IMPROVING THE EFFICIENCY OF POWER CONSUMPTION IN HYBRID VEHICLES (4 WHEELERS) THROUGH ENGINE SWITCHING

PROJECT REPORT

18CSC202J - OBJECT ORIENTED DESIGN AND PROGRAMMING LABORATORY

(2018 Regulation)

II Year/ III Semester

Academic Year: 2022 -2023

By

ANJANA (RA2111029010017)

MEENAKSHI GAYATHRI (RA2111029010009)

Under the guidance of

Dr. K. Deepa Thilak

Assistant Professor

Department of Networking and Communications

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Kancheepuram

NOVEMBER 2022

BONAFIDE

This is to certify that **18CSC202J - OBJECT ORIENTED DESIGN AND PROGRAMMING LABORATORY project report** titled "**IMPROVING THE EFFICIENCY OF POWER CONSUMPTION IN HYBRID VEHICLES (4 WHEELERS) THROUGH ENGINE SWITCHING"** is the bonafide work of **ANJANA (RA2111029010017)** and **MEENAKSHI GAYATHRI (RA2111029010009)** who undertook the task of completing the project within the allotted time.

Signature of the Guide

Advisor

Dr. K. Deepa Thilak

Assistant Professor

Department of NWC,

SRM Institute of Science and Technology

Signature of the II Year Academic

<<Name>>

<<Designation>>

Department of NWC

SRM Institute of Science and Technology

About the course:-

18CSC202J/ 8AIC203J - Object Oriented Design and Programming are 4 credit courses with **L T P C as 3-0-2-4** (Tutorial modified as Practical from 2018 Curriculum onwards)

Objectives:

The student should be made to:

- Learn the basics of OOP concepts in C++
- Learn the basics of OOP analysis and design skills.
- Be exposed to the UML design diagrams.
- Be familiar with the various testing techniques

Course Learning Rationale (CLR): The purpose of learning this course is to:

1. Utilize class and build domain model for real-time programs
2. Utilize method overloading and operator overloading for real-time application development programs
3. Utilize inline, friend and virtual functions and create application development programs
4. Utilize exceptional handling and collections for real-time object-oriented programming applications
5. Construct UML component diagram and deployment diagram for design of applications
6. Create programs using object-oriented approach and design methodologies for real-time application development

Course Learning Outcomes (CLO): At the end of this course, learners will be able to:

1. Identify the class and build domain model
2. Construct programs using method overloading and operator overloading
3. Create programs using inline, friend and virtual functions, construct programs using standard templates
4. Construct programs using exceptional handling and collections
5. Create UML component diagram and deployment diagram
6. Create programs using object oriented approach and design methodologies

Table 1: Rubrics for Laboratory Exercises

(Internal Mark Splitup:- As per Curriculum)

CLAP-1	5=(2(E-lab Completion) + 2(Simple Exercises)(from CodeZinger, and any other coding platform) + 1(HackerRank/Code chef/LeetCode Weekend Challenge)	Elab test
CLAP-2	7.5=(2.0(E-lab Completion)+ 2.0 (Simple Exercises)(from CodeZinger, and any other coding platform) + 3.5 (HackerRank/Code chef/LeetCode Weekend Challenge)	Elab test
CLAP-3	7.5=(2.0(E-lab Completion(80 Pgms)+ 2.0 (Simple Exercises)(from CodeZinger, and any other coding platform) + 3.5 (HackerRank/Code chef/LeetCode Weekend Challenge)	**2 Mark -** E-lab Completion **80 Program** Completion from 10 Session (Each session min 8 program) **2 Mark -** Code to UML conversion GCR Exercises **3.5 Mark - HackerRank** Coding challenge completion
CLAP-4	5= 3 (Model Practical) + 2(Oral Viva)	• **3 Mark** – Model Test • **2 Mark** – Oral Viva
Total	25	

COURSE ASSESSMENT PLAN FOR OODP LAB

S.No	List of Experiments	Course Learning Outcomes (CLO)	Blooms Level	PI	No of Programs in each session
1.	Implementation of I/O Operations in C++	CLO-1	Understand	2.8.1	10
2.	Implementation of Classes and Objects in C++	CLO-1	Apply	2.6.1	10
3,	To develop a problem statement. 1. From the problem statement, Identify Use Cases and develop the Use Case model. 2. From the problem statement, Identify the conceptual classes and develop a domain model with a UML Class diagram.	CLO-1	Analysis	4.6.1	Mini Project Given
4.	Implementation of Constructor Overloading and Method Overloading in C++	CLO-2	Apply	2.6.1	10
5.	Implementation of Operator Overloading in C++	CLO-2	Apply	2.6.1	10
6.	Using the identified scenarios, find the interaction between objects and represent them using UML Sequence diagrams and Collaboration diagrams	CLO-2	Analysis	4.6.1	Mini Project Given
7.	Implementation of Inheritance concepts in C++	CLO-3	Apply	2.6.1	10
8.	Implementation of Virtual function & interface concepts in C++	CLO-3	Apply	2.6.1	10
9.	Using the identified scenarios in your project, draw relevant state charts and activity diagrams.	CLO-3	Analysis	4.6.1	Mini Project Given
10.	Implementation of Templates in C++	CLO-3	Apply	2.6.1	10
11.	Implementation of Exception of Handling in C++	CLO-4	Apply	2.6.1	10
12.	Identify the User Interface, Domain objects, and Technical Services. Draw the partial layered, logical architecture diagram with UML package diagram notation such as Component Diagram, Deployment Diagram.	CLO-5	Analysis	4.6.1	Mini Project Given
13.	Implementation of STL Containers in C++	CLO-6	Apply	2.6.1	10
14.	Implementation of STL associate containers and algorithms in C++	CLO-6	Apply	2.6.1	10
15.	Implementation of Streams and File Handling in C++	CLO-6	Apply	2.6.1	10

LIST OF EXPERIMNENTS FOR UML DESIGN AND MODELLING:

To develop a mini-project by following the exercises listed below.

