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

## Introduction

An autonomous vehicle, a driverless or robot car, can sense its environment and travel with minimal or no human intervention (Reddy, 2019). The benefits of AVs are vast, including improvements in road safety, efficiency, and environmental impact. By reducing human error, AVs could decrease the number of accidents; however, as accidents will still occur, ethical dilemmas must be considered regarding the AV's decision-making (Deema et al., 2023). Moreover, autonomous vehicles are expected to optimize traffic flow and reduce congestion, which could lead to lower emissions and a significant reduction in the ecological footprint (Zewe, 2022).

Autonomous vehicles use sophisticated sensors, including LiDAR, RADAR, and cameras, to retrieve the necessary data about the environment. The data includes obstacle detection, lane recognition, and traffic signal status, which are crucial for navigation and safety. Machine learning algorithms process the data, enabling the vehicle to recognize patterns, make decisions, and react to countless driving scenarios (Gupta et al., 2021). Autonomous vehicles rely on global positioning systems (GPS) and inertial navigation systems to track their position precisely. AVs require software infrastructure to support these complex operations (Reddy, 2019). Zhou and Sun (2019) explore the reliability of these technologies through metamorphic testing. This allows for predicting errors in LiDAR sensors of self-driving cars to avoid potential accidents.

The technological landscape for AVs is rapidly evolving; however, successfully integrating AVs into daily life will require considering societal impact and finding solutions to these technological, legal, infrastructure, and ethical issues (Hancock et al., 2019).

**Driverless Car Design**

The Society of Automotive Engineers (SAE) defines five levels of car automation. This report will focus on the design of a Level 4 car where the vehicle can automatically drive, detect obstacles, and adjust the speed and route based on changing circumstances. However, the driver can still override some of the decisions if he chooses to (Bosch, 2024).

The design will involve creating an object-oriented program in Python that interacts with users through a User Interface and allows data collection and processing. Data structures such as lists and dictionaries will be integrated to manage the flow and storage of data within the system.

Given the complexity of AV software, this report will focus on operations, classes, and methods that can be built with object-oriented programming within the module's scope. It simplifies the design based on the software that will be built for the next assignment.

The paper focuses on designing the following three operations for the Autonomous Vehicle: Obstacle Detection, Navigation and route Planning, and a User Interaction system.

