of interactions (they are the equivalent to samples in supervised learning). If we are accepting that the main driver for utilizing model-free algorithms is that we need solutions in a shorter deploy time, and with a lower influence of experts, it is seriously problematic to need large datasets for each task that the robot will learn how to perform. One way to mitigate these drawbacks is **learning by demonstration**, where a robot agent learns the optimal policy by observing an expert agent (that is expected to have the near-optimal policy). We will address this learning as **Imitation Learning** (IL).

Learning behaviors by IL, especially considering that we can learn from humans and/or other robots, is one of the most promissory new fields in robotics. This approach can significantly decrease the training time and, hopefully, in the future, have robots that can learn daily tasks just by watching them being performed on the Internet. Although nowadays robots apply a feature extraction process to guide policy search while learning with the master' demonstration, in the future, an end-to-end approach based on vision is expected to prevail.

In this work, we aim at studying how Imitation Learning can help Reinforcement Learning to address the problem of learning to perform tasks in the aerial robotics context. By doing this, we aim at contributing to the state-of-the-art in the search for intelligent agents that can quickly grasp new behaviors and infer what they could (and what they need to) learn by absorbing other agents' knowledge. The chosen algorithm is Generative Adversarial Imitation Learning (GAIL) since we consider that it is the proper algorithm to tackle this problem by utilizing expert (state, action) trajectories.

Although the research focus is related to IL, having a trained optimal policy $\pi*$ that will work as a reference for the imitated behavior is mandatory. To tackle this necessity, we chose two state-of-art algorithms, one on-policy - the PPO (Proximal Policy Optimization) [2] - and the other off-policy - SAC (Soft

Actor-Critic)[3], [4] - to perform the tasks and then to use the rollouts of the expert trajectories as the imitation process input.

As contributions of this work, we can cite: i) an open-source version of GAIL applied to robotics agents in more complex scenarios than those available in the literature; II) the first use of SAC to learn a low-level control policy for Quadrotors; iii) a framework to use the Drone agent developed by [5][6] with the plugin PyRep [7], that is a Python wrapper for the API of the new Coppelia Simulator [8].

2 Related Work

The literature for research in UAVs is vast. However, we are mainly interested in its intersection with Reinforcement Learning and Imitation Learning applied to UAVs. Generally, DRL is applied in UAVs in high-level control, like in Navigation [9][10][11], Autonomous Landing [12][13][14][15] and Target Tracking [16]. For low-level control, however, few works have been proposed. Zhang, et. al [17] designed a low-level control using MPC with guided-policy-search. The MPC control, mapping raw sensor data to rotor velocities, is used only in the training phase (since it is computationally expensive). Then, they employed a supervised approach to learning the final policy.

In [18], Koch et. al propose an open-source high-fidelity simulation environment, GymFC. They compared a PID controller to three model-free DRL approaches PPO, TRPO, and DDPG in an attitude control task (focusing only on the propellers' thrust and the agent's angular velocities). Xu et al. [19] used DRL (PPO) to perform model-free learning to automatically change from fixed-wing to multicopter setup and vice-versa and model-agnostic use of the controller, employing it on different drone configurations.

Hwangbo et al. [20] proposed one of the first low-level controllers with DRL. They used DRL to control a UAV, but also a PD controller to help the training phase. They employed a model-free deterministic policy gradient approach. This approach leads to very sample inefficient methods that could not be trained in more complex simulators with better dynamic models. Our work builds upon the research made by Lopes et al. [6], where a first stochastic low-level controller based on DRL (PPO) was proposed. The authors trained a Parrot simulated robot in Coppelia for a similar task as in [20].

3 Material and Methods

To fulfill our goals, we first learn optimal policies with an on-policy formulation (PPO) and off-policy (SAC), saving its trajectories as tuples of (state, action) for each episode. Then, we use GAIL to train a control policy with the expert trajectories. Finally, we observe whether the result is similar to the traditional RL methods since IL alleviates the strenuous task of reward-shaping (finding the best reward-function that best performs under the desired situation).

