Assignment title	**Assignment 1: Software Design Proposal for Driverless Car**
Discussion	System Architecture
Student	Miguel Maide Bezares (ID 12694153)
Module	OOP_PCOM7E November 2023
Due date	15/01/2023

Glossary of Terms and Acronyms

GNSS	Global Navigation Satellite System
V2X	Vehicle to everything
GUI	Graphical user interface
OOP	Object Oriented Programming

Introduction

The idea of autonomous vehicles, especially cars, has been gaining strength and interest over the last decade. This paper will propose a simplified system architecture that defines interrelation of different modules that work together achieving a self-driving experience in Python.

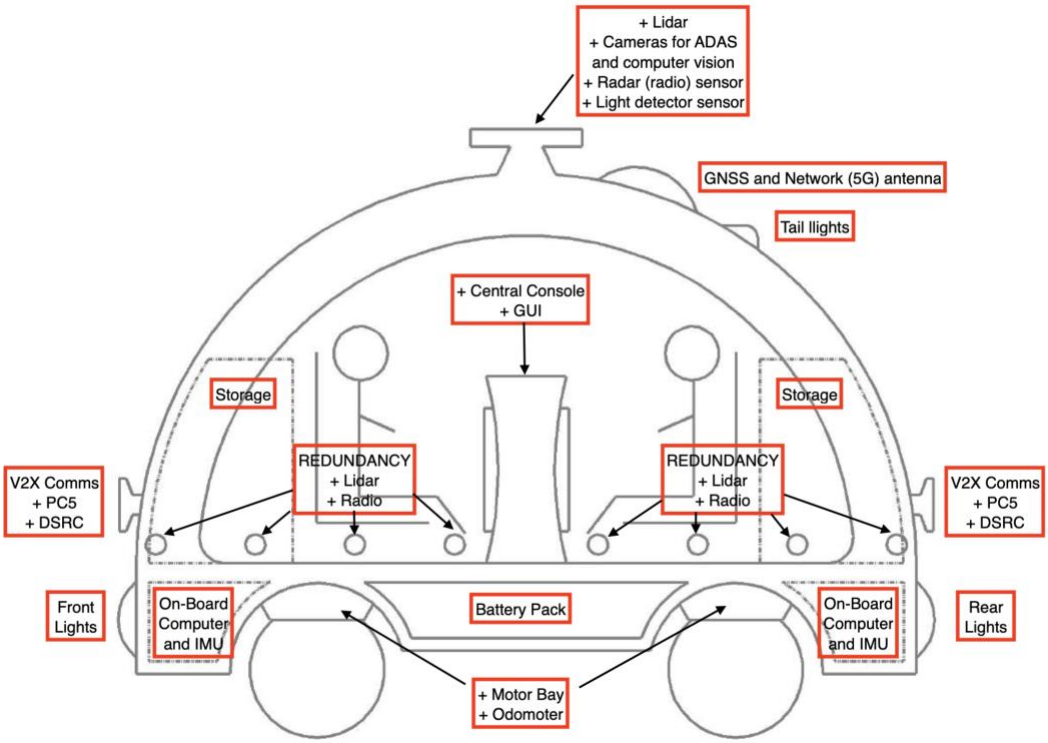


Figure 1: System Overview

Proposed solution

As a simplified approach (like the one described in Reddy, 2019), vehicle was equipped with a set of cameras, obstacle detection technology (Radar and Lidar), antennas for communication (GNSS and 5G) and revolutionary V2X technology using PC5 and DSRC antennas (Noor-A-Rahim et al., 2022). TE (2018), clearly raises the need for this technology to be supported on 5G or better connectivity, due to its low latency and high transmission speeds.

