Abstract

Recent technological advances have led to the increased development of inexpensive unmanned aerial vehicles (UAVs) for exploration and monitoring in applications such as geophysical mapping, route and agricultural surveillance, natural disaster planning and response, and environmental data collection has led to the need for the development of coordinated path-planning algorithms. Deep reinforcement learning (RL) is one approach that may be useful in generating policies that enable robust drone performance in uncertain environments. **In this work, we trained deep RL polices such as Soft Actor-Critic (SAC) and Deep Deterministic Policy Gradient (DDPG) within a gym environment we developed.**

We consider a model problem in which a swarm of UAVs are tasked with mapping targets within a domain. Each individual drone is subject to crashing into obstacles, other drones, the ground, and the boundaries of the domain, which the swarms must learn to avoid. The basic UAV swarm dynamics are derived from Newton's laws of motion, where individual drones are subject to drag forces and are able to emit a constant propulsion force. The *control inputs* for each drone include a normalized force vector that determines the direction of propulsion. In this model problem, we assume that the swarm has *full observability* of the environment (domain boundary, drones, targets, and obstacles), and that the objective/reward function is well defined.

To compare the performance of DDPG and SAC, both were run for 10000 episodes, each of which had a maximum rollout length of 300 steps. Training began after 10000 transitions were collected into the memory buffer, with 10 gradient update steps run between each 1000 rollout transitions collected. The learning curves for DDPG and SAC are shown in the results section of the report, and indicate some initial success in training, as both actor and critic losses largely trended downward as training continued.

We developed and tested a physics-based drone simulator capable of deploying a UAV swarm to capture targets while avoiding obstacles. Our experiments suggest that deep reinforcement learning algorithms, such as DDPG and SAC, can achieve reasonable results and demonstrate the feasibility of using RL for controlling a drone swarm. However, there are still areas for improvement and further research.

Github repository:

Exploration and Optimal Path Planning for UAV Swarms

Trajectory optimization via deep reinforcement learning vs genetic algorithm

COMPSCI 285 - Deep Reinforcement Learning, Decision Making, and Control

Maya Horii, Brian Howell, Reece Huff

University of California, Berkeley

December 2022

Github repository:

1 Introduction

Background

Recent advances in lithium-ion battery, camera, 3D printing, LiDAR, and hyper-spectral imaging technologies have led to the increased development of inexpensive unmanned aerial vehicles (UAVs) for a large number of applications. In particular, the use of a large number of UAVs (or swarms) for exploration and monitoring in applications such as geo-physical mapping, route and agricultural surveillance, natural disaster planning and response, and environmental data collection has led to the need for the development of coordinated path-planning algorithms.

Currently, UAV exploration range still remains an expensive task due potential collisions in uncertain environments and the limited power supply for each drone. The development of a digital twin/replica can aid in simulating trajectories of UAVs within a particular environment many thousands of times faster than in real time due parallelization of fast numerical models. This framework enables engineers to inexpensively iterate in simulation a large range of possibilities to ensure robust control mechanisms for operation in the real world.

Model problem and assumptions

The modeling of UAV swarms may contain varying degrees of fidelity, including incorporation of the physics of power supply to rotors, the dynamics of linear and angular momentum, the effects from drag, the interaction between drones, and the coordination to achieve a collective objective. Optimal trajectories, however, can rapidly be computed by focusing primarily on effects from drag, interactions and objective, while treating individual drones as point masses. This enables simulations that can be performed 1000x faster than real time, while still obtaining reasonable flight trajectories. A visualization of a potential initial configuration is shown in Figure 1.

We consider a model problem in which a swarm of UAVs are tasked with mapping targets within a domain. Each individual drone is subject to crashing into obstacles, other drones, the ground, and the boundaries of the domain, which the swarms must learn to avoid. The basic UAV swarm dynamics are derived from Newton's laws of motion, where individual drones are subject to drag forces and are able to emit a constant propulsion force. The *control inputs* for each drone include a normalized force vector the determines the direction of propulsion. In this model problem, we assume that the swarm has *full observability* of the environment (domain boundary, drones, targets, and obstacles), and that