1. To develop a problem statement.
2. Identify Use Cases and develop the Use Case model.
3. Identify the conceptual classes and develop a domain model with UML Class diagrams.
4. Using the identified scenarios, find the interaction between objects and represent them using UML Sequence diagrams.
5. Draw relevant statechart and activity diagrams.
6. Identify the User Interface, Domain objects, and Technical services. Draw the partial layered, logical architecture diagram with UML package diagram notation.

Suggested Software Tools for UML:

StarUML, Rational Suite, ArgoUML (or) equivalent, Eclipse IDE and Junit.

ABSTRACT

The main objective of this project is designing a software that works towards improving the efficiency of power consumption in vehicles (4 wheelers) by introducing hybrid engine switching. In other words, this idea enables the software implementation of interchanging of fuel powered engines and electric engines within a vehicle (typically of four wheels or more) through analyzing the internal and external surrounding conditions of the vehicle. Listed below are certain factors that were taken into consideration as motives behind the project: 1) Vehicle performance (that is, traction, stability and maneuverability), pavement friction, roadway infrastructure, crash risk, traffic flow get affected due to terrestrial and weather based conditions. For instance, brake time is slower in bad weather, so leaving more room to stop is imperative. 2) The frictional force between the road and tire is what allows the tire to "push" off the road, thus moving the car forward (Newton's third law — the action is the pushing frictional force, the reaction is the forward movement of the car). 3) Electric cars store the electricity in rechargeable batteries that power an electric motor, which turns the wheels. They accelerate faster than vehicles with traditional fuel engines – so they feel lighter to drive. They are also environmentally friendly. 4) Although superior to gas powered vehicles, disadvantages of EVs include finding charging stations, charging times, higher initial costs, limited driving range, and battery packs can be expensive to replace. 5) Electric cars do not charge while driving. However, some EVs will charge when you use the breaks. Thus, we come up with the solution of implementing both engines into one vehicle, and switching between them as required by the user, and/or automatically, based on the surrounding environmental conditions, as detected by the sensors.

MODULE DESCRIPTION

Apart from the description given in the abstract, some other important points regarding the project are covered below:

When the vehicle is started, the electric engine is activated by default. The user is given a prompt as to whether they would prefer automatic switching of the engines, as will be decided by the system based on predefined optimal conditions, or if they would prefer manual switching of the engines; that is, the user can choose which engine they want to activate depending on their preferences. Information from the sensors will still be displayed to the user, and they can change the modes of engine switching as per their will.

Additional functions in terms of the programming aspect based on conditions and status responses from the sensors are as follows:

- Slowing down and speeding up the vehicle
- Charging the electric engine when Fuel based engine is in use
- Filling in Fuel based engine when the electric engine is in use

The predefined conditions based on which decisions are made by the software include the temperature of the area outside the vehicle, as well as the temperature of the engines within the vehicle, the location and traffic situations of the region, and the nature of the terrain that the vehicle has to go through. Information on these conditions is gathered by sensors that are integrated within the vehicle, including temperature sensors which determine the temperatures, GPS sensors which display locations and crowd density, tire pressure sensors which calculate appropriate gauge values through the friction that is experienced by the tires, and even oxygen sensors that determine the amount of oxygen left in the fuel powered engine, as well as voltage sensors that are meant for the electric engine.

The conditions and sensors are specified as follows:

Conditions:

Weather based: Fog, temperature, rainfall / hail, snow / sleet, dust.

Land/Terrestrial: Wet surface (due to rains/any other reason), heavy traffic, friction, slopes, uneven roads, sudden changes in road width.

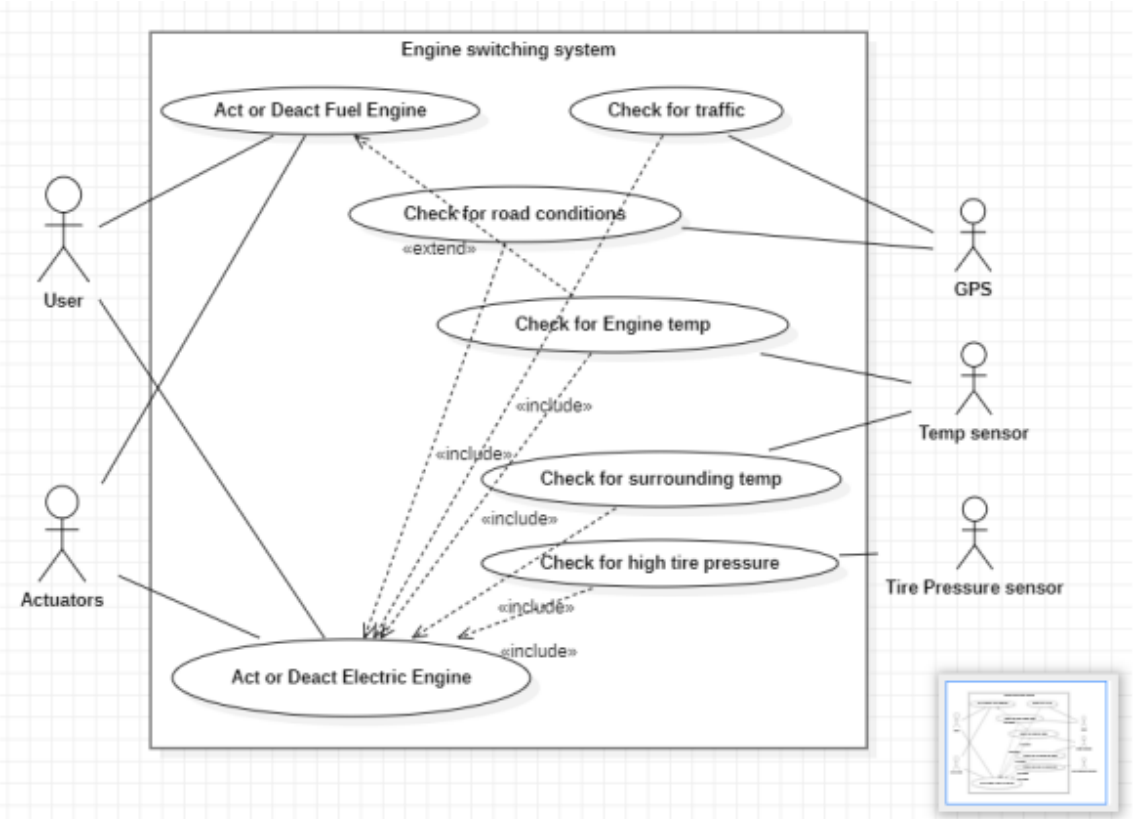
Sensors:

Temperature sensors (for monitoring surrounding (weather) and engine temperatures), *GPS* (detects traffic conditions), *Tire pressure sensors* (Temperature of air in the tire increases due to friction of the tire with the road - the tire pressure varies based on the terrain), *Oxygen/other gasses' sensor* (to track the amount of oxygen/other gasses left for combustion), *Voltage sensors* (to detect engine idling*), and *Electronic battery sensor* (informs the car of the exact battery status, measures the temperature and controls the charging voltage and charging current accordingly.)

*(*Idling is when vehicles' engines are running, but the vehicles are not in motion. Idling cars and trucks produce 130,000 tons of carbon dioxide each year. EVs can be used here to prevent this and also enable the user to avail the benefits of air conditioning and the radio.*)*

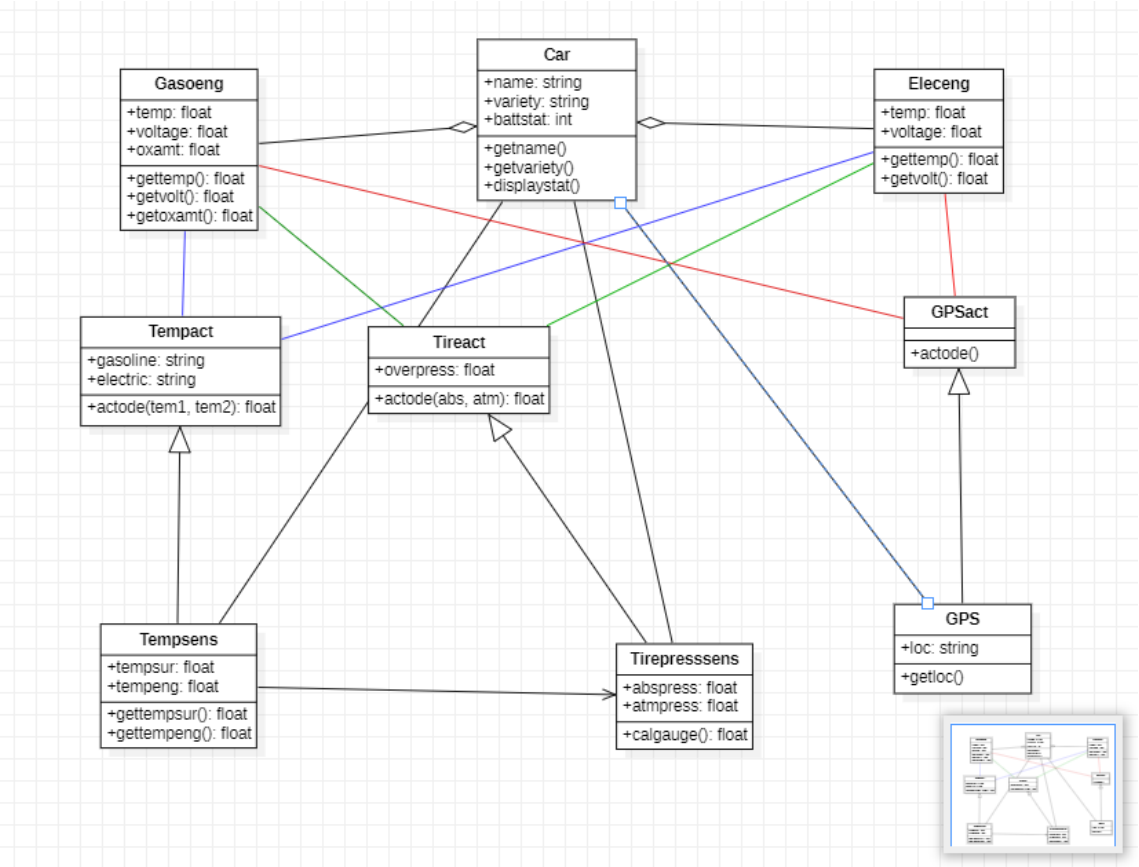
The information received from these sensors is then put to use through the use of actuators, which enable the engine switching software to switch the engines by activating or deactivating them appropriately based on the conditions, to be as efficient as possible, in terms of improving the efficiency of the vehicle, while also potentially ensuring the betterment of the environment through reductions in pollution.