3.1 Problem Formulation

We want to train a policy control, using DRL and IL to a **go-to-target** task. Figure 1 shows our target scene with an immobile target in the center (position

$[x, y, z]$). At each training start, the drone is dropped in the air in different positions and orientations. The quadrotor target following task can be naturally formulated as a sequential decision-making problem under the RL framework. At each particular timestep, the agent receives an observation s_t from the environment (the onboard sensors or, in our case, the state information given by the simulator), and acts according to a policy $\pi(s)$ and then observes a reward r_t

The agent's goal is to learn an optimal policy π^* that maximizes the expected sum of discounted rewards R_t, and we use both SAC and PPO for this purpose evaluating many different formulations of the MDP. Later, we use the best policy learned by PPO and imitate it using GAIL.

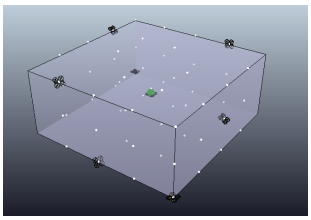

Fig. 1: Possible initial positions for the drone that cover all quadrants of the relative agent-to-target position vector.

Reward Function As reward shaping is a significant element to define the quality of the policy learned by an agent, we have tested more than 20 different reward-functions. For the purpose of this work, we will use the reward function defined next. We consider success in our task by getting close to the target (distance reward) as well as robustness and stability (zeroing the angular velocity when the UAV is at the target location).

The proposed reward function is defined by: $r_t(s) = r_{alive} - 1.25\epsilon_t(s)$, where ϵ_t is the position error between the target position and the quadrotor's position at timestep t. r_{alive} is a constant (alive bonus) used to assure the quadrotor earns a reward for flying inside a limited region (radius of 3.2 meters from target) and that r_t is always positive. In this reward-function, $r_{alive} = 4.0$.

Initializations We have tested three different initialization methods, two of them are the same as proposed in [5]. The target is initialized in a fixed position, located at the point $\xi_{target} = (x = 0.0, y = 0.0, z = 1.7)$ [m] for all initialization setups and it stands still for the whole duration of the simulation training phase.

I1: Initialization - Already on the goal. This initialization follows the simpler setup, leaving the drone in the same position as the target: $[x, y, z] = [0, 0, 1.7]$ and $[\phi, \theta, \psi] = [0, 0, 0]$.

I2: Initialization - Gaussian. In this initialization method the quadrotor starts in: $[x, y, z] = [\mathcal{N}(0, 0.3), \mathcal{N}(0, 0.3), 1.7 + \mathcal{N}(0, 0.3)]$ and $[\phi, \theta, \psi] = [\mathcal{N}(0, 0.6), \mathcal{N}(0, 0.6), \mathcal{N}(0, 0.6)]$.

I3: Initialization - Discretized Uniform. While testing and training several experiments with the initialization $I2$: Gaussian, we have noticed that not

all episodes were coming to an end (time $\geq$ time horizon), causing a higher variance in our Average Rewards for each batch size. We then performed many episodes with a trained deterministic policy and come to the conclusion that episodes that were starting with higher rotation angles, on average, were more likely to fail. So, we hypothesized that this was due to the long tails in a gaussian distribution. Hence, causing the drone to spend the majority of the training starting in easier setups. The initialization I3 is a uniform distribution where we can parametrize the *bound_limit* of the array and how many (*chunk_num*) chunks this array is divided in:

- $[x, y]$ were draw from the uniform discrete distribution $\mathcal{U}(-1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5)$
- $[z]$ from $\mathcal{U}(1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2., 2.1, 2.2)$
- $[\phi, \theta, \psi]$ from $\mathcal{U}(-44.69, -36.1, -26.93, -17.76, -9.17, 0.0, 9.17, 17.76, 26.93, 36.1, 44.69)$

States The state representation in our MDP is defined as $S_t = \{x, y, z, \phi, \theta, \psi, R_{11}, R_{12}, R_{13}, R_{21}, R_{22}, R_{23}, R_{31}, R_{32}, R_{33}, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}, a1_{t-1}, a2_{t-1}, a3_{t-1}, a4_{t-1}\}$, where (x, y, z) are the relative position from the drone to the target, (ϕ, θ, ψ) are the relative orientations, $(\dot{x}, \dot{y}, \dot{z})$ and $(\dot{\phi}, \dot{\theta}, \dot{\psi})$ are the linear and angular velocities, $(R_{11}, R_{12}, R_{13}, R_{21}, R_{22}, R_{23}, R_{31}, R_{32}, R_{33})$ are the rotation matrices and an_{t-1} are the actions taken in the last time step for all motors n.

