Software Architecture Document

Project Name:

The Smart Titan Navigation and Steering System (STNSS)

Team Name:

Titan Scrum

Team Members:

Annamalai Arumugam

Vinay Manish Shah

Kishore Shankar Abhimanyu

Nitish Lele

Hetal patel

Keshav Lingala

Sneha Priyadarshan

Thomas Han

Benjarit Hotrabhavananda

Last Revised Date: 19/10/2022

v1.1

Revision History

Date	Version	Description	Author
17/10/2022	1.0	Updated Software Architecture Document contents and diagrams with the help of the SAD template	Smart Titan Navigation and Steering System.
19/10/2022	1.1	Update references section and grammar issues.	

Table of Contents

Table of Figures

1. Introduction

We have developed an architecture for the Smart Titan Navigation and Steering System in which the entire automatic operation of the car is described by taking all feasible inputs for a steering systeminto account. We looked into various automated steering models to get a general idea of how the model functions and how it could be changed to address flaws in existing systems. The architecture is developed by keeping all these major purposes, and scopes in mind, so the output generated will be very appropriate and effective.

1.1 Purpose

The product's goal is to make it easier for the user's car to traverse city streets and highways without needless user input, allowing them to unwind while driving without any intervention or problem from the system. The main purpose is that the car is very safe and reliable so that a maximum number of users can drive it. The sensors and the connection in the car design should work without any error when the user makes use of its functionalities.

1.2 Scope

We have designed the architecture for automated steering system product in such a way that it can cover the maximum functionalities of an automatic steering and navigation system that it can access and process. With the use of the COTS microcontroller chip, it will help to pick the user's action very easily and the user will be able to operate the functionalities very properly. Through the UI the user will easily be able to understand how the car operates and how the functionalities take place in action.

1.3 Definitions, Acronyms, and Abbreviations

This document has used very few definitions and acronyms. which are as follows,

- SAD (Software architecture document),
- UI(User Interface),
- STNSS(Smart titan navigation and steering system), and
- GPS(Global Positioning System).

1.4. References

Following are the documents used for the references for building this architecture of the product:

- https://www.tesla.com/support/autopilot
- https://www.synopsys.com/automotive/what-is-autonomous-car.html
- https://www.thezebra.com/resources/driving/how-do-self-driving-cars-work/
- https://www.jdpower.com/cars/shopping-guides/levels-of-autonomous-driving-explained

1.5. Overview

In this SAD, we have documented the representation of the product developed with the help of the 4+1 views along with the representation of it in a proper manner. Along with the views, we have also explained the size, performance, and quality of the document. In the use case diagram, each and every

process that gets executed is explained through a diagrammatic approach to get an overview of the product functionalities.

2. Architectural Representation

The architecture of The Navigation and Automatic Steering System (NASS) is shown in this document from various views, including the use case view, logical view, process view, development view, and deployment view. These are views on an underlying Unified Modeling Language (UML) including a Package diagram, State diagram, Activity diagram, and Sequence diagram.

3. Architectural Goals and Constraints

The final goal of our Smart Navigation and steering system is to get a flawless automatic system giving the driver the best results without any problem to achieve maximum effortless and optimal experience.

Domain-Driven Design

Domain-Driven Design (DDD) is an way to implement software that centers the development around a domain model which has a fair understanding of the processes and rules of a business domain, encapsulating business centric logic in a core layer of the solution, while having less out of domain dependencies. An architectural goal of this Case is to adhere to DDD principles when dealing with data modification.

4. Use-Case View

The use-case view defines our whole designed architecture in a very well format. It covers all information from the driver, computer, and GPS system to the satellite station’s view. All the process that takes place in each and every user’s case are described in depth in a use-case diagram which can be viewed as a third party here.