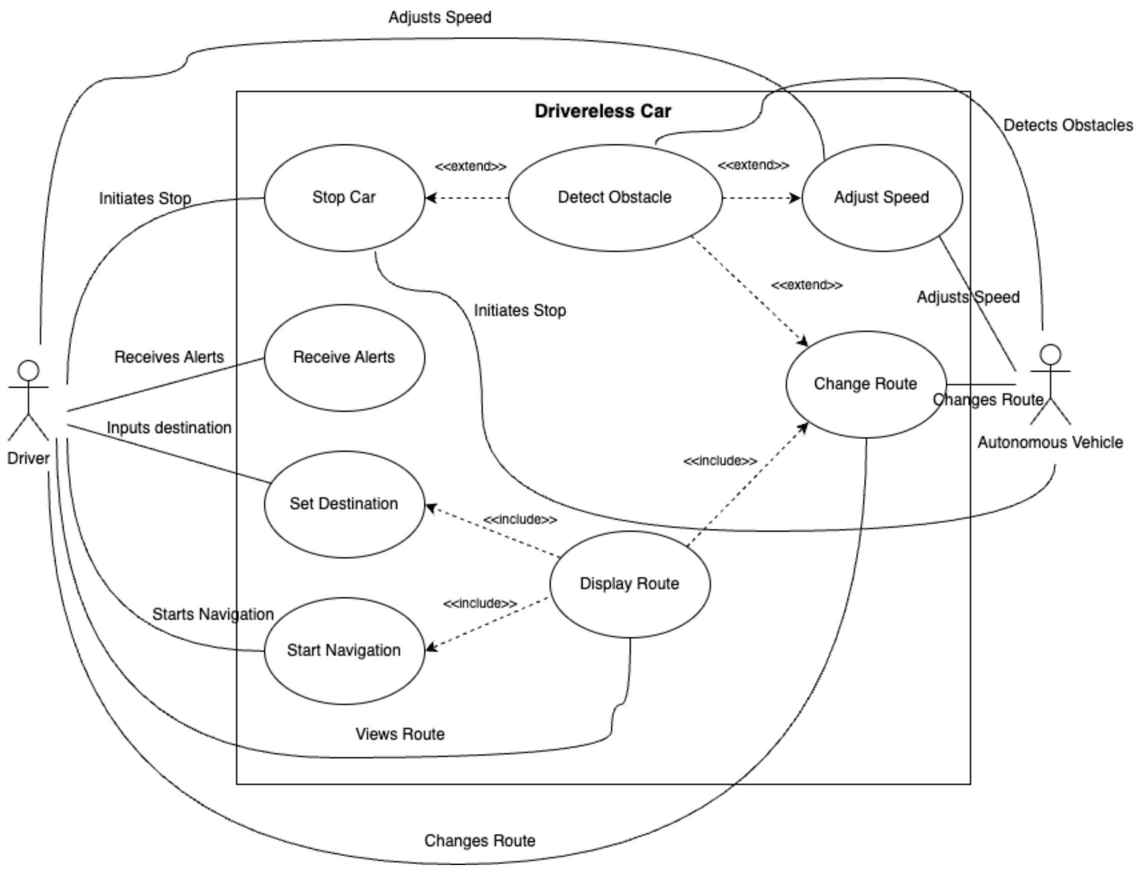
**Obstacle Detection** can demonstrate how self-driving cars process sensor data to avoid collisions. Implementing this allows for encapsulating sensor data processing, obstacle identification, and response strategies into distinct classes. Classes like Sensor, ObstacleDetector, and Obstacle will be created where the Sensor class will simulate data input, the ObstacleDetector will process the data and identify potential obstacles, and the Obstacle will represent the detected objects with attributes like Type and Distance of the object.

**Navigation and Route Planning** are central to an autonomous vehicle's functionality. They include real-time data processing and decision-making, which can be a distinct module in an object-oriented program. Classes such as Map, Route, and Navigator will be used. A map will manage static geographical data, Route will handle the planning of paths, and Navigator will use these components to plot and update the driving path based on obstacle detection.

