PG Dip - Computer Science
Module: OOP
Tutor: Dr Nadeem Qazi
Student: Jean-Gaëtan De Souza

Design Document

For a Self-Driving Car

Operations:
The design of this software system of a driverless car focuses around three essential operations: Obstacle Detection and Avoidance, Traffic Sign Recognition, and Automated Parking.

Research Keywords:
The keywords “autonomous vehicle”, “driverless car”, “robotic car”, “radar driverless car”, “GPS driverless car,” “odometry driverless car,” and “ultrasound sensor parking” were selected for the initial research phase of the design of this software, and are an essential part of the technologies that empower the three selected operations. The technologies such as radar, GPS, and odometry are essential for precise positioning and navigation, while ultrasound sensors play a key role in proximity detection (essential for obstacle avoidance and autonomous parking). As I began the design phase of the Obstacle Detection and Avoidance system, I was quickly confronted with the ethical and legal aspect driverless cars. This led me to broaden my research to include key concepts like the "trolley dilemma for autonomous vehicle" and "legal responsibility for autonomous vehicle". These topics are crucial for the design of driverless cars software, especially when the vehicle confronts scenarios where collisions are imminent and inevitable, highlighting these ethical and legal challenges.

Rationale of key elements:

Class: ObstacleDetector

- Attributes:
  detectedObjects: Stack
  objectType: string
  objectID: int
  isAvoidable: boolean
  fragilityIndex: float
  avoidancePriority: Queue

- Operations:
  gatherData()
  processData()
  setIsAvoidable()
  getIsAvoidable()
  setObjectType()
  getObjectType()
  setObjectID()
  getObjectID()
  setFragilityIndex()
  getFragilityIndex()
  addToAvoidancePriority()
  getAvoidancePriority()

avoidObstacle()

The obstacleDetector class detects and manages objects and obstacles the vehicle encounters. In the context of the software, we consider that an object becomes an obstacle when it is in the vehicle's path. It uses a stack (in LIFO order for quick re-evaluation) to keep track of detected objects, allowing for quick updates as objects come into the sensor range. The queue for avoidance priority helps in determining the order in which obstacles should be avoided. A boolean defines if the obstacle is avoidable. If not, the program defines a fragility index of the main obstacle, and defines the fragility index of surrounding objects in order to calculate if swerving the vehicle and hitting other objects would reduce the harm (for example, hitting a public bench to avoid hitting a pedestrian). The methods seen in this class, aligns with the ethical and decision-making considerations discussed in Schäffner's (2020) study on moral decision-making in self-driving car dilemmas.

Class: TrafficSignRecognizer

- Attributes:
  recognizedSigns: Queue
  isVehicleCompliant: Boolean

- Operations:
  recognizeSigns(sensorData)
  interpretSigns()
  adaptVehicleBehaviour()

The TrafficSignRecogniszer class handles the interpretation of traffic signs and signals, ensuring compliance with traffic laws. The queue holds recognized signs for sequential processing, and a boolean indicates compliance status of the vehicle. Traffic Sign Recognition (TSR) systems can reach an accuracy rate of up to 99.91% (Triki, Karray and Ksantini, 2023).

Class: ParkingAssistant

- Attributes:
  parkingSpace: AvailableParking
  parkingRequested: Boolean

- Operations:
  requestParking()
  findSpace()
  parkVehicle()

The ParkingAssistant class manages the parking operations for the vehicle. It utilizes a boolean to track if parking is requested by the user and launches the parking search as well as the necessary manoeuvres to park the vehicle if a parking space is found.

Class: AvailableParking

- Attributes:
  parkingType: string
  depth: float
  width: float
  isAccessible: boolean

- Operations:
  availableParking(parking Type: string, depth: float, width: float, isAccessible: boolean)
  getParkingType(): string
  setParkingType(parkingType: string)
  getDepth(): float
  setDepth(depth: float)
  getWidth(): float
  setWidth(width: float)
  setIsAccessible(isAccessible: boolean)

Rationale: This class represents the potential parking spot available for the vehicle. Attributes include the dimensions and accessibility of the parking space, ensuring that the vehicle can determine if it can fit and comply with any accessibility requirements. The Improved “YOLOV5-OBB” algorithm described by Chen et al. (2023), introduces an enhanced algorithm for detecting parking spaces. By encapsulating the properties of a detected parking spot such as type, dimensions, and accessibility, this class is suited for this algorithm.

