A Design Proposal of Software to Support Operation of a Driverless Car

The software development for an autonomous vehicle is both broad and complex. This report considers three operations of a self-driving car:

1. Drive to location
2. Collision avoidance
3. Emergency stop

The operations will be designed to comply with regulations for England, according to the Highway Code. The software design will be an object-oriented paradigm designed to show the car and the wider environments.

Classes and objects within the system

The Vehicle Class:

The system will consider driverless vehicles as a class with the following attributes and methods:

Vehicle
reg: string make : string model: string drive: Movement
locate_road() detect_obstacle() get_speed_limit(road) get_stopping_distance() steer() accelerate() brake()

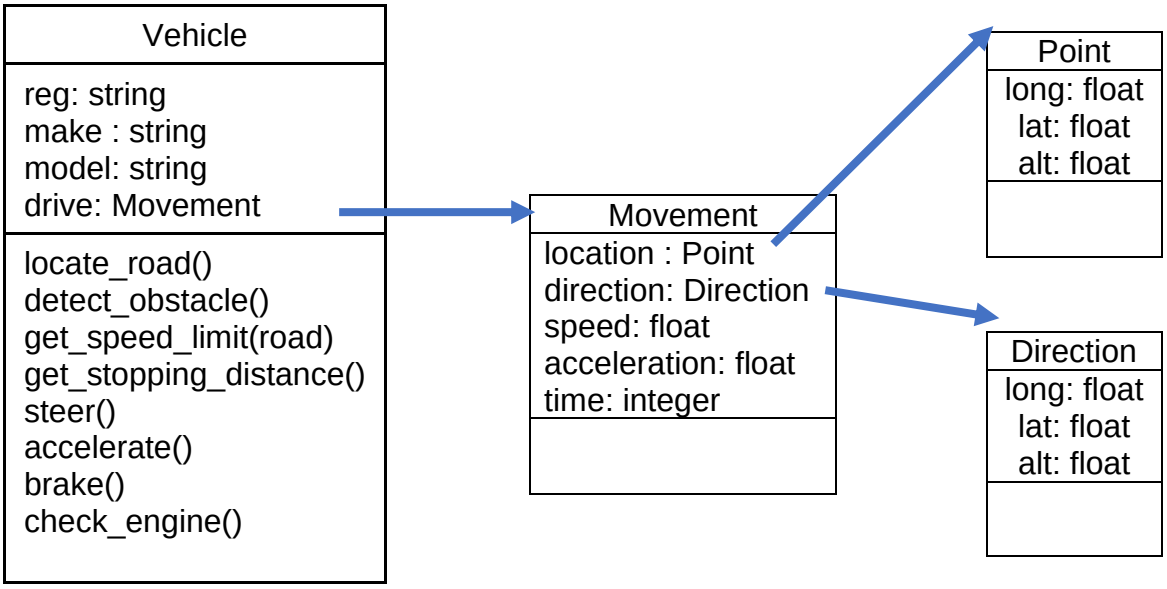
The reg, make and model will describe the vehicle, while the drive attribute will hold the data concerning the position and movement of the vehicle at a point in time. The Movement class is illustrated below:

Movement
location : Point direction: Direction speed: float acceleration: float time: integer

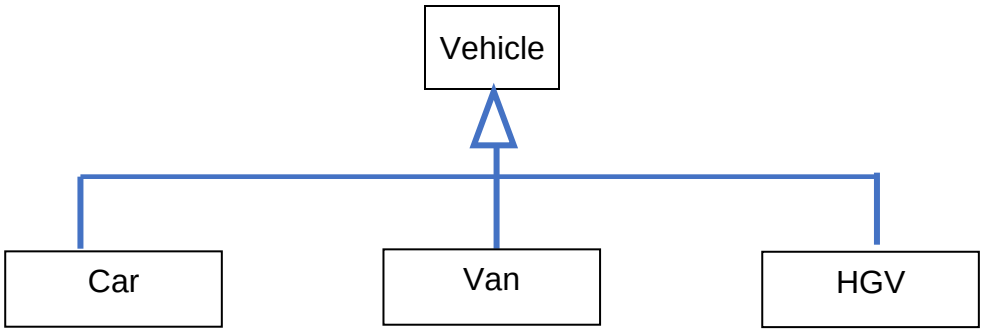
The Movement class introduces the Point and Direction classes. The Point class is illustrated below:

Point
long: float lat: float alt: float

The Point class represents coordinates for longitude, latitude and altitude and would be used to locate the vehicle's potion on the map. The Direction class represents the 3-dimensional vector, used to show the direction of travel. While this is a vector rather than a coordinate, the attributes are the same as the Point class, meaning that the Direction class can inherit the Point class attributes. The diagram below shows the Vehicle class with its associated classes of Movement, Point and Direction.

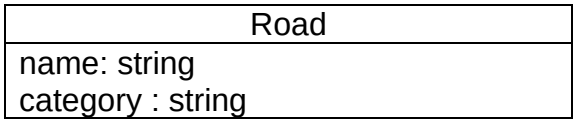


There are several types of vehicles to be considered: Cars, Vans and Heavy Goods Vehicles (HGV) as they all have different speed limits depending on the type of road they are using. The types of vehicles are shown below:



The Road Class:

In order for the autonomous vehicle to drive on the road, the vehicle must interact with the road network and each road will be an object of the Road class as shown below.



segments: Road_Segment

Each Road is made up by multiple road segments, of the Road-Segment class. The road segment is defined by a start and end locations from which the distance and gradient are determined. Each road segment is modelled as a straight section of road.

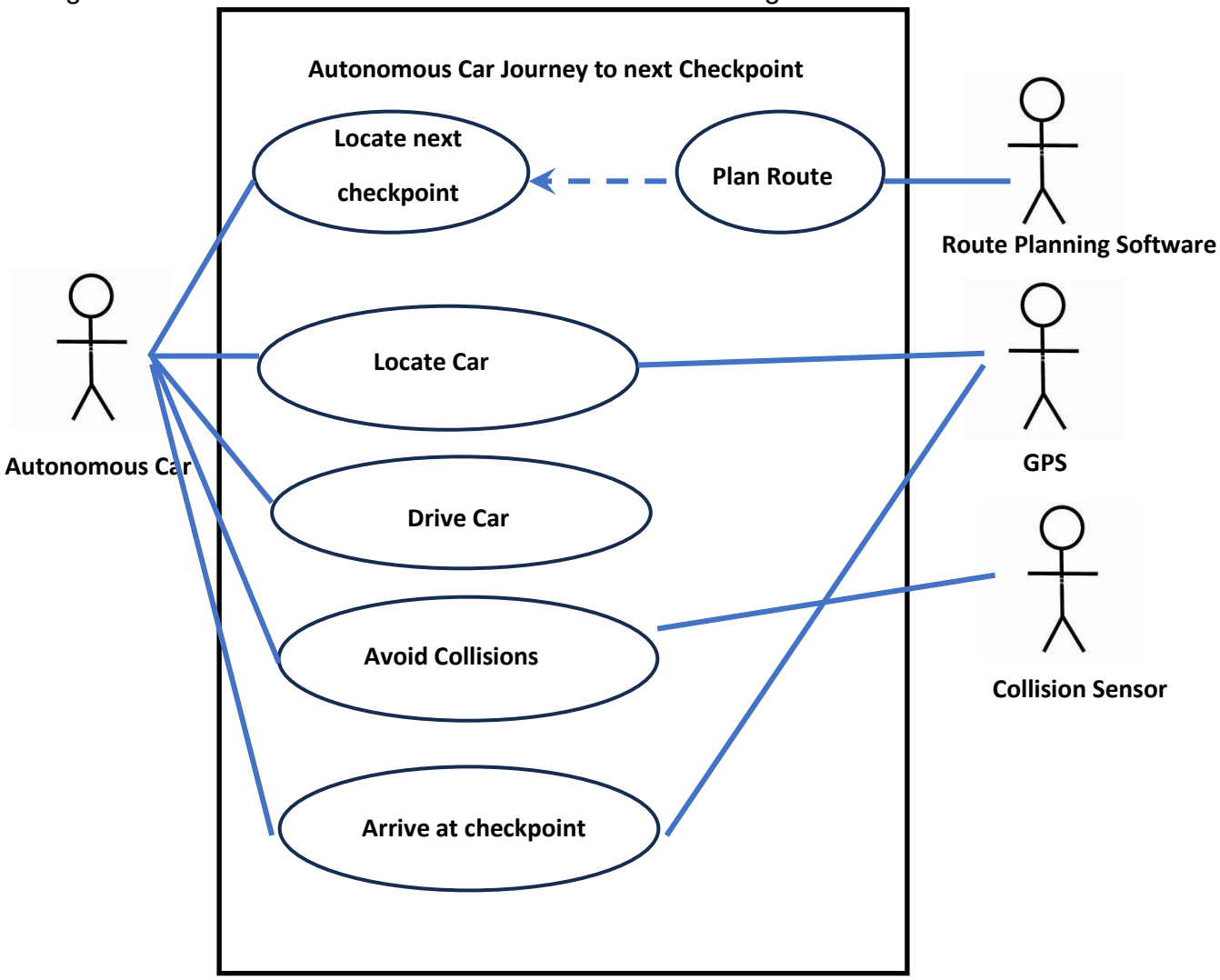
Road_Segment
index: integer location_start : Point location_end:Point direction:Direction distance: int gradient: int category: string
get_distance() get_category(road) get_direction() get_gradient()

