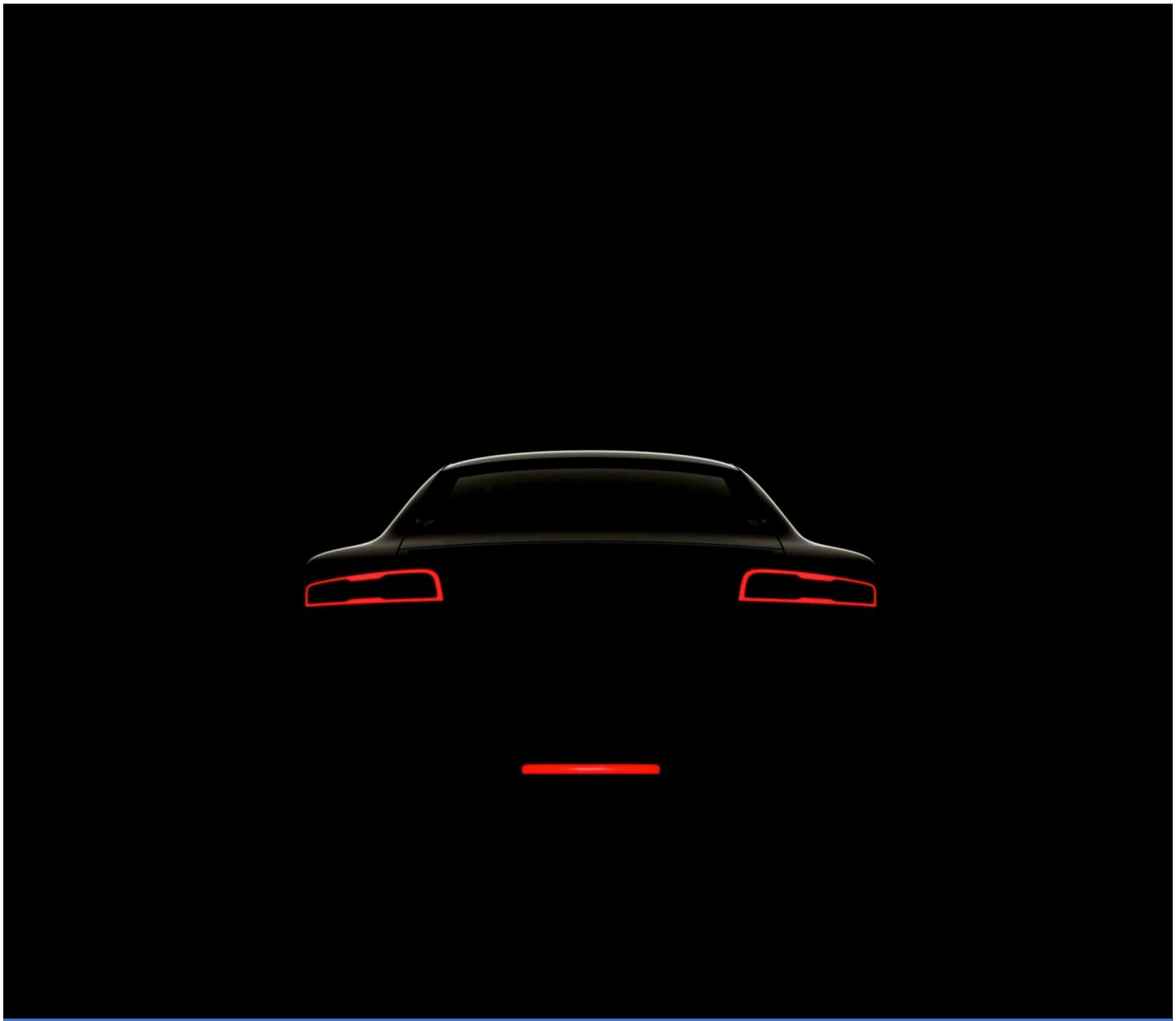


CAR HEALTH MONITOR

MECH 550C

Contents

Table of Figures

PROJECT PROPOSAL

Goals:

To design a tunable car health monitoring system which notifies when the car needs maintenance along with the predicted maintenance cost via an email and a simulator of the cars sensors.

