Ex. No: 16	**CARLA Simulation**

Aim:

Use CARLA to simulate any experimental CARLA setup for autonomous vehicles.

Reference GitHub link:

https://github.com/Geniussh/Self-Driving-Car-Projects/tree/main/Motion%20Planning

Explaination:

Agent:

Your agent serves as the core decision-making entity in your autonomous driving system. It integrates various modules to navigate through complex environments:

1. **Behavioural Planner:** This component is responsible for high-level decision-making. It manages scenarios like stop signs at T-junctions using a state machine. It transitions between different states based on the current situation and predefined rules.
2. **Local Planner:** This module generates detailed trajectories or paths for the vehicle. By utilizing a spiral path generation technique based on given waypoints and employing optimization techniques from the scipy library, it ensures the vehicle follows smooth and feasible routes.
3. **Collision Checker:** Safety is paramount. This module verifies that the paths generated by the local planner are free from collisions. By implementing circle-based collision checking, it provides an additional layer of security.
4. **Path Selector:** Once potential paths are generated, this module evaluates them based on an objective function. This ensures the selection of the safest and most feasible path for execution.
5. **Velocity Planner:** This component is crucial for determining how fast the vehicle should travel along the selected path. It considers various scenarios, including stop signs and dynamic obstacles, to generate a velocity profile for the controller.

Environment:

The CARLA simulator provides a virtual urban environment that accurately mimics real world driving scenarios. It includes streets, intersections, vehicles, pedestrians, and other objects. This serves as the testing ground for your autonomous driving system.

Sensor:

The setup relies on LIDAR measurements as a key sensory input. These measurements provide a detailed 3D representation of the environment, which is vital for tasks like generating the occupancy grid.

Algorithmic Model:

The motion planning stack combines various algorithms, enabling your agent to make informed decisions:

1. **State Machines:** Used in the behavioural planner, state machines allow for structured decision-making by transitioning between predefined states based on the current situation.
2. **Optimization with scipy:** The local planner leverages optimization techniques from the scipy library to generate smooth and optimal paths based on given waypoints.
3. **Circle-based Collision Checking:** This algorithm ensures that the generated paths are collision-free by checking for intersections with obstacles represented as circles.
4. **Objective Function Evaluation:** The path selector uses objective functions to evaluate and rank potential paths, ensuring the selection of the most suitable path for execution.
5. **Dynamic Obstacle Handling for Velocity Planning:** The velocity planner adapts the speed profile based on dynamic obstacles and other scenarios, ensuring safe and efficient motion.

Interactions with the CARLA Environment:

1. **Sensor Data Reception:** The agent receives LIDAR measurements, which serve as a critical source of environmental perception.
2. **Behavioural Planner Operations:** The agent uses the behavioural planner to handle specific scenarios, such as stop signs at T-junctions. State transitions occur as needed to navigate through intersections.
3. **Local Path Generation:** The local planner uses optimization techniques to generate detailed paths based on provided waypoints. This ensures precise vehicle navigation.
4. **Collision Checking:** The collision checker verifies that the generated paths are free from collisions, adding an additional layer of safety to the planning process.
5. **Path Selection:** The path selector evaluates potential paths based on an objective function, ultimately choosing the best-suited path for execution.
6. **Velocity Profile Generation:** The velocity planner determines the speed profile along the selected path, considering various scenarios like stop signs and dynamic obstacles.
7. **Controller Commands:** The controller updates the vehicle's control commands (e.g., throttle, brake, steering) based on the computed path and velocity profile, ensuring the planned motion is executed accurately.