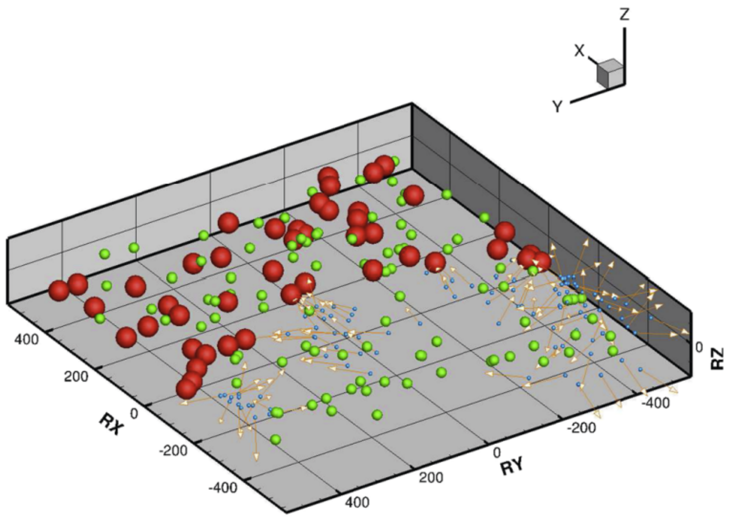


Figure 1: Example drone environment obtained from [1]. Agent (blue) seek to tag targets (green) while avoiding obstacles (red).

objective/reward function is well defined. For this project, we developed a gym environment that that enables test trajectories provided from a variety of approaches, including deep RL policies and physics-based models.

Objective of work

Optimal trajectories for the model problem above can be obtained through a variety of methods. In this project, we will compare and contrast genetic algorithm and deep reinforcement learning (RL) approaches for obtaining optimal path trajectories, and discuss their strengths and weaknesses and providing examples of their use in trajectory optimization. For deep RL, we explore the Soft Actor-Critic (SAC) and Deep Deterministic Policy Gradient (DDPG) methods for developing policies to control swarm of drones to successfully tag targets within 3D domain while avoiding potential crashes. These results are benchmarked against previous work using genetic algorithms (GA) that optimize the path planning determined by an exponential attraction-repulsion model. We also *attempt* to compare the policies obtained by SAC, DDPG and physics-based GA under similar environments. These results can be used to compare these methods to each other in terms of effectiveness and efficiency.

2 Methods

Genetic algorithm

Previously, Zohdi [1] proposed a simple physics-based model by approximating a swarm of UAVs as point masses with Newtonian states position $\boldsymbol{r}$, velocity $\boldsymbol{v}$ and acceleration $\boldsymbol{a}$. This work provided a path-planning algorithm with the objective of mapping a desired domain by means of modeling drone-target, drone-obstacle, drone-drone interactions throughout the domain:

$$\hat{\boldsymbol{n}}_{i\to j}^{\text{tar}} = (w_{t1}e^{-a_1 d_{ij}^{dt}} - w_{t2}e^{-a_2 d_{ij}^{dt}})\boldsymbol{n}_{i\to j} \tag{1}$$

$$\hat{\boldsymbol{n}}_{i\to j}^{\text{obs}} = (w_{o1}e^{-b_1 d_{ij}^{do}} - w_{o2}e^{-b_2 d_{ij}^{do}})\boldsymbol{n}_{i\to j} \tag{2}$$

$$\hat{\boldsymbol{n}}_{i\to j}^{\text{drone}} = (w_{d1}e^{-c_1 d_{ij}^{dd}} - w_{d2}e^{-c_2 d_{ij}^{dd}})\boldsymbol{n}_{i\to j} \tag{3}$$

.

Specifically, these equations model the weighted directions from a drone to a desired target, or away from obstacles or other drones, where $\boldsymbol{d}_{ij}$ is the euclidean distance between an interaction, and the parameters ($\boldsymbol{w}_t$, $\boldsymbol{w}_o$, $\boldsymbol{w}_d$, $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$) are parameters learned by a genetic algorithm with an objective function defined as

$$\min_{\Lambda} \quad w_1 M^* + w_2 T^* + w_3 L^* \tag{4}$$

where M^* indicates the percent of unmapped targets, T^* indicates the total time, L^* indicates the percent of crashed drones, and Λ represents a vector of parameters in Equations 1-3 to be learned.

Deep reinforcement learning

In this work, we consider the soft actor-critic (SAC) and deep deterministic policy gradients (DDPG) methods to accommodate the continuous action space (normalized propulsion vector). However, given the time constraints of the project, several simplifications and modifications had to be made. First, the problem was simplified to a 2D domain with a single target with two agents. This simplification greatly reduced the computational cost enabling us to more rapidly debug our RL implementations.

