Reward-Based Reinforcement Learning for Autonomous Vehicle Navigation in a Circular Track

Cedric Slavin

11/8/2024

Abstract

This project implements a reinforcement learning approach to navigate an autonomous vehicle on a circular track without colliding with other vehicles. The vehicle is trained with a Q-learning algorithm that rewards safe driving actions and penalizes behaviors leading to potential collisions or deviations from the track boundaries. This abstract discusses the reward-based system, Q-learning implementation, and initial results from training the vehicle to achieve collision-free navigation.

1 Introduction

Autonomous vehicle navigation requires dynamic decision-making to ensure safety and efficiency. This project uses reinforcement learning (RL) to train an autonomous vehicle to navigate a circular track environment. Through a Q-learning algorithm, the vehicle learns an optimal policy that balances speed and steering adjustments based on a reward system. The system awards points for safe navigation and penalizes actions leading to unsafe conditions.

2 Reinforcement Learning Framework

2.1 State and Action Space

The state is defined by the vehicle's relative position to track boundaries and other vehicles, while the action space includes acceleration, braking, and steering adjustments.

Listing 1: State and Action Definitions

```
actions = ["accelerate", "brake", "steer_left", "steer_right"]

def get_state(sensor_data):
    return tuple(sensor_data.values())
```

In this setup, the vehicle's position is represented as discrete states based on sensor data, while the actions control the vehicle's motion.

2.2 Reward System

The reward function provides positive rewards for safe driving and penalties for approaching other vehicles or deviating from the track.

Listing 2: Reward Calculation

```
def get_reward(sensor_data):
    if sensor_data["front"] < 2:
        return -10  # Penalty for potential collision
    elif sensor_data["left"] < 1 or sensor_data["right"] < 1:
```

```
        return -5  # Penalty for track boundary proximity
    else:
        return 1  # Reward for safe navigation
```

2.3 Q-learning Algorithm

The Q-learning algorithm is implemented by iteratively updating the Q-values for each state-action pair. The Q-value update rule considers both immediate rewards and expected future rewards.

Listing 3: Q-learning Update

```
def update_Q_table(state, action, reward, next_state):
    action_idx = actions.index(action)
    if next_state not in Q_table:
        Q_table[next_state] = np.zeros(len(actions))
    best_next_action = np.argmax(Q_table[next_state])
    td_target = reward + discount_factor * Q_table[next_state][best_next_action]
    Q_table[state][action_idx] += learning_rate * (td_target - Q_table[state][action_idx])
```

In this implementation, the Q-values are updated based on the action taken, the reward received, and the best possible future rewards in the next state.

3 Simulation Results

After training the Q-learning agent, the autonomous vehicle demonstrated improved ability to navigate the circular track without collisions. Figure 1 shows a snapshot from the simulation, highlighting the vehicle's trajectory as it successfully avoids nearby vehicles and track boundaries.

4 Conclusion

The reinforcement learning approach used in this project successfully trained the vehicle to navigate the circular track environment, with Q-learning as the core algorithm for learning safe navigation policies. The reward-based system incentivizes collision-free behavior, establishing a foundation for more advanced algorithms in future levels.

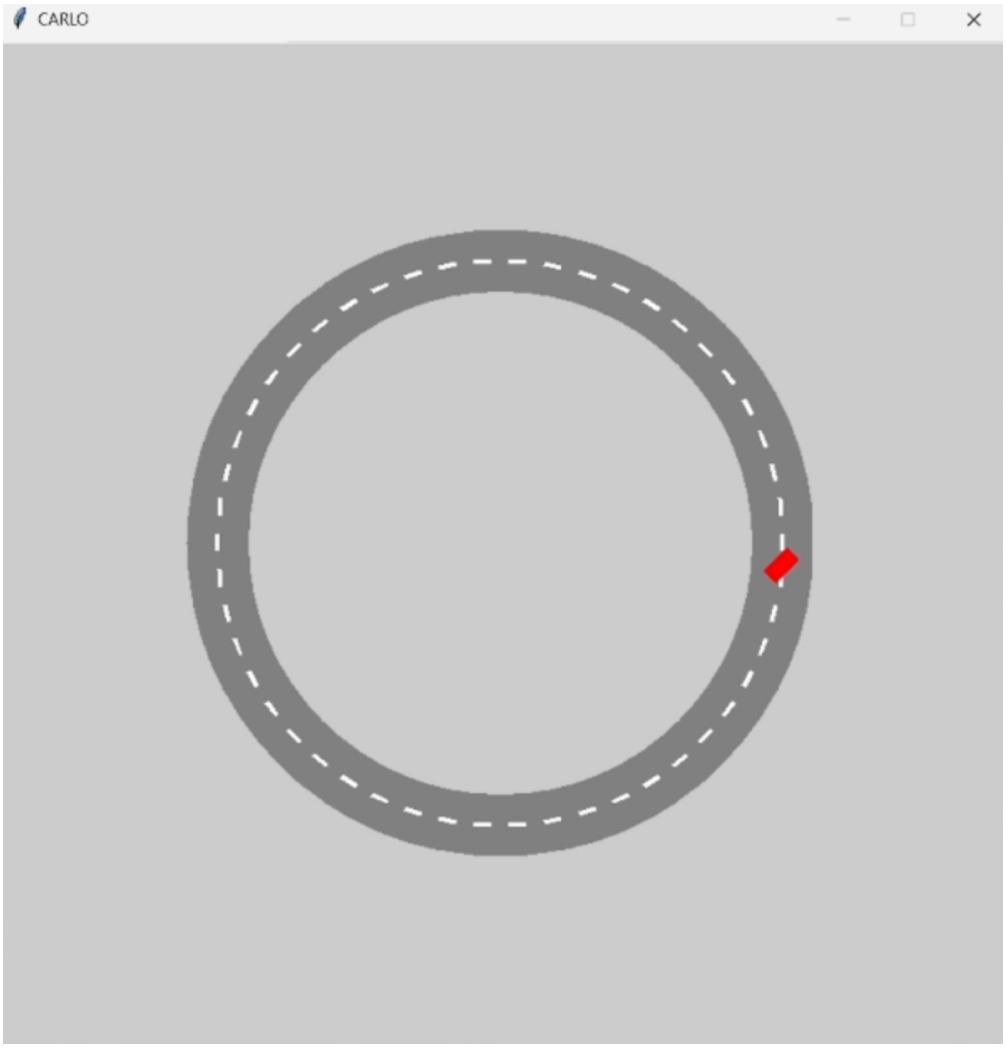

Figure 1: Simulation results showing the vehicle navigating safely around the track.