Use case diagram with explanation



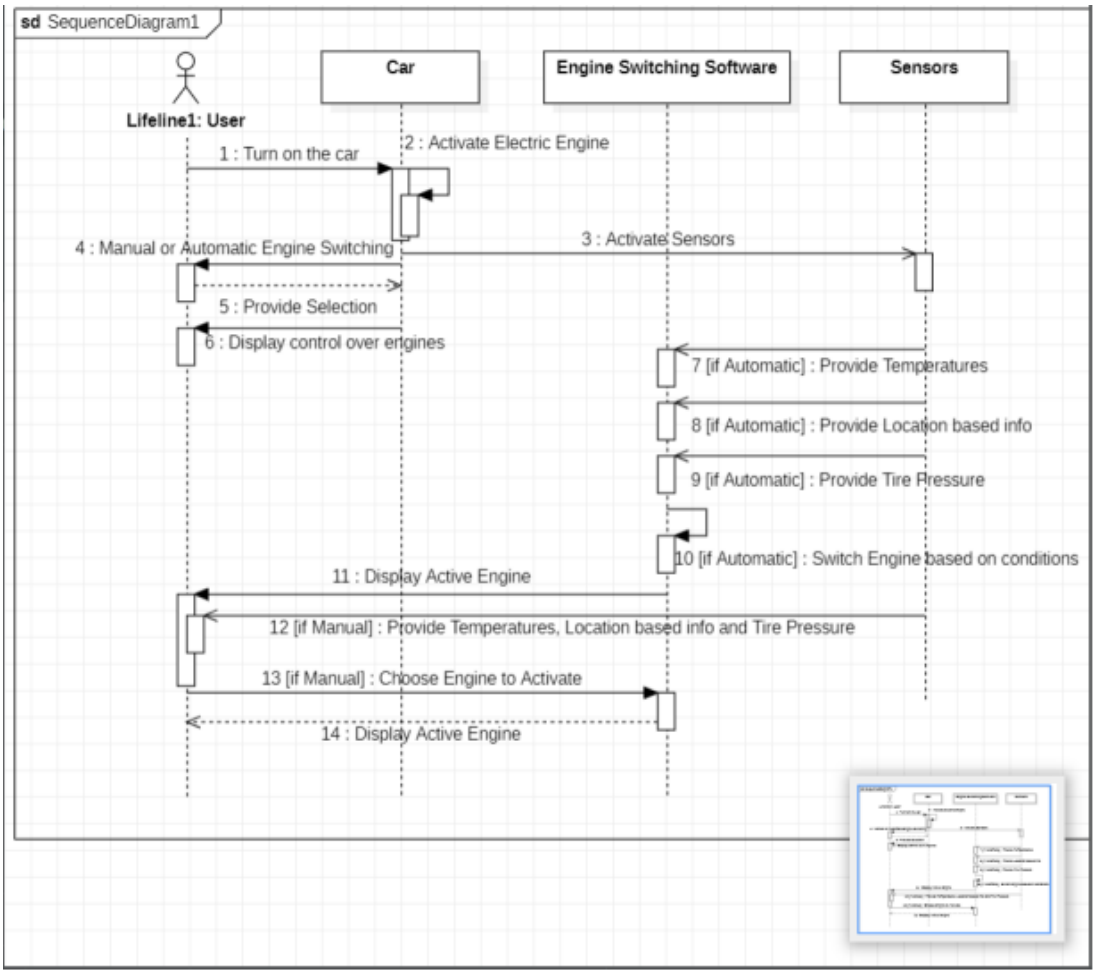
The primary actor, as depicted in the diagram above, is the user (person driving the vehicle), followed by the others, which are the actuators, and their respective GPS, temperature, and tire pressure sensors. “Act or Deact” refers to the activation and deactivation of the engines as per the requirements to be specified by the program. The sensors check for the surrounding conditions, and provide the same information to the actuators, as well as the user. In the case of the switching being automatic, the engine switching occurs automatically based on the optimal conditions; if the switching is manual, the user picks which engine they want to activate or deactivate, and are provided with information from the sensors as well. The <<include>> statements indicate that the activation or deactivation of the engines is dependent on the information provided by the sensors, and the <<extend>> statement illustrates that the temperatures of the engines are checked internally, following which the engine switching is carried out if and as required.

Class diagram with explanation



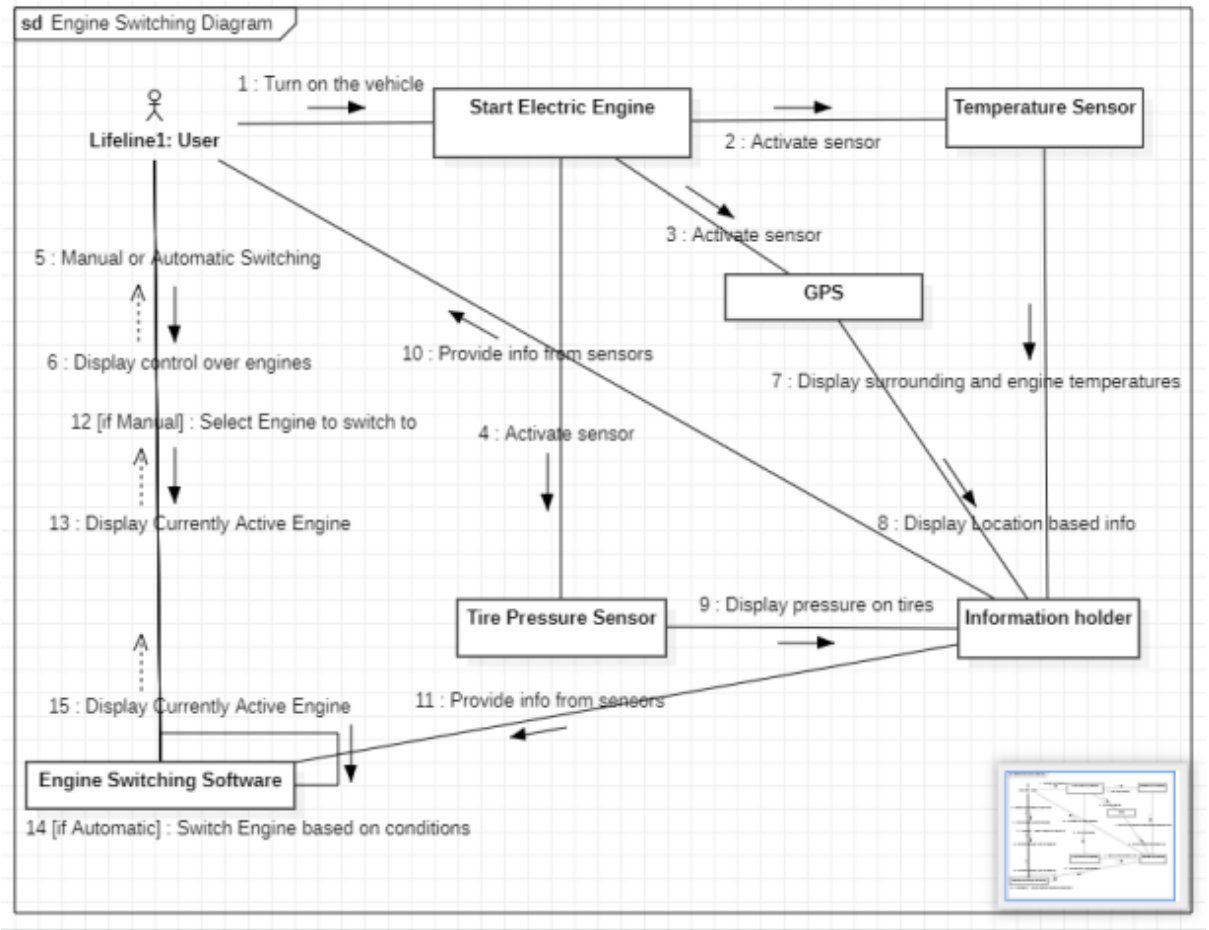
The above diagram represents the classes involved in the switching of engines in a 4 wheeler (car). The classes involved are two different engines (gasoline and electric engine), sensors and actuators. Each sensor senses the required temperature, tire pressure and GPS location and passes the information to both the engines which results in the switching of engines either manually or automatically. Each and every class has its own functions for fetching in the details of temperature from both internal and external environment, GPS location and tire pressure. Specific associations are mentioned between the classes above(i.e) sensors and actuators are connected with inheritance concept.