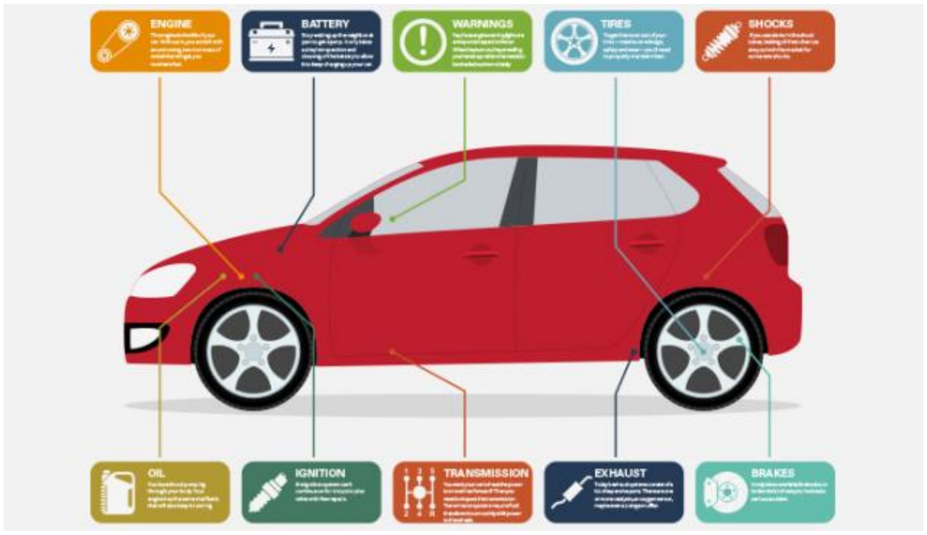
Tasks to be carried out:

- Collect data about sensors values and reasons for car maintenance to decide checking the parameters and generate data to train predictive model.
- Design GUI for the simulator of the car's sensors which streams data packets of sensor information.
- Setup AMQP for streaming this data.
- Design a GUI that displays the sensor value in a graphical form.
- Build monitoring system which decodes the packets of sensor information and checks if the car requires maintenance based on the checking parameters set by a .txt file and passes the sensor information to the predictor system.
- Create a predictive model (Regression model) that uses multiple features such as the cars make, the cars model and the sensors output (that is received from the monitoring system) to estimate the maintenance cost.
- Create a Notification layer that uses Google Mail API to extract all the previously collect information and notify the user via email.

Introduction:

Contemporary vehicles are fascinating art of machinery and are manufactured using latest technology and a lot of engineering. Every upcoming model is mastered to eclipse the previous versions in terms of various parameters like fuel efficiency engine performance etc.

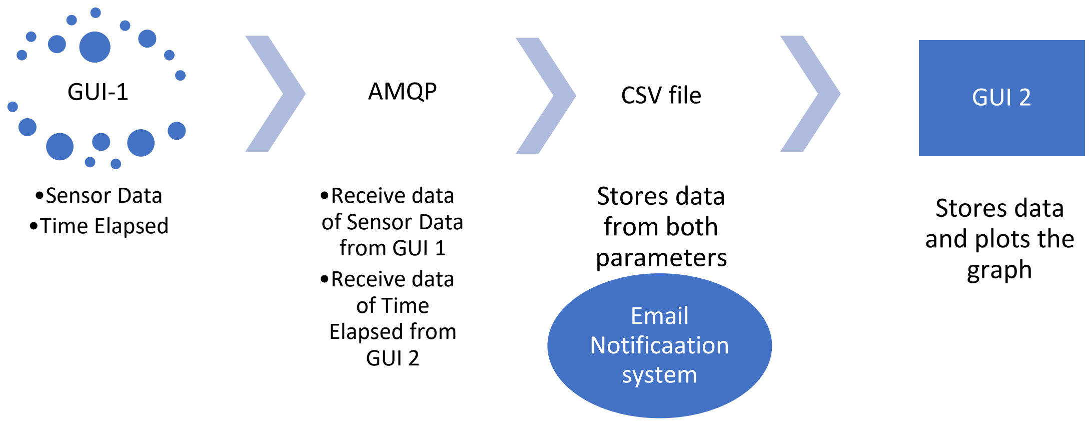
However, there is no denying fact that it is necessary to maintain these beautiful machines in the best possible conditions. As these are machines there is no certain time when these machines are going to stop working. Sometimes laxity in nipping the evil from the bud about car problems can lead to car breakdown and a complete failure.