Class: SensorManager

- Attributes:
  sensorsData: Queue

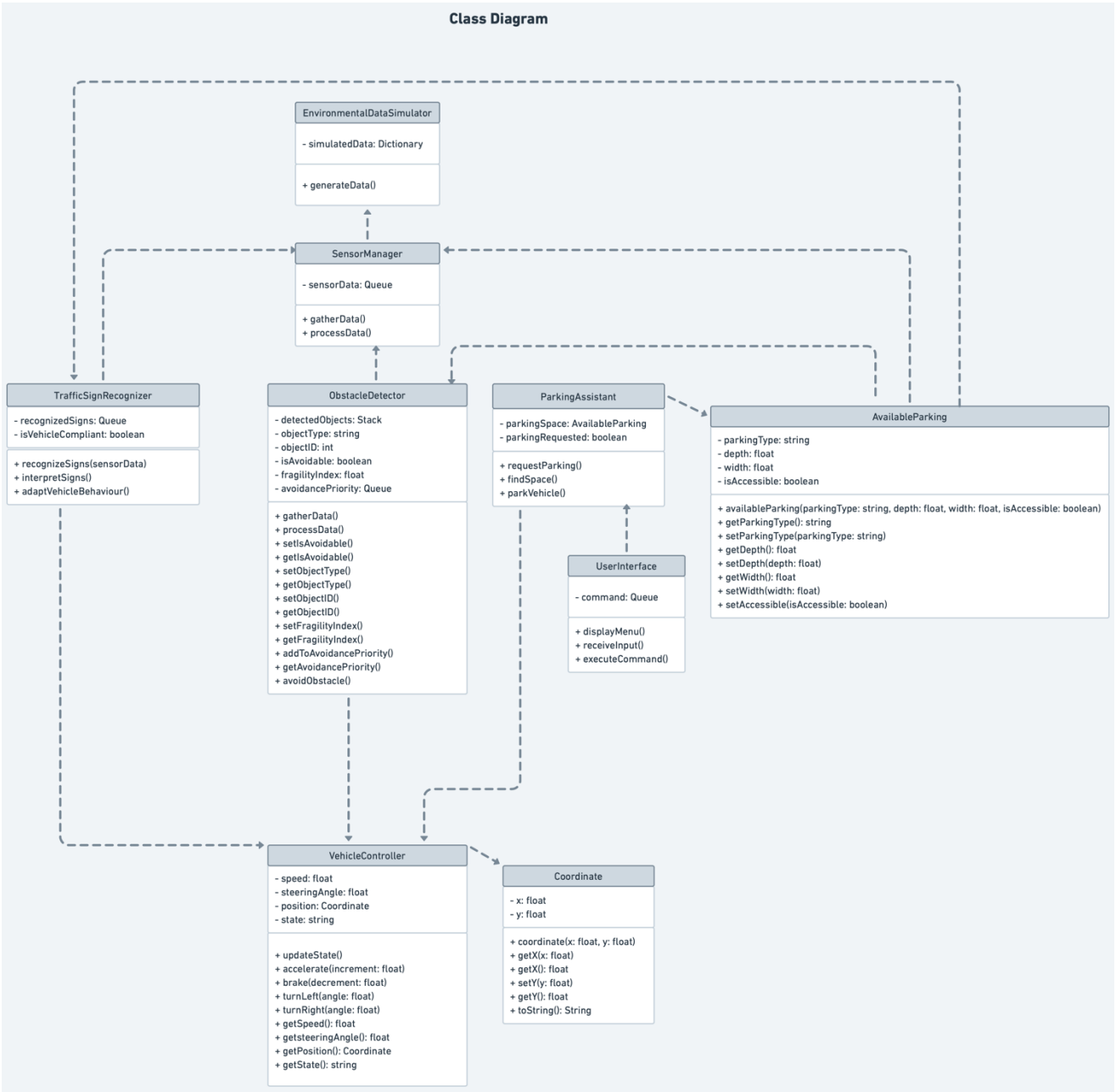
- Operations:
  gatherData()
  processData()

