Autonomous car

Implementation description

This application emulates three functions of an autonomous car. It provides a menu with four options. These options allow us to define the shortest route, show the use of emergency brakes, and simulate lane-centring occurrences. Each of these options will be explained in detail in the instructions section.

Data structures

Shortest route

We use a dictionary data structure storing point objects to generate a map of locations. Each location corresponds to a point in the Cartesian plane. Using a dictionary responds to the need for unique indexes that are not required to be consecutive. Additionally, the search operation gets reduced to a check on the existence of the index when the application needs to find out if a location is currently on the map.

The shortest distance between two points is a straight line for the emulation. This distance is calculated using the Pythagorean theorem in the Euclidian plane.

To store the points in the shortest route, we used a list whose elements are point objects extracted from the map. The list of objects provides an ordered sequence that the car can follow until reaching the destination.

Collision avoidance

For this function, we use a list of objects that could appear near the car and another list containing distances. Both lists combine to produce a list of random tuples containing a type of object and its distance to the car.

In a real scenario, this function can also be provided by the sensor called RADAR but for this application we have used only the function provided by the sensor called LIDAR. The fact that both sensors can provide the same function is a case of Polymorphism.

Lane centring

To emulate the lane centring function, we create a lane map based on a list of tuples representing the left and right distances from the lane to the car. The chosen tuple is refreshed every second during execution.

Development approach evaluation

The approach used to develop this application was based on the UML standards. This supports current and future development as the UML supports this and potentially to a different type of sub-systems.

UML is not a development methodology which can bring advantages and disadvantages. One of its advantages is that it can be used in different stages of development, so it can also be used in the testing stage to build up test cases.

In terms of programming, Object-oriented Programming provided the concept and implementation of Inheritance and Polymorphism for this application.

Reusability is also another important concept that the development of this application considered. The use of classes to encapsulate the data and the behaviour makes it possible to reuse all the classes in this application.

Reflections

We have followed the suggested learned UML standard to support the development process of this application. This has included the design of use cases, state, activity, sequence, and class diagrams.

The use of UML made it easy to understand the application's behaviour and structure; therefore, the development was driven by the provided standards in Object Oriented approach.

The Object-oriented Programming provided modularity to expand the functions in a simple and organised way. Being the objects independent entities, we are able to work in multiple objects without interfering with other developers.

Main function

The interface of this application is provided by the main function, which controls the execution of all four available options. To ensure continuous interaction, the main function is run at the end of the execution of each selected option.

ANSI escape codes

To emphasise the messages and give the interface a different look and feel, we introduced ANSI escape codes wrapped in a class.

Environment

This application needs a Python3 environment to run.

No external libraries are required.

Built-in libraries

`sys`. Provides exceptions to prevent the abrupt termination of the program.

`os`. To ensure the ANSI escape codes work on a windows environment.

`random`. This library provides random numbers that are used in the generation of areas. These areas are required by the seeder function to generate a map of coordinates. The library is also used to generate random distances for the objects surrounding the car.

`time`. This library provides functions that allow operations on the time. This is required to execute the `sleep` function during emulation.

Automated test

Custom assert statements have been written to test the class methods. No testing frameworks have been used for this application. Data for the tests are provided mainly by the Seeder class.

Instructions

Execution

Use the terminal to execute the application.

```
python3 autonomouscar.py
```

Main menu

1. Route planning

2. Collision avoidance

3. Lane centring

4. Exit

Route planning

This option will first generate a map of areas and their corresponding coordinates. Then, it will calculate a route to be followed by the car to reach its destination using the shortest path.

Collision avoidance

After generating the shortest route, this option will allow for emulating the collision avoidance event. The car will stop and will not resume after. The random numbers picked may make the function terminate after a few seconds so please try again if necessary.

Lane centring

For this function, we use a combination of the collision avoidance functionality and adding distances from the car to the lane markers. The system will automatically centre the car if the distance to the lane markers or road edges is less than 30 cm. The random numbers picked may make the function terminate after a few seconds so please try again if necessary.

Exit

This option terminates the execution of the application.