

Autonomous Drone Path Finder

Guide: Himanshu Sarma

Team (HS04):

Vanam Hemanth (S20170010172)

Nandala Neelakanta Sriram (S20170010102)

Puppala Hrithik (S20170010115)

Contents

- Introduction
- Motivation
- Problem Statement
- Simulation tools
- Literature Review
- Work done
- Future plan
- References

Introduction

- Drones technology is booming in the coming years.
- Drones are used in many fields such as in agriculture, construction, mining and many more.



Image link:

<https://www.piston.my/2019/09/21/audi-airail-quattro-brings-autonomous-driving-to-off-road-travels/>

Motivation

- Since ages, many people are missing in the forests and dense areas.
- Many people go missing every year in America's 419 national parks and monuments.
- Yosemite National Park tops the ISB's cold case list with 10 unsolved cases, followed by Great Smoky Mountains National Park and Grand Canyon with five each.
- We want to design a multi-drone model to distribute them over an area to rescue people by creating a virtual map which the experts can use to plan the possible locations of the missing person.

Problem Statement

- The main aim of our project is to train a group of autonomous UAV or drones to explore a given area and create a virtual map of the area based on the terrain information in the simulation.

Challenges:

- ❑ Train a drone model to fly autonomously.
- ❑ Train a group of drones to explore the area efficiently by communicating with each other drone.
- ❑ Create virtual terrain map based on the data obtained from the drones exploration.

Simulation Tools

- We searched for the possible simulation tools so that we can implement the training in an appropriate environment.
- The Reinforced learning platform simulators we researched are as follows:
 - ◆ Arcade Learning Environment
 - ◆ VizDoom
 - ◆ MuJoCo
 - ◆ DMLab-30
 - ◆ Hard Eight
 - ◆ AI2Thor
 - ◆ OpenAI Retro
 - ◆ DMControl
 - ◆ ProcGen
 - ◆ Project Malmö



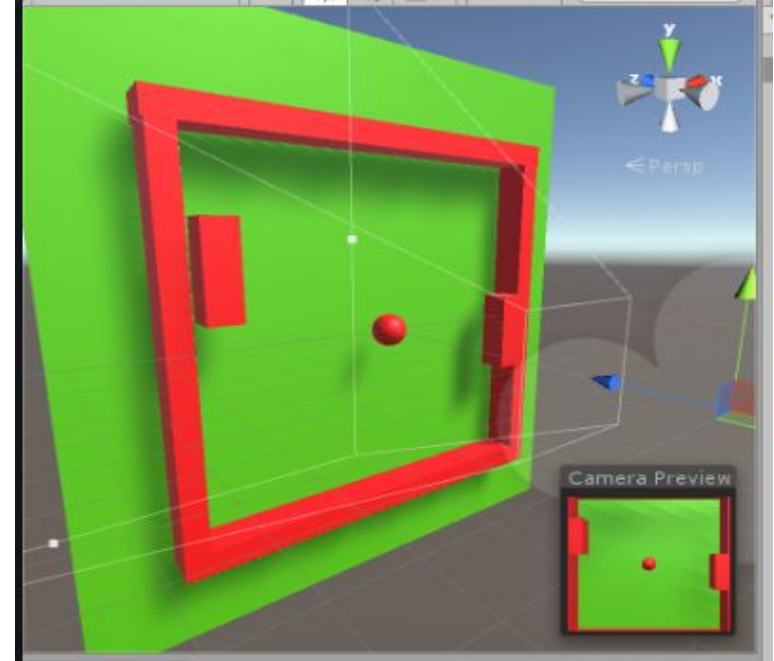
ALE learning the breakout game

Simulation tools

- Most of the simulation platforms lack the 3d realistic physics implementation or realistic visual rendering technology.
- Even if they do have 3d realistic physics , the ability to most of the simulation platforms are made for problem specific. So there is no flexibility to create our own environments.

Game engines as simulators

Since the modern game engines such as cryengine, unreal engine, unity3d engine, snowdrop engine, frostbite engine have come so much in the modern technology where the rendering is almost hyper realistic and the physics implementation is also so accurate , we choose to go with an game engine for reinforced learning.



Pong game simulation in unity game engine

Why Unity?