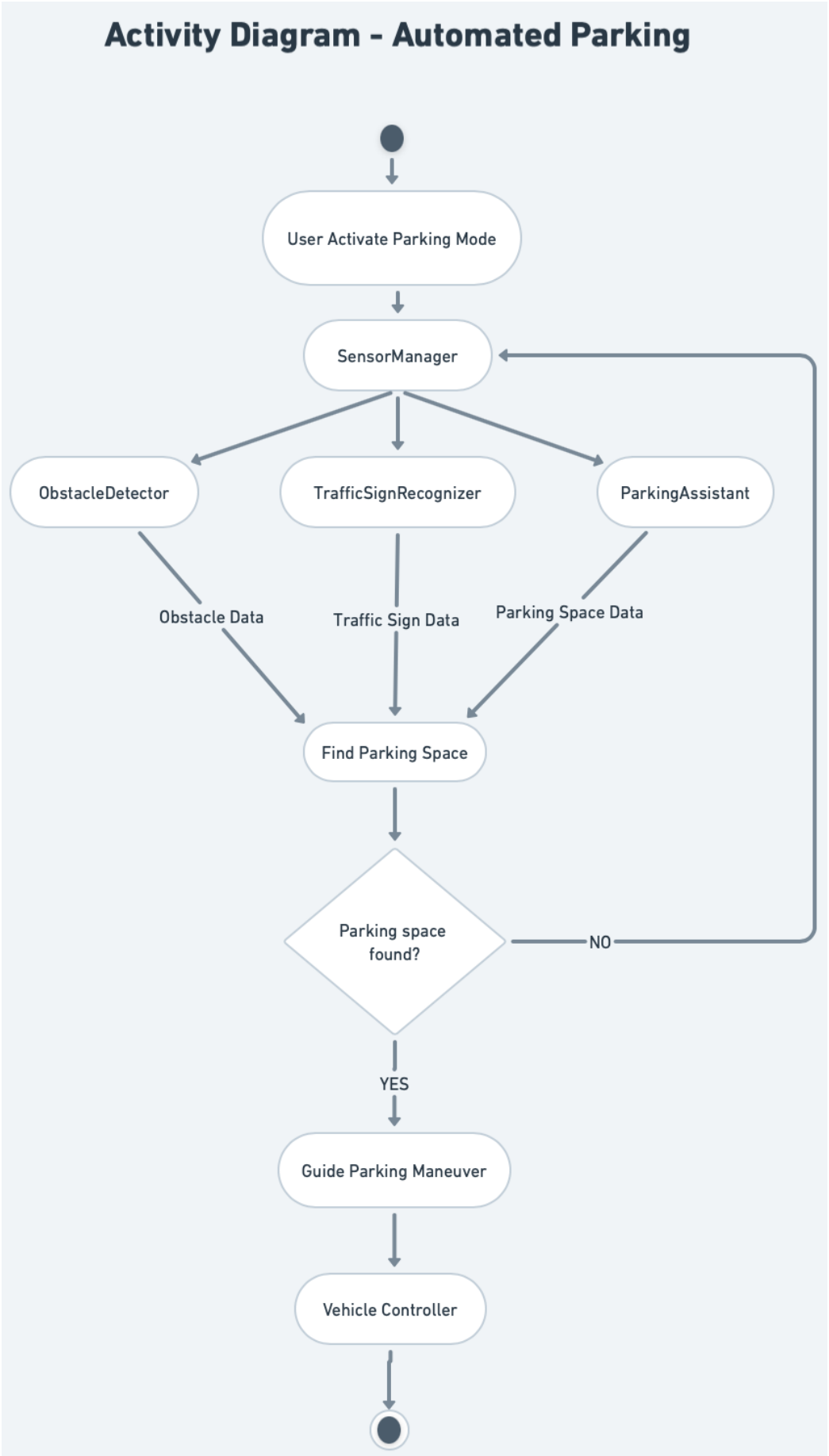
The SensorManager class Manages the data collected from sensors. Using a queue (First In, First Out) ensures that data is processed in the order it's received, which is crucial for real-time applications like autonomous driving.