Use Case Diagram

Despite the vehicle being fully autonomous, it still requires of interaction with external bodies. These can either trigger vehicle functionalities (left side) or support them (right side):

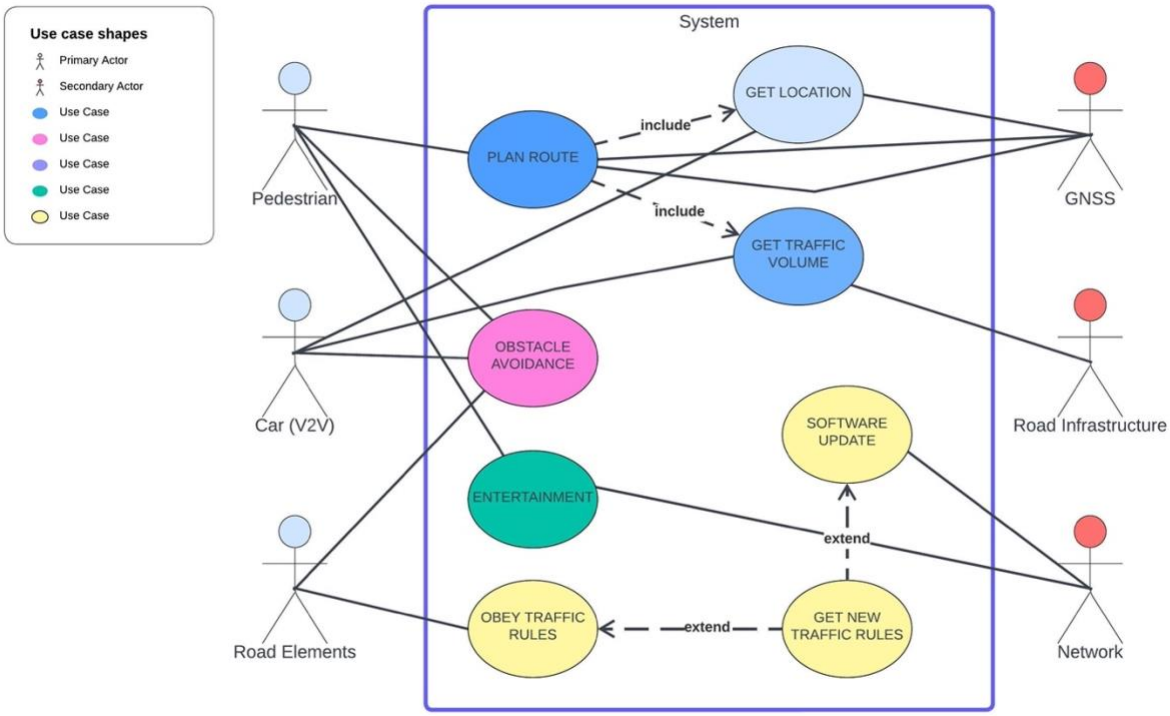


Figure 2: Use Case Diagram

Class Diagram

The class diagram organises the working modules rather efficiently, as well as the interactions and relations between them:

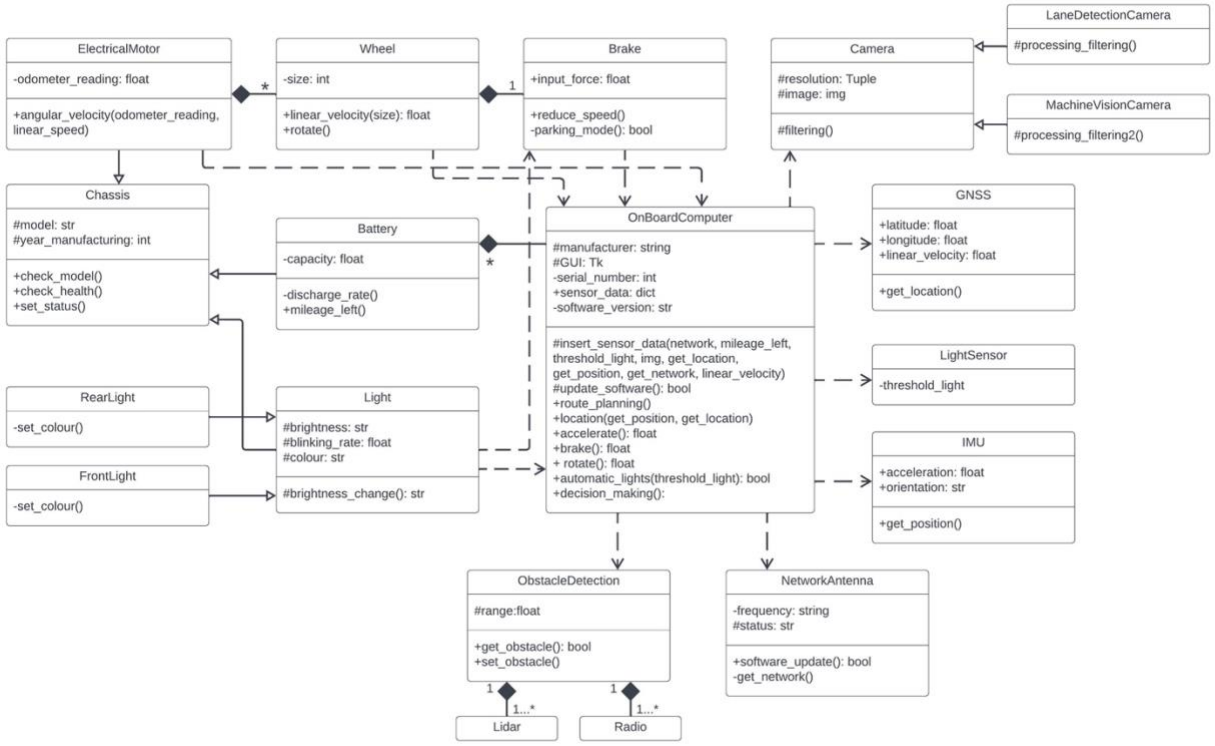


Figure 3: Class Diagram

All the dependency relations (dashed arrows) are <<use>> types of dependency as much of the data is calculated by class functions and stored in dictionaries used by the OnBoardComputer. Here a simple method to relate for instance the OnBoardComputer and the LightSensor classes:

```
class LightSensor:
    def __init__(self):
        self.data = dict(light_threshold=1, condition="good")

    @property  # Getter method
    def value(self):
        return self.data


class OnBoardComp:
    def __init__(self, light_sensor):
        self.to_process = light_sensor

    def processor(self, key):
        data = self.to_process.value
        result = data[key]  # Use the provided key to access the value in the
dictionary
        return result
```

Here, the computer object from the OnBoardComp class, is initialized using an object with all its attributes from a dependent class. This basically equates the attribute “to_process” to all the

attributes of the sensor (in this case only 1 which is data). However to access this attribute, as if it was the path to a file, you search for the attribute data that was inherited, either with self.to_process.value or self.to_process.data. Then you look for the key and assign it to processor. With “processor” assigned, time to use the property within your class (Lott and Phillips, 2021).

Activity Diagram

These four diagrams define simple processes that the vehicle follows to complete some of its tasks.