In order to avoid curse of car failure, an application is designed to monitor the car health based on the sensor data. The application will be useful to predict the failure of the part beforehand and notify the user about the part failure and prevent it.

Correcting the serious car problems is the work of a trained mechanic but monitoring car performance is a good practice for any user.

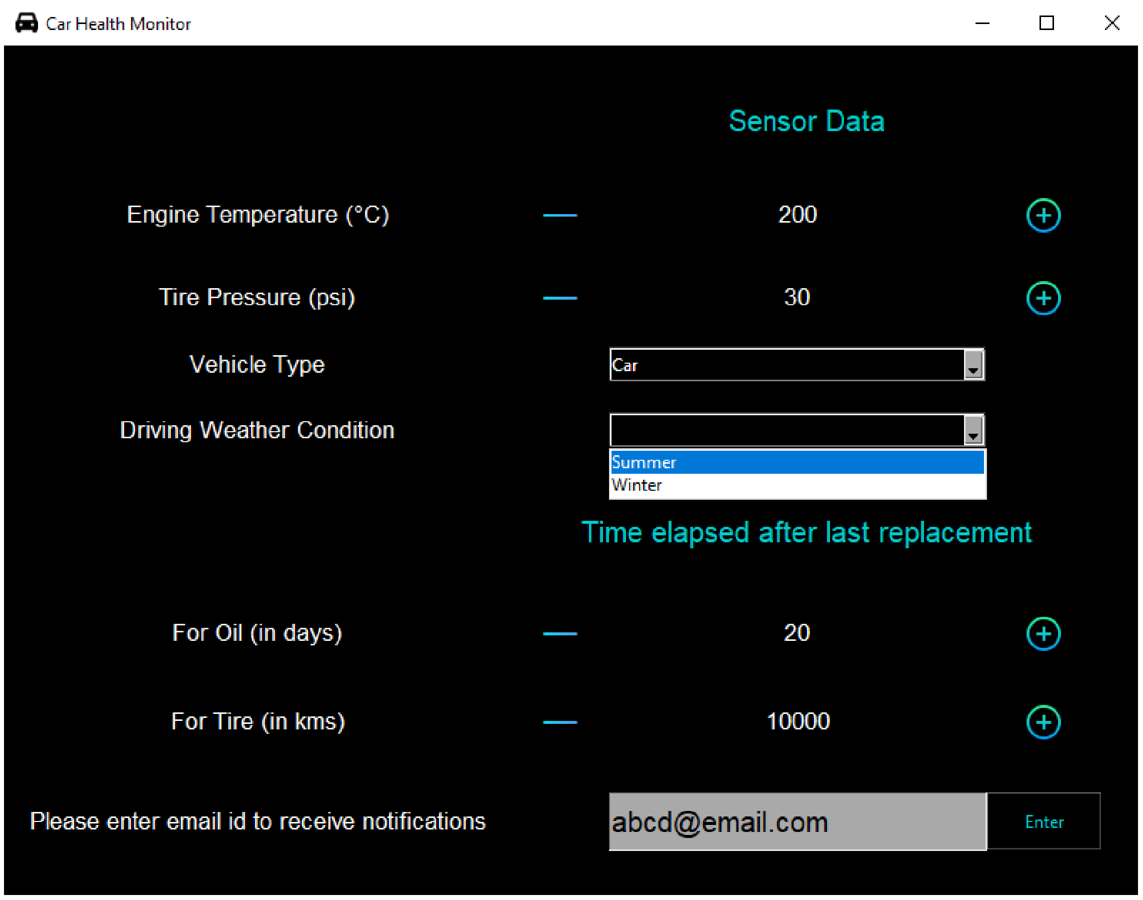
WORKING:


GUI-1
•Sensor Data
•Time Elapsed
AMQP
•Receive data of Sensor Data from GUI 1
•Receive data of Time Elapsed from GUI 2
CSV file
Stores data from both parameters
Email Notificaation system
GUI 2
Stores data and plots the graph

- Car Health Monitor is an application which is used to monitor different parameters of the car and notify the user about the component that has high probability of failure in coming future.
- Car Health Monitor is used to monitor the parameters Engine Oil, dependent engine temperature and Tire Health with the help of the sensor data.
- The sensor data is sent to a CSV file through Advanced Message Queuing Protocol(AMQP).
- The data is read from the .CSV file and is used for plotting *live failure curves* for both parameters.
- On plotting the graph if the probability of failure is above certain value, the user gets notified by an email.
- The user will also get notified if the cureent engine temperature or tire pressure is showing any abnormal behabiour.
- The email will notify which parameter of the car needs to be serviced.
- This will help the user to predict the failure of the part and to avoid unpredicitve break down of the car.

Following are the steps of execution of this application:

Graphic User Interface (GUI) - 1:



Car Health Monitor

Sensor Data

Engine Temperature (°C) — 200 +

Tire Pressure (psi) — 30 +

Vehicle Type: Car

Driving Weather Condition: Summer / Winter

Time elapsed after last replacement

For Oil (in days) — 20 +

For Tire (in kms) — 10000 +

Please enter email id to receive notifications: abcd@email.com Enter

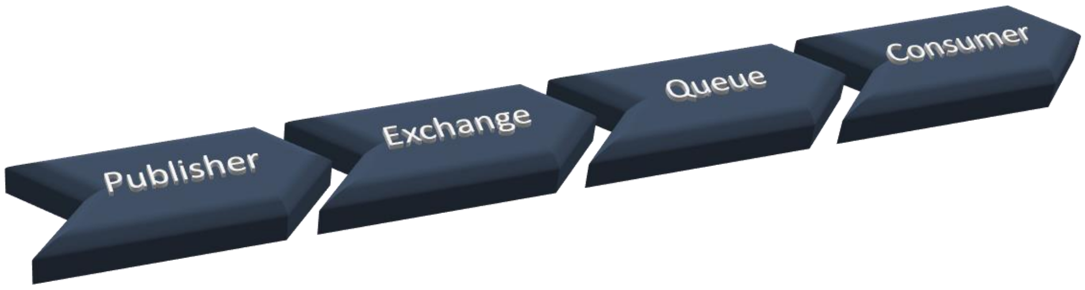
Figure 1: Car Health Monitor GUI 1

- The sensor data is the live *simulated* data that is received from the sensor. Taking actual sensor data is out of scope of the project and hence been simulated.
- The push buttons are used to simulate the increase or decrease the readings.
- Different types of vehicles have different threshold limits and those limits are also based on the ambient conditions the vehicle is driven in.
- Accordingly there a drop down menu is provided which is used to select the model of the car and other one is used to select the region where the car is driven.
- The second part is simulating the time elapsed since the last replacement of Tire or engine Oil. Based on the time the pasrt is used for, different failure graphs will be plot.
- The simulated time elapsed and sensor data is recorded after the last replacement and this data is processed and sent to the .CSV file through AMQP.
- A text box is provided where the user is requested to eneter the email-id on which the alert/notifications will be sent in case of any alarming sensor values.