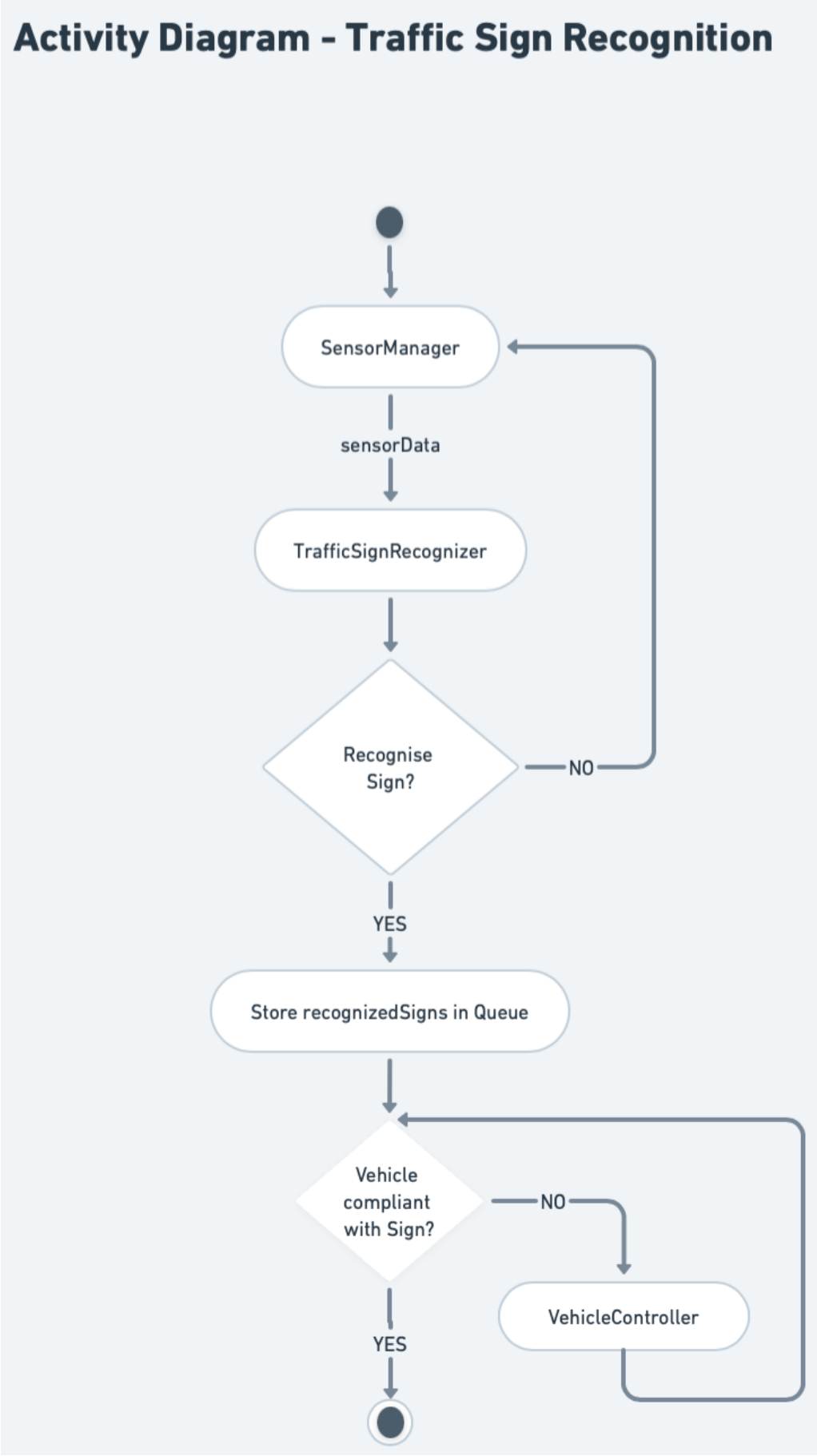
Class: EnvironmentalDataSimulator

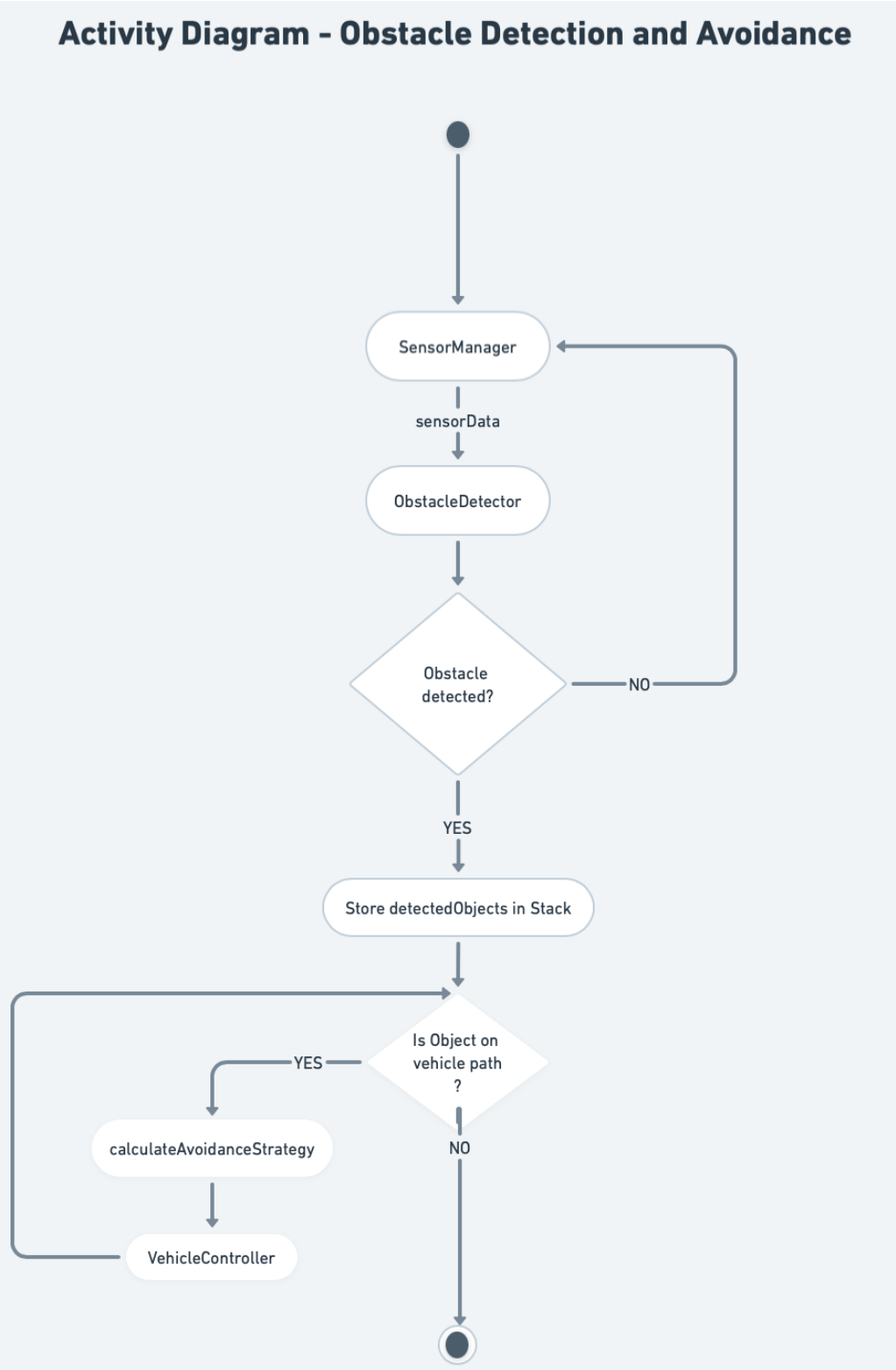
- Attributes:
  simulateData: Dictionary
- Operations:
  generateData()

The EnvironmentalDataSimulator class is responsible for simulating environmental data that the driverless car might encounter, for test purposes, which is crucial in the context of driverless cars, as highlighted by Zhou and Sun (2019). This class also facilitates the implementation of the “safety score” as proposed by Zhao et al. (2020), by generating simulated environmental data and ensuring the autonomous vehicle's computing system can make safe, timely decisions in varied real-world scenarios.