Actions The action space for the problem is, for all experiments, defined by $A = \{a1, a2, a3, a4\}$ where each represents the value sent to each propeller (0-100%).

3.2 Simulation Framework

Our setup consists of robotics simulator **Coppelia Simulator**, a plugin called **PyRep**, and an environment **Environment** that access the full simulator API with through Pyrep and resembles the common RL interface from OpenAI Gym. The **Environment** corresponds to the MDP (state, action, reward) and general parameters modeled for each experiment. Our robot/agent is a simulated version of the AR Parrot drone.

4 Experiments

To evaluate the quality of policies learned by on-policy and off-policy methods, we followed distinct environment initialization (as described in Section 3.1) and compared the resulting episodes played with our deterministic PPO and SAC π^*. We then trained GAIL to learn to imitate the best expert policy generated by PPO, evaluating its resulting system in trajectories not presented during learning. Finally, we evaluate SAC, PPO, and GAIL robustness. Due to paper length restrictions, we will show GAIL results directly compared to PPO learned expert policies. SAC results will be presented when we compare all algorithms.

4.1 GAIL on On-Policy expert trajectories

After training PPO with distinct initialization methods, we perform imitation learning on top of the PPO optimal policy. For the first experiment (Figure 2),

where PPO was trained with initialization $I1$, we achieved results very similar to the ones achieved by the original policy. Indeed, when comparing PPO and GAIL policies, we can see almost no difference in the resulting behavior.

We also trained GAIL to mimic a PPO expert policy trained with initialization $I2$. As we can see in Figure 3, GAIL demonstrated very similar behavior to the expert policy, especially related to steady-state. For all six variables (position and angle) that we look at GAIL, it presents a step-response with more overshoot at the beginning of the episode (Figure 3). While learning from expert trajectories sampled from different initial pose configurations, it can be hard to properly evaluate what, in a complex dynamics setup like a quadrotor falling from different relative positions and orientations, the ideal response should look like. The mimicked policy shows a smaller steady-state error in x but overall depicts very similar results. This can show that, maybe the original policy has a difficult time optimizing to survive the initial conditions and fine-tune the steadiness with zero vibration. We point to the reader that more research is needed in this regard, including trying curriculum learning first to learn how to fly without falling and get closer to the target and then fine-tune to erase the steady-error and vibration.