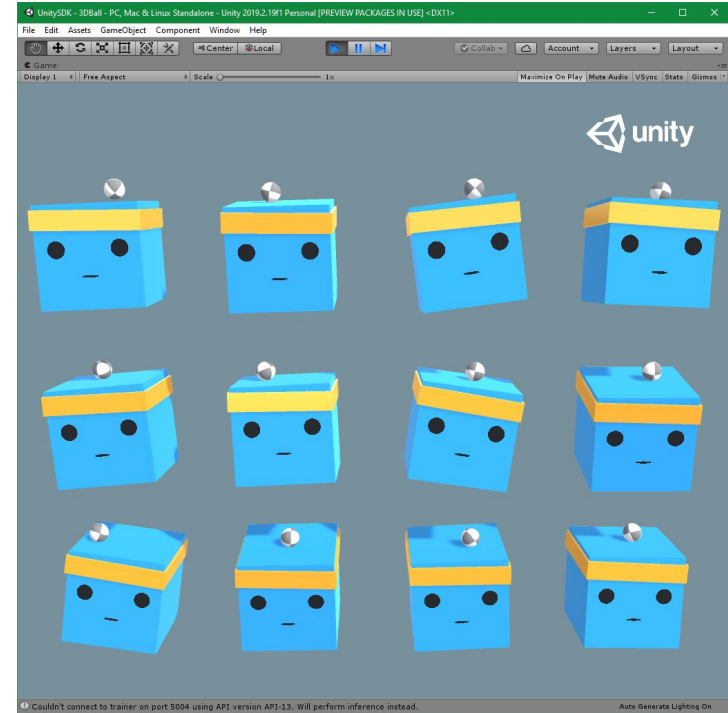
- Even though the game engines provide the physics implementation, flexibility to create our own environments, realistic visual graphics, lightings etc, not many provide the API's for connecting to external applications.
- Some game engines like cryengine has to be purchased if we want to work with that software.
- Unity however is free and provide well maintained documentation for the scripting APIs.

Unity

- **Sensory Complexity:** The Unity engine enables high-fidelity graphical rendering. It supports pre-baked as well as real-time lighting and the ability to define custom shaders, either programmatically or via a visual scripting language.
- **Physical Complexity:** Unity environments can be simulated with either the Nvidia PhysX or Havok Physics engines. There are plugins from unity asset store for Bullet and MuJoCo physics engines as alternatives to PhysX.
- **Task Logic Complexity:** The Unity Engine provides a rich and flexible scripting system via C# language.
- **Flexible Control:** It is possible to control most aspects of the simulation programmatically. For example, GameObjects can be conditionally created and destroyed in real-time.
- **APIs:** It provides the api for communicating with external applications which will be helpful for connecting to the Python API for reinforced learning.

Unity ML-Agents Toolkit

- We can use the python package for the unity to connect to python shell and implement the machine learning on the agents.



Architecture MLAgents

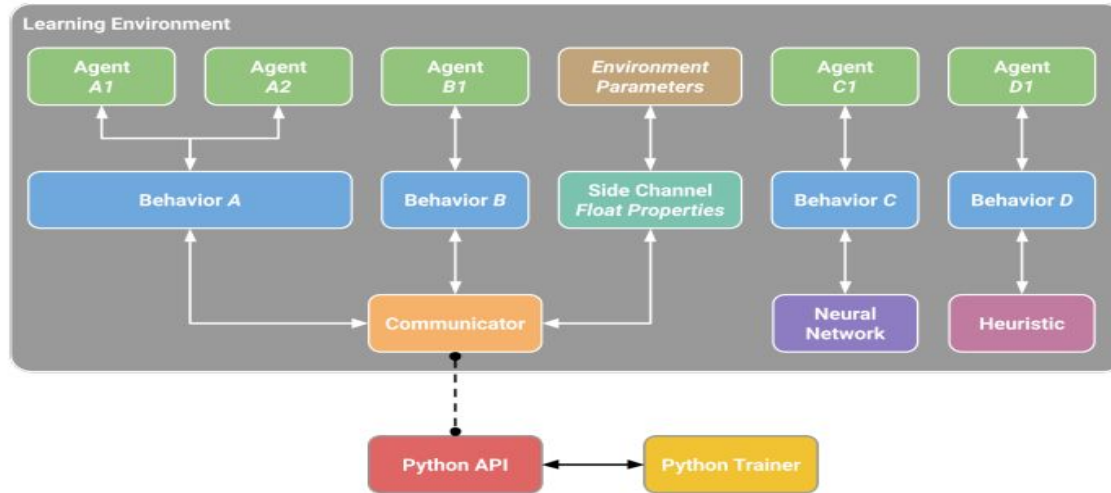


Figure 2: A Learning Environment (as of version 1.0) created using the Unity Editor contains Agents and an Academy. The Agents are responsible for collecting observations and executing actions. The Academy is responsible for global coordination of the environment simulation.

