RL For Satellite Scientific Discovery

Vanessa Bellotti, Tanay Nistala

Scientific Discovery in Orbit around Enceladus

Overview

Understanding the problems

Project objective

PPO

Reward Function

Evaluation

Project objective

Develop an innovative RL model tailored for online learning within the confines of a simulated environment. The specific focus is on the detection of plumes, such as those found on Enceladus. The model's multifaceted objectives encompass not only the identification of these plumes but also the dynamic guidance of a spacecraft towards their source for comprehensive analysis. The RL model must exhibit a capacity for continuous adaptation and learning from its interactions with the environment.

Overview

- Develop a PyBullet environment that captures the scenario of a satellite or drone in orbit and targeting a plume for scientific discovery
- Train the agent to navigate towards the plume
- Try to engineer a multi plume environment to test the agent's decision making
- Develop a reward function that captures both accuracy and speed so the mission can be successful while also prioritizing fuel and efficiency
- Evaluate the agent based on average reward per episode and other metrics

Understanding the problems

01 Penalties: Negative rewards for collisions with obstacles or incorrect analysis.

02 Exploration: provide rewards for finding previously unvisited plumes

03 Adaptability: Reward for successfully adapting to changes in the environment.

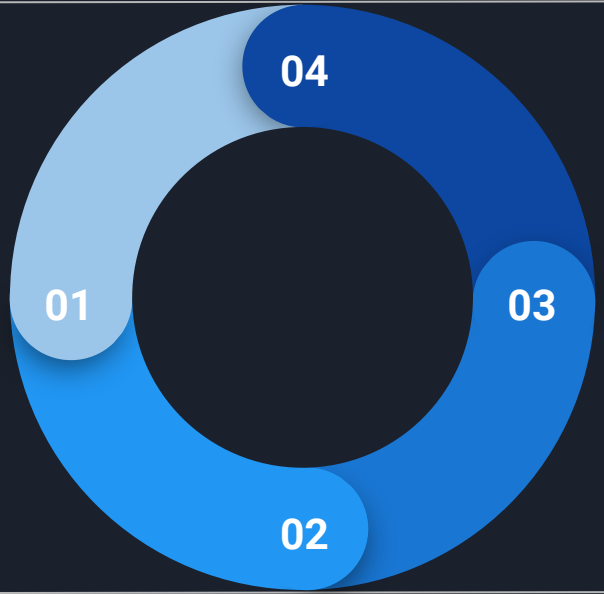
Planned Project Steps

Environment

Design the environment with one satellite and a) single plume b) multiple plumes

Reward function

Engineer a reward function that encompasses both efficiency and accuracy



Agent Evaluation

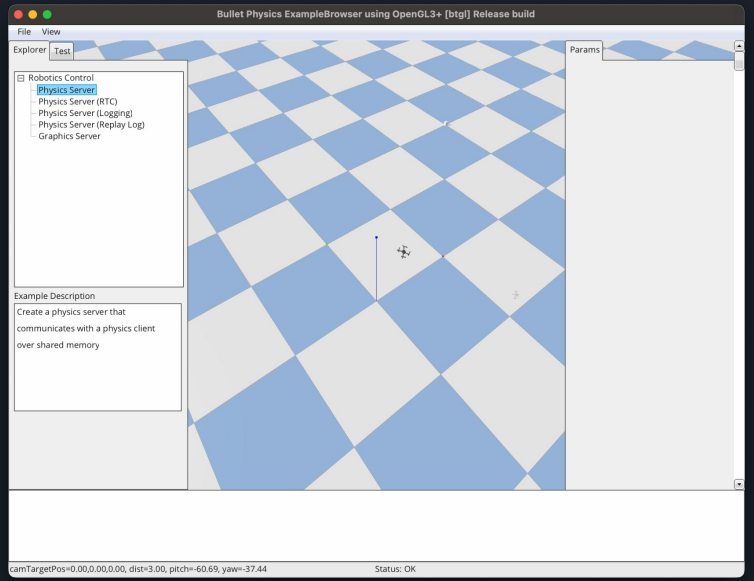
Metrics may include average reward per time step

Model Training

DQN or PPO from Stable Baselines

Demo

- In this screenshot, we can see the PyBullet environment, the singular satellite agent we have, and also the plume target that the agent is navigating towards
- The agent is fixed on the z axis so that it will only move in the x and y planes and the reward function defines success as the satellite coming within a concentration threshold of 0.8 or closer to the plume



Proximal Policy Optimization (PPO)

- PPO is an on-policy stochastic training method

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

  - Exploration involves sampling actions based on the latest stochastic policy
  - Randomness in action selection depends on initial conditions and training procedure
  - Policy gradient method clip function: constrains policy updates from extreme magnitudes