- Advanced Message Queuing Protocol (AMQP):

Figure 2: AMQP

- The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for message-oriented middleware. The defining features of AMQP are message orientation, queuing, routing.
- The AMQP is implemented on RabbitMQ.
- The message is received from the publisher and the AMQP routes the message to the consumer.
- The publisher in this application is the GUI 1



- Exchanges are entities where messages are sent. Exchanges take a message and route it into zero or more queues. The routing algorithm used depends on the exchange type and rules called bindings.
- The exchange type selected is default exchange, in this you specify the routing key and the message is delivered to the respective queue.
- Queues in the AMQP 0-9-1 model are very similar to queues in other message- and task-queueing systems: they store messages that are consumed by applications.
- The consumer in this application is the .CSV file

Graphic User Interface (GUI) 2:

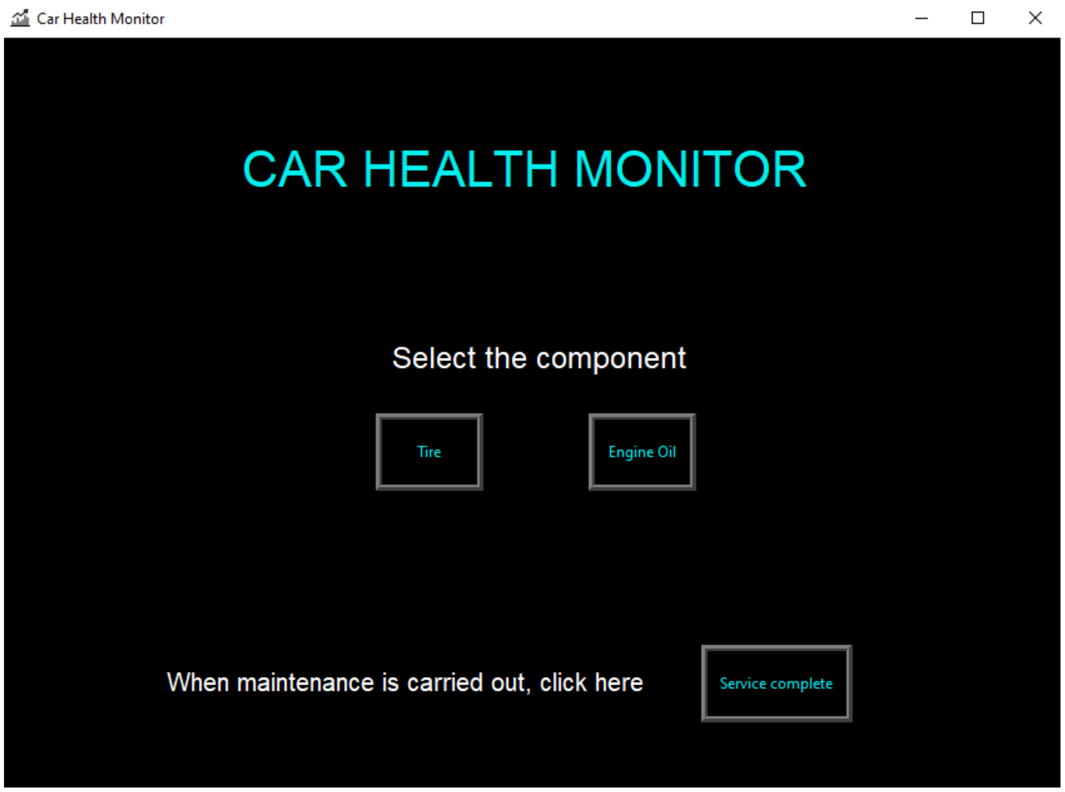


Figure 3: GUI 2 Home Page

- This is the home page of the second user interface which will take us through the display of live failure graphs.
- The home page provides you with the choice of component selection for observing live plots.
- It also instructs user to click the button “Service complete” whenever a maintenance is carried out by the user.
- The component failure graphs are governed by the following equation:

$$F(t) = 1 - e^{-\left(\frac{t}{\alpha}\right)^{\beta}}$$

  Where, β =shape parameter
  α =scale parameter
- Failure graphs of different components have different values of α and β.