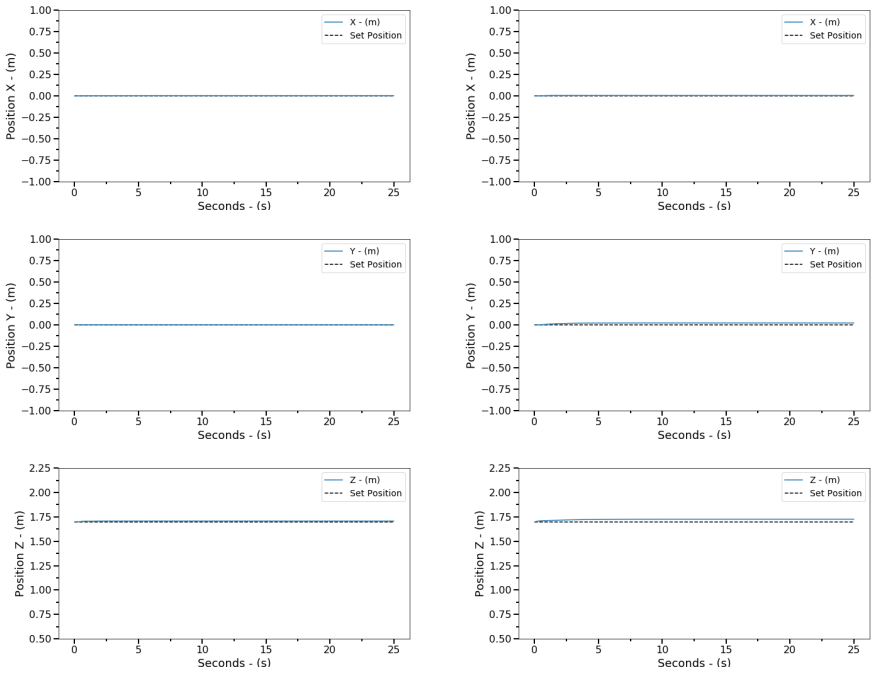

Fig. 2: Distances for PPO and GAIL trained upon PPO with initialization I1. Left - PPO positions x, y, and z. Right - GAIL positions x, y, and z. All in (m).

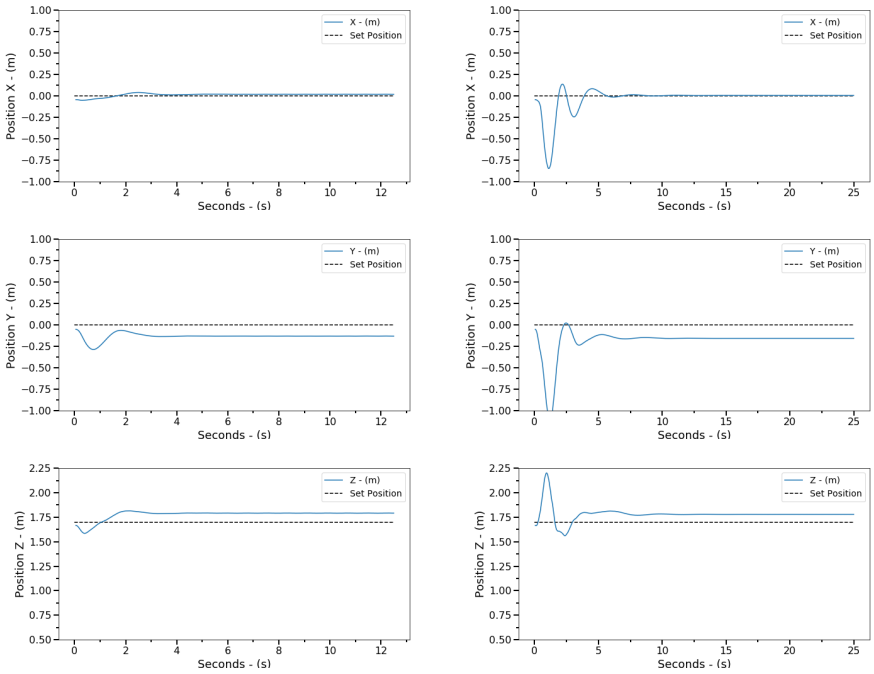

Fig. 3: Distances for PPO and GAIL trained upon PPO with initialization I2. Left - PPO positions x, y, and z. Right - GAIL positions x, y, and z. All in (m).

4.2 GAIL's control performance on different trajectories

Imitation Learning usually has a well-know effect, especially in behavior cloning. Almost any attempt to imitate the expert's trajectories would generate a policy that cannot be abstract to unseen tasks. However, to show that the learned policy was able to generalize to more complex behavior, we run them to follow a moving target describing three distinct paths.

Line. For this experiment, we run the same GAIL policy depicted in Figure 2 to follow a moving target with two different speeds in a rectilinear path. The result is presented in Figure 4. We can see an overshoot in the beginning and after a steady control of the quadrotor for the low-velocity task. For the higher velocity task, the trajectory is not very rectilinear, showing that maybe the policy trained with IL may have a harder time abstracting to the unseen task of a fast mobile goal. This line trajectory evidences what we have discussed earlier regarding problems at the beginning of the episodes. This was a PPO problem amplified in GAIL mainly because the GAIL policy was trained with expert trajectories from PPO. Although the target only moves in x, the agent has a significant movement in y at the start of the episode. The graph in Figure 4 b) shows that we have a more substantial latency in our trajectory since the agent never recovers the time lost with the clumsy start.