As each road is made up of many road segments, this is represented below.

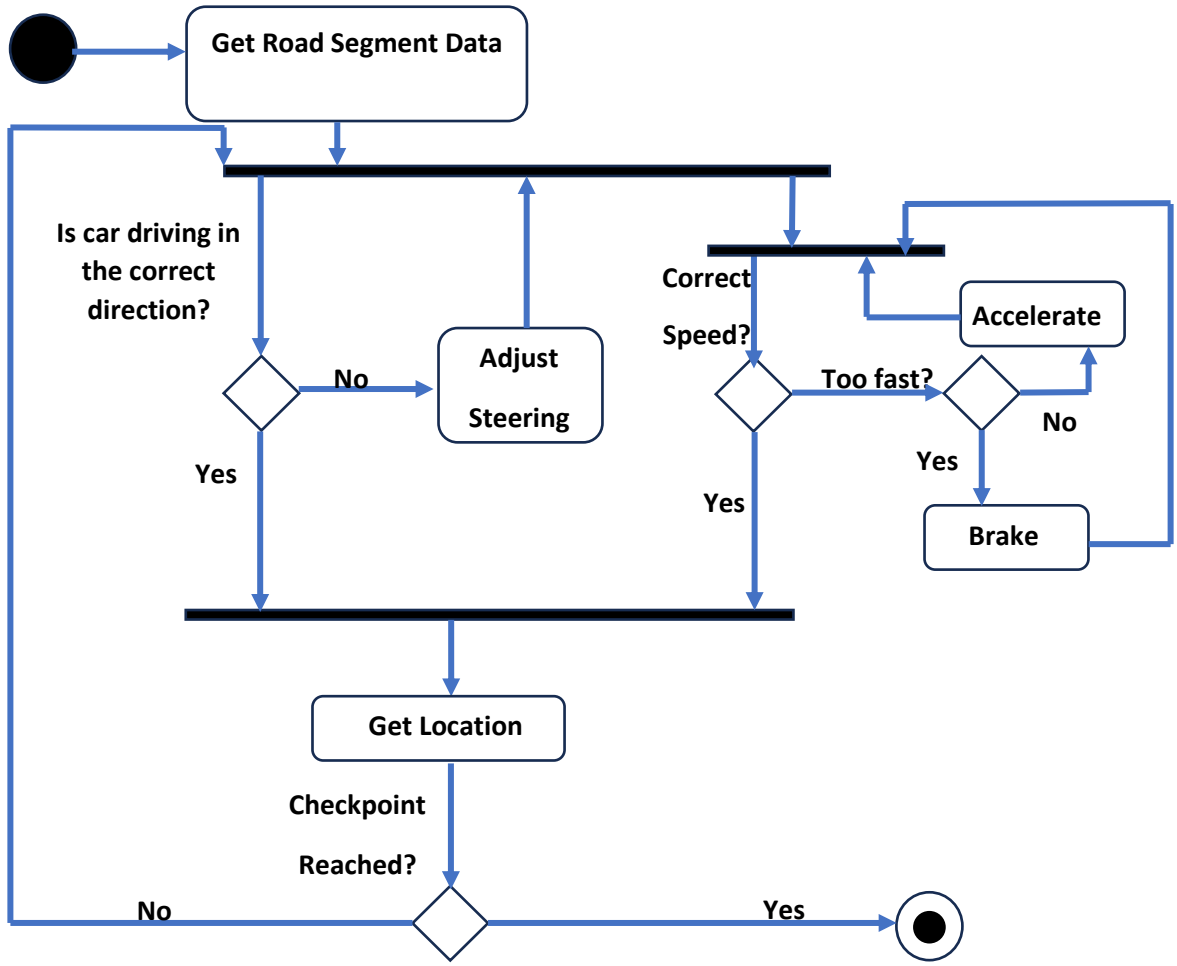
The roads and road segments will be stored as part of a road network database. Route planning software can be used to determine a route which can be stored as a list of road segments with the end of each segment representing a checkpoint in the journey. A route has the same attributes as a road as it is made up of road segments. As such a route could be considered a special type of road. The route does not have roads as that would imply that all of the road segments from a road are included in the route.