The second modification of the original problem was a redefinition of the reward function. To accommodate a more continuous reward signal, the following reward function was implemented for both SAC and DDPG methods:

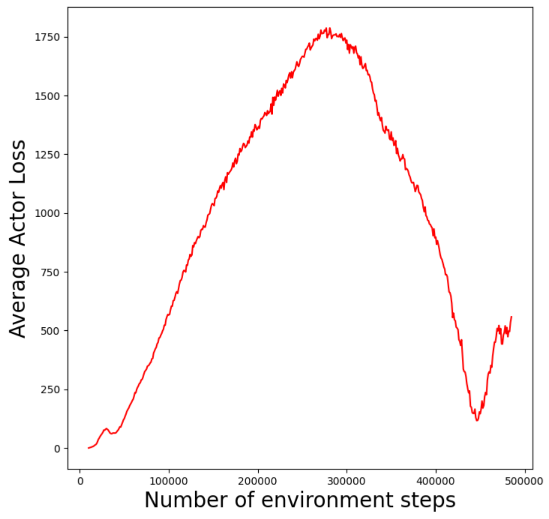
$$r = -\sum_{i=1}^{n_{agents}} \min_j \left(|a_i - t_j|\right) + d_{max}\left(n_{mapped} - n_{crashed}\right) \tag{5}$$

where a_i is the i-th agent, t_j is the j-th target, d_{max} represents the maximum possible distance in the domain between a possible agent and target, n_{mapped} represents the number of mapped targets, and $n_{crashed}$ represents the number of crashed UAVs. Intuitively, this reward function penalizes the UAV swarm by summing every agents distance to its nearest target, adding reward scaled by the maximum possible distance between an agent and a target in the specified domain, as well as a negative penalty for every crashed drone also scaled by the maximum distance between an agent and a target. The latter two terms are to account for the sudden decrease in reward due to the first term after target is tagged and no longer considered in the simulation.

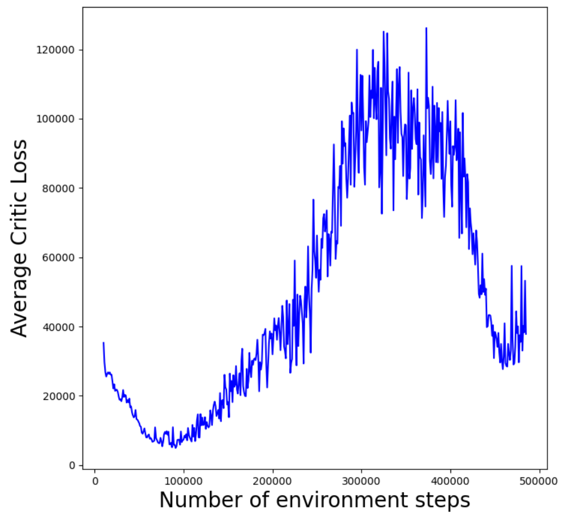
The soft actor-critic implementation was based heavily off of [2].

3 Results

To compare the performance of DDPG and SAC, both were run for 10000 episodes, each of which had a maximum rollout length of 300 steps. Training began after 10000 transitions were collected into the memory buffer, with 10 gradient update steps run between each 1000 rollout transitions collected. The learning curves for DDPG and SAC are shown in Figure 2 and 3 respectively, and indicate some initial success in training, as both actor and critic losses largely trended downward as training continued.