- Advantages of PPO
  - Balances between performance and comprehension
  - Simplicity
  - Stability
  - Sample efficiency

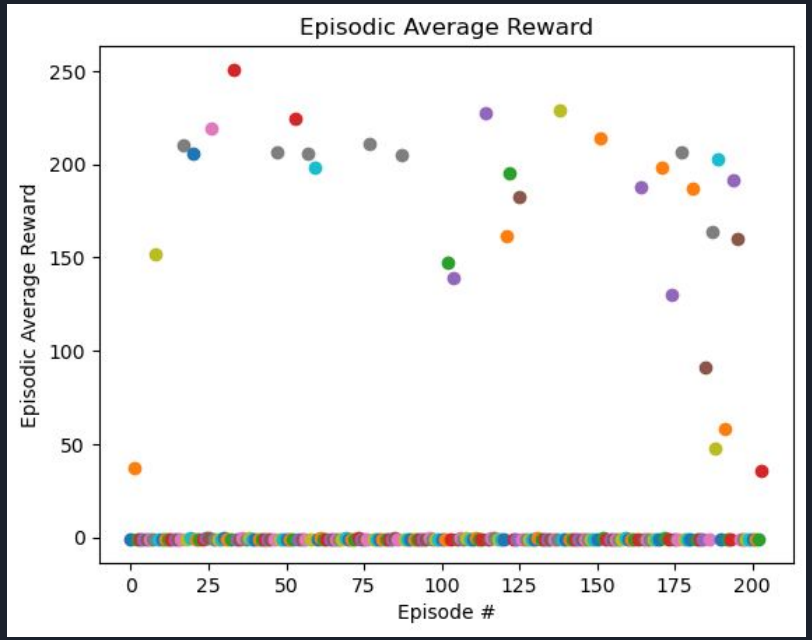
- Advantage function calculation: A = discounted sum (Q) - baseline estimate (V)
  - discounted sum: total weighted reward for the completion of a current episode
    - calculated after an episode's completion so program records episode's outcome
    - calculating advantage is essentially an unsupervised learning problem
  - baseline estimate: value function that outputs the expected discounted sum of an episode starting from the current state.
  - A > 0: actual return of the action is better than the expected return from experience
  - A < 0: how bad the actual return is based on the expected return

Reward Function

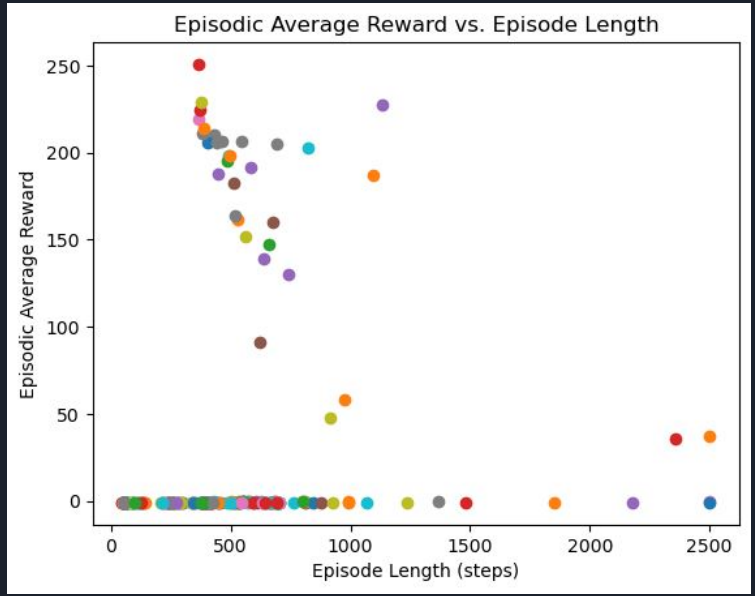
- Reward function engineering
  - Singular plume environment
  - Multi-plume environment
- Overview of the reward
  - Strong positive reward for goal target
  - Negative reward correlated to distance away from the nearest targeted plume
- Future adjustments to make
  - Better splitting of ties between plumes
  - Dynamic concentration threshold: decrease threshold as more plumes are discovered
  - Include penalty ahead of episodic truncation

```python
def _computeReward(self):
    """
    Function to calculate reward at each step.

    Parameters
    ----------
    None

    Returns
    -------
    int reward
    """

    min_distance = np.inf
    closest_plume = None
    for drone_position in self.get_current_positions():
        for plume_position in self.plume_positions:
            distance = np.linalg.norm(drone_position[:2] - plume_position)
            if distance < min_distance:
                min_distance = distance
                closest_plume = plume_position

    concentration = self.get_concentration_value(self.get_current_positions()[0])
    if concentration > 0.8:
        if closest_plume in self.visited:
            print('Already visited this plume')
            reward -= 1_000
        else:
        # Generate a positive reward for finding the goal
            reward += 1_000
            self.visited.add(closest_plume)
            print(f'found plume at position: {closest_plume}')
    else:
        # Generate a negative reward for not finding the goal
        reward -= 1 - concentration

    return reward
```