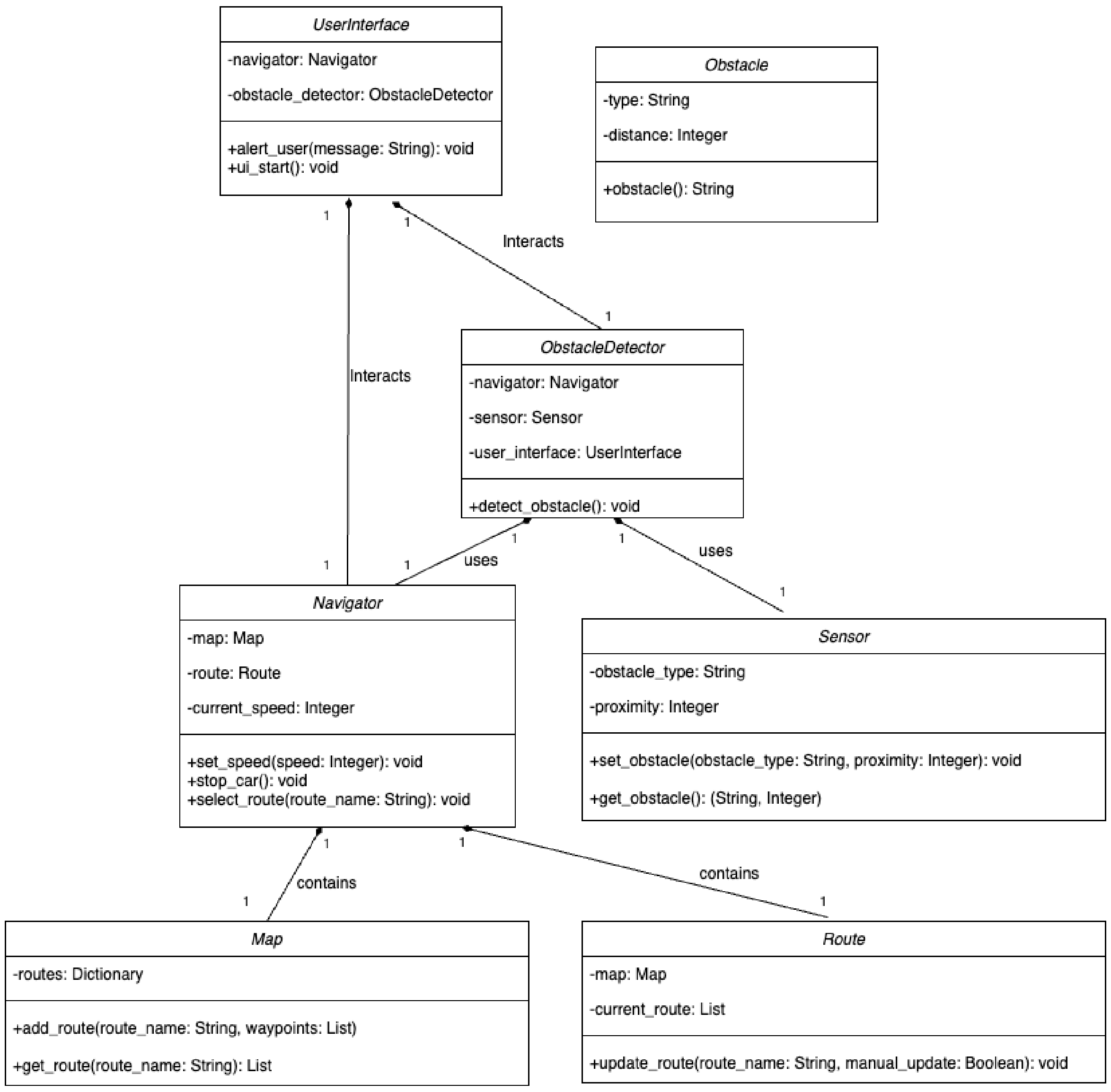
**The User Interaction System** will be used to update the driver on changes in route and speed, take input, and allow the user to set a destination. A User Interface class will be created to manage all user interactions, process the inputs to the system, and give feedback, such as status updates or alerts on obstacles.

Each class under the three operations has a distinct responsibility that aligns with object-oriented design principles such as encapsulation and modularity. Since interacting with real vehicle sensors or maps is beyond the scope, the program will support an interface - to simulate the input data to which the program will react for navigation and obstacle detection.