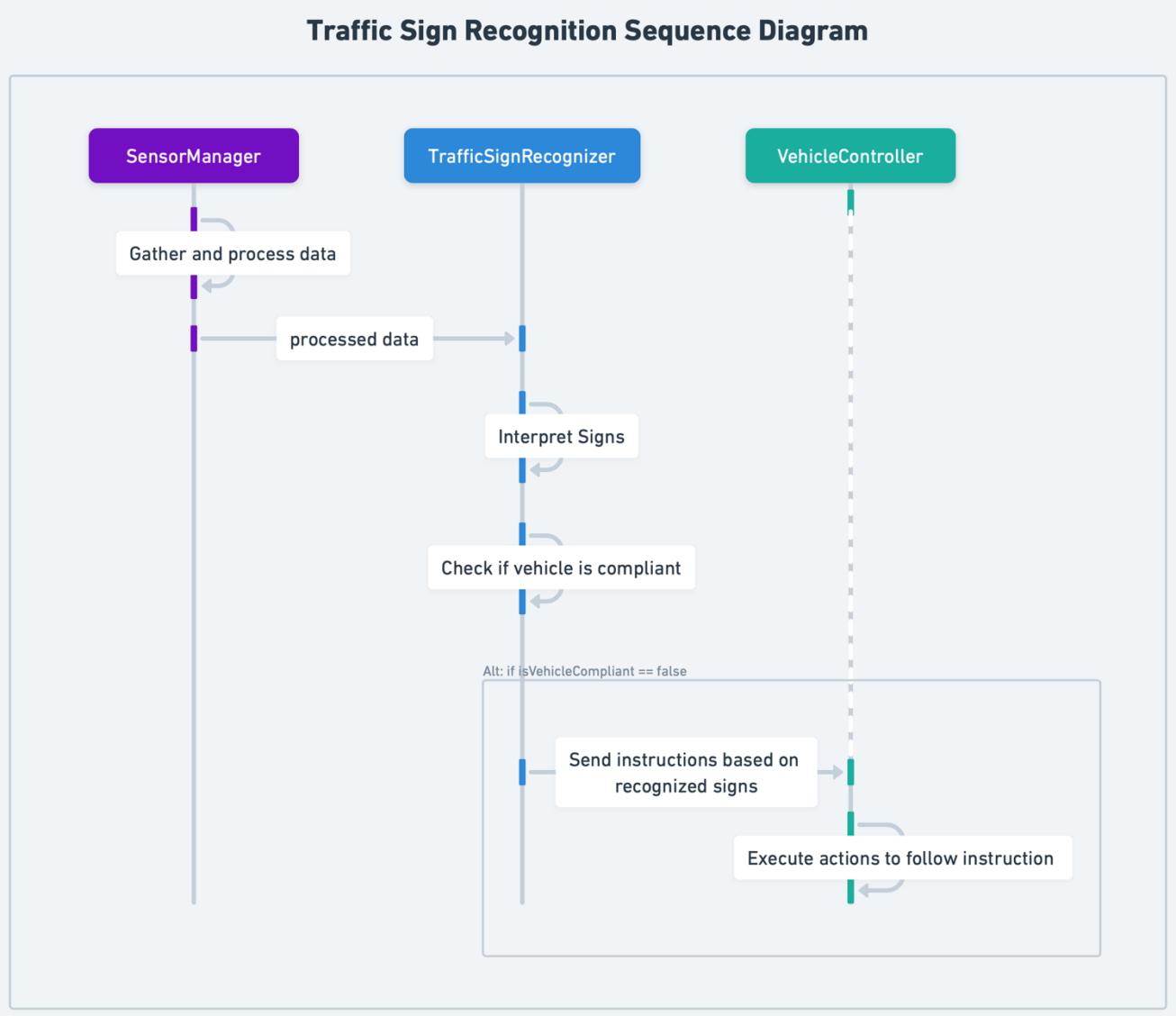

Class Diagram
EnvironmentalDataSimulator
- simulatedData: Dictionary
+ generateData()
SensorManager
- sensorData: Queue
+ gatherData()
+ processData()
TrafficSignRecognizer
- recognizedSigns: Queue
- isVehicleCompliant: boolean
+ recognizeSigns(sensorData)
+ interpretSigns()
+ adaptVehicleBehaviour()
ObstacleDetector
- detectedObjects: Stack
- objectType: string
- objectID: int
- isAvoidable: boolean
- fragilityIndex: float
- avoidancePriority: Queue
+ gatherData()
+ processData()
+ setIsAvoidable()
+ getIsAvoidable()
+ setObjectType()
+ getObjectType()
+ setObjectID()
+ getObjectID()
+ setFragilityIndex()
+ getFragilityIndex()
+ addToAvoidancePriority()
+ getAvoidancePriority()
+ avoidObstacle()
ParkingAssistant
- parkingSpace: AvailableParking
- parkingRequested: boolean
+ requestParking()
+ findSpace()
+ parkVehicle()
AvailableParking
- parkingType: string
- depth: float
- width: float
- isAccessible: boolean
+ availableParking(parkingType: string, depth: float, width: float, isAccessible: boolean)
+ getParkingType(): string
+ setParkingType(parkingType: string)
+ getDepth(): float
+ setDepth(depth: float)
+ getWidth(): float
+ setWidth(width: float)
+ setAccessible(isAccessible: boolean)
UserInterface
- command: Queue
+ displayMenu()
+ receiveInput()
+ executeCommand()
VehicleController
- speed: float
- steeringAngle: float
- position: Coordinate
- state: string
+ updateState()
+ accelerate(increment: float)
+ brake(decrement: float)
+ turnLeft(angle: float)
+ turnRight(angle: float)
+ getSpeed(): float
+ getsteeringAngle(): float
+ getPosition(): Coordinate
+ getState(): string
Coordinate
- x: float
- y: float
+ coordinate(x: float, y: float)
+ getX(x: float)
+ getX(): float
+ setY(y: float)
+ getY(): float
+ toString(): String

