Report: Decision Making for Autonomous Vehicles on Highways --Group 41

Github: https://github.com/yYang-bit/ME5418.git

Introduction:

This report details the development and implementation of the SumoGym environment, which is designed to simulate and train the vehicle decision-making on highways. The SumoGym class extends the gym.Env class from OpenAI Gym. This environment models an ego vehicle (car_0) navigating a multi-lane highway, making decisions such as lane changes and speed adjustments to maximize rewards while avoiding collisions. The class encapsulates methods for initializing the simulation, subscribing to vehicle contexts, retrieving observations, executing actions, handling collisions, and resetting the environment.

Code Structure and Explanation

Initialization (__init__)

- Ego Vehicle Identification: The ego vehicle is designated as "car_0".
- Simulation Parameters: Includes lane change duration, maximum and minimum speeds, acceleration, deceleration rates, step counter, and observation radius.
- Observation Space: Defined using spaces.Box to include the ego vehicle's position, lane ID, velocity, and the availability of adjacent lanes.
- Action Space: Defined using spaces.Discrete with five possible actions: change left, change right, idle, accelerate, and decelerate.
- SUMO Configuration: Specifies whether to display the SUMO GUI and the path to the SUMO configuration file.

Context Subscription and Data Retrieval

- subscribe_context Method: Utilizes TraCI (Traffic Control Interface) to subscribe to specific vehicle variables within a defined radius around the ego vehicle. This method monitors the position and lane index of surrounding vehicles.
- get_subscribed_data Method: Retrieves the subscribed context data, excluding the ego vehicle itself to focus on nearby traffic.
  Ego Vehicle State (get_ego_state)

Ego Vehicle State (get_ego_state)

This method returns the ego vehicle’s position, lane ID, and speed using TraCI commands, forming the basis for decision-making.

Observation Generation (get_observation)

- Ego Position: The longitudinal position of the ego vehicle.
- Ego Lane ID: The current lane index.
- Ego Velocity: The vehicle's speed.
- Left Lane Availability: Indicates if a lane change to the left is feasible.
- Right Lane Availability: Indicates if a lane change to the right is feasible.
- Change Lane Flag: Suggests whether a lane change is advisable based on surrounding

traffic conditions.

Action Execution (step)

- Action 0 (Change Left) and Action 1 (Change Right): Attempts lane changes while checking for boundary conditions. Rewards are adjusted based on the necessity and success of the maneuver.
- Action 2 (Idle): Rewards maintaining the current speed and lane.
- Action 3 (Accelerate) and Action 4 (Decelerate): Modifies the vehicle's speed within set limits, rewarding beneficial speed adjustments.
- Reward Mechanism: Incorporates penalties for collisions and rewards for desirable behaviors like successful lane changes and maintaining optimal speeds. In addition, the velocity reward encourages the larger speed. The closer to maximum speed, the better.

Rest and Close Methods

Resets the environment for a new simulation episode, restarting SUMO and returning the initial observation. Terminates the SUMO simulation, ensuring no background processes remain active.

Libraries and Tools Utilized

OpenAI Gym

OpenAI Gym provides a standardized environment framework, allowing the integration of RL algorithms with SumoGym.

SUMO (Simulation of Urban Mobility)

SUMO is an open-source traffic simulation tool that models vehicle dynamics and road networks. The TraCI interface allows external programs to control SUMO simulations in real-time.

NumPy and TraCI

NumPy is used for handling numerical operations, particularly in defining the observation and action spaces. TraCI enables real-time communication between the environment and SUMO.

Reflections

In the initial stages of writing the code, a simple method was used to gather observation space data by retrieving the IDs of all vehicles and calculating the distance between each vehicle and the ego car, filtering those within the observation range. However, testing revealed that this approach was inefficient, taking too much time for high-frequency observation updates. To improve efficiency, the subscribed_context and get_subscribed_data methods were later implemented, allowing the ego car to receive observation data in a more timely manner.

This highlights an important consideration: when designing algorithms, it is crucial to account for computation time. **For data that needs to be accessed frequently, further optimization of the algorithm is necessary to ensure real-time performance in high-frequency tasks.**