A GPS system can be used to locate the position of the car on the route and adjust the steering, acceleration, and braking methods to drive the car along the planned route. As the car passes the checkpoint (the end of each road segment) it continues to drive through the next road segment stored in the que, until it reaches the destination.

However, there may be other vehicles on the road and the car may have to adapt to ensure that a safe distance is maintained. Other vehicles, or hazards can be located using a collision sensor. This is shown in the use case diagram below.

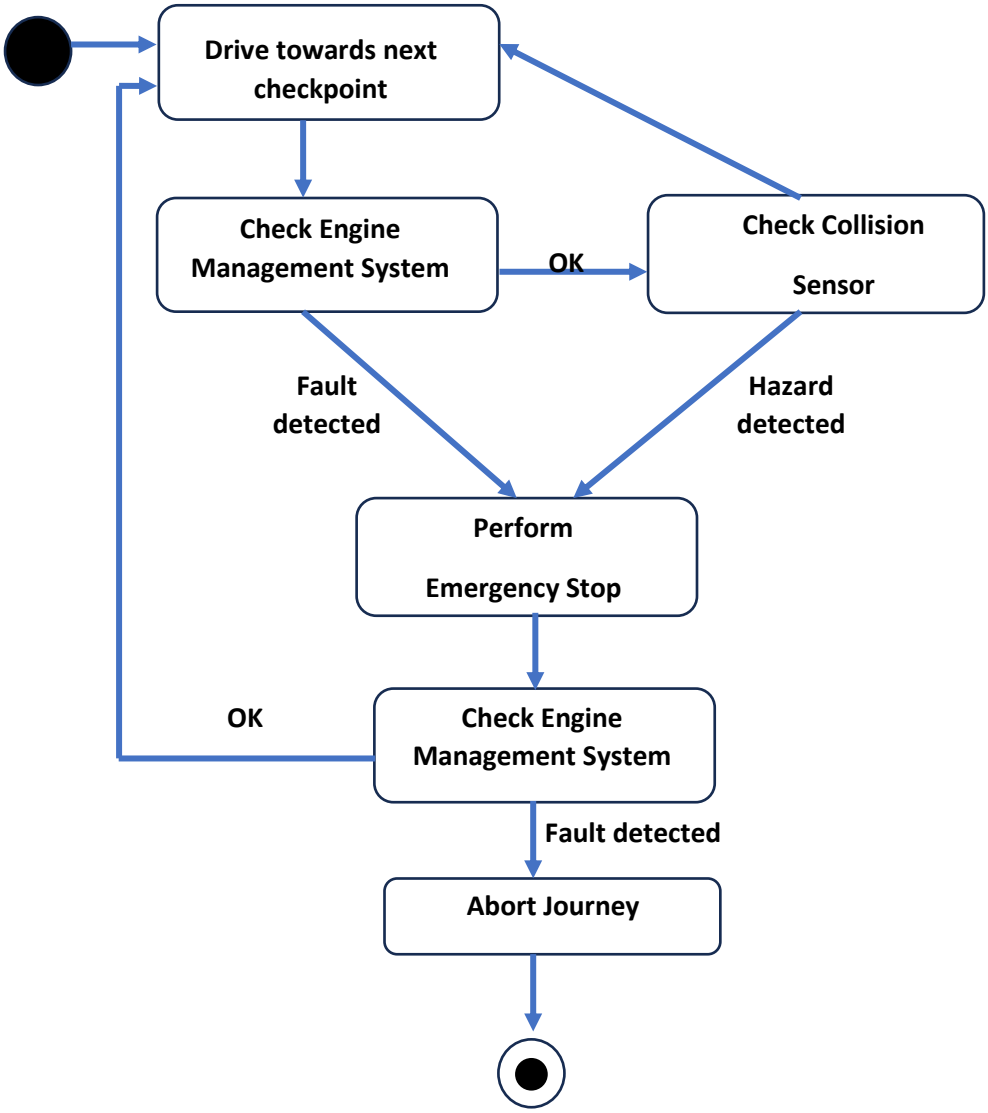


As the car is driving through each road segment, the steering, acceleration and braking activity will need to occur. The steering and acceleration/braking can happen simultaneously. This is shown in the activity diagram below.

