Project 2 (By two person)

You should implement unfolded Algorithms, Provide Visuals, Provide as possible graphical results

Below is a step-by-step guide on how to train a reinforcement learning agent (e.g., using **SAC**, **DDPG**, **PPO**, or **TD3**) with **Gym-PyBullet-Drones**. This guide assumes you're using **Python 3.x** and will leverage **Stable Baselines3** for the RL algorithms.

1. Install the Required Packages

1. **Create/Activate a Python Virtual Environment** *(recommended)*:

```
python -m venv venv
source venv/bin/activate  # On Linux/Mac
# or
venv\Scripts\activate.bat  # On Windows
```

2. **Install the necessary libraries**:

```
pip install –upgrade pip
pip install pybullet
pip install stable-baselines3
pip install git+https://github.com/utiasDSL/gym-pybullet-drones.git
```

This will install:

• **pybullet** (physics simulation)

• **stable-baselines3** (popular RL algorithms like SAC, TD3, PPO, DDPG)

• **gym-pybullet-drones** (the custom Gym environments for drones)

2. Choose an Environment

Gym-PyBullet-Drones provides several environments (or "aviaries"), for example:

• hover-aviary-v0: Single-drone environment where the goal is to hover at a specified position.

• takeoff-aviary-v0: Single-drone environment where the drone must take off and reach a target height.

• flythrugate-aviary-v0: Single or multi-drone environment for flying through gates (a more complex task).

• tune-aviary-v0: An environment specifically designed for classical (PID) control tuning.

For simplicity, we'll use hover-aviary-v0 in the example below.

3. Minimal Example: Training With SAC

Here is a short Python script (train_sac.py) illustrating how to train **SAC** on hover-aviary-v0:

```
import gym
import gym_pybullet_drones  # This registers the drone environments
from stable_baselines3 import SAC
from stable_baselines3.sac.policies import MlpPolicy

def main():
    # 1. Create the environment
    env = gym.make("hover-aviary-v0")

    # 2. Instantiate the RL model (SAC in this case)
    model = SAC(
        policy=MlpPolicy,
        env=env,
        verbose=1,       # Set to 1 (info) or 2 (debug)
        learning_rate=1e-3,
        buffer_size=100000,
        batch_size=256,
        gamma=0.99,
        tau=0.02
    )

    # 3. Train the agent
    timesteps = 50000
    model.learn(total_timesteps=timesteps)

    # 4. Save the model
    model.save("sac_hover_aviary")

    # Optional: close the environment if needed
    env.close()

if __name__ == "__main__":
    main()
```

Key Points

• **Policy**: MlpPolicy means a Multi-Layer Perceptron (fully-connected neural net).

• **Hyperparameters**: The above defaults often work decently, but you may want to tune learning_rate, batch_size, etc., based on performance.

• **Timesteps**: Adjust 50000 (or more) depending on your computational resources and how complex the task is.

4. Training With Other Algorithms

In **Stable Baselines3**, you can replace SAC with other continuous-control algorithms such as **DDPG**, **TD3**, or even a discrete-friendly algorithm like **PPO** (though PPO is widely used in continuous action spaces as well). The code structure is the same; only the import changes:

• **DDPG**:

• **TD3**:

• **PPO** (though typically used for discrete or continuous):

Everything else (environment creation, saving, etc.) remains basically the same.

5. Monitoring Training & Plotting

1. **Logging**:

• By default, Stable Baselines3 logs to the console.

• You can optionally pass a tensorboard_log parameter to log data for TensorBoard:

```
model = SAC("MlpPolicy", env, verbose=1, tensorboard_log="./tensorboard/")
model.learn(total_timesteps=50000)
```

• Then, run tensorboard –logdir ./tensorboard/ to visualize training curves in your browser.

2. **Plotting**:

• You can record rewards by wrapping your environment or by using callback functions.

• See [Stable Baselines3's documentation] on how to implement callbacks and track custom metrics.

6. Evaluating the Trained Model

After training, you likely want to see how the drone performs. Here's a quick script (evaluate_sac.py) to load the model and visualize the drone in PyBullet:

```
import gym
import gym_pybullet_drones
from stable_baselines3 import SAC

def evaluate_model():
    # 1. Load the environment
    env = gym.make("hover-aviary-v0", gui=True)  # gui=True to see PyBullet
UI

    # 2. Load the trained model
    model = SAC.load("sac_hover_aviary", env=env)

    # 3. Run episodes
    obs = env.reset()
    for _ in range(1000):  # 1000 steps
        # Get the action from the model
        action, _states = model.predict(obs, deterministic=True)
        obs, reward, done, info = env.step(action)

        # Optionally, you can print or log the reward
        # print("Reward:", reward)

        if done:
            obs = env.reset()

    env.close()

if __name__ == "__main__":
    evaluate_model()
```

Key Points

• gui=True: In gym-pybullet-drones, specifying gui=True will render the environment so you can visually watch the drone.

• model.predict(obs, deterministic=True): By default, SAC is stochastic, so setting deterministic=True means you see the "greedy" (exploiting) version of the policy.

7. Tips for Successful Training

1. **Simplify the Task**: If the drone is too unstable:

• Start with an easier environment like takeoff-aviary-v0 or a smaller action range.

• Use a well-shaped reward function (e.g., negative distance to a hover point, plus penalties for excessive tilt).

2. **Hyperparameter Tuning**:

• Learning rate, batch_size, buffer_size, and exploration strategies (for DDPG/TD3) can dramatically affect training.

• Start with default Stable Baselines3 hyperparameters, then adjust systematically.

3. **Check Observations & Actions**:

• Make sure you understand the observation space (state vector) and the action space (motor thrusts, velocity commands, or orientation) provided by the environment.

• If your agent never learns, you might need to modify these or re-scale the action space.

4. **Logging & Debugging**:

• Inspect reward logs frequently. If rewards don't improve or remain negative, adjust your environment or reward shaping.

5. **Speeding Up Training**:

• Increase the n_envs for parallel environments if your hardware supports it.

• Use GPU acceleration for the policy network if possible.

8. Next Steps & Extensions

• **Multiple Waypoints**: Extend the environment to require the drone to pass through multiple waypoints.

• **Obstacle Avoidance**: Add static or moving objects in the environment with collision penalties.

• **Domain Randomization**: Randomize gravity, wind, or drone mass to encourage robust policies.

• **Multi-Agent**: Gym-PyBullet-Drones supports multi-drone setups—train multi-agent RL with cooperation or competition tasks.

Final Remarks

That’s it! With these instructions, you should be able to:

1. Install and set up Gym-PyBullet-Drones.

2. Train with an RL algorithm of your choice (SAC, TD3, DDPG, PPO, etc.) using Stable Baselines3.

3. Evaluate and visualize the learned drone behavior in PyBullet.

This workflow is a great starting point for experimenting with more complex drone control tasks and reward structures. Good luck with your training, and feel free to ask if you have any more questions!

Or run and train Bipedal waslking robot in Open AI gym

Then you will get maximum 80 points