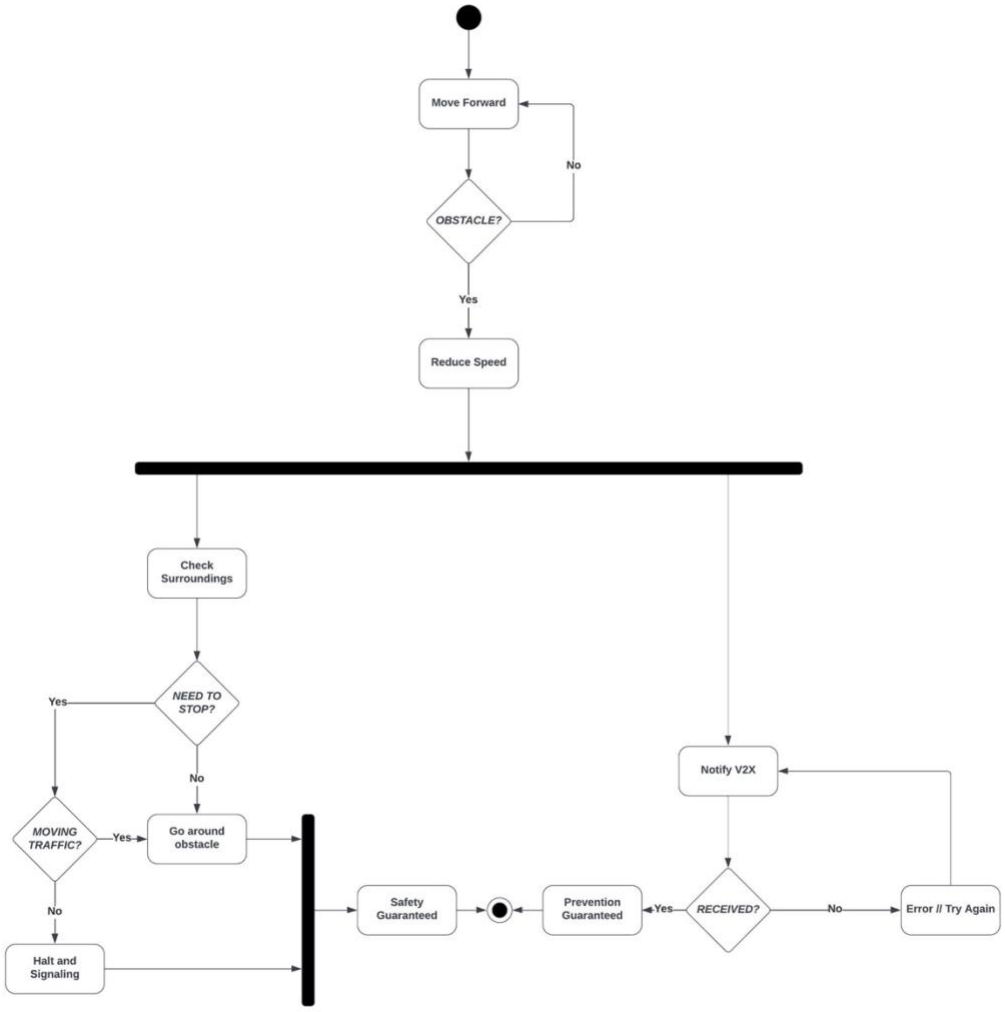


Figure 4: Obstacle Detection

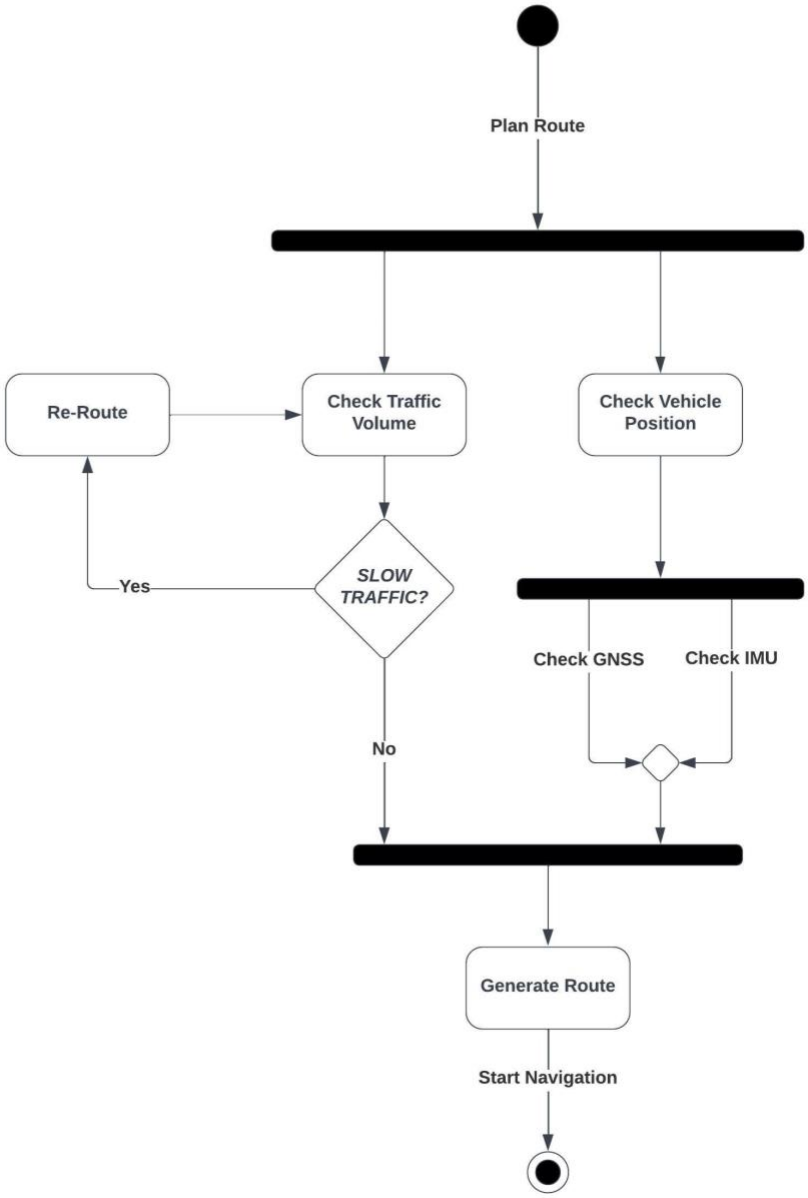


Figure 5: Route Planning

Note in figure 6, many of the machine learning steps and image processing has not been considered for fluency purposes. However, Yang(2023) reinforces the need for better image filtering, left behind for future implementation.