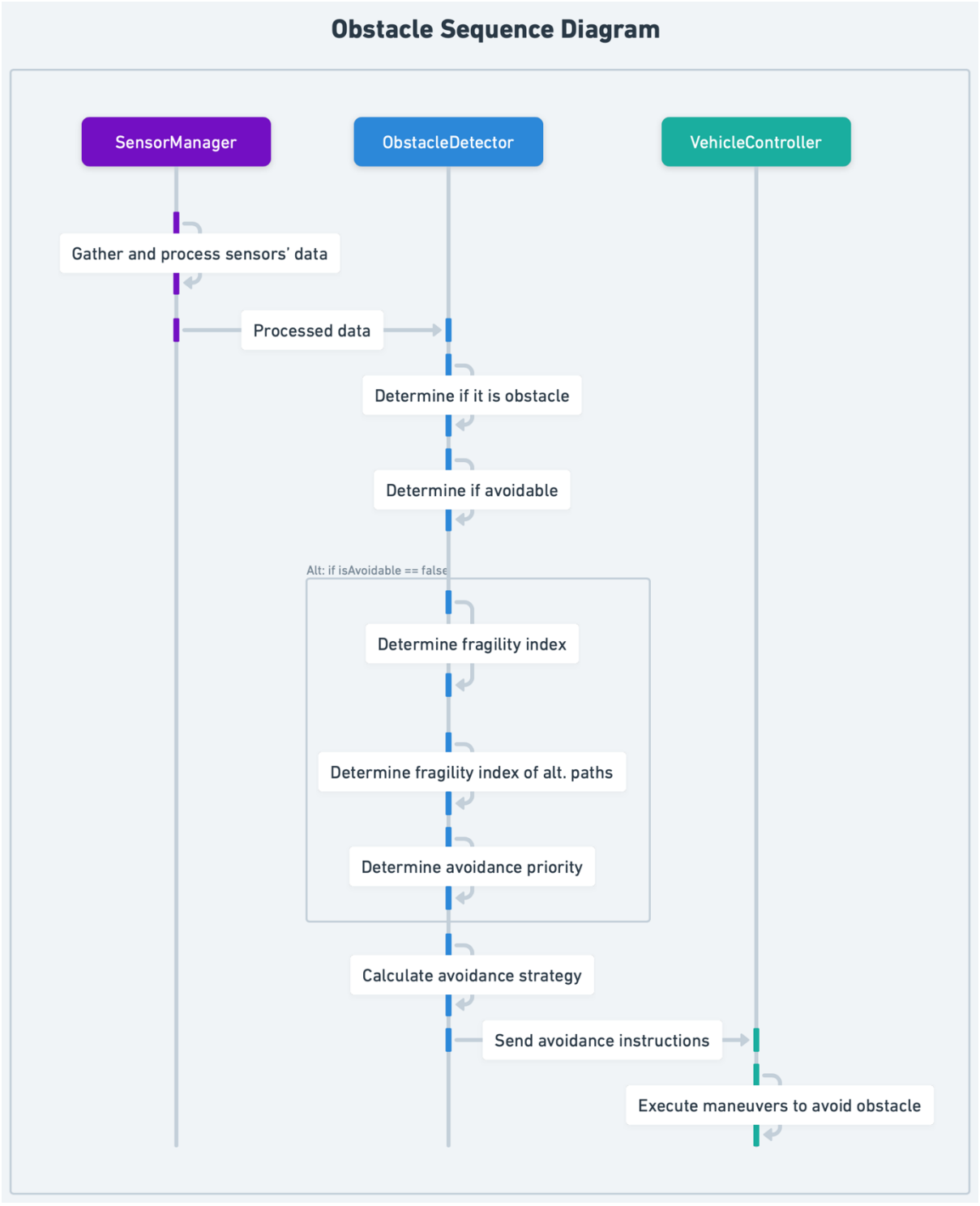

Activity Diagram - Automated Parking
User Activate Parking Mode
SensorManager
ObstacleDetector
TrafficSignRecognizer
ParkingAssistant
Obstacle Data
Traffic Sign Data
Parking Space Data
Find Parking Space
Parking space found?
NO
YES
Guide Parking Maneuver
Vehicle Controller

