Learning Drone Light Show Formations using Mutil-Agent RL

Shweta Agrawal
Stanford University
ashweta@stanford.edu

Abstract

This work considers the problem of learning flight paths for drones in a Drone LED light show, given a set of positions for the drones to start from and target end positions to reach. With a simplifying set of assumptions of a 2D grid, deterministic state transitions, and a relatively smaller number of drone agents, we evaluate various approaches of reinforcement learning to solve this problem. Starting with a centralized learning approach, we implement a custom grid environment with a central MDP model for the actions and observations of all N agents, and learn a simple RL policy using Q-learning. Then we model the problem as a Markov Game, and evaluate three different approaches of Multi-Agent Deep Reinforcement Learning, including, PPO [1], DQN [2], and DDPG [3]. For this, we use PettingZoo's parallel multi-agent MPE environment, tweaking the reward function and termination conditions to suit the application. We show that post some reward structure tuning, Proximal Policy Optimization (PPO) works well for this application, and outperforms other methods evaluated.

1 Introduction

Drone light shows have become fairly popular in recent years, making their appearance in various music festivals and holiday avenues. They are a safer, environment friendly, and fascinating alternative to fireworks. These drone shows are performed by illuminated, synchronized, and choreographed groups of LED drones that arrange themselves into various aerial formations. Today the flight paths for these drones are choreographed using flight planning with a 3D animation software like Blender (https://www.blender.org), and it can take weeks or even months to design this choreography.

This work tries to learn a Multi-Agent Reinforcement Learning model to enable the drones to automatically and cooperatively navigate the path from any given set of starting positions to any given set of landmarks as quickly as possible, avoiding collisions. This is done using a reward function that rewards negatively based on how far drones are from the desired configuration, penalizes heavily for collisions with each other, and optimizes for the overall time it takes to get to the desired final state using a discount factor on the finishing positive reward. [1]

To limit the scope of the project, we assume a 2D grid, deterministic state transitions and a relatively smaller set of drones. To serve a real world application, this work can to be extended to a 3D grid, will need to use a simulation model for drones with realistic transition probabilities, and needs to be scaled to a larger number of drones, as discussed in Future Work.

2 Related work

In the last few years, there has been a lot of exciting developments in multi-agent reinforcement learning, and previous work has been done to extend a number of single agent reinforcement learning algorithms to multi-agent learning.

[4] considers the problem of learning cooperative policies in partially observable domains and extends three classes of single-agent deep reinforcement learning algorithms based on policy gradient, temporal-difference error, and actor-critic methods to cooperative multi-agent systems. We borrow the idea from this paper that if the agents are homogeneous, their policy may be trained more efficiently using parameter sharing. This allows us to train a single policy using experiences of all agent drones. We too evaluate the above three classes of deep reinforcement learning algorithms for the cooperative drone navigation task. While we evaluate DQN [2] and DDPG [3] as in the paper, we use PPO (Proximal Policy Optimization) [1] instead of TRPO (Trust Polity Region Optimization) [5]. PPO algorithms have some of the same benefits of TRPO, but they are much simpler to implement, more general, have better sample complexity (empirically), and have been shown to outperform other policy gradient methods.

We also looked at MADDPG [6] that extends DDPG into multi-agent policy gradient algorithm where decentralized

[1] code: https://github.com/ashweta1/dronerl

agents learn a centralized critic based on the observations and actions of all agents. Instead of learning a single policy for all agents, MADDPG maintains an approximation of other agents' policies, to eliminate the non-stationary nature of the environment in multi-agent domains. Learning separate policies for each agent allows MADDPG to also be used with heterogeneous agents. While MADDPG is an interesting approach, it is complicated for homogenous agents environment, especially because of the use of a replay buffer. We skip this approach for this work, with potential to evaluate in future.

[6] however also provides the original implementation of OpenAI Gym's Multi Particle Environments (MPE), a set of communication oriented environments where particle agents can (sometimes) move, communicate, see each other, push each other around, and interact with fixed landmarks. In our work we use these environments through PettingZoo. PettingZoo [7] is a library that supports diverse set of Multi-Agent environments. PettingZoo's API, while inheriting many features of Gym, is unique amongst MARL APIs in that it's based around the novel AEC games model, and is also now home to MPE (https://pettingzoo.farama.org/environments/mpe/). We use MPE's $simple\_spread$ cooperative navigation environment in this work, making adjustments to its reward structure, as appropriate for the application.