Square. For this experiment, we run the same GAIL policy depicted in Figure 2 to follow a moving target with two different speeds in a square path. With the trajectory performed by the slow target, the agent has a slow beginning with its step-response and than achieves a right trajectory, but present a delayed maneuver every corner of the square. The target is not an object that the simulator calculates its dynamics, so it does not have any problems with a sharp 90 degrees curve, but the UAV with the GAIL-trained policy has difficulties doing so. In Figure 5 b) and c), we can observe that the agent has a problematic start that creates a gap between the agent and the target, and this gap increases in each time the target gets close to the corner of the square. Figure 5 d), on z-position, depicts the problem that the GAIL policy has it this trajectory since the curve performed by the agent is not parallel with the transversal plane (xoy-plane with equation z=1.7). This problem is increased with the fast-moving target making the trajectory hardly resemble the one of a square.

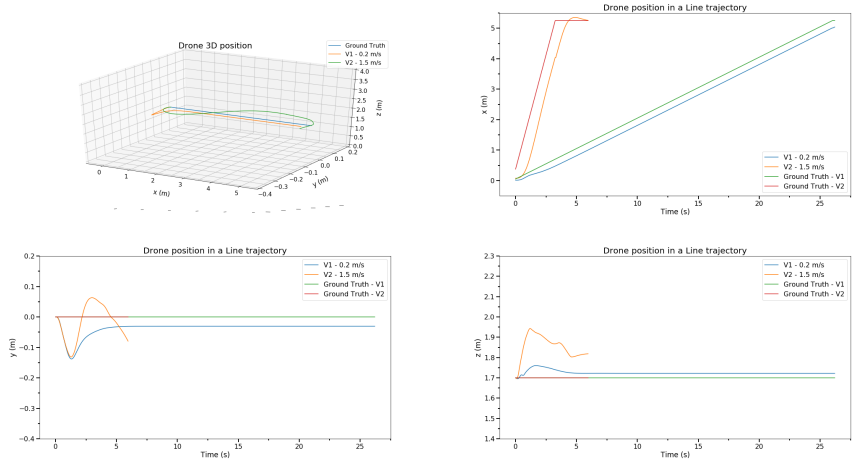

Fig. 4: GAIL trained with PPO. (a) GAIL trained upon PPO 3D position for the Line trajectory (b) Position x (c) Position y (d) Position z.

Sinusoidal. For this experiment, we run the same GAIL policy depicted in Figure 2 to follow a moving target with two different speeds in a Sinusoidal path. The Sinusoidal trajectory brings more light into the policy's problems using GAIL (Figure 6). The slow-target path following starts once again with overshoot and then follows along a good sinusoidal trajectory, with a steady-error. It is clear that when the agent is dropped-off at the start of the training, its policy quickly acts to survive, but then it does follow the target without compensating the beginning. The reason may be that the expert trajectories were drawn from a policy that emphasizes stability and low angular velocities. In the fast-target trajectory, the drone shows that it does not have a proper stable path in higher velocities since it travels in the transversal (this time the