	α	β
Engine Oil	5190	1.55
Tire	41667	1.37

- Upon selecting one of components we are led to one of the following frames.

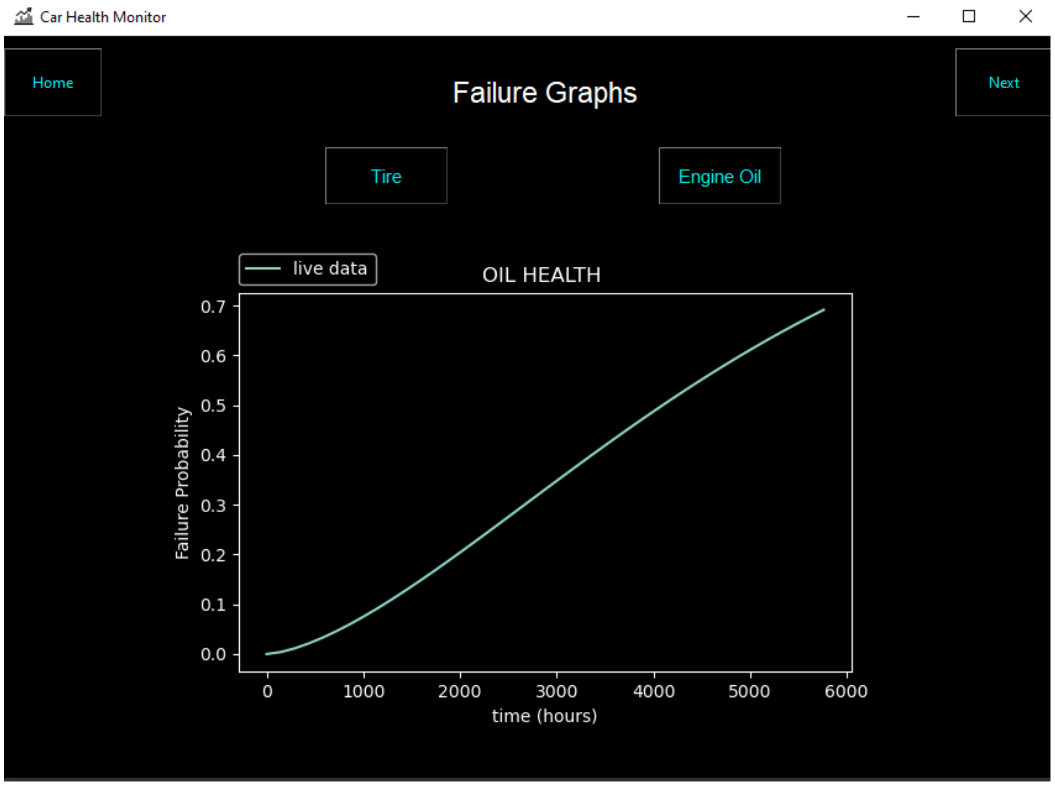


Figure 4 Failure graph of engine oil

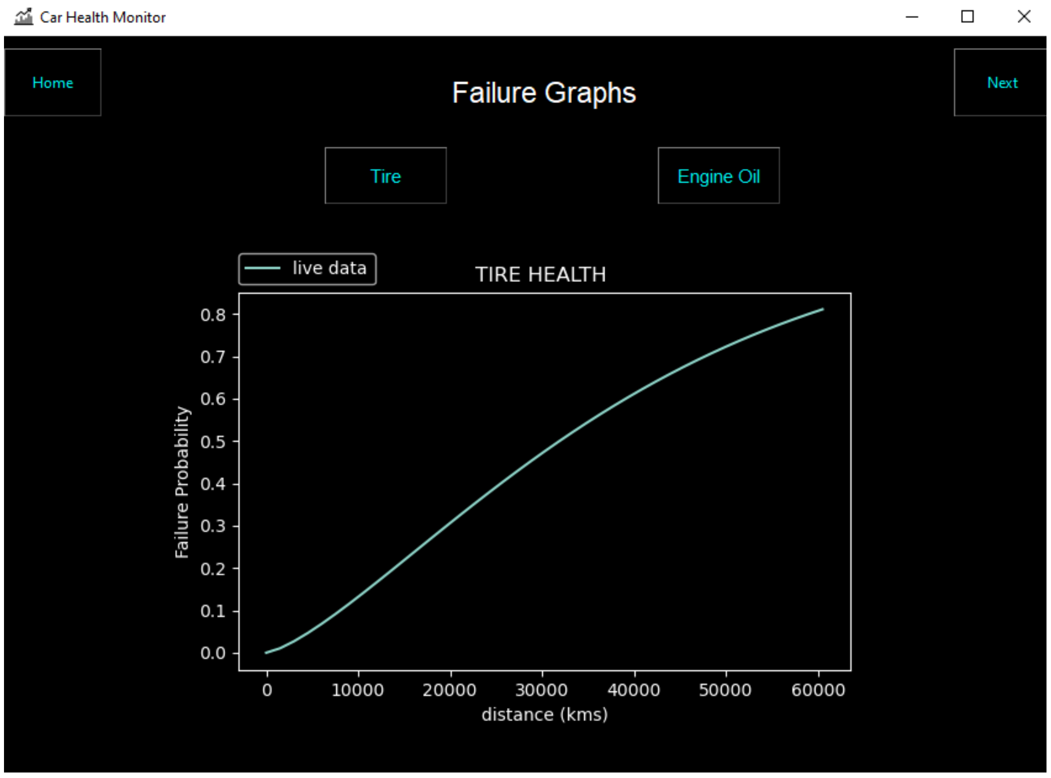


Figure 5 Failure graph of Tire

- As we can see the live failure plots of the components are plot.
- Options are provided to switch to a different graph, go to the home page or go to the next page.
- As the duration on the x-axis increases, the probability of failure on the y-axis goes on increasing from 0 to 1.
- When failure probability of any component is over 60%, an email notification is sent to the user indicating that a change/maintenance is required.
- The graph is replotted with the most recent data values after every 10secs.
- In this way, a user is well informed about the failure of certain components in the car and can plan an early maintenance.
- T-kinter module in Python3 has been used in order to develop both the GUIs.

Email Notification System:

The email notification system is responsible for checking if there is maintenance required of any component or there has been a breakdown.
To achieve this the system reads the sensor data and the simulated time information, then processes this data and checks if the user needs to be notified.

This is achieved by two files (alertsys.py and alertloop.py)

Alertsys.py

Alertsys.py contains the code for the main function of the notification system. It uses Simple Mail Transfer protocol (SMTP) integrated with Gmail for mailing services and uses pandas for reading data from the csv file and further processing it.