Sequence diagram with explanation



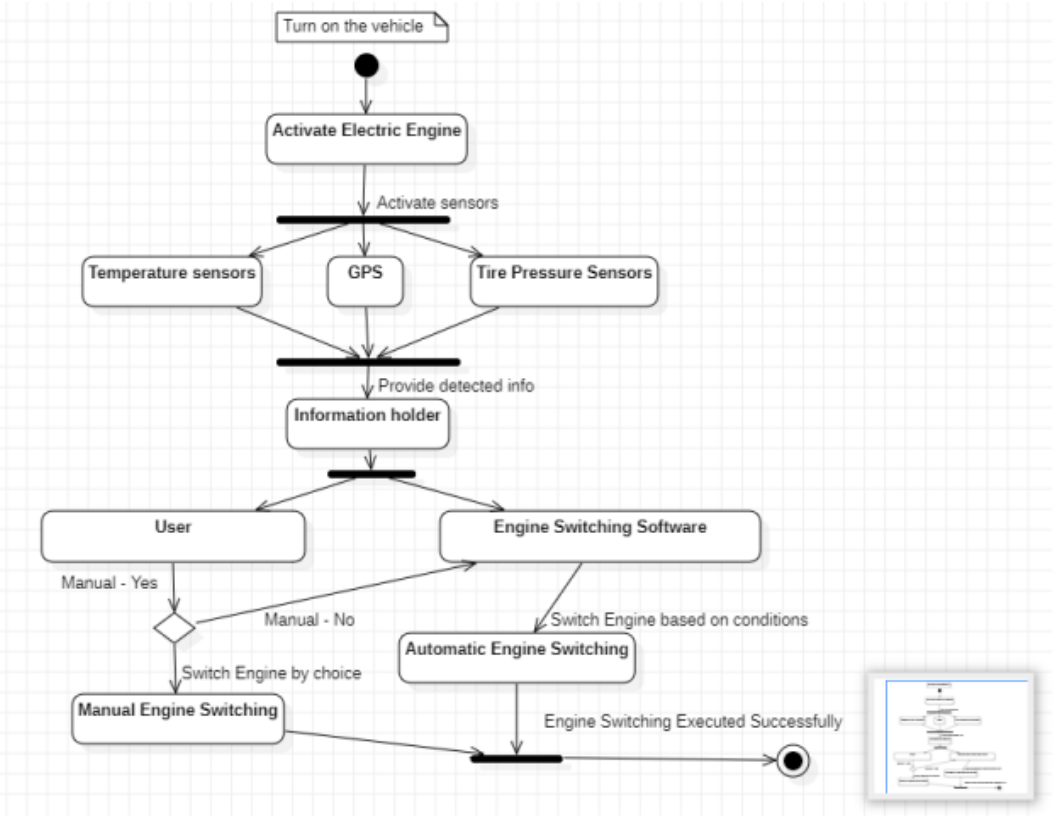
The sequence diagram above illustrates the communication between each object as per the desire of the user/ requirements of the engine switching software. The user turns on the vehicle, and selects the automatic or manual mode of engine switching. When the user chooses the switching mode the sensors get activated and provide the required conditions like temperature, tire pressure and location (terrain) based on which the engines are switched. The guard statement shows the condition of whether the user has the switching control, or if the vehicle is to automatically switch the engines. The switching system also has a 'self' message, which shows that it switches the engine by itself, based on the information that it receives from the sensors, as well as the required conditions.