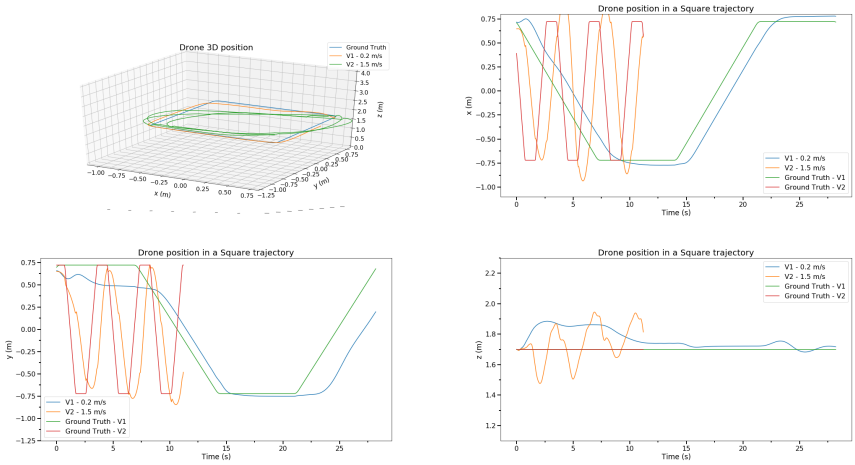

Fig. 5: GAIL trained upon PPO. (a) GAIL trained upon PPO 3D position for the Square trajectory (b) Position x (c) Position y (d) Position z.

zox-plane) plane of the trajectory in our case, the y-axis in Figure 6. Figure 6 c), of y-position, shows again the problem that GAIL has at the start of the episode, which causes a significant movement in y, something that was not supposed to happen. The graph in Figures 6 b) and d) show a good sinusoidal trajectory, with expected differences in the peaks and valleys caused by the agent. The learned trajectory is delayed in these positions due to the problems at the start of the episode.

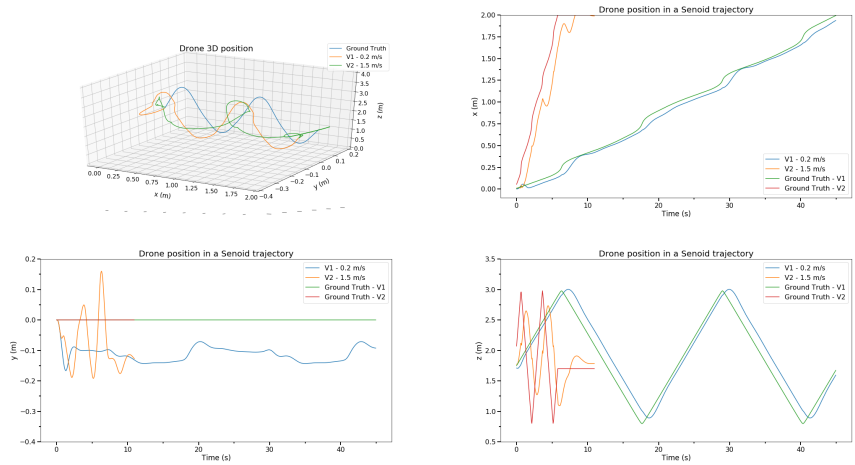

Fig. 6: GAIL trained upon PPO. (a) GAIL trained upon PPO 3D position for the Sinusoidal trajectory (b) Position x (c) Position y (d) Position z.

4.3 Algorithm Comparison

For summary purposes, we show the comparison for the three best policies learned by SAC, PPO, and GAIL in the trajectories not previously seen in training. For the Line trajectory, in Figure 7, it gets more apparent what we once said about the overshoot of the start in GAIL and PPO, something that is less prevalent with our SAC policy. The training with SAC did not use normalization at all. Still, PPO and GAIL used a running_normalizer, something common in Policy Gradient formulations, which could influence the initialization of on-policy algorithms. It is important to remark that GAIL, for this task, performed better than the PPO expert used as a reference. We can also notice that, although PPO and GAIL move significantly in the y-direction at the beginning of the episode, PPO is more efficient in correcting its trajectory. For the Square trajectory, although SAC's path is more stable and parallel to the ground plane (while still with the steady-state error on the z-direction), the paths of PPO and GAIL are closer to the Ground Truth. At the beginning of the task, the overshoot, familiar to PPO and GAIL, did not occur in this trajectory. For the Sinusoidal task, the start made the trajectory even dephased with the ground truth, which does not occur with SAC. However, we can see that, although SAC performs better here, GAIL mimics the behavior of PPO even for unseen trajectories.