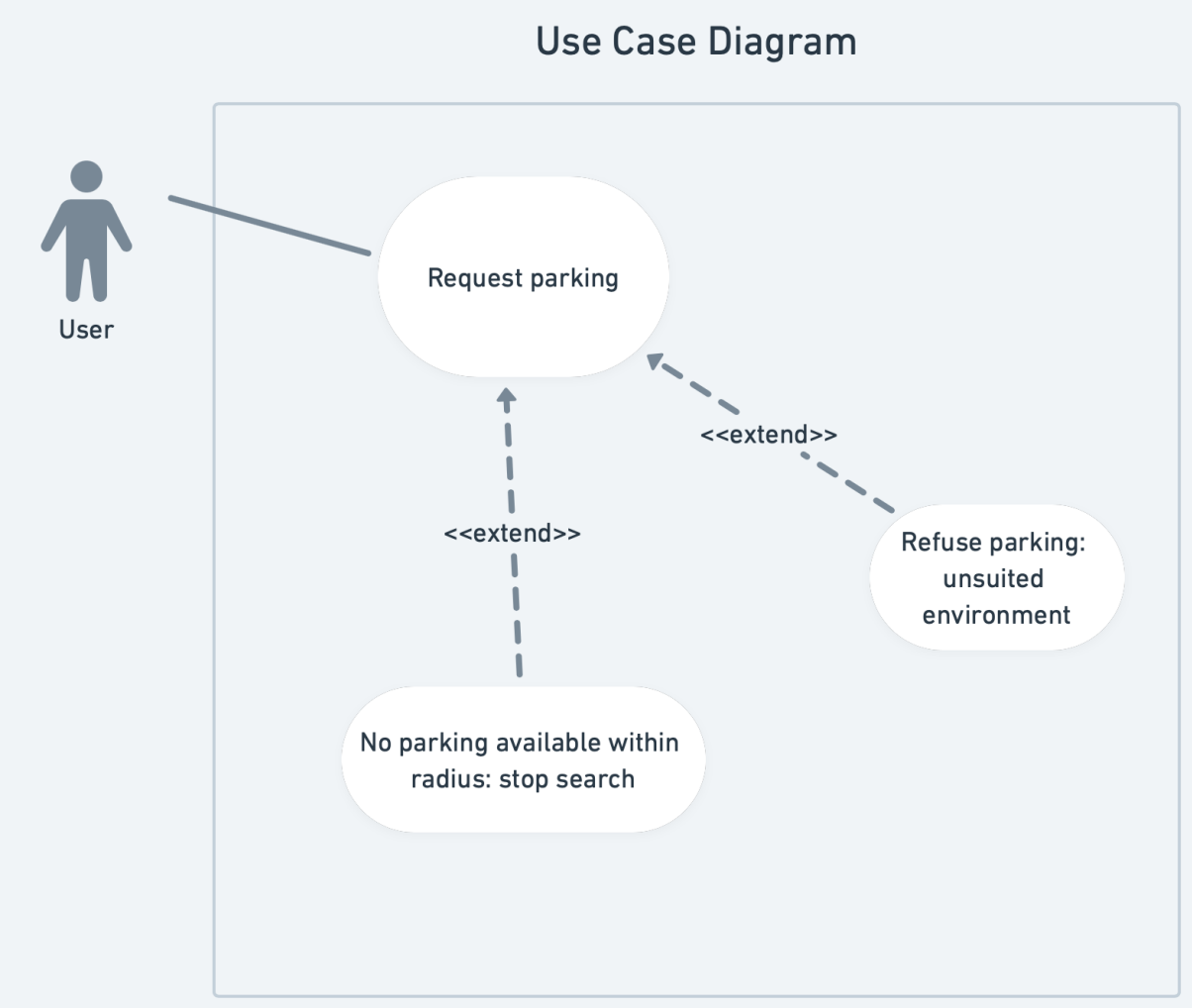

Activity Diagram - Traffic Sign Recognition
SensorManager
sensorData
TrafficSignRecognizer
Recognise Sign?
NO
YES
Store recognizedSigns in Queue
Vehicle compliant with Sign?
NO
VehicleController
YES