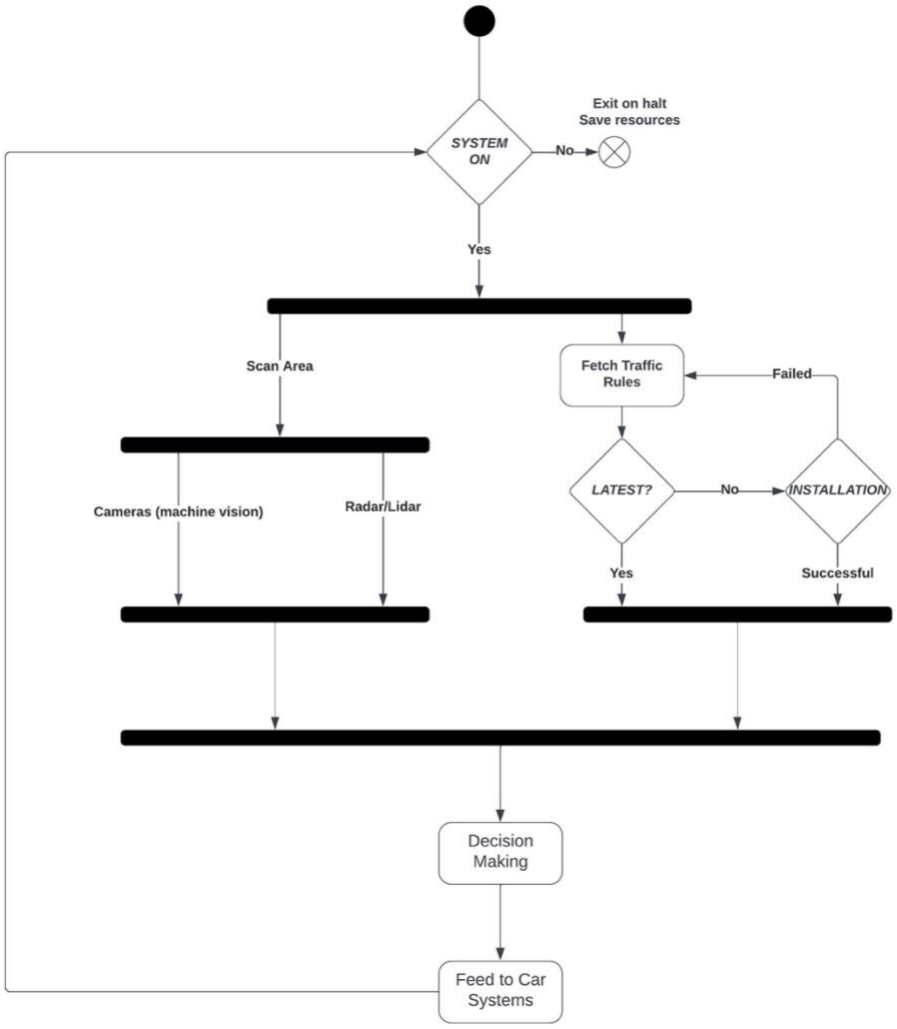


Figure 6: Traffic Rules Simple Algorithm

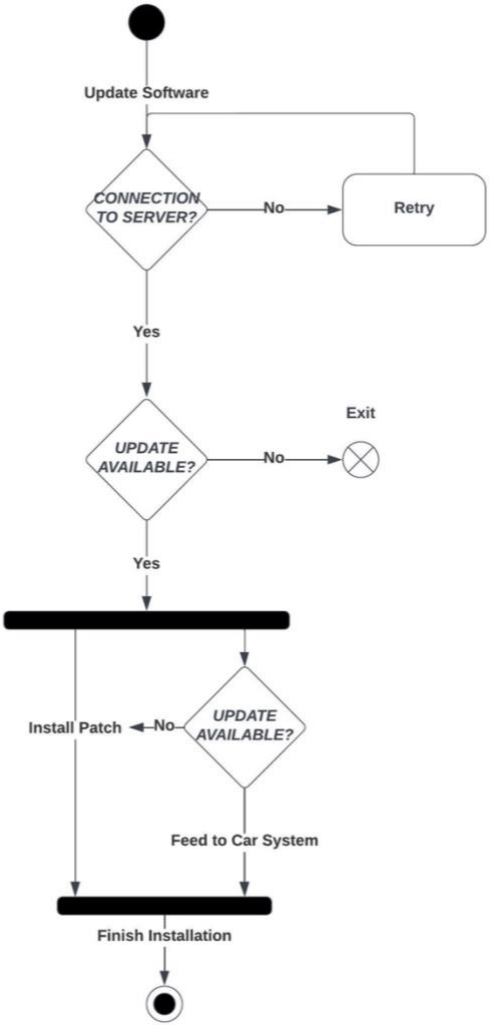


Figure 7: Software Update

Sequence Diagram

This simplified diagram represents a series of feedback loops during the vehicle's operation over time for obstacle avoidance, decision making and parking mode.

