Samuel Weiss
Jacob Brendemuehl
Dr. Sujan Kumar Roy
DATA4991
10/01/2025

DATA 4991 Capstone Project Report A

The current Unity environment can be found at the GitHub repository in the folder UnityEnv (https://github.com/zecerman/UAV-Collision-Detection). This folder can be opened directly in Unity and is recognized as a complete project by the editor. Please do open the github link and read the programs as they are referenced in this report, they constitute much of the proof of our work that you asked for in the project assignment.

§1. Version Control

We encountered many version mismatch issues, which we have managed to resolve at this time. We settled on using python version 3.1 and Unity version 6000.2.2f1 with the following

- mlagents==1.1.0
- torch==2.2.2
- protobuf==3.20.3
- cattrs==1.6.0
- tensorboard
- numpy

Our team does not expect to encounter any further issues of this kind.

§2. Physics Environment

It is difficult to reference any actual file given how Unity programs are stored, if you want to follow along with this section please open “UnityEnv” in your Unity editor to follow along.

The drone simulated in Unity will be based on the Amazon MK 30 drone and its parameters (rigidbody, forces, constraints, control, etc.) The drone is currently a temporary model and doesn’t accurately represent the MK 30 as of right now, but it is a work in progress to get closer to a real drone.

- Weight
    - The average weight of the Amazon MK 30 drone is 35-38 kg (depending on payload)
    - In Unity, the rigid body mass is determined by the physics engine, which automatically applies a downward force equal to mass * gravity.
    - This is a baseline force that the propellers will counteract just to hover.
    - The center of gravity on the drone is manually adjusted down for the time being which increases the drones stability and is more faithful to the final model.
- Gravity
    - Gravity is handled by the Unity environment automatically and is a function of the UAV's weight variable.
    - This will be implemented to accurately represent how much propeller thrust is needed to counteract gravity and weight.
- Propeller Thrust
    - Our drone currently has 4 propellers to conform to the transfer learning data, while the MK 30 has 6. We will have to update the drone model to better match the model.
    - Yaw, pitch, and roll control come from the differences in rotor thrust.
        - Ex. Speeding up the left rotor will cause the drone to roll.
    - Instead of modeling blades, we will apply forces in the direction each propeller pushes.
- Balance of Forces
    - In hover: upward thrust equals downward weight
    - In motion: if thrust is greater than weight, the drone accelerates upward; if thrust is less than weight, it sinks.

§3. Reinforcement Learning Loop

This recording captures the current state of the reinforcement learning loop:

https://drive.google.com/file/d/1TjStqzn7T3Q8bzM-i8uFOR1eqEt1gqLn/view?usp=sharing
To summarize, the drone's AI is driven by two main scripts and the training loop is handled by a third script. Beginning with the loop, the file can be found in "UnityEnv/config/drone_goal.yaml". This config file specifies the neural network structure and hyperparameters for the training library we are using "ml-agents". Currently, the .yaml file calls for the construction of a simple two body feed forward perceptron. The results of each training episode are saved to "UnityEnv/results/drone_goal_xxx" and can be seamlessly loaded to continue training from where it left off. The NN architecture in the .yaml file is the minimum required model for Unity and ml-agents to compile and train, it is a placeholder and must be developed further.

Moving on to the drone's AI, it is controlled by DroneStabilizer.cs and FirstML.cs which can both be found in "UnityEnv/Assets/Scripts". Teaching a machine learning agent to balance a drone is itself a very difficult problem, and not a particularly enlightening one either. Our concern is with the more complicated issue of teaching a drone to avoid obstacles, which requires balance as a prerequisite. To address this issue, we are using a simple calculus-based balancing algorithm which calculates deviations in the UAV's tilt and fights to correct them by increasing the output of certain propellors. This script (renamed from DroneAI in previous sprints to DroneStabilizer for clarity) exposes three controls to the machine learning agent in the second script "FirstML". The only strings that the ML agent is allowed to pull are "tilt_x", "tily_y", and "climb_z". By allowing the nudge the pitch, roll, and vertical height of the UAV, it can effectively steer it anywhere. This script also contains basic drive code for instantiating each episode (such as spawning the goal in a random nearby location) and defining a basic reward function (steps are penalized, touching the goal is rewarded, etc…). Both scripts need to be revisited, with a bulk of our coming development time being dedicated to improving the ML agent program.