4.1. Use-Case Diagrams

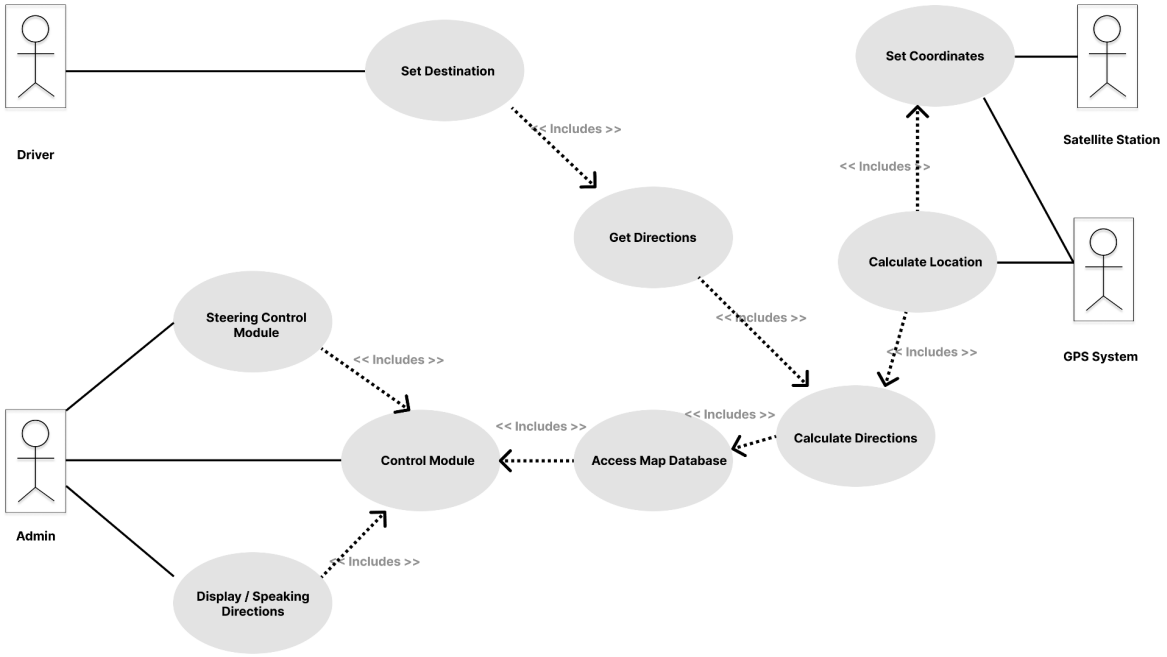


Figure: Use Case Diagram

The Following are the main components of the use case diagram, which are reliable on each other for the operation to take place:

Driver: Operator or person behind the steering wheel

Admin: Admin user will be responsible for the admin panel and debug the failures in any of the controller.

Gps System: It is used to get the current coordinates of the Vehicle with the help of satellite.

Satellite Station: The coordinates given by the GPS system will be mapped by the satellite and fix the final location will be passed to the GPS system and will further go to the computer system and will be checked by the driver.

The System Flow Goes

- The driver sets the destination in the System.
- The System takes in destination input, gets directions, calculates the location, gets coordinates from GPS, and then accesses the map database for additional information.
- Then, The System will pass all the Data about the maps and directions to the central control module, which then displays it on the computer screen and sends it to the steering control module.
- The Steering Control Module then senses the obstacles and then navigates the car appropriately.

5. Logical View

The three primary components, User Interface, System Services, and System Objects, make up the logical representation of the STNSS. It is communicated between the server and client components through the messages that they exchange. Below given is the logical view of our STNSS system design.