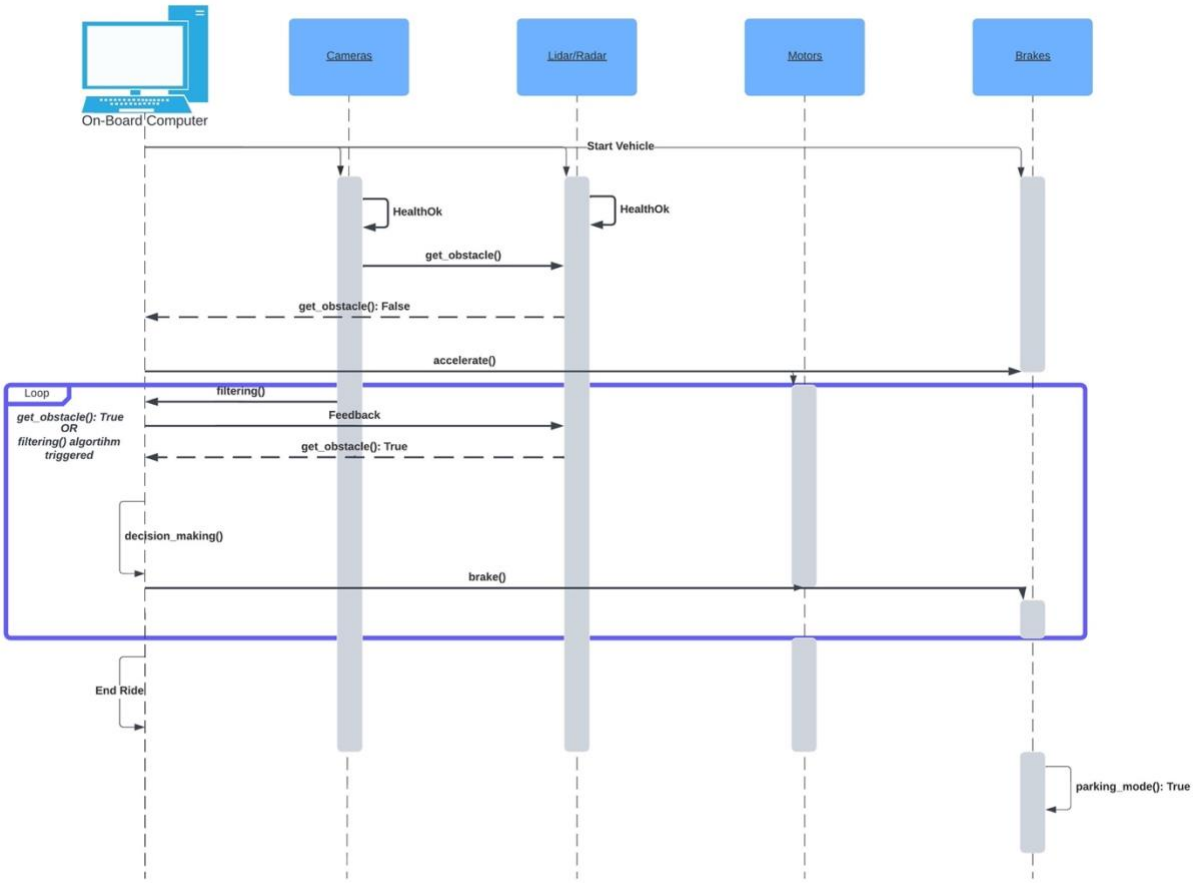
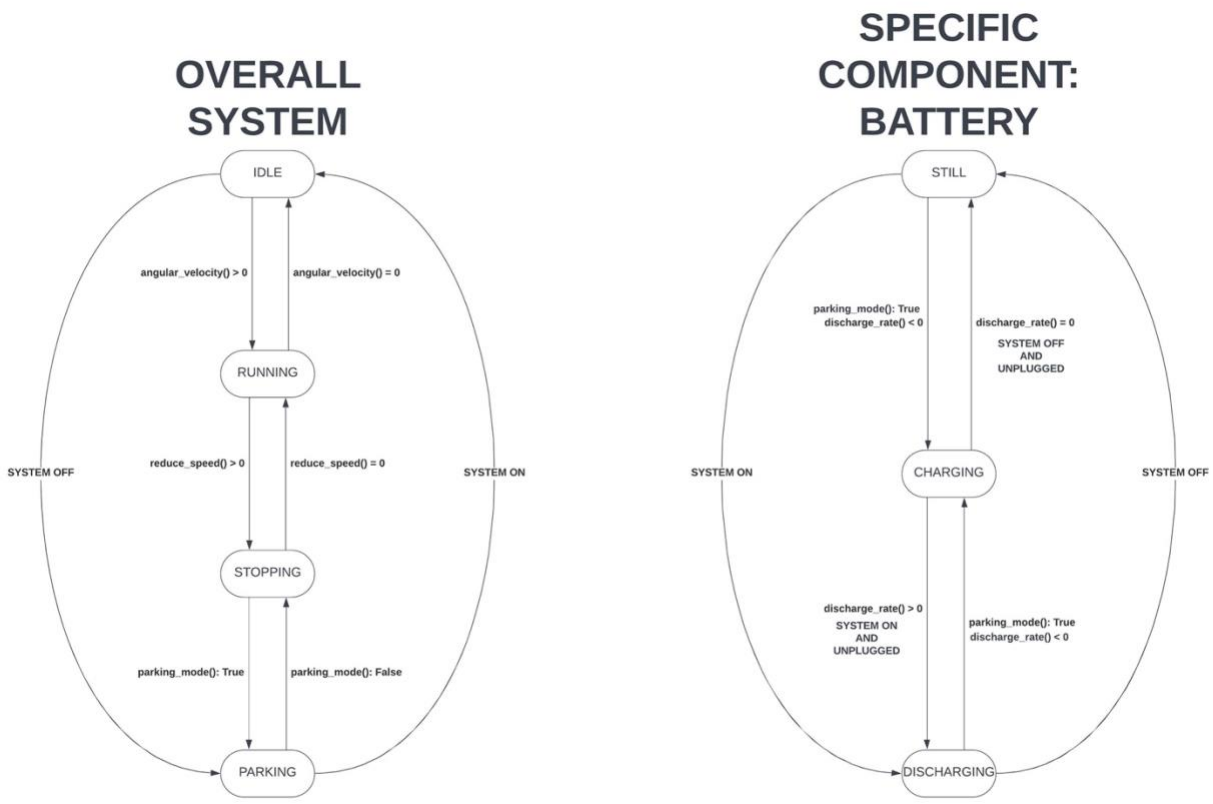


