**README**

This project was accomplished thanks in large part to the tutorial provided below, which was adapted to a newer version of ML-agents for performing quadcopter stabilization in Unity:

[Training a Virtual Drone Using Machine Learning](https://www.youtube.com/watch?v=6LxjUvXOo74)

Additionally, the ml-agents unity asset package allowed the reinforcement learning algorithms to run in Unity, found at the following link:

*ML-agents extension package installation instructions:*

[*https://github.com/Unity-Technologies/ml-agents/blob/release\_8/docs/Installation.md#install-the-comunityml-agents-unity-package*](https://github.com/Unity-Technologies/ml-agents/blob/release_8/docs/Installation.md#install-the-comunityml-agents-unity-package)

*ML-agents version required: 1.0.8*

The ml-agents package should be downloaded and placed in a directory accessible via a terminal shell to start training. More details are provided in their installation instructions in the above link.

To prepare this package, place the DroneHoverEnv.yaml file within the ml-agents/config/ppo directory.

*Running the simulator:*

1. Start Unity and load the DroneControl package.
2. Install ML-agents version 1.0.8 in the Unity package manager.
3. In a terminal shell, change directories into the ml-agents package and activate the virtual python environment (3.7). Prepare a virtual environment if you have not already done so.
4. To start training from the beginning, within the directory containing ml-agents:

$ mlagents-learn ml-agents/config/ppo/DroneHoverEnv.yaml --run-id=drone\_test

1. To monitor the training process, use Tensorboard to visualize the cumulative reward and value estimates:

$ tensorboard --logdir summaries

*Environment Setup:*

During each episode of training, if the agent (drone) achieves its goal (reaching a destination cube), falls off the map / crashes, or reaches the time limit, the episode terminates and the goal is relocated to a new random position. The scene is then randomized to promote learning in a variety of conditions.

A reference to the Rigidbody component of the agent is needed to reset the agent’s velocity and apply actions (forces) to its actuators. There are therefore 4 continuous actions, a force applied to each thruster of the quadcopter. The observations made by the agent’s sensors are sent to the ‘Brain’, analogous to a trained policy, where decisions (actions) are made based on the observations. This sensor data is used as input to a neural network as a feature vector. In this implementation we follow an approach similar to the one taken in [6], where the state observations include the agent’s displacement relative to the target cube, the agent’s velocity, and its orientation.

The agent model needs the following essential components (scripts):

1. <Custom> Agent:
   1. Target: Target (Transform)
   2. Force Multiplier: 10
2. Behavior Parameters:
   1. Behavior Name: DroneAgent
   2. Vector Observation:
      1. Space Size: 10
      2. Stacked Vectors: in scenarios where a vector of observations need to be remembered or compared over time, stacking the vectors provides the agent with memory without using an RNN.
   3. Actions:
      1. Continuous Actions: 4
      2. Discrete Branches: 0
3. Decision Requester:
   1. Decision Period: 1