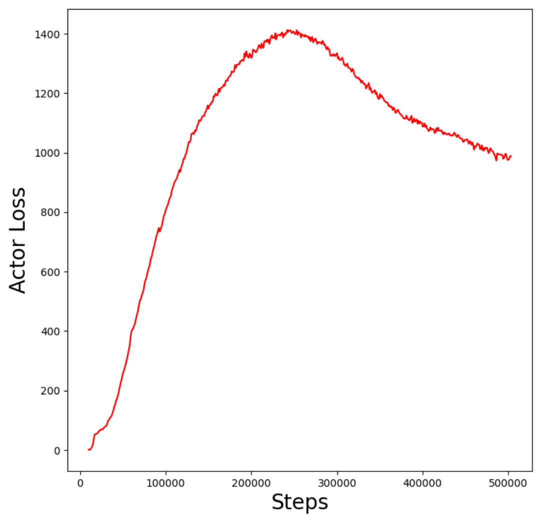


(a) Actor loss

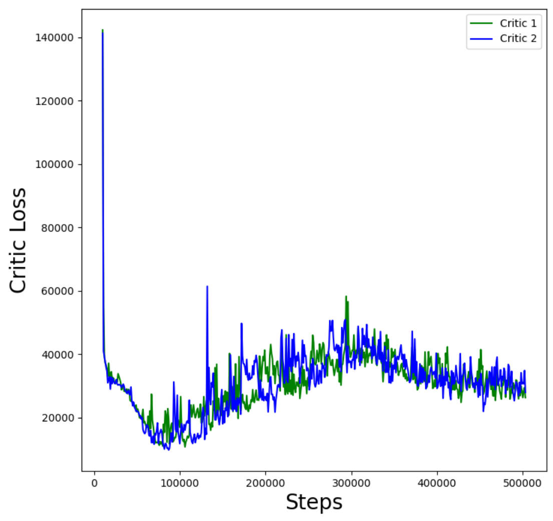


(b) Critic loss

Figure 2: Learning curves for DDPG implementation



(a) Actor Loss



(b) Critic Loss

Figure 3: Learning curves for SAC implementation

To compare the performance of the final policies generated with SAC and DDPG, both policies were run in the same 100 randomly initialized environments. The average reward was calculated over time for each episode. The mean and standard deviation of the average rewards are shown in Figure 4. The resulting performance of SAC and DDPG appear to be extremely similar, with SAC having a slightly higher mean. Both policies failed to consistently capture the target – DDPG captured the target once, SAC captured the target twice (out of 100 episodes). However, a high amount of variability was observed in SAC runs initialized with different random seeds, and at least one SAC-generated policy was able to consistently capture the target. An example from this high-performing policy is shown in Figure 5.

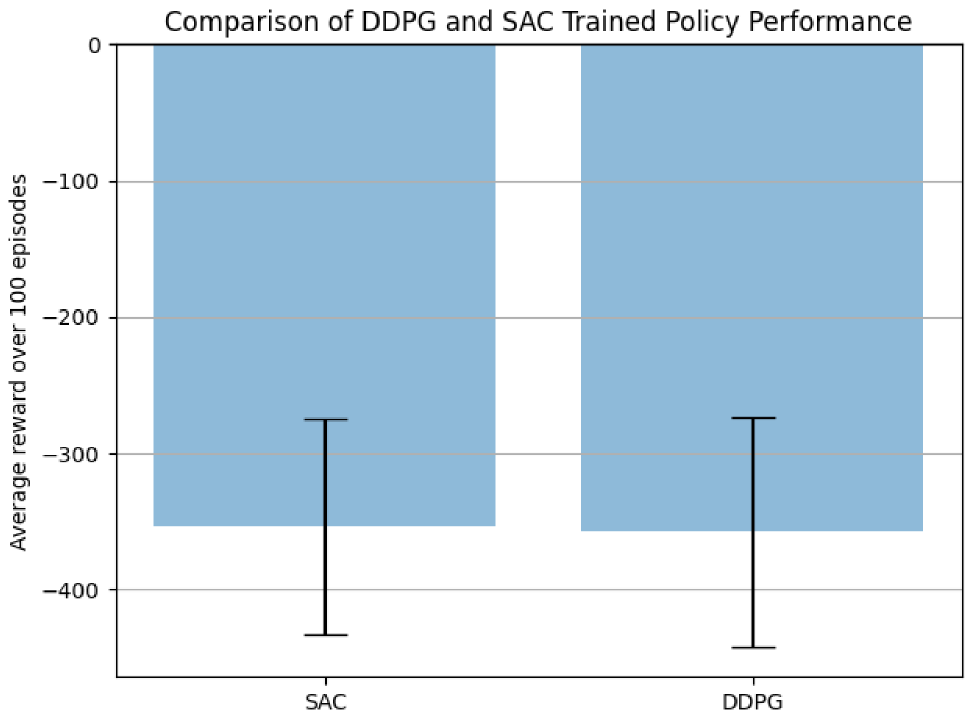


Figure 4: Comparison of mean and standard deviation of average episode reward over 100 episodes between SAC and DDPG