Figure 8: Sequence Diagram

State Transition Diagram

Depiction of the system states and functions that trigger them, for the vehicle's movement and a specific component like the battery. This can be applied to other components like wheels or lights, but same principle applies:



Data management

The On-Board computer decision making process uses a queued list, as instructions are piled and executed with a priority order. For instance, the machine learning algorithm will want to turn right, but to do that, positioning the car on the right lane goes first. Then if there is a crosswalk, pedestrians are given way as top priority again (heaviest node). On the other hand, sensors like the odometer in the motor work with stacked values in a list, as the one with the most value is the latest.

Dictionaries are the main source of data storage while the program is running. These are initialized as empty dictionaries and filled by the class functions. Functionalities and data are fed to the dependant class by passing the right key in a processing function like the one described above.

References

Yang, M. (2023). Lane Detection Methods Survey for Automatic Driving. *Journal of Physics: Conference Series*, 2547(1), pp.012015–012015. doi:https://doi.org/10.1088/1742-6596/2547/1/012015.

TE (2018). *THE ROAD TO AUTONOMOUS DRIVING*. [online] Available at: https://www.te.com/content/dam/te-com/documents/automotive/global/TE-TP-Autonomous-Driving-0218-FINAL-EN.pdf [Accessed 13 Jan. 2024].

Noor-A-Rahim, Md., Liu, Z., Lee, H., Khyam, M.O., He, J., Pesch, D., Moessner, K., Saad, W. and Poor, H.V. (2022). 6G for Vehicle-to-Everything (V2X) Communications: Enabling Technologies, Challenges, and Opportunities. *Proceedings of the IEEE*, 110(6), pp.712–734. doi:https://doi.org/10.1109/jproc.2022.3173031. - (Noor-A-Rahim et al., 2022)

Lott, S.F. and Phillips, D. (2021). *Python Object-Oriented Programming*. Packt Publishing Ltd.

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

Bibliography

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

Eider, M., Bodenschatz, N., Berl, A. and Danner, P. (2020). *A Novel Approach on Battery Health Monitoring*. [online] Available at: https://mediatum.ub.tum.de/doc/1462977/50212155899.pdf [Accessed 12 Jan. 2024].