Literature Review

Proximal Policy Optimization:

- In reinforced learning the network which converts the input frames to output frames are called policy network.
- Doesn't store state transition data in the replay buffer like popular Q learning algorithm but learns directly from the events the agent encounters in the environment.
- This ability of PPO algorithms help remove the computational complexity of the Q table.
- Policy Gradient Loss:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$$

where:

- π_{θ} Is probabilities from the output of the policy network.
- \hat{A}_t is the estimate of the relative value of the selected or simply called advantage value

Advantage function

$$\hat{A}_t = \text{Discounted sum of rewards} - \text{baseline estimate}$$

- Discounted sum of rewards or the return is the weighted sum of all the rewards the agent got in each time step in the episode.
- Baseline estimate or the value function is the estimate of the discounted sum of rewards in this current episode calculated at the start of episode.
- The Advantage value tells how much better was the action that the agent took based on the expectation of what would normally happen in the state that the agent was in.
- Advantage value greater than zero implies that the action is better than expected.
- Advantage value is less than zero implies that the action is worse than expected.


Proximal Policy Optimization

$$r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)},$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right]$$



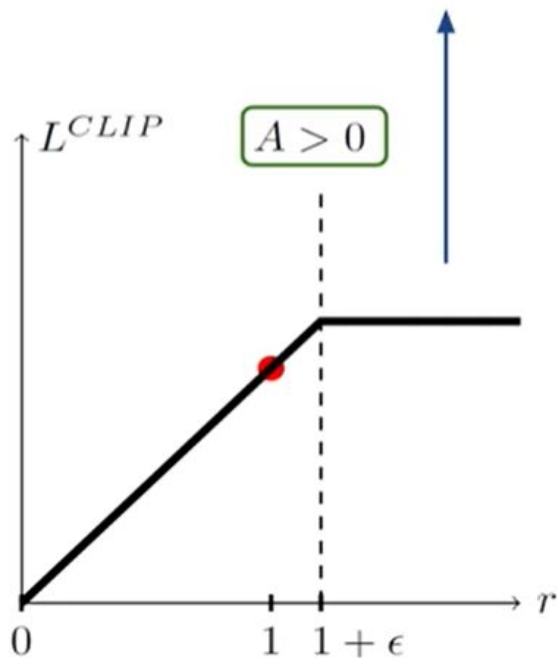
Normal policy gradient objective



Clipped version of normal gradients objective where epsilon is a small number like 0.2.

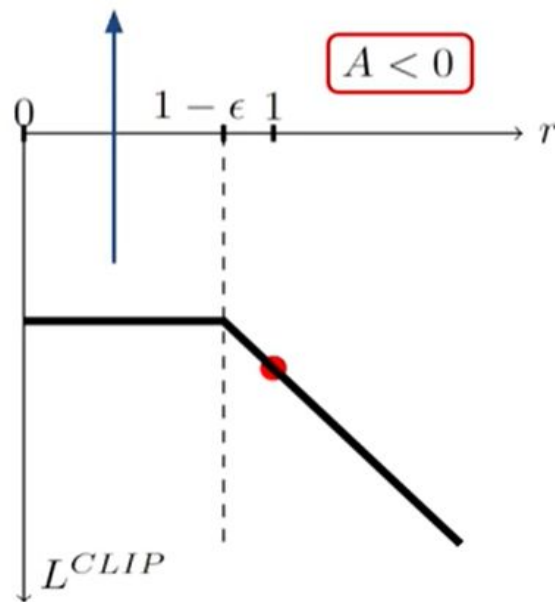
If the action was **good**...

... and it became a lot more probable after the last gradient step, don't keep updating too much or else it might get worse!