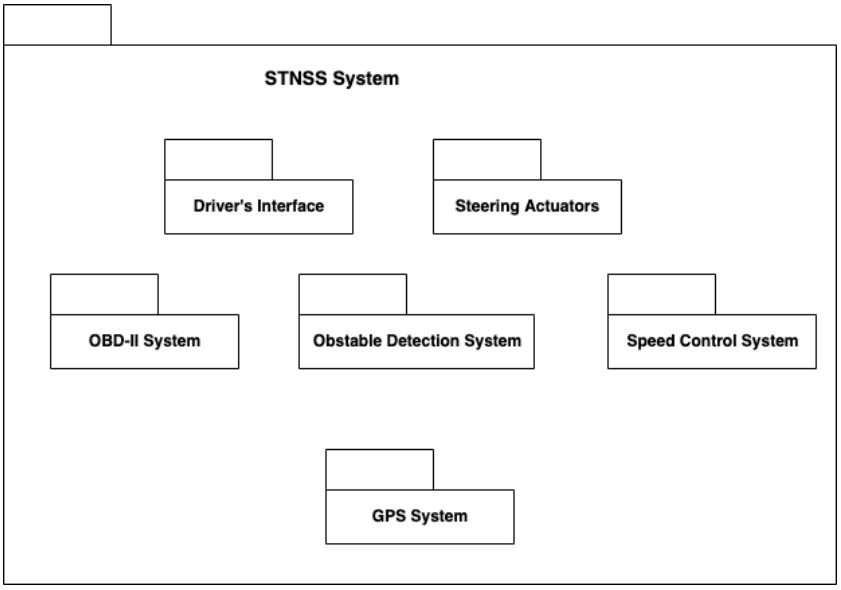


Figure: Logical View

5.1. Overview

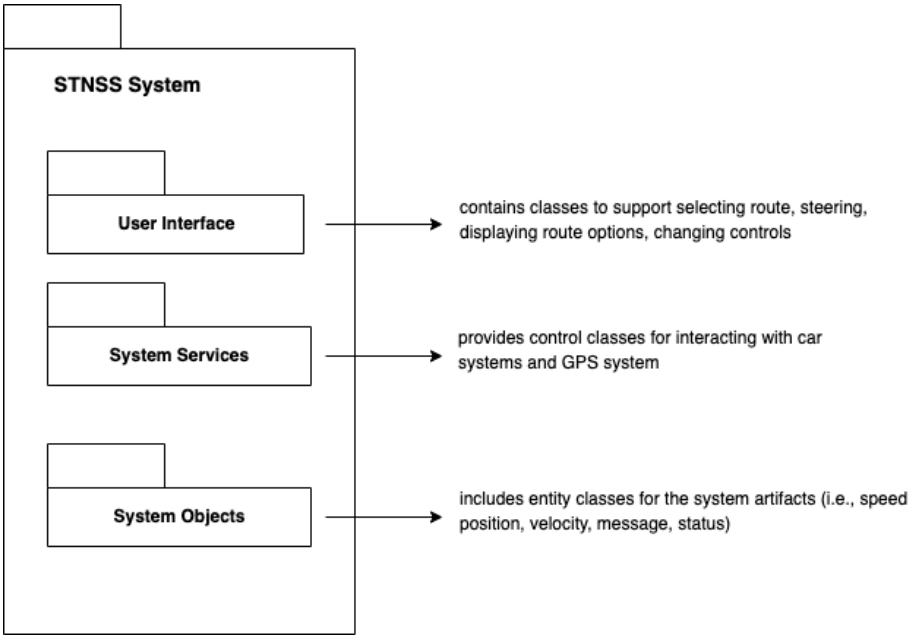


Figure: Logical View Overview

5.2. Architecturally Significant Design Packages

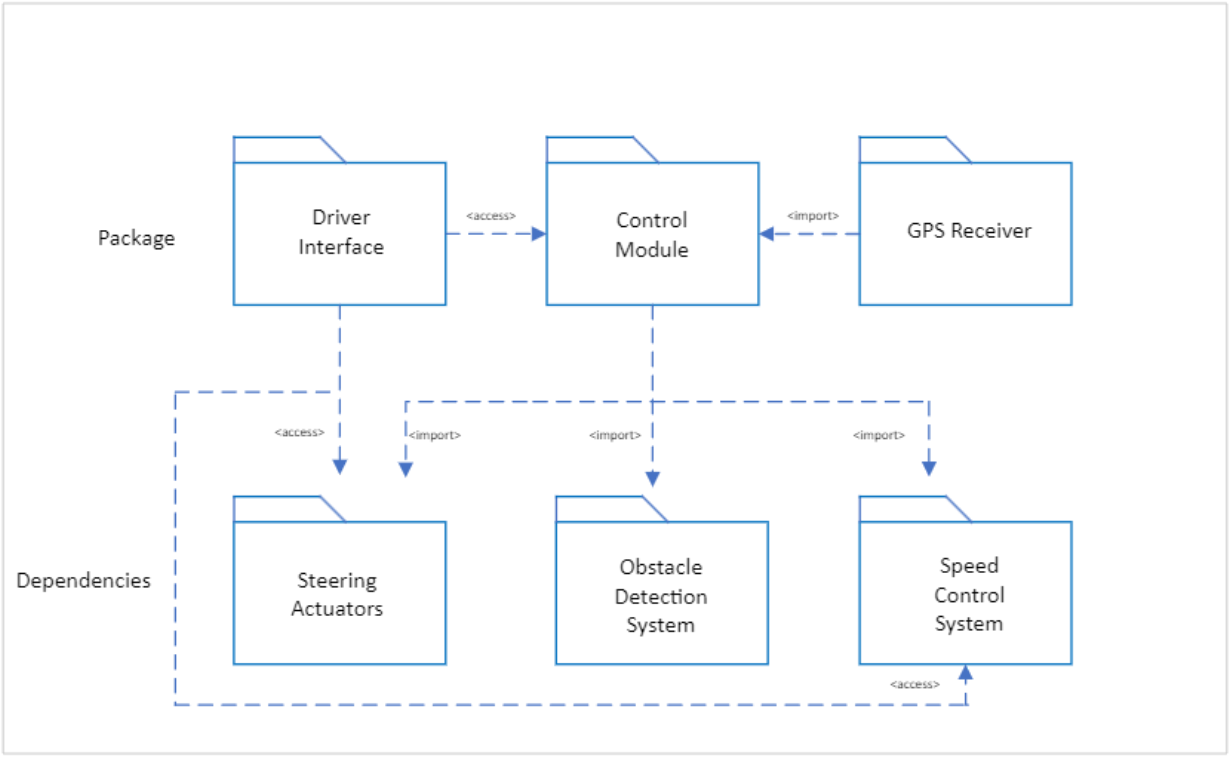


Figure: Architectural Design Packages

The Architecturally Significant Design Packages for the Navigation and Automatic Steering System (NASS) include the following package and their dependencies:

Package:

- Driver Interface: The control module can be accessed by the driver interface package to obtain location and velocity data from the GPS receiver and to identify any obstructions so that the message can be updated appropriately. In order to obtain steering control information and speed control information, it can access the steering actuators of the dependent modules.
- Control Module: The control module package, which joins the other two packages and manages communication between them, is the brains of the program. It is dependent on the sensor systems that provide the correct alerts, errors, and information needed for the application's safety functionality.
- GPS Receiver: All of the application's time and location-related data that the control module package can access is contained in this package. All satellite-based communications inside the application are handled by this package.

Dependencies:

- Steering Actuators: When the system stops the automatic steering capabilities, the driver interface will use the steering actuator to take over the manual steering control. It chooses the route as necessary and updates the position, velocity, and speed as needed.
- Obstacle Detection System: In order for the control system to perform the appropriate action, it detects every impediment the program encounters, whether it is stationary or moving. Several

sensors are utilized for Obstacle Detection, including lidars, radars, cameras, and ultrasonic sensors.

- Speed Control System: The driver has control over the speed system when in manual mode. When using automatic steering, the system displays warnings if the speed limit is exceeded as well as messages about speed management.