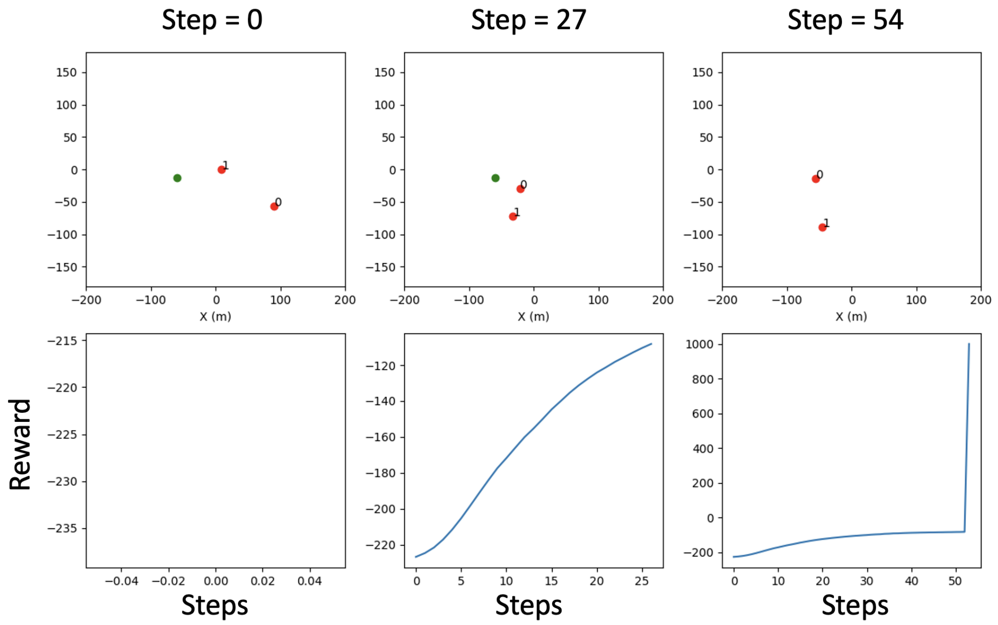


Figure 5: Example of rollout from high-performing policy generated with SAC. Red dots are drones, green dot is target.

4 Implications

For the simplified 2D problem with two agents and a single target, the reward function defined in Equation 5 appears to be valid based on the results above. However, generalization to n agents and m targets may require more fine tuning of this reward function. Consider the case in which two agents approach the same target and one drones tags it. An immediate step penalty is applied to both drones which is not compensated by the second term in Equation 5. Additionally, the reward function provides no motivation for the swarm to spread out. To address both of these problems, and addition term can be added to the reward function:

$$r = -\sum_{i=1}^{n_{agents}} \min_j \big(|a_i - t_j|\big) + d_{max}\big(n_{mapped} - n_{crashed}\big) + \sum_{i=1}^{n_{agents}} \min_j \big(|a_i + a_j|\big) \tag{6}$$

where the additional term promotes repulsion term between each of the drones.

5 Conclusion and future work

We developed and tested a physics-based drone simulator capable of deploying a UAV swarm to capture targets while avoiding obstacles. Our experiments suggest that deep reinforcement learning algorithms, such as DDPG and SAC, can achieve reasonable results and demonstrate the feasibility of using RL for controlling a drone swarm. However, there are still areas for improvement and further research.

One direction for future work is to investigate the use of advanced offline RL algorithms, such as CQL to see if they can improve upon the performance of DDPG and SAC. Training and testing offline may enable us to fine-tune some of the more challenging aspects of our task, such as the reward function and the neural network architecture of the agent. In addition, DDPG and SAC have many hyperparameters making them difficult to tune, and therefore we could test simply policy gradient algorithms such as Proximal Policy Optimization (PPO).

6 Contributions

- Gym environment + genetic algorithm benchmark - Brian Howell
- Implementing + training SAC method - Maya Horii
- Implementing + training DDPG method - Reece Huff
- Report - Equal contribution from all

7 References

[1] TI3927120 Zohdi. “The Game of Drones: rapid agent-based machine-learning models for multi-UAV path planning”. In: *Computational Mechanics* 65.1 (2020), pp. 217–228.

[2] Pranjal Tandon. “pytorch-soft-actor-critic”. In: Github: https://github.com/pranz24/pytorch-soft-actor-critic (2021).