If the action was **bad**...

... and it just became a lot less probable, don't keep reducing it's likelihood too much for now!



Algorithm of PPO

Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1,2,... do
  for actor=1,2,...,N do
    Run policy  $\pi_{\theta_{\text{old}}}$  in environment for  $T$  timesteps
    Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
  end for
  Optimize surrogate  $L$  wrt  $\theta$ , with  $K$  epochs and minibatch size  $M \leq NT$ 
   $\theta_{\text{old}} \leftarrow \theta$ 
end for
```

Final training objective in PPO:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t [L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)], \quad (9)$$

where c_1, c_2 are coefficients, and S denotes an entropy bonus, and L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{\text{targ}})^2$.

Results

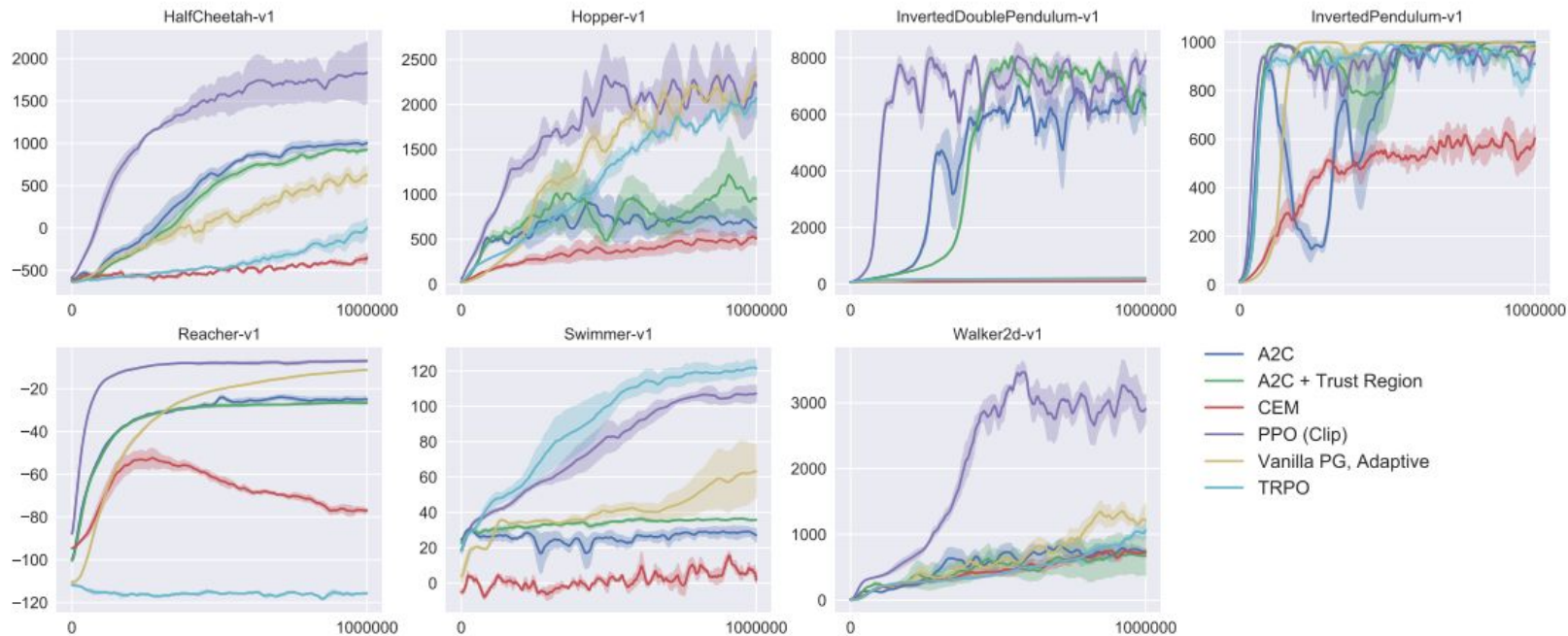


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

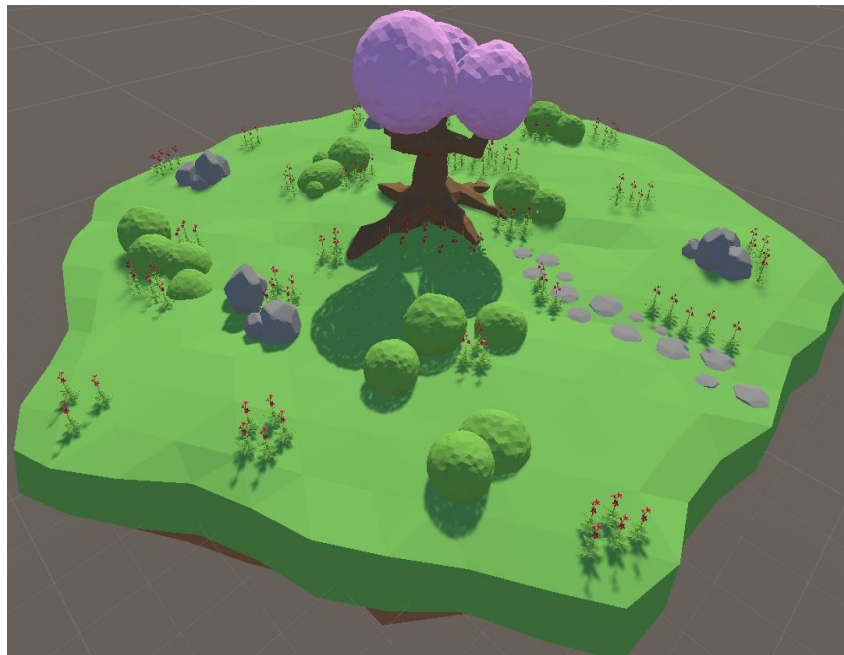
Work done

- The time to train a complex drone model to fly takes a lot of time and the training can't be implemented in online google colab like advanced technologies as that would require unity instance to run along with the python trainer.
- We first wanted to implement the PPO algorithm with a more simple environment.
- The simple environment:
 - ◆ The aim of this environment is to make a hummingbird model which collects the nectar from the nearest flower and once nectar of that flower is exhausted move to the next nearest flower.



On Episode begin

- Whenever the episode begins the position of the flower plants are randomly selected.
- The position of the agent (hummingbird) is randomly selected (a safe position not colliding with any flowers or rocks or other game objects) or simply spawned right in front of a flower bud with a probability of 0.5 for each.