5.3. Architecturally Significant Design Classes

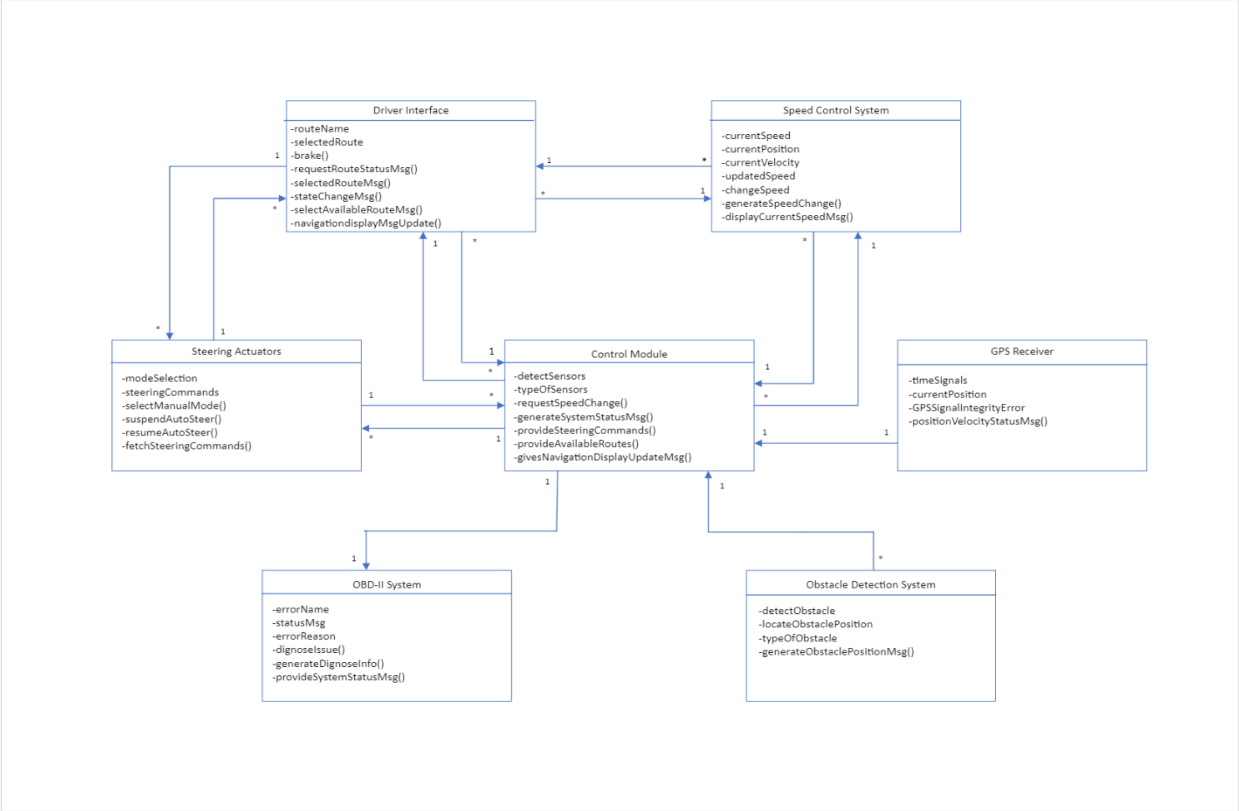


Figure: Architectural Design Classes

The Architecturally Significant Design Classes for the Navigation and Automatic Steering System (NASS) include the following:

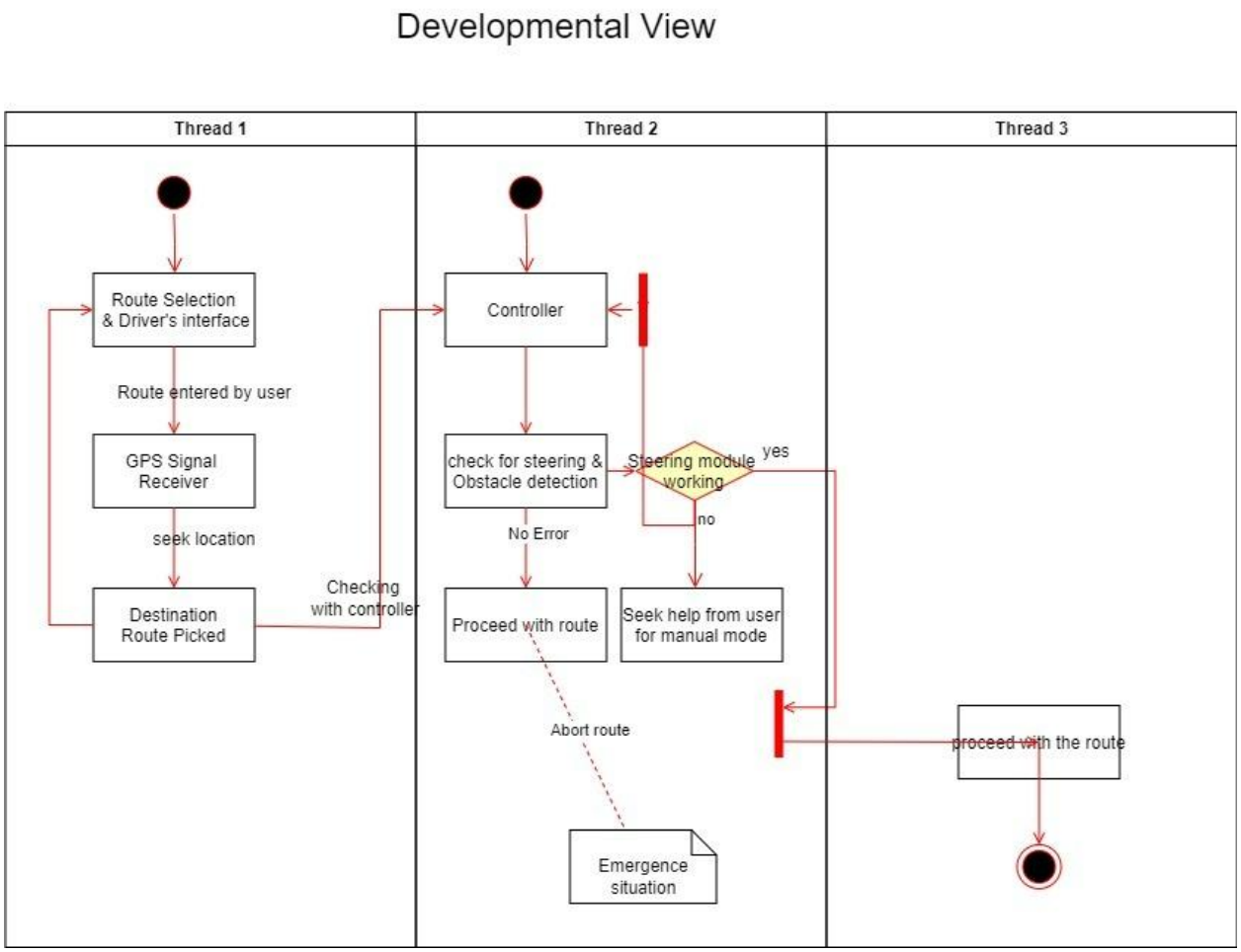
- Driver Interface: The driver class consists of a few control variables and methods that request and provides the selected route, available routes, status like valid or invalid for the requested route, change of state such as the vehicle is resumed or suspended, and updated navigation display messages to locate the vehicle, its speed, velocity, and steering condition.