3 Environment

3.1 MDP

While not scalable to any reasonable number of drones, as a proof of concept and for better understanding, we first implement our own environment for this problem as a centralized Markov Decision Process (MDP), with following properties:

- State/Observation: Joint observation provides the location of all N agents (and N immovable targets).
- Action: Joint action provides one of 5 discrete values for all N agents: right/left/up/down/do-nothing.
- Termination:
  - when every agent has reached one, and only one, target (success), or
  - max number of time-steps have been taken (failure),

  whichever is earlier.
- Reward:
  - High positive reward when every agent has reached one, and only one, target.
  - High negative reward when two agents collide.
  - A small negative reward on every state based on total minimum distance of agents from targets.
- Discount factor < 1 to incentivize agents to reach the final state quickly.
- No transition probabilities: deterministic state transitions on action.

We simulate this MDP by implementing a new 2D grid environment DroneGridEnv extending GridWorldEnv from OpenAI Gym library [8]. The implemntation extends the environment to support multiple agents and multiple targets instead of 1, implementing the required step, reward and done conditions.

We note that the major limitation of this MDP model is scale. The joint observation and action space explodes exponentially by number of agents. For a $5 * 5$ grid and 3 drones alone, observation space is $(25)^3 = 15625$ dimensional, and action space is $5^3 = 75$ dimensional. For even a slightly bigger grid like $20 * 20$, the observation space dimension is 64M!

3.2 Markov Game

MDP model in the previous section provides us a base formulation that can be extended to a partially observable multi-agent Markov Game [9]. Instead of one joint observation and one joint action, a Markov game (or Stochastic game) has N agents, a set of observations $O_1, O_2, ...O_N$ for each agent, and a set of actions $A_1, A_2, ...A_N$ to be taken at each step. The agents use a stochastic policy that maps a given observation and action O_i, A_i to a probability $[0, 1]$. This policy is learnt using the methods described in the next section.

We use the OpenAI Gym's MPE environment $simple\_spread.py$, through PettingZoo library for representing this Markov Game. $simple\_spread$ is a cooperative navigation task multi-agent environment, that is very well suited for our purpose. We make following tweaks in the environment to represent the application better, and encourage the agents to achieve the final state faster instead of just trying to minimize the distances.

- Add high positive reward for final state.
- Experiment with higher penalty for collisions.
- Add termination condition for when all targets or landmarks have been reached by one agent each.

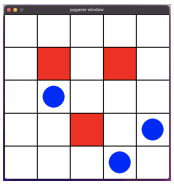

Figure 1: Custom DroneGridEnv

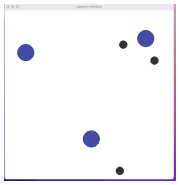

Figure 2: PettingZoo MPE environment

4 Methods

4.1 Single-Agent Q-Learning

In Value Iteration[10] methods, the *quality* $Q(s, a)$ of a state-action pair is defined as the total expected discounted

reward attained by state s on taking action a and following the optimal policy from there on.

$$Q(s,a) = R(s,a) + \gamma \Sigma_{s' \in S} T(s,a,s') V(s') \quad (1)$$

where *value* $V(s)$ of a state s is the total expected discounted award attained by optimal policy starting from state s.

$$V(s) = \max_{a' \in A} Q(s,a') \quad (2)$$

Traditionally, value iteration involves applying updates in these equations synchronously using transition function T. Q-learning [11] performs these value updates asynchronously by choosing the next action using current policy (starting with random) and performing the update whenever an agent receives a reward r when making a transition from s to s' by taking action a, the probability of which turns out to be $T(s,a,s')$ anyways.

We implement Q-learning to learn the centralized MDP model for the DroneGridEnv. Note that we do not have non-deterministic transition probabilities so it is equivalent to value iteration learning in our case.

4.2 Multi-Agent Deep Reinforcement Learning

4.2.1 Parameter Sharing

In our multi-agent environment, all the agents are homogenous. This allows all the agents to share the parameters of a policy and the policy can be learnt using observations and actions of all agents. Note that this still allows different behaviour for different agents since they get different observations. We primarily evaluate this shared learning (but decentralized control) policy using Proximal Policy Optimization (PPO). For comparison, we also try two more deep RL approaches in multi-agent environment: DQN and DDPG, in the same parameter sharing setting.

4.2.2 Proximal Policy Optimization (PPO)

This work focuses on Proximal Policy Optimization (PPO) as the primary method for learning a Deep Multi-agent RL policy for the cooperative drone navigation task.