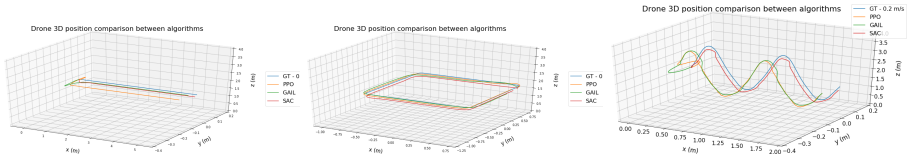

Fig. 7: Drone 3D position comparison between algorithms in the Line, Square and Sinusoidal trajectories.

4.4 Robustness of the trained deterministic policies

One common problem in DRL resources is the lack of robustness of the trained policies and brittleness of the algorithms in the training phase. To evaluate our trained policies' robustness, we perform 216 episodes with the most extreme values in the initialization setup Discretized_Uniform (section 3.1) I3. The total of possible timesteps is $216 * 250 = 54000$, but the failed episodes are halt before completing 250 timesteps. The limit starting positions are: $[x, y]$ from the uniform discrete distribution $\mathcal{U}$([-1.5, 1.5]), $[z]$ from $\mathcal{U}$([1.2, 2.2]), $[\phi, \theta, \psi]$ from $\mathcal{U}$([-44.69, 44.69]). Table 1 presents the results from this experiment. PPO achieved a good result, with 850.72 mean and 904.16 median. It is important to notice that the failed episodes are lower than the median value. As episodes start in different positions, high variance in the rewards does not necessarily mean low policy quality. We can see that GAIL did not present a good result in terms of robustness, with only 54% of successful runs. As we saw in the Results (Figures 4.1 and 4.2) discussion, the overshoot of GAIL at the start of each episode was probably because the algorithm did not quite precisely learn an optimal policy for the harsh initial phases of our training task. SAC achieved a perfect score in this experiment, completing 100% of the episodes attempted. Figure 8 shows an example of SAC performing over harsh untrained initial conditions.

Table 1: Results for testing robustness.

Algorithm	**Mean**	**Median**	**% Successful runs**
PPO	850.72	904.16	93.98
SAC	886.13	886.81	100.0
GAIL	476.34	810.42	54.16

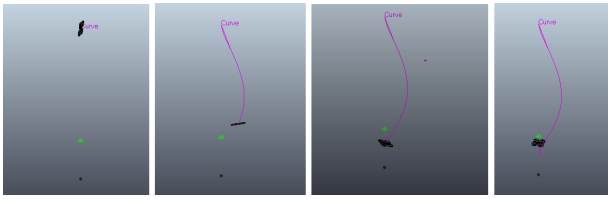

Fig. 8: Recovery trajectory after initialization with $[x, y, z] = [0, 0, 6]$ and $[\phi, \theta, \psi] = [0, 90°, 0]$ using SAC π^{*s}. The green sphere is the reference target.

5 Conclusion

In this work, we extensively approached the literature of DRL, Imitation Learning, and DRL applied to UAV control. We trained quasi-optimum policies with PPO and SAC for the go-to-target task for a UAV low-level control. Then, we used GAIL to learn by imitating a policy based on the expert trajectories of our PPO best policy. We also evaluated the policies in a similar task for which the policies were not trained (moving targets). We assessed, discussed, and evaluated all of our results. We demonstrated that it is possible to train an efficient low-level controller for quadrotor flight using both DRL and Imitation Learning. For that purpose, we first trained a reference policy using PPO, a state-of-art model-free algorithm that is on-policy. Although it was not straight forward and we had to use our modifications in existing open-source frameworks, we have validated the hypothesis in [5] that a PPO could be used for training a quadrotor controller. We also show that we can use SAC, state-of-art model-free off-policy algorithm, to train a low-level controller for quadrotor flight. As far as we are concerned, this is the first work to do so. SAC was also a better choice for a control agent than PPO, achieving better results in terms of control stability and robustness than PPO. Finally, we also showed that one could use GAIL to train an optimal policy almost as good as the original expert policy without the need for reward shaping. We have seen that the resulting agent mimics the expert policy with all its generalization capabilities. We evaluated the impact of using distinct state representations, reward functions, and training initialization procedures in the learned policy during the work, both in reward optimization and control policy quality. We also have created a wrapper for the environment proposed in [5] using the PyRep framework and performed several tests to assert that the two ends are similar using a 10^{-5} tolerance.