Collaboration (Communication) diagram with explanation



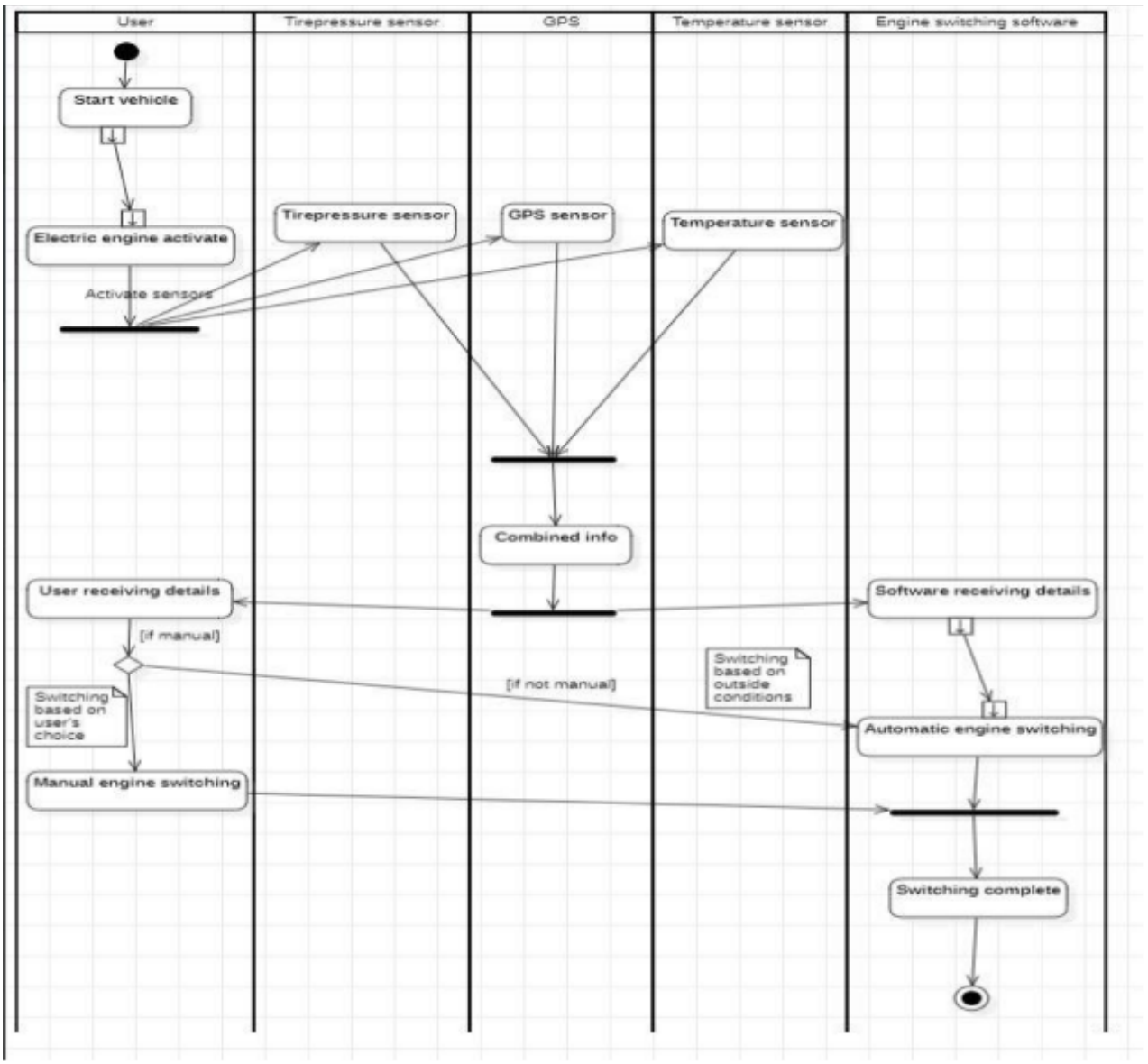
The communication diagram above illustrates the connections between the various objects that take part in the process. The user is depicted as the main lifeline, who initiates the events by turning on the vehicle, and selecting the automatic or manual mode of engine switching. Appropriate messages as shown above are sent from one object to another; here, from the activation of the sensors, to the provision of information to the information holder, to the information being provided to the engine switching software and the user, and the respective replies to the messages that are being sent, such as displaying the currently active engine to the user, are all of significance. Moreover, guard statements with ‘if’ conditions have been provided in the cases of the switching being automatic or manual, and the switching system employs a ‘self’ message as well, where it switches the engine by itself, based on the information from the sensors and the predefined optimal conditions.

State chart diagram with explanation



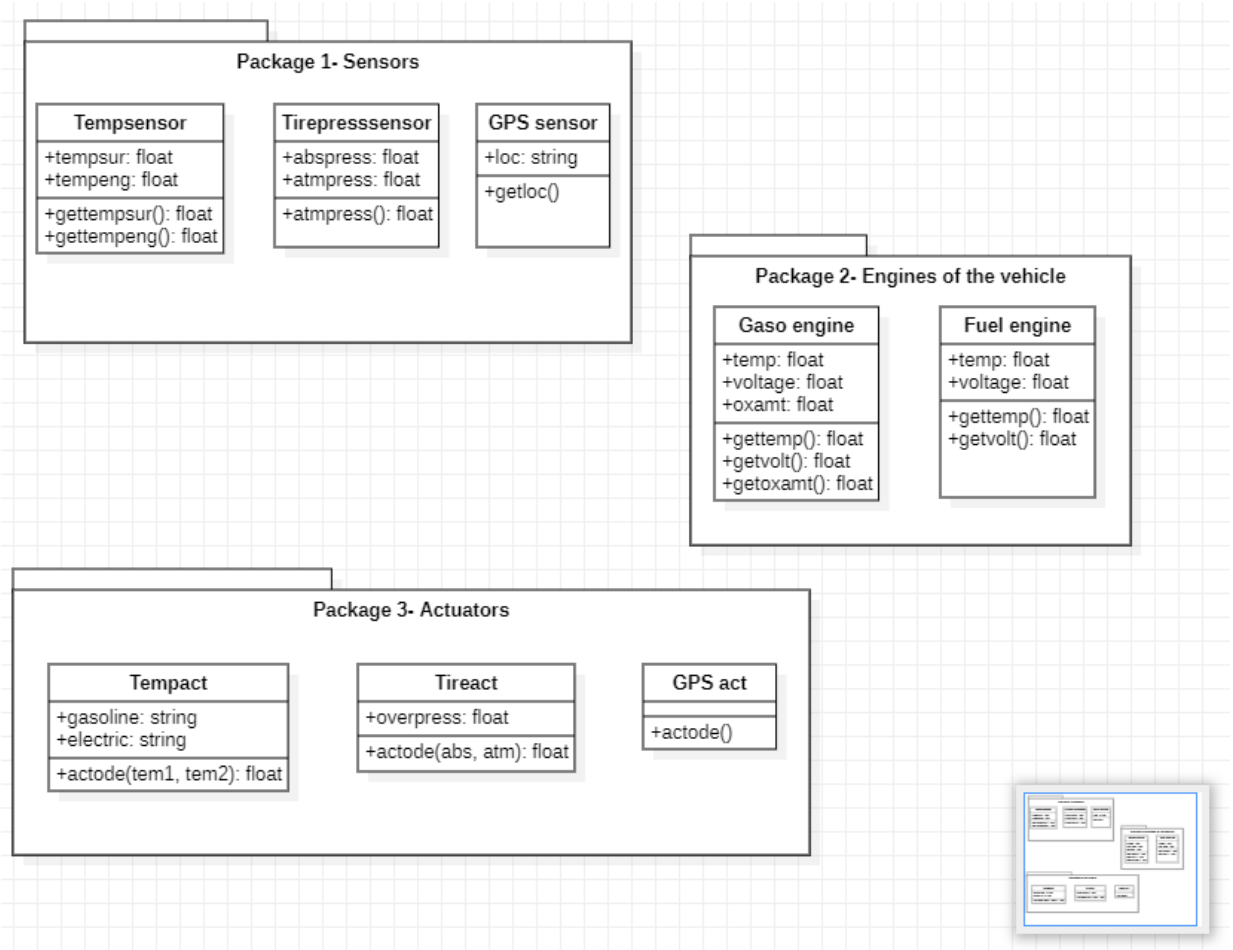
The states of the various objects within the life cycles of the process are demonstrated in the statechart diagram above. Turning on the vehicle (denoted by the 'initial state' symbol) enables the activation of the electric engine (by default as mentioned before), and 'forking' is illustrated, where the activation of the engine leads to the activation of the sensors. Following this, at the stage where the sensors have received information from the respective sources, 'joining' is illustrated, where all the information is sent to the information holder. 'Forking' is implemented once again, where the information is displayed to the user and provided to the engine switching software, following which a 'decision' symbol is used to show the if-else condition of manual or automatic switching of engines, through which the engine switching software and/or the user proceed to take action. The end of each cycle is denoted by the 'final state' symbol, by when the engine switching would have been executed successfully.