PPO, first introduced in [1] by OpenAI, is a policy gradient method. Policy gradient methods are a type of RL techniques that rely on learning a parameterized stochastic policy π_θ optimizing θ for expected return (long-term cumulative reward) using gradient ascent. The most common gradient estimator is of the form:

$$\hat{g} = E_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right] \quad (3)$$

where $\hat{A}_t$ is an estimator of the advantage function (function mapping gain in expected reward) at timestep t.

Trust Region Policy Optimization, or TRPO [5], is a policy gradient method that avoids parameter updates that change the policy too much, by maximizing a "surrogate" objective subject to a KL divergence constraint on the size of the policy update.

$$J(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] \quad (4)$$

subject to

$$\hat{E}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \leq \delta \quad (5)$$

PPO modifies this objective by "clipping" the changes to the policy.

$$J^{clip}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \quad (6)$$

PPO methods have some of the same benefits as TRPO, however, they are simpler to implement, more general and are known to outperform other policy gradient methods.

4.2.3 Deep Q-Network (DQN) and Deterministic Policy Gradient (DDPG)

We evaluate these two methods only to provide a comparison against PPO, and without going into a lot of detail, describe them briefly.

DQN [2] or Deep Q-network uses a neural network to approximate the *quality* function $Q(s,a)$ of state-action pair, relying on an experience replay dataset that stores agents' experiences, in order to avoid correlations between observations.

Deep Deterministic Policy Gradient (DDPG) [3] combines the actor-critic and DQN approaches to learn policies in continuous action spaces by maintaining a parameterized actor function that deterministically maps states to actions, while learning a critic $Q(s,a)$ that estimates the value of state-action pairs.

5 Experiments

In this work, we run several experiments to train a policy for the environments described above. For each of these experiments, we study the convergence of expected reward across training episodes, and then evaluate the policy on the respective environment through multiple randomized trials. We then report the goodness of the learnt policy in the form of

- Success rate: % of trials where all landmarks were reached by the agents.
- Avg steps to success: average number of time-steps needed by agents to achieve success.
- Collision rate: % of trials that resulted in one or more agents colliding with each other.

Each of these training episodes and test trials is limited to a max of 25 timesteps, at which point the environment terminates and the episode/trial is declared a failure.

5.1 Q-Learning: for single-agent centralized MDP

We implement and learn a very simple RL policy for 3 agents and 3 given fixed targets using Q-learning. We use a discount factor γ of 0.95 and learning rate of 0.1. The episode mean reward seems to converge in $\approx 60K$ training episodes Figure 3, and the policy performs reasonably well at 81% success rate, as shown in Table 1. A demo of the learnt model can be accessed here: `https://youtu.be/aYm5vWn4qRk`

At first glance, the performance of the policy looks decent. However, this model and policy has some major limitations:

- The policy is learnt given fixed target landmarks. A new policy will need be learnt for a new set of landmarks. This approach does not scale to randomized landmarks, unlike the more general multi-agent deep learning in the next section.
- The grid size used if very small $5 * 5$. With joint observation and action model, the observation and action space is exponential w.r.t. grid/state size, hence scaling the grid is difficult.

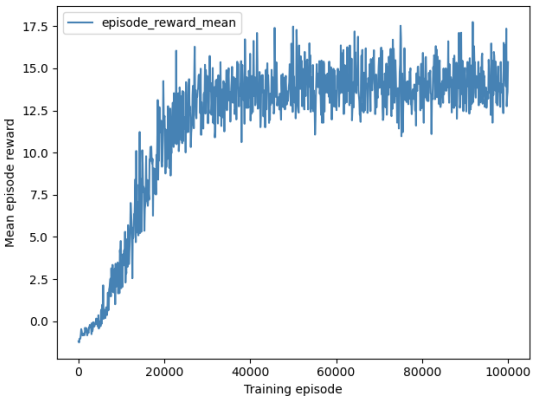

Figure 3: Q-learning for a fixed target pattern in a 3 drone environment.