References

1. R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, 2nd Edition, MIT Press, Cambridge, MA, USA, 2018.

2. S. John, W. Filip, D. Prafulla, R. Alec, K. Oleg, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).
3. T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, CoRR abs/1801.01290 (2018). arXiv:1801.01290.
4. T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, S. Levine, Soft actor-critic algorithms and applications, CoRR abs/1812.05905 (2018). arXiv:1812.05905.
5. G. C. Lopes, Intelligent control of a quadrotor using reinforcement learning with proximal policy optimization, 2018.
6. G. Lopes, M. Ferreira, A. Simoes, E. Colombini, Intelligent control of a quadrotor with proximal policy optimization reinforcement learning, in: Latin American Robotic Symposium, 2018, pp. 503–508.
7. S. James, M. Freese, A. J. Davison, Pyrep: Bringing v-rep to deep robot learning, arXiv preprint arXiv:1906.11176 (2019).
8. E. Rohmer, S. P. N. Singh, M. Freese, Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework, in: IEEE IROS, 2013.
9. Y. Xu, Z. Liu, X. Wang, Monocular Vision based Autonomous Landing of Quadrotor through Deep Reinforcement Learning, in: 2018 37th Chinese Control Conference (CCC), 2018, pp. 10014–10019, iSSN: 1934-1768.
10. R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, A. Cangelosi, Autonomous Quadrotor Landing using Deep Reinforcement Learning, arXiv:1709.03339 [cs]ArXiv: 1709.03339 (Feb. 2018).
11. C. Sampedro, A. Rodriguez-Ramos, I. Gil, L. Mejias, P. Campoy, Image-Based Visual Servoing Controller for Multirotor Aerial Robots Using Deep Reinforcement Learning, in: IEEE IROS, IEEE, Madrid, 2018, pp. 979–986.
12. C. Wang, J. Wang, Y. Shen, X. Zhang, Autonomous Navigation of UAVs in Large-Scale Complex Environments: A Deep Reinforcement Learning Approach, IEEE Transactions on Vehicular Technology 68 (3) (2019) 2124–2136.
13. B. Zhou, W. Wang, Z. Liu, J. Wang, Vision-based Navigation of UAV with Continuous Action Space Using Deep Reinforcement Learning, in: 2019 Chinese Control And Decision Conference (CCDC), 2019, pp. 5030–5035, iSSN: 1948-9447.
14. S. Krishnan, B. Borojerdian, W. Fu, A. Faust, V. J. Reddi, Air Learning: An AI Research Platform for Algorithm-Hardware Benchmarking of Autonomous Aerial Robots, arXiv:1906.00421 [cs]ArXiv: 1906.00421 (Jun. 2019).
15. S. Shah, D. Dey, C. Lovett, A. Kapoor, Airsim: High-fidelity visual and physical simulation for autonomous vehicles, in: Field and Service Robotics, 2017.
16. S. Li, T. Liu, C. Zhang, D.-Y. Yeung, S. Shen, Learning Unmanned Aerial Vehicle Control for Autonomous Target Following, arXiv:1709.08233 [cs] (2017).
17. T. Zhang, G. Kahn, S. Levine, P. Abbeel, Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search, arXiv:1509.06791 (2016).
18. W. Koch, R. Mancuso, R. West, A. Bestavros, Reinforcement Learning for UAV Attitude Control, ACM Transactions on Cyber-Physical Systems 3 (2) (2019) 1–21.
19. J. Xu, T. Du, M. Foshey, B. Li, B. Zhu, A. Schulz, W. Matusik, Learning to fly: computational controller design for hybrid UAVs with reinforcement learning, ACM Transactions on Graphics 38 (4) (2019) 1–12.
20. J. Hwangbo, I. Sa, R. Siegwart, M. Hutter, Control of a quadrotor with reinforcement learning, IEEE Robotics and Automation Letters 2 (4) (2017) 2096–2103.