Activity diagram with explanation



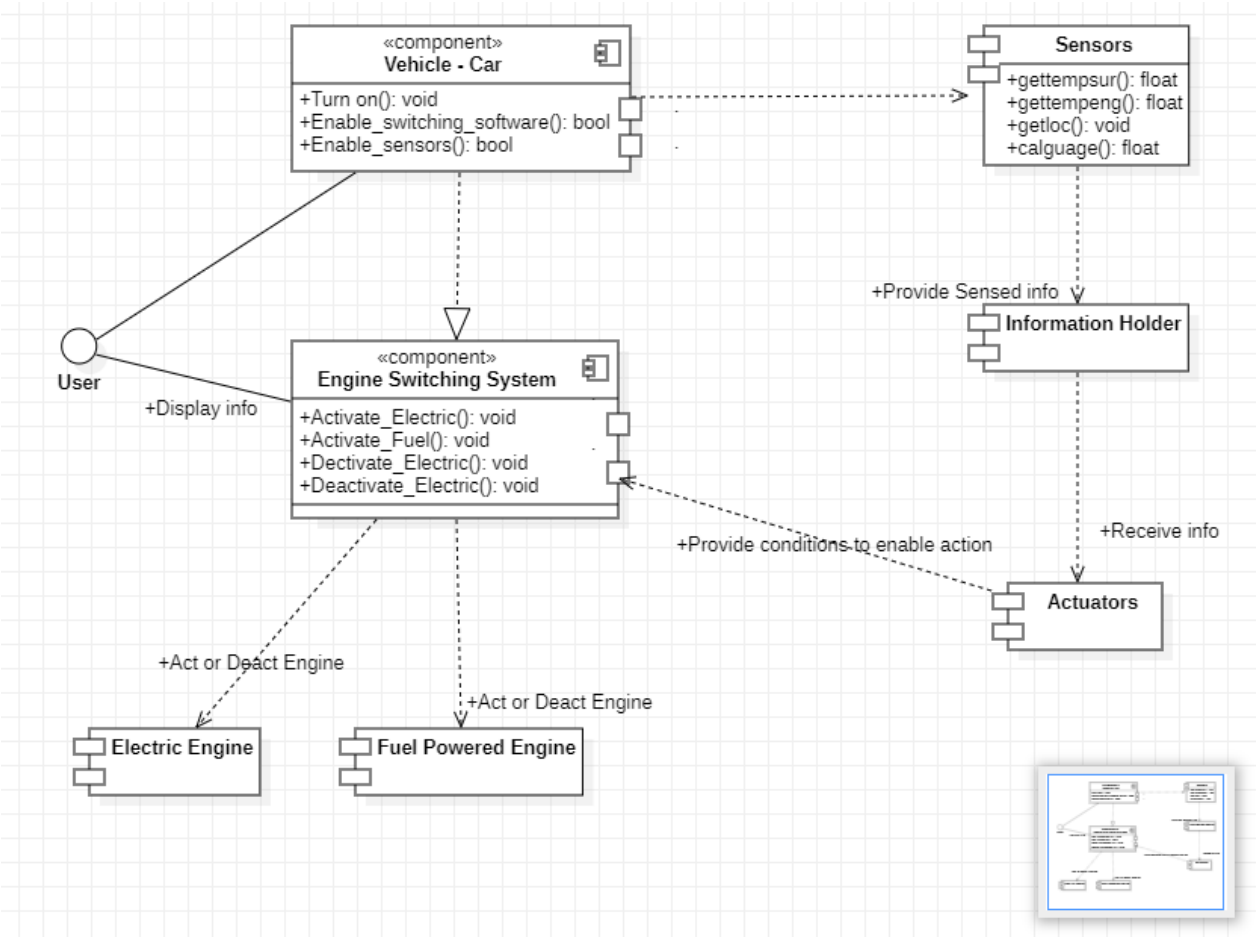
The execution of the process starts with the user turning on the vehicle (start symbol) which in turn results in the activation of the sensors in the vehicle. As switching on the vehicle results in the activation of the 3 sensors, the operation/ condition is considered to be a fork. The combined information which is obtained by the information holder is then passed on to both user and the software based on which if the user chooses manual or automatic mode of switching based on his wish and will (end/finish symbol).

