Assignment 2: System Implementation

Word count: 660

Project Description and Module Overview

The developed software supports the following systems:

1. Object Detection System (*./system/objectdetection.py*)
    a. Responsible for detecting and locating objects including measuring distance from vehicle.
2. Path Planning System (*./system/pathplanning.py*)
    a. Responsible for routing, including generating, recalculating and following the route.
3. Locator System (*./system/locator.py*)
    a. Responsible for fetching vehicle's current position.

Additional system modules store:

1. LiDAR (*.system/lidar.py*) sensor object, which in real life is required by the Object Detection System (Liu et al., 2023) and is an addition to the original design from Assignment 1, and
2. GPS (*.system/gps.py*) and IMU (*.system/imu.py*), required by Locator and Path Planning systems.

Functions in *VehicleInterface* and *Vehicle* classes, located in __*main*__*.py* module, support operations such as start, stop and drive, which update engine and moving status of vehicle. The focus of this assignment was to build primitive building blocks of sensing capabilities and simulate their operations.

Additional helper modules are present to support the systems:

1. *./constants.py*

   a. Contains constant values to support the Locator and Planner classes for navigation and positioning of vehicle.

2. *./util.py*

   a. Contains utility functions for generating random values and objects to support the overall testing of the system.

3. *./system/geometry.py*

   a. Contains the Point object class with distance method.

The updated Class UML diagram is present in Figure 1.

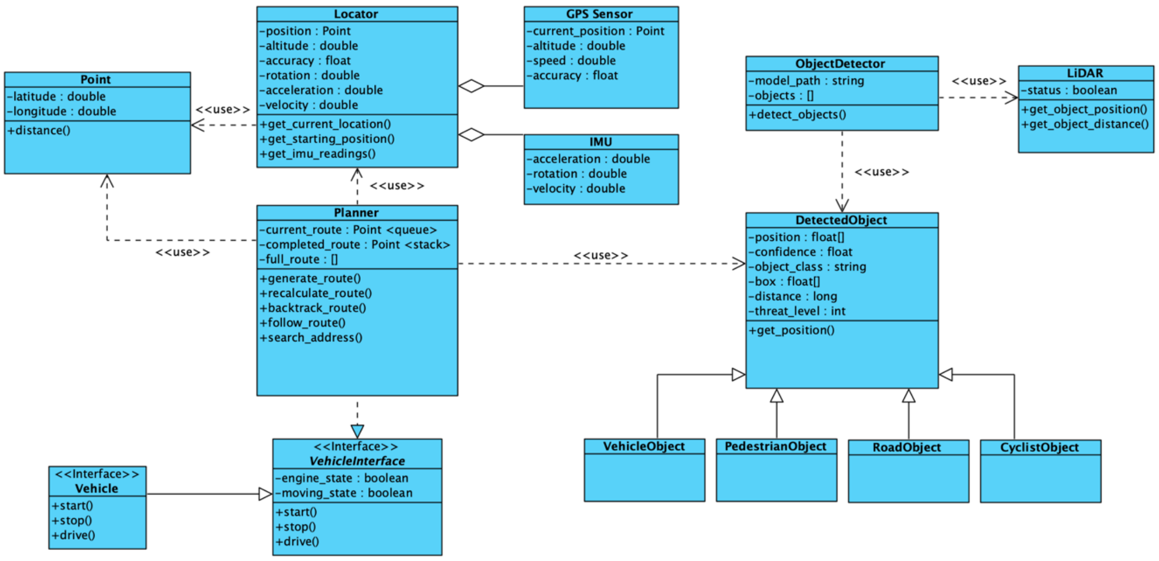

Figure 1. Class UML diagram of Autonomous Vehicle.

Design

The system design attempts to combine multiple design patterns:

1. **Singleton** pattern, applied for class *Vehicle* using the recommended Python implementation to ensure a single vehicle instance exists (Rhodes, n.d.).
2. **Status** pattern, also applied to class Vehicle, via updating engine and moving status throughout vehicle operations.

3. **Strategy and Observer** patterns, based on vehicle behaviours and responses to its environment such as routing, rerouting, adjusting position.

System Testing Approach

Given the dynamic nature of designed systems, testing was achieved by simulating real-life environments where the vehicle moved from a starting coordinate to an end coordinate. This was achieved using:

1. Randomly generated coordinates
2. Randomly generated route that permits left and right turns and straight runs
3. Randomly generated object attributes for object detection

A random seed of 15 is set constant throughout the script to ensure test values always remain the same. The system combines unit tests and simple assertions within the scripts. Additional logic was tested and validated using external websites and references such as GeoMidpoint (n.d.) and Carvalho et al. (2022) for coordinates.

Running the script

To run the main script, navigate to the directory containing the project in the terminal and use the -m option followed by the package name. For example,

```
$ cd path/to/autonomous_vehicle
$ python -m autonomous_vehicle
```

To run the test script, navigate to the test directory containing test_coordinates.py and execute the following command in your terminal:

```
$ python -m unittest test_coordinates.py
```

Notes and Reflections

This was a highly involved assignment given the complexities of an autonomous vehicle system; however, it was crucial to explore the various aspects involved, particularly the simultaneous navigation and detection of objects to ensure safe passage from A to B. Additional consideration and development would be required to make the vehicle responsive to its environment, i.e., if threat level is high based on detected object and vehicle positions, the vehicle would slow down, and vice versa.

A simple routing solution was created for the purposes of this exercise. In a real-world scenario, the Travelling Salesman problem would be relevant to tackle an extension of this assignment. Additionally, this exercise assumes Earth is a sphere, rather than an ellipsoid. It is important to highlight that dedicated libraries have been developed to simplify operations with coordinates, bearings and distances, such as Shapely, GeoPandas and GeoPy amongst others.

Generally, for an exercise of this scale and complexity, it is vital to keep the scope controlled whilst capturing the necessary detail. It would have been beneficial to have additional time to explore the DetectedObjects class and its children and how these interact with the vehicle.

References:

Carvalho, H.S., Pilastri, A., Novais, R. & Cortez, P. (2022) RanCoord—A random geographic coordinates generator for transport and logistics research and development activities. *Software Impacts*, 14: 100428.

GeoMidpoint (n.d.) Bearing and Distance Calculator

Liu, H., Wu, C. & Wang, H. (2023) Real time object detection using LiDAR and camera fusion for autonomous driving. *Scientific Reports*, 13(1): 8056.

Raspberry Pi Foundation (n.d.) Finding the distance between two points on the Earth. Raspberry Pi Foundation. Available from: https://projects.raspberrypi.org/en/projects/fetching-the-weather/6 [Accessed 21 May 2024]

Rehrl, K. & Gröchenig, S. (2021) Evaluating localization accuracy of automated driving systems. *Sensors*, 21(17): 5855.

Rhodes, B. (n.d.) The Singleton Pattern. Python Design Patterns. Available from: https://python-patterns.guide/gang-of-four/singleton/ [Accessed 25 May 2024]

Veness, C. (n.d.) Calculate distance, bearing and more between Latitude/Longitude points. *Movable Type Scripts*. Available at: http://www.movable-type.co.uk/scripts/latlong.html [Accessed 21 May 2024]