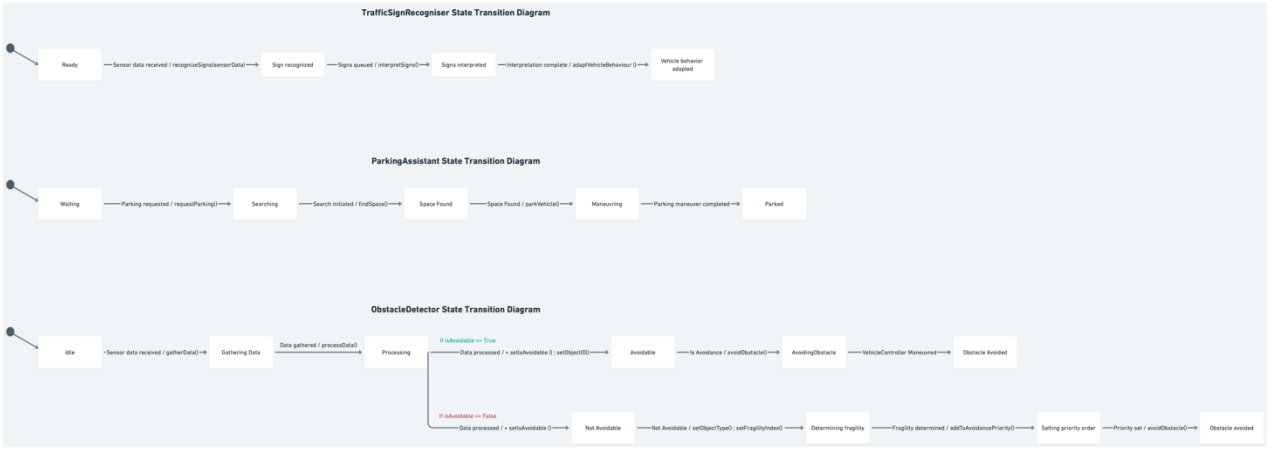

Activity Diagram - Obstacle Detection and Avoidance
SensorManager
sensorData
ObstacleDetector
Obstacle detected?
NO
YES
Store detectedObjects in Stack
Is Object on vehicle path ?
YES
NO
calculateAvoidanceStrategy
VehicleController


Parking Sequence Diagram
UserInterface
ParkingAssistant
SensorManager
ObstacleDetector
TrafficSignRecognizer
AvailableParking
VehicleController
Request parking
Find parking spot
Send sensor data
Send sensor data
Send sensor data
Feedback on obstacles
Relevant signs
Identify suitable spot
Instruct to park car
Complete parking


Traffic Sign Recognition Sequence Diagram
SensorManager
TrafficSignRecognizer
VehicleController
Gather and process data
processed data
Interpret Signs
Check if vehicle is compliant
Alt: if isVehicleCompliant == false
Send instructions based on recognized signs
Execute actions to follow instruction


Obstacle Sequence Diagram
SensorManager
ObstacleDetector
VehicleController
Gather and process sensors' data
Processed data
Determine if it is obstacle
Determine if avoidable
Alt: if isAvoidable == false
Determine fragility index
Determine fragility index of alt. paths
Determine avoidance priority
Calculate avoidance strategy
Send avoidance instructions
Execute maneuvers to avoid obstacle


Use Case Diagram
User
Request parking
<<extend>>
<<extend>>
Refuse parking: unsuited environment
No parking available within radius: stop search


TrafficSignRecogniser State Transition Diagram
Ready
Sensor data received / recognizeSigns(sensorData)
Sign recognized
Signs queued / interpretSigns()
Signs interpreted
interpretation complete / adaptVehicleBehaviour ()
Vehicle behavior adapted
ParkingAssistant State Transition Diagram
Waiting
Parking requested / requestParking()
Searching
Search initiated / findSpace()
Space Found
Space Found / parkVehicle()
Maneuvring
Parking maneuver completed
Parked
ObstacleDetector State Transition Diagram
Idle
Sensor data received / gatherData()
Gathering Data
Data gathered / processData()
Processing
If isAvoidable == True
Data processed / + setIsAvoidable () ; setObjectID()
Avoidable
Is Avoidance / avoidObstacle()
AvoidingObstacle
VehicleController Maneuvred
Obstacle Avoided
If isAvoidable == False
Data processed / + setIsAvoidable ()
Not Avoidable
Not Avoidable / setObjectType() ; setFragilityIndex()
Determining fragility
Fragility determined / addToAvoidancePriority()
Setting priority order
Priority set / avoidObstacle()
Obstacle avoided

Reference list

Chen, Z., Wang, X., Wei Wei Zhang, Yao, G., Li, D. and Zeng, L. (2023). Autonomous Parking Space Detection for Electric Vehicles Based on Improved YOLOV5-OBB Algorithm. *World Electric Vehicle Journal*, 14(10), pp.276–276. doi:https://doi.org/10.3390/wevj14100276.

Reddy, P. (2019). *EasyChair Preprint Driverless Car : Software Modelling and Design Using Python and Tensorflow*. [online] Available at: https://easychair.org/publications/preprint_open/k7wj.

Schäffner, V. (2020). Between Real World and Thought Experiment: Framing Moral Decision-Making in Self-Driving Car Dilemmas. *Humanistic Management Journal*. doi:https://doi.org/10.1007/s41463-020-00101-x.

Triki, N., Karray, M. and Ksantini, M. (2023). A Real-Time Traffic Sign Recognition Method Using a New Attention-Based Deep Convolutional Neural Network for Smart Vehicles. *Applied Sciences*, 13(8), p.4793. doi:https://doi.org/10.3390/app13084793.

Zhao, H., Zhang, Y., Meng, P., Shi, H., Li, L.E., Lou, T. and Zhao, J. (2020). Safety Score: a Quantitative Approach to Guiding Safety-Aware Autonomous Vehicle Computing System Design. *IEEE*. doi:https://doi.org/10.1109/iv47402.2020.9304602.

Zhou, Z.Q. and Sun, L. (2019). Metamorphic Testing of Driverless Cars. *Communications of the ACM*, 62(3), pp.61–67. doi:https://doi.org/10.1145/3241979.