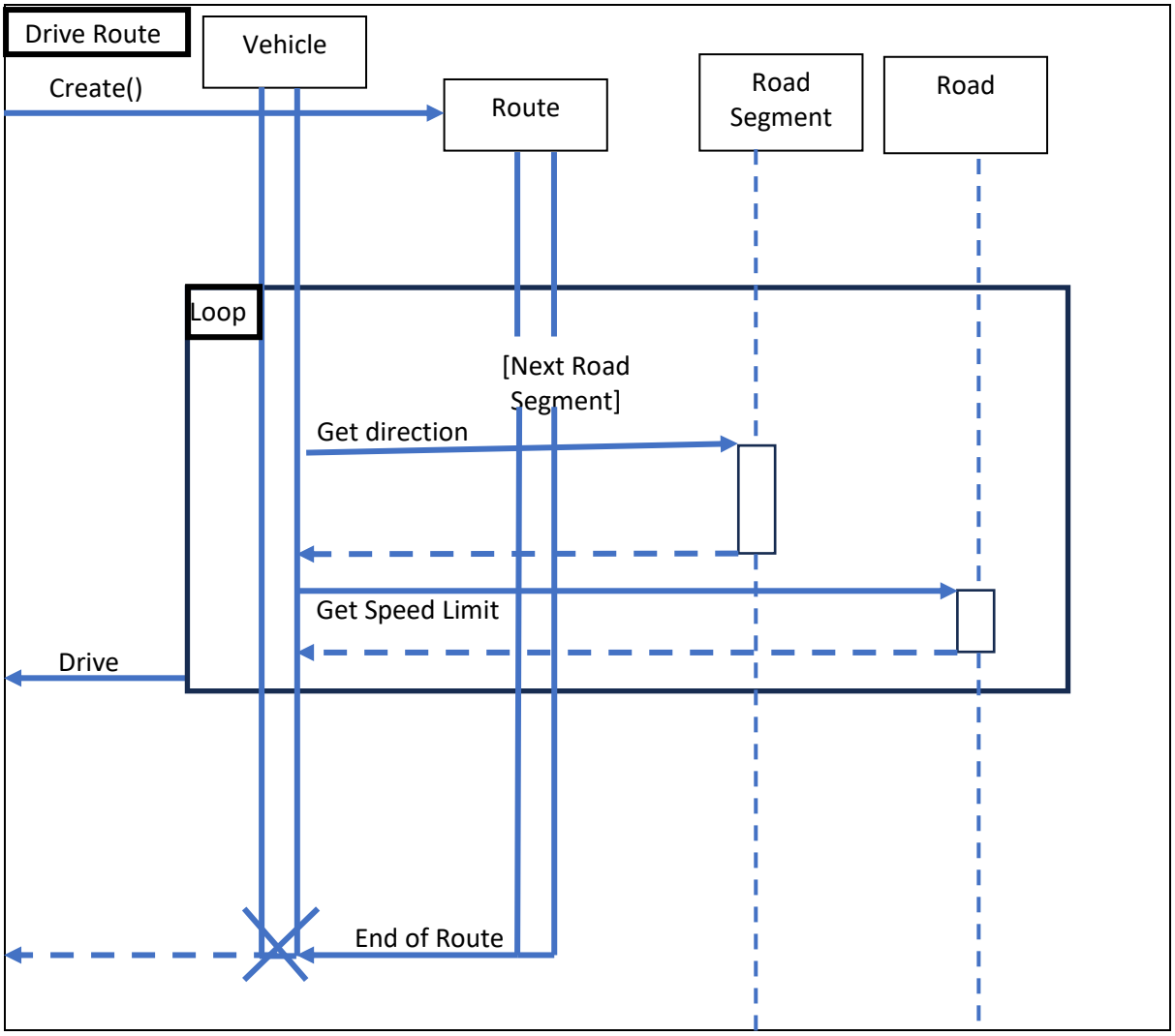


During the journey, the car may be required to perform an emergency stop due to a mechanical issue or avoiding a collision. The engine management system will store a time-stamped log of the vehicle's data. This is processed as a stack, so that the latest available information is being acted on, but a history of the vehicle's health is recorded throughout the journey. If the car is required to perform an emergency due

to a fault, the journey is aborted. This would represent a change of state and is shown in the state transition diagram below. If the vehicle performs an emergency stop to avoid a collision and there is no mechanical fault, the journey can be continued.



The complete journey sequence is summarise in the sequence diagram below.



Summary and Limitations

The system described is very basic and does would does not include many factors from the real world. For example, the system would need developing to consider lane control, indicators, lights and other driving features. In addition, there are other external factors that have not been included such as weather or road condition. However, the adaptability of the object oriented paradigm are such that a more complex system could be developed over time. The nature of the modular approach would also allow a team of developers to collaborate on more complex systems.

References

Reddy, P. P. (2019) Driverless Car: Software Modelling and Design using Python and Tensorflow.

Zhou, Z. Q. & Sun, L. (2019) Metamorphic testing of driverless cars. Commun. ACM 62(3): 61–67. DOI: https://doi.org/10.1145/3241979.

Joque, J. (2016) The Invention of the Object: Object Orientation and the Philosophical Development of Programming Languages. *Philosophy & Technology* (29):335 -356.

Rumbaugh, J., Jacobson, I. & Booch, G. (2005) *The Unified Moddeling Language Reference Manual. Second Edition.* Boston: Addison-Wesley

Phillips D. (2018) *Python 3 Object-Oriented Programming*. Third Edition. Birmingham: Packt Publishing Ltd.

Glinz, M. (2000) 'Problems and deficiencies of UML as a requirements specification language', *Tenth International Workshop on Software Specification and Design.* San Diego, 07 November 2000. San Diego: IEEE. 11-22.

Babaei, P et al. (2023) Perception System Architecture for Self-Driving Vehicles: A Cyber- Physical Systems Framework. Available from: https://www.researchsquare.com/article/rs-3777591/v1 [Accessed 25 April 2024]