**Use Case Diagram for all three operations:**



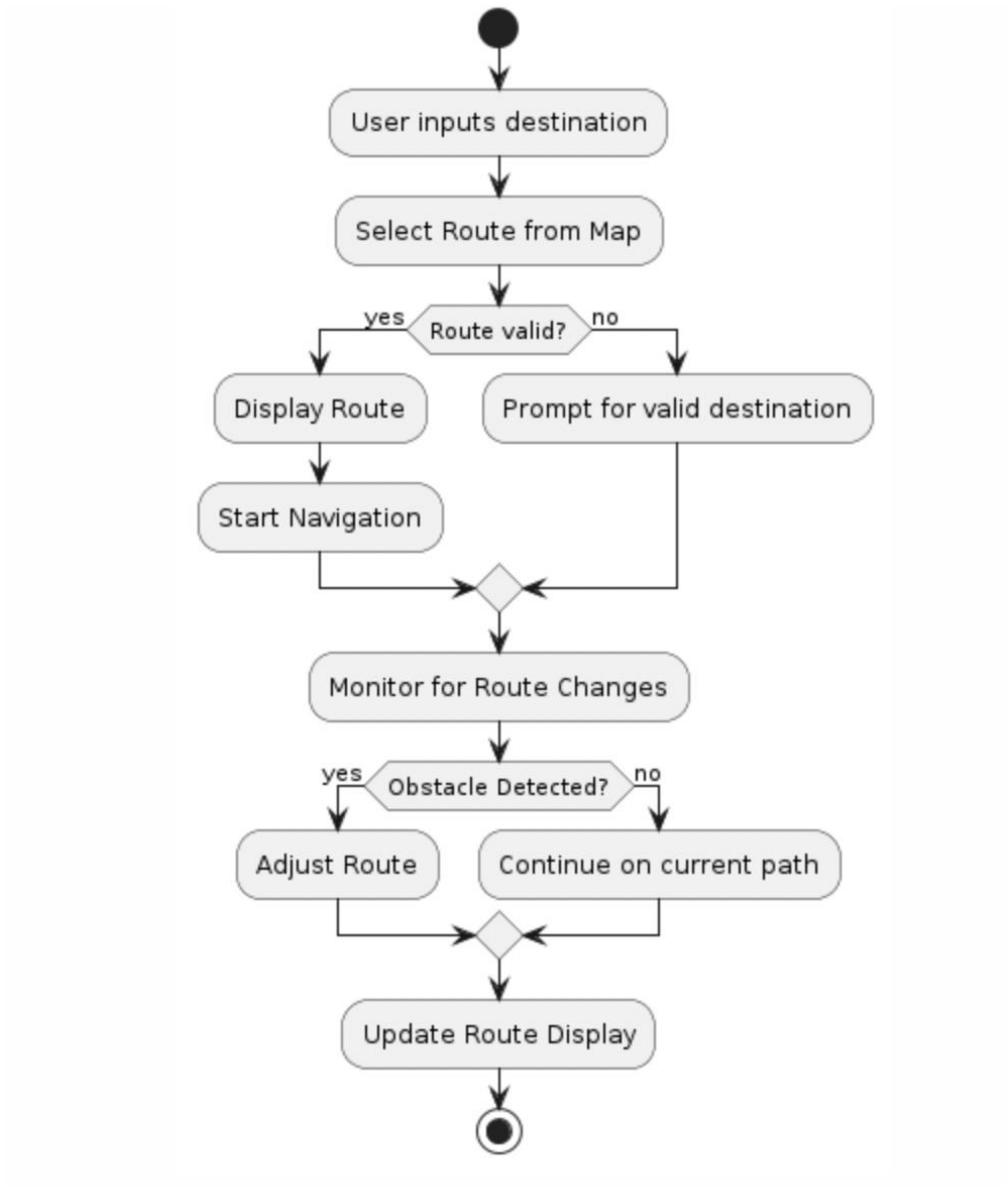
## Class Diagram for all three operations:


UserInterface
-navigator: Navigator
-obstacle_detector: ObstacleDetector
+alert_user(message: String): void
+ui_start(): void
Obstacle
-type: String
-distance: Integer
+obstacle(): String
Interacts
Interacts
ObstacleDetector
-navigator: Navigator
-sensor: Sensor
-user_interface: UserInterface
+detect_obstacle(): void
uses
uses
Navigator
-map: Map
-route: Route
-current_speed: Integer
+set_speed(speed: Integer): void
+stop_car(): void
+select_route(route_name: String): void
Sensor
-obstacle_type: String
-proximity: Integer
+set_obstacle(obstacle_type: String, proximity: Integer): void
+get_obstacle(): (String, Integer)
contains
contains
Map
-routes: Dictionary
+add_route(route_name: String, waypoints: List)
+get_route(route_name: String): List
Route
-map: Map
-current_route: List
+update_route(route_name: String, manual_update: Boolean): void
1