Evaluation - Average Reward per time step per episode



Evaluation - Episodic Average Reward vs Episodic Length



Evaluation - Satellite Episode Rewards for each time step

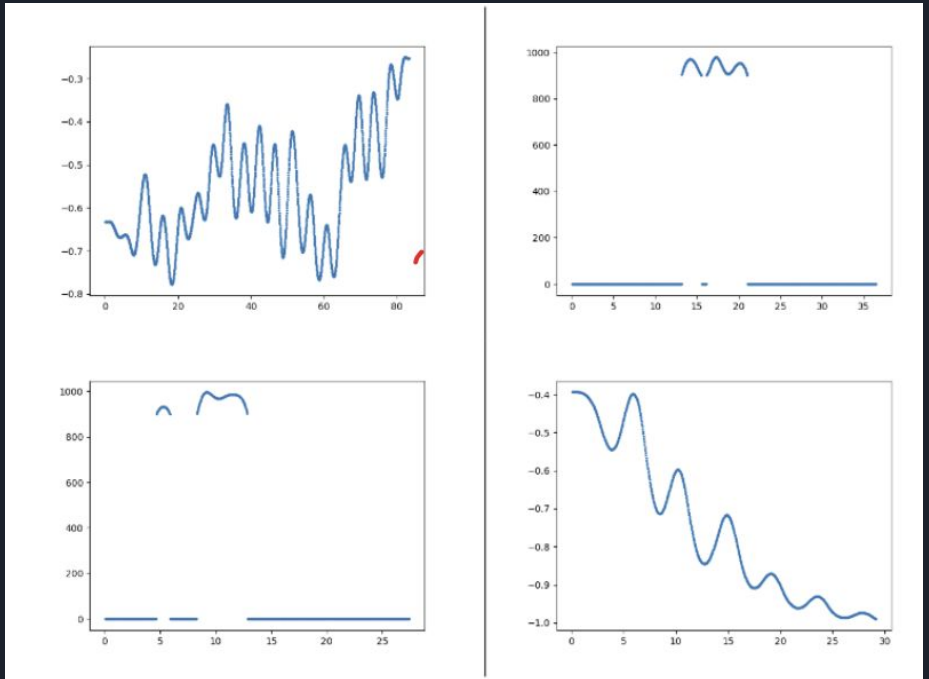


Table 1: Episode rewards at each time step

Evaluation - Satellite Trajectories and Oscillation

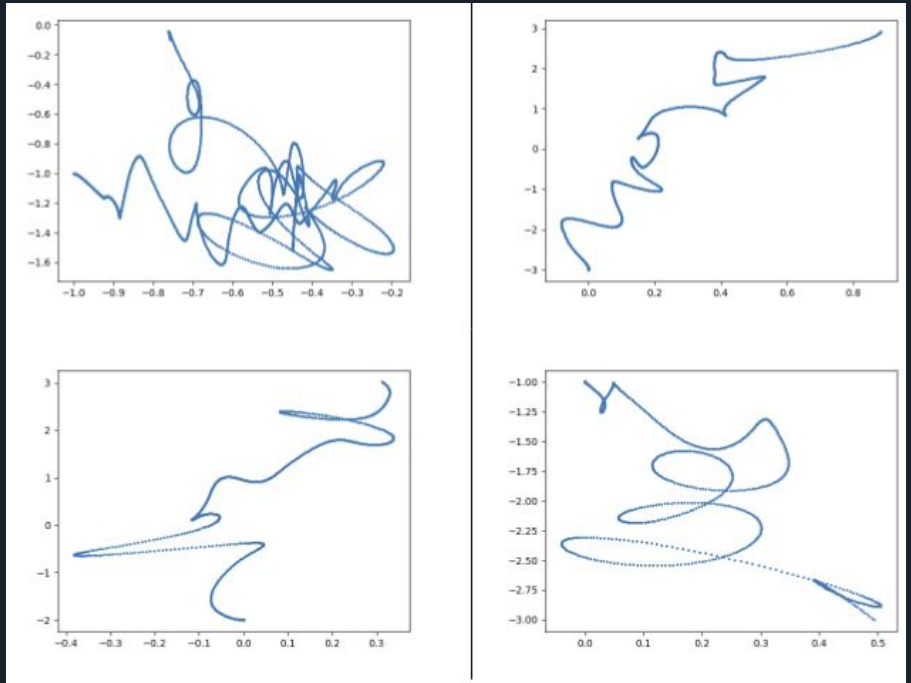


Table 1: Episodic Drone Trajectories

Thank you!