Package diagram with explanation



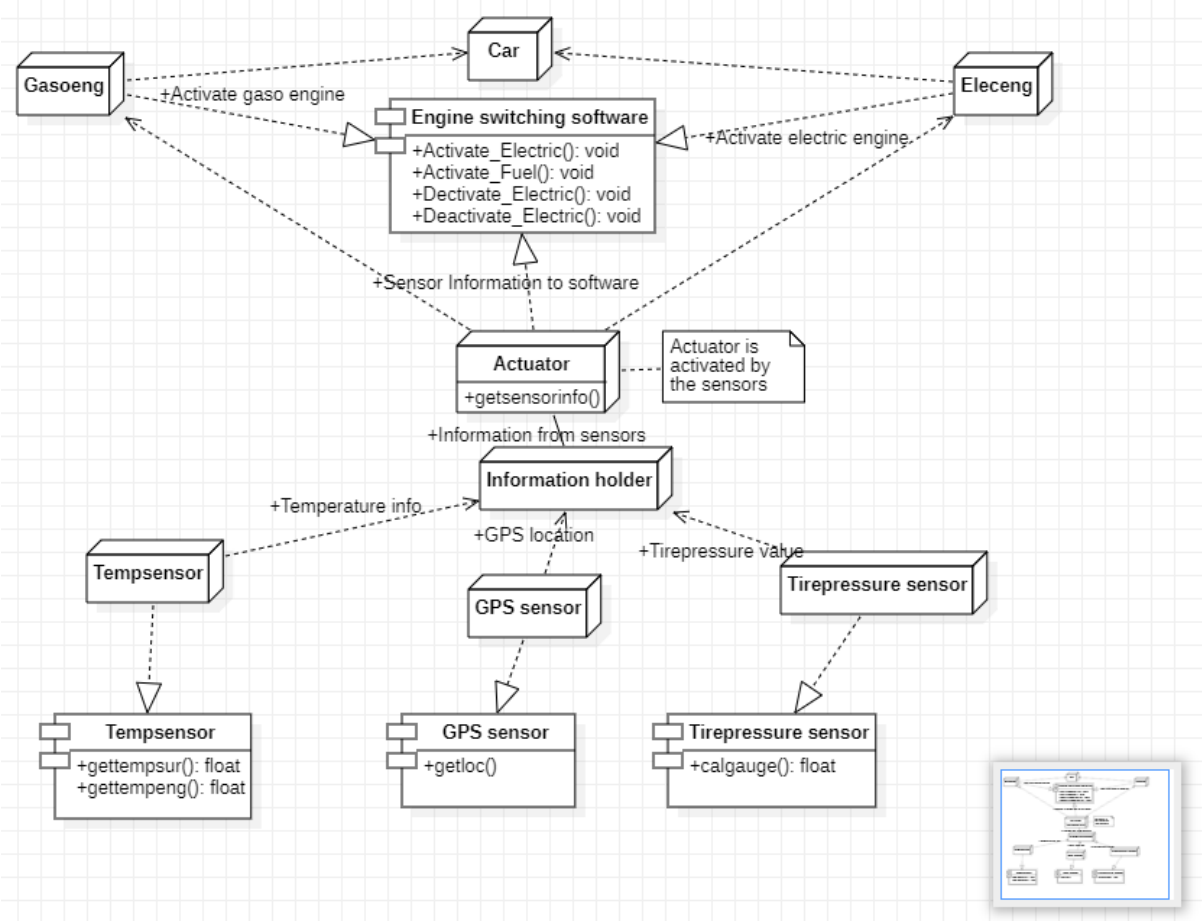
The above diagram represents a package which consists of class diagrams as components in it. The engines, sensors and actuators that were illustrated in the class diagram have been grouped into a single package respectively, where the specific attributes and operations related to them are mentioned. It is to be noted that connections or relations have been avoided here as there are some common functions (operations) as well as non-common operations in the classes belonging to their respective packages, which would lead to potential ambiguity in the illustration.

Component diagram with explanation



The main components involved in the project are illustrated in the component diagram above. They include the vehicle, the sensors, the engine switching system, the actuators, and the electric and fuel powered engines themselves as well. The user is depicted to be using an interface in order to communicate with the above components and carry out the required actions as per their will, in the case that the switching of the engines is set to the manual mode. The information received from the sensors is shown to be displayed to the user, and the activation and deactivation of the engines is shown to be carried out by the engine switching software as well.

Deployment diagram with explanation



The main nodes/hardware components involved in the project are illustrated in the deployment diagram above. They include the vehicle (car), the sensors, the engine switching system, the actuators, and the electric and fuel (gasoline) powered engines. The engines depend on the actuators, which rely on the information holder, which receives information from the sensor outputs (shown by the dependency arrows). The operations of sensors are shown as components, which are connected to their respective nodes. This type of connection shown here is the implementation of component realization (behavior of sensors mentioned). Based on the final output from the information holder, the switching of engines from either gasoline to electric or vice versa happens either manually (if that is the mode chosen by the user) or automatically.

Conclusion

Thus, through thorough analyses and designing of the UML diagrams as depicted above, designing a software that works towards improving the efficiency of power consumption in vehicles (4 wheelers) by introducing hybrid engine switching (mostly in terms of software implementation) has been demonstrated successfully. This project can be further expanded by bringing in other types of sensors that would make the functioning of the engine a lot more efficient, considering that efficiency was the prime goal of this project. One can also take up other scenarios that revolve around vehicles and their impact on the environment, and the focus can be further diverted into the betterment of the environment as well. This project idea is a basic step towards achieving the globally pursued goal of efficiency in our world by utilizing available resources as optimally as possible.

References

- https://www.sciencedirect.com/topics/engineering/hybrid-electric-vehicle
- https://www.elprocus.com/different-types-of-sensors-used-in-automobiles/
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4217221/
- https://creately.com/blog/diagrams/deployment-diagram-tutorial/
- https://developer.ibm.com/articles/the-sequence-diagram/
- https://www.youtube.com/watch?v=O3o9oOWBwb0&t=63s