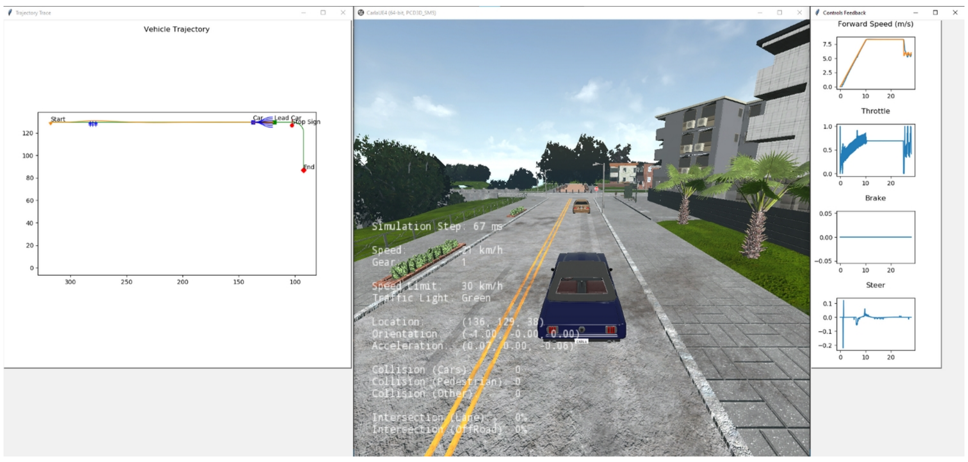
Occupancy Grid Generator:

This component utilizes LIDAR measurements to create an occupancy grid belief map. The inverse scanner measurement model and iterative logodds updates contribute to an accurate representation of the environment's occupancy probabilities.

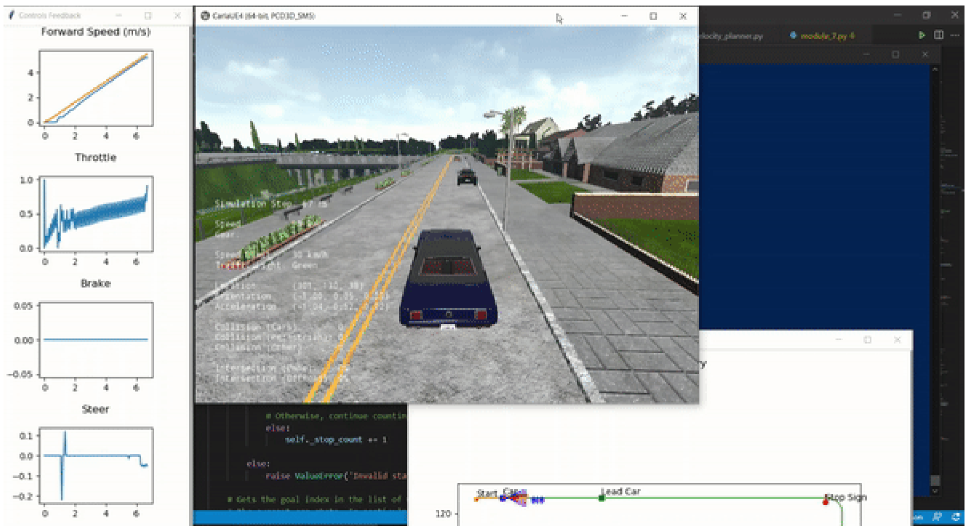
Mission Planner:

The mission planner incorporates Dijkstra's and A* search algorithms to plan routes on a road network in Berkeley, California. The use of OSMNX and NetworkX libraries for generating Open Street Map data and determining the correct shortest path serves as a solid foundation for route planning.

Screenshots/Simulations:



```
Simulation Step: 67 ms

Speed:              21 km/h
Gear:               1

Speed Limit:        30 km/h
Traffic Light:      Green

Location:           (136, 129, 38)
Orientation:        (-1.00, -0.00, 0.00)
Acceleration:       (0.07, 0.00, -0.06)

Collision (Cars):           0
Collision (Pedestrian):     0
Collision (Other):          0

Intersection (Lane):        0%
Intersection (OffRoad):     0%
```



```
            # Otherwise, continue countin
            else:
                self._stop_count += 1

        else:
            raise ValueError('Invalid sta

    # Gets the goal index in the list of
```

Result:

Therefore, successfully simulated an experimental setup for autonomous vechiles using CARLA.