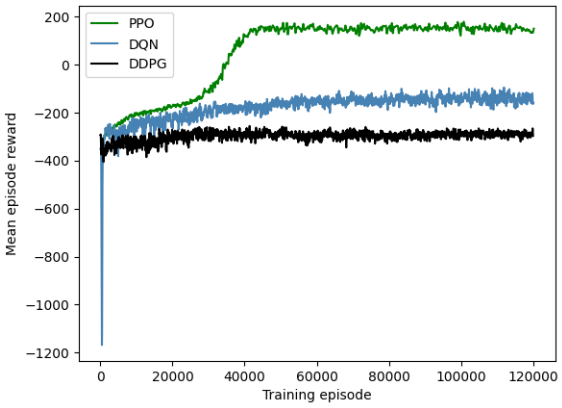

Figure 4: Training reward convergence for PPO vs DQN and DDPG

5.2 PPO: Multi-Agent Deep RL for partially observable Markov Game, vs DQN, DDPG

The PettingZoo's MPE parallel environment [12] has N agents, where each agent gets their own observations. Each agent's observation contains a vector composed of the agent's position and velocity, other agents' relative positions and velocities, landmarks' relative positions, landmarks' and agents' types, and communications received from other agents. Note that the velocities are only used in case continuous_actions are chosen. We use discrete actions for all methods, except DDPG.

We train a PPO policy for the environment using ray[rllib] on tensorflow [13]. For the neural network, we use default configuration, and a discount factor $\gamma = 0.9$, running the training on a cloud machine with 6 CPUs. PPO is able to learn a reasonable generic policy that is able to navigate the agents to any given target positions with a success rate of 90+%! A demo of the trials with the MPE environment, using the learnt PPO policy can be accessed here: `https://youtu.be/rpW3ANoM2J8`).

On tuning the value of collision penalty, we find that there is a trade-off between success rate (what percent of trials all landmarks are landed by the agents), and collision rates. A collision penalty of -10.0 seems to perform the the best as shown in Table 2.

We also try to learn DQN and DDPG policies in the same parameter sharing setting. However, those two policies fail to converge to a positive reward value, as seen in Figure 4.

6 Conclusion and Future work

Proximal Policy Optimization (PPO) seemed to perform well with 90+% success rate, which is encouraging. Given more time, further tuning of learning rate and discounting factor would be interesting to explore to improve performance.

We failed to get DQN and DDPG to converge to a good reward value in training, and these policies showed very results during testing. A big factor might be the inability of these methods to learn a good controller in the decentralized setting given the non-stationary nature of the problem, as the authors in [4] also note. However, more tuning of training parameters and the underlying CNN configuration could also likely help get better performance from these methods.

We also made a number of simplifying assumptions for this cooperative navigation task, like using a 2D grid, only 3 drone agents and deterministic state action transitions. To be able to use this model to guide the flight paths in a real world Drone Light Show, we will need to

- Extend the environment to a 3D grid.
- Scale the model and learning to 3D environment and much larger (20-2000) drone agents. This might need

Algo	Collision penalty	Episodes to converge	Success rate	Avg steps to success	Collision rate
Q-Learning	-10.0	$\approx 60k$	81%	6.19	6%

Table 1: Performance of a Q-learning policy for a fixed target pattern in a 3 drone environment over 1000 trials.

Algo	Collision penalty	Episodes to converge	Avg Reward	Success rate	Avg steps to success	Collision rate
PPO	-1.0	100k	180.0	90.2%	12.50	26.9%
PPO	-10.0	100k	151.6	91.6%	12.29	18.2%
PPO	-50.0	100k	86.9	88.4%	14.11	11.3%

Table 2: Success and Collision rates on 1000 trials for generic target PPO policies trained for a 3-drone environment, with varying reward penalty on collisions

more resources and also investigation into PPO scaling techniques. For example, using Decentralized Distributed Proximal Policy Optimization (DD-PPO) [14])

- Use a Drone simulation model to get (or simulate) transition probabilities.

Overall, Drone Light Shows is an exciting application of Multi-agent Reinforcement Learning. A drone cooperative navigation model, once extended to a real world scale, will not only save a lot of time to choreograph these shows, but also allow for drones to react better to environmental factors, avoid collisions with each other, and even enable creating new formations on-the-fly during the show.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[4] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In Gita Sukthankar and Juan A. Rodriguez-Aguilar, editors, *Autonomous Agents and Multiagent Systems*, pages 66–83, Cham, 2017. Springer International Publishing.

[5] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015.

[6] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017.

[7] J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning, 2020.

[8] OpenAI Gym. Make your own EnvGrid World Env. `https://www.gymlibrary.dev/content/environment_creation/`. [Online; accessed 08-Dec-2022].

[9] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*, 1994.

[10] Dimitri P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., USA, 1987.

[11] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[12] PettingZoo. Multiparticle Env. `https://pettingzoo.farama.org/environments/mpe/`. [Online; accessed 08-Dec-2022].

[13] rllib. Rllib PPO. `https://docs.ray.io/en/master/rllib/rllib-algorithms.html#ppo`.

[14] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames, 2019.