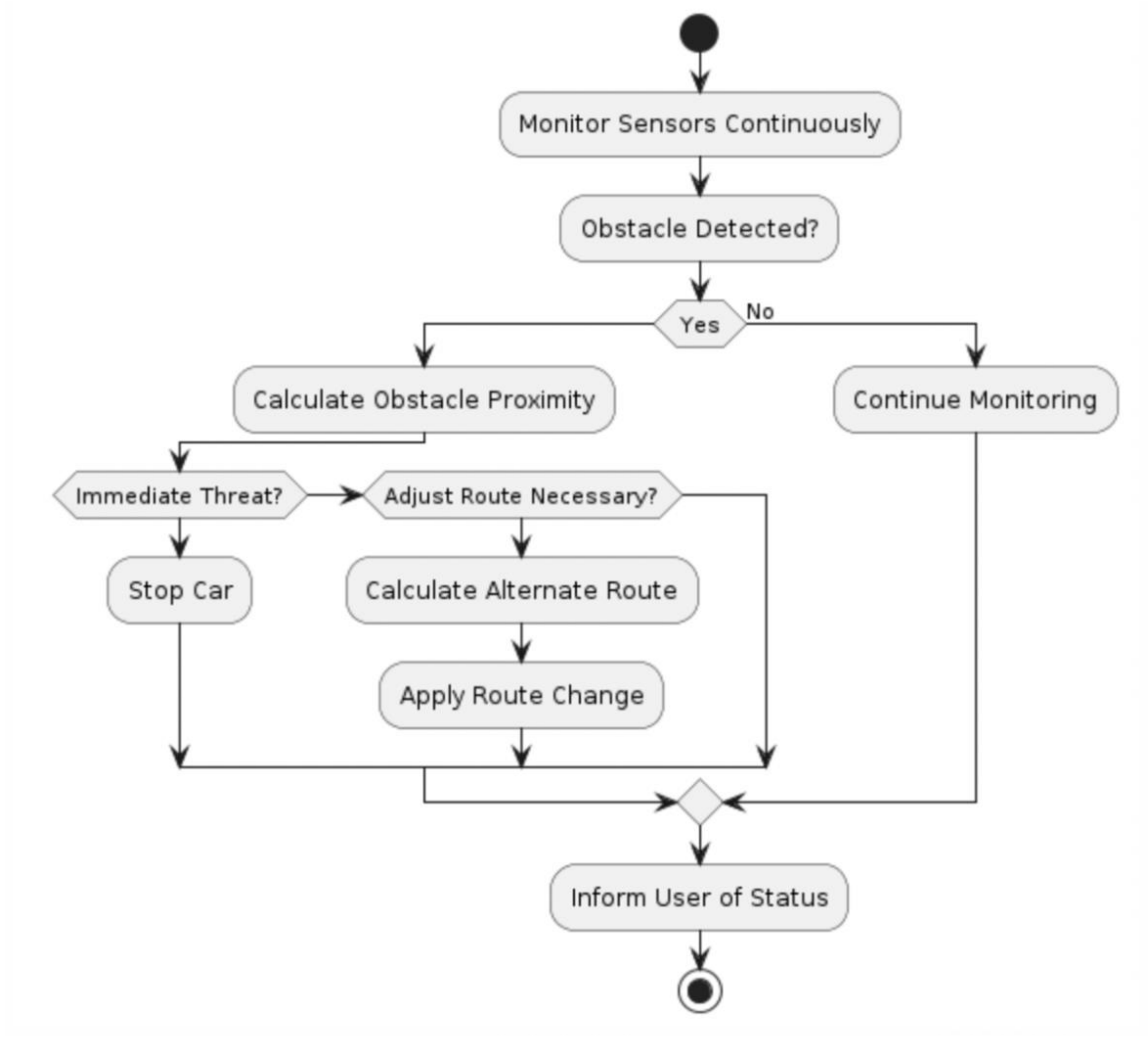
The three operations consist of the above classes, methods, and attributes. While each class has a separate functionality, all classes have interdependencies and are connected within the software. For example, ObstacleDetector can notify the Navigator of obstacles. The Navigator then recalculates the route or adjusts the vehicle's speed based on the type and proximity of the obstacle. The detectObstacle() in the ObstacleDetector class triggers updateRoute() or setSpeed() methods in the Navigator class.

Similarly, the ObstacleDetector interacts with the user interface to alert the user about detected obstacles. The method alertUser() in the UserInterface class is triggered by detectObstacle() in the ObstacleDetector class.