- Speed Control System: This class gives the current position, speed, and velocity. It also provides updated and changed rates. It receives the speed change message from the control system and updates its speed accordingly to provide the safety of the application.

- Steering Actuators: This class works on the mode selection of the steering. The driver can select manual as well as automatic steering. However, in a few conditions, the control system ceases the intuitive steering functionality, and the driver then operates the car manually for the proper application flow. The steering actuator class fetches the steering command upon which it resumes or suspends the automatic steering feature.

- OBD-II System: The OBD-II system is the system that self-diagnoses the issue. It diagnoses the problem within the system and provides the appropriate error message. It also gives the reason for the raised issue.
- Obstacle Detection System: This sensor class considers a few control variables that detect the obstacle, its location, and its type, such as stationary or moving in the application. It has a method that generates the obstacle position message required by the control system for detailed information regarding the obstacle and to take further action.
- GPS Receiver: This satellite communication class only interacts with the control system. It provides all the necessary methods and variables to locate the current position and time signals needed for the system. It uses clock signals for time accuracy. It gives the valid or in-valid position velocity message required by the system to enable the auto-steering feature.
- Control Module: This is the application's main class which connects to all the other courses. It receives the information from the GPS and provides the action the system needs to perform. It gives the steering commands required by the steering system to operate the steering functionality. It requests the speed change whenever the speed system exceeds the safety speed limit. This class provides the available route and navigation display update message to the driver interface to select the new way and update its speed, position, and velocity accordingly. Moreover, it works with the obstacle detection sensor to sense the obstruction and uses the OBD-II system to diagnose the reason behind it and give the system a status message.

6. Implementation/Development View



6.1 Threads:

System uses multithreaded model to take the appropriate routes and takes automatic decisions.

6.2 Route Selection threads (Thread 1):

This thread is responsible for the selecting the route once the user sets their destination and it picks all the break points in the routes like signs, signal stops, pedestrians.

6.3 Controller threads (Thread 2):

This thread fetches the routes and its outputs as its input then takes the optimistic decision of faster route to destination and pass this to the steering module.

6.4 Steering module threads (Thread 3):

Each thread has its own starting point across different process but its synchronization are managed by controller. Steering module thread gets started by main process and it waits for the controller to finish its decisions. Then, it fetches all the inputs and then it takes periodic inputs from velocity and traffic details from GPS module then it also does fetches input from obstacle detection sensor and navigates the car.