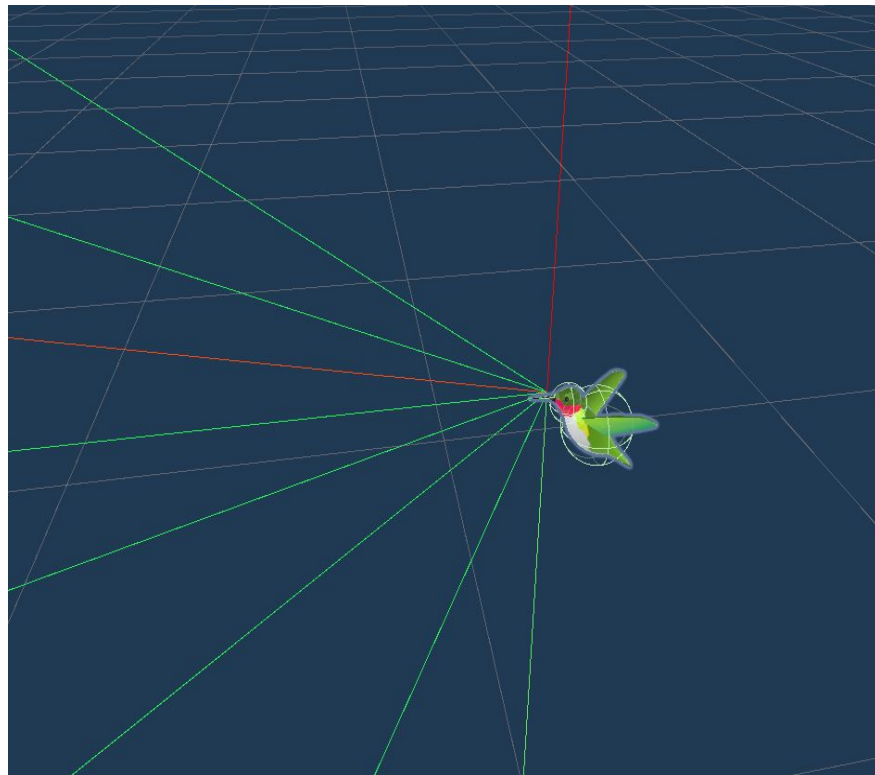
Work done

Inputs to the neural network:

- The agents normalized local rotation (four float values).
- The direction to the nearest flower normalized (three float values).
- Agents beak tip direction with respect to flower to know whether the agent is facing the flower (one float value).
- Whether the beak tip is in front of the flower or behind the flower (one float value).
- Distance of the flower from the agent (hummingbird) divided by total environment diameter (one float value).
- Additionally agent also has ray perceptions which act like lidar.
 - 7 forward ray perceptions with an angle of 60 with each other.
 - One ray perception facing directly upwards the agent.
 - One ray perception facing directly downwards the agent.

Rewards

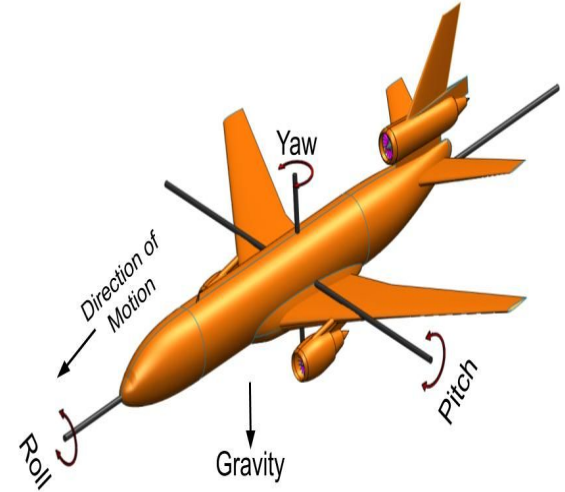
- A small 0.01 + bonus reward for the getting the nectar from the flower.
 - ◆ Bonus is the normalized value of how much of the beak tip forward direction is directly towards the flower nectar chamber multiplied by 0.02.
- A negative 0.5 reward whenever the agent crosses the boundary of the environment.



Actions

The action space (output of the neural network) consists of a float vector of length 5.

- Index 0: moves agents x coordinate (+1 = right, -1 = left)
- Index 1: moves agents y coordinate (+1 = up, -1 = down)
- Index 2: move agents z coordinate (+1 = forward, -1 = backward)
- Index 3: moves agents pitch angle (+1 = pitch up, -1 = pitch down)
- Index 4: moves agents yaw angle (+1 = turn right, -1 = turn left)



Hyperparameters for PPO

Batch Size	2048
Layers	2
Beta (learning rate)	0.005
Max steps per episode	5000