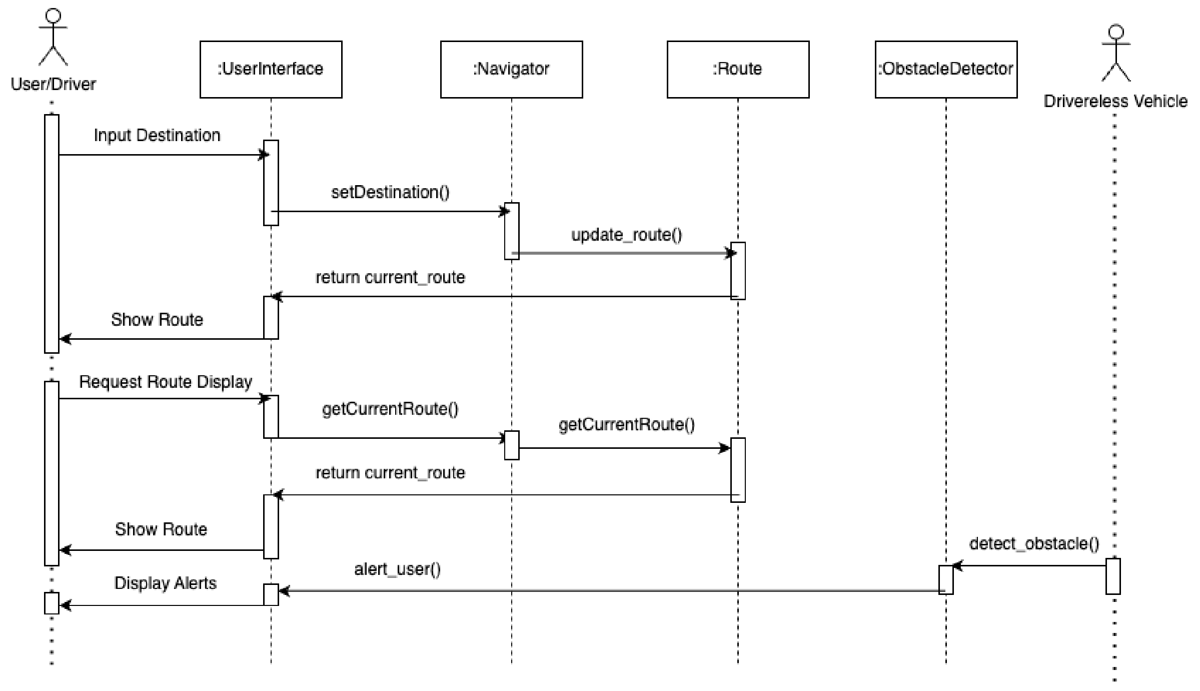
**Activity Diagram for Navigation & Route Planning:**