7. Process View

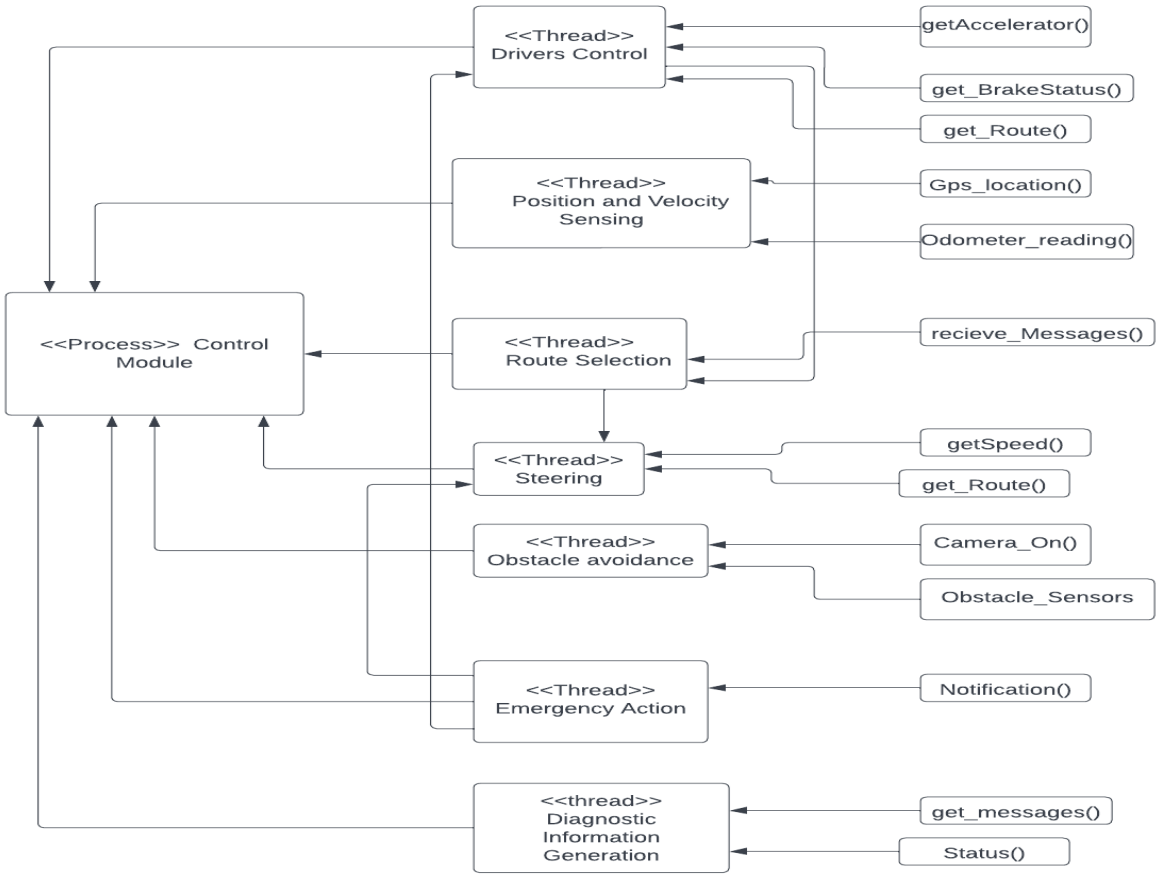


Figure: Process View

7.1 Control Module

Process the data from multiple inputs and send out messages to specific threads.

7.2 Driver Control

Thread that controls sending steering and speed controls for the car to move and send messages to stop steering and speed change.

7.3 Route Selection

Route selection thread provides multiple routes to reach the particular destination.

7.4 Position and Velocity sensing

This thread takes an odometer and GPS reading to find the location of the car and find the speed of the car at a particular time.

7.5 Steering

Steering decides the angle by which the car should turn at different situations. Steering takes input from the control module to decide the angle by which it should rotate.

7.6 Obstacle Avoidance

Obstacle avoidance thread finds out the status of the obstacle. Input for this thread is from camera and other obstacle finding sensors fixed in the car.

7.7 Emergency Action

Emergency action thread takes care of the error in the system and turns off the automatic steering in case of emergency.

7.8 Diagnostic Information Generation

Provides the user with the necessary information regarding the operation of the car and notifies the user regarding the emergency situation or situation which needs user attention.

8. Deployment/Physical View

8.1 External Server

External server focuses on multiple requests from the vehicle for the GPS system, Ml model processing, and general internet activity.

8.2 Internal vehicle Server

Internal device Server focuses on more urgent activity in navigation like a rapid brake system in case of emergency. It also focuses on obstacle detection on the road and instructs the main controller system on the basis of requests.

8.3 GPS System

GPS helps in showing the precise location of the vehicle on the world map. It also helps in navigation systems.

8.4 On-system control system(For steering control)

The system control system manages the steering wheel on the basis of navigation commands from the Internal vehicle Server.