Hidden units	256
Epsilon	0.2
Lambda (reward weight)	0.95

Work done



- We have found some drone models on which we will implement the training in the future.
- We have also searched for the different types of drones and their various properties which will be helpful in creating the realistic model.

Environment



- We have found some nice urban + forest environment from the unity asset store which can be used to implement the training for drones.

Future Plan

→ By the next evaluation

- Create a drone model which flies autonomously also use different algorithms to train the model like Soft Actor-Critic to see which yields best results.

→ By the final evaluation

- Create a model which deploys a group of drones and get the terrain information from the drones and generates the virtual map of the area.

References

- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov, "Proximal Policy Optimization Algorithms", arXiv:1707.06347v2 [cs.LG] 28 Aug 2017.
- Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar and Danny Lange, "Unity: A General Platform for Intelligent Agents " arXiv:1809.02627v2 [cs.LG] 6 May 2020.
- Huy Xuan Pham, Hung Manh La, David Feil-Seifer and Ara Nefian, "Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage", arXiv:1803.07250v2 [cs.RO] 16 Sep 2018.
- A. Anwar and A. Raychowdhury, "Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning," in IEEE Access, vol. 8, pp. 26549-26560, 2020, doi: 10.1109/ACCESS.2020.2971172.
- Yuchao Hu and Wei Meng , "ROSUnitySim : Development and experimentation of real-time simulator for multi-unmanned aerial vehicle local planning", in Industrial Electronics Society, IECON 2015 - 41st Annual Conference of the IEEE, DOI: 10.1109/IECON.2015.7392488.

Thank you !!