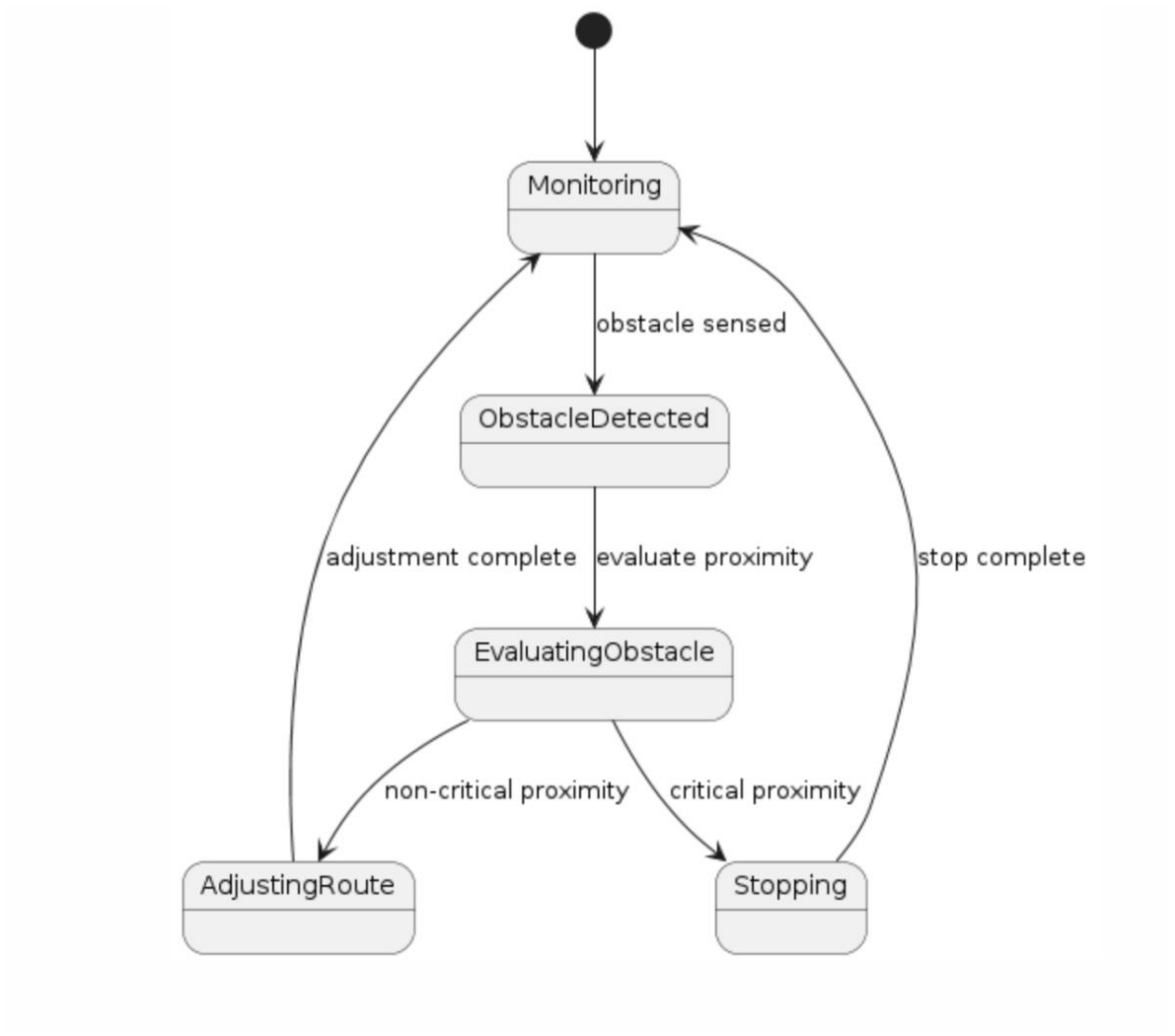
**Activity Diagram for Obstacle Detection:**



## Sequence Diagram for User Interface:



**State Diagram for Obstacle Detection:**



## References:

Almaskati, Deema., Kermanshachi, Sharareh., Pamidimukkula, Apurva. (2023) Autonomous vehicles and traffic accidents. Journal of *Transportation Research Procedia,* 321–328. DOI: https://doi.org/10.1016/j.trpro.2023.11.924

Bosch (2024) The five steps of automated driving. Available from: https://www.bosch-mobility.com/en/mobility-topics/the-five-steps-of-automated-driving/ [Accessed 28 April 2024]

Hancock, P.A., Nourbakhsh, I., & Stewart, J. (2019). On the future of transportation in an era of automated and autonomous vehicles. DOI: https://doi.org/10.1073/pnas.1805770115

Gupta, A., Anpalagan, A., Khwaja, A., & Guan, L. (2021) Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. Journal of *Array.* DOI: https://doi.org/10.1016/j.array.2021.100057

Reddy, P. (2019) Driverless Car: Software Modelling and Design Using Python and Tensorflow. Available at: https://easychair.org/publications/preprint_open/k7wj [Accessed 28 April 2024]

Zewe, A. (2022) On the road to cleaner, greener, and faster driving. Available from: https://news.mit.edu/2022/ai-autonomous-driving-idle-0517 [Accessed 28 April 2024].

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