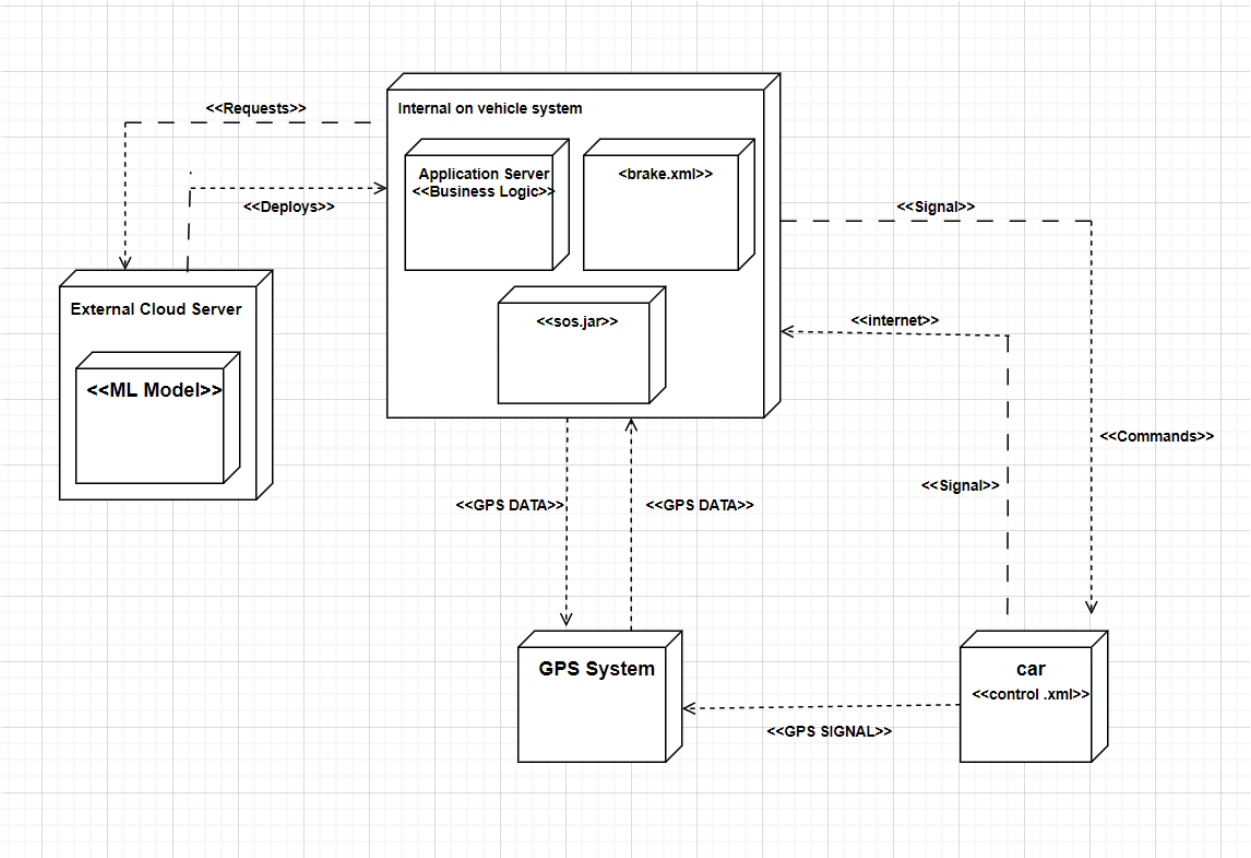


Figure:Physical view diagram

9. Size and Performance

Here are the following performance of our product which needs to be followed in order to better output from the system:

1. The system should correspond to the user's input within 5 seconds of being called.
2. User should not get any kind of error while using the UI of the system dashboard.
3. The satellite connection should be well established while the sensors are activated and asking for coordinates.
4. Sensors should work properly when activated for better performance.

Since the microcontroller chip is fixed inside the car there is no such size of people to use this application since it is embedded inside the car and the users driving it can only use the functionalities of the system.

10. Quality

The software architecture supports the quality requirements, which are explained given below:

1. The desktop user-interface shall be Windows compliant.
2. The user interface of the STNSS shall be designed for ease of use and shall be appropriate for a any user community with no additional training on the System.
3. Each feature of the STNSS shall have built-in online help for the user. Online Help shall include step-by-step instructions on using the System. Online Help shall include definitions for terms and acronyms.
4. The STNSS can be able to work for 24 hours a day, 7 days a week. There shall be no more than 4% downtime.
5. Mean Time Between Failures shall exceed 300 hours.
6. Upgrades to the PC client portion of STNSS shall be downloadable from the UNIX Server over the internet. This feature enables users to have easy access to system upgrades.

< End of Document >