In the body of alertsys.py two functions are defined, function *send_email()* accepts from address, to address, cc address, subject, message, login, app password and smtp server address as parameters using which it starts a local server, publishes the email and quits the server when the function is called.

The second function *checkcond()* accepts six parameters namely flag (which is to check if we have already hit the condition, thres1, thres2, thres3, thres4 (the threshold values to compare the sensor data and the simulation time with) and email (the email to which the notification will be sent to). In this function we read the two .csv files where the data is stored (SensorData.csv and TimeElapsed.csv) and save it as a pandas data frame while giving column names to the data frames (data1 and data2). For the sensor data, we use the data frame object (data1) to extract the last 10 values and then compute their mean and compare it to the threshold values (thres1 and thres2). In a similar way we use the pandas data frame object data2 we retrieve the simulation times and compare it to the respective threshold values (thres3 and thres4). If any of the values exceeds their respective thresholds, we set the flag and we call the send email () function and pass which components need maintenance or need replacement alongside which email to notify.

Additional Features:

- For the sensor data the mean of the last ten values is compared rather than comparing it to the individual values, this enables the system to avoid noisy spikes in data and requires the data to maintain the increase for a set amount of time.
- Debug mode: Allows the namespace to be checked internally by itself by calling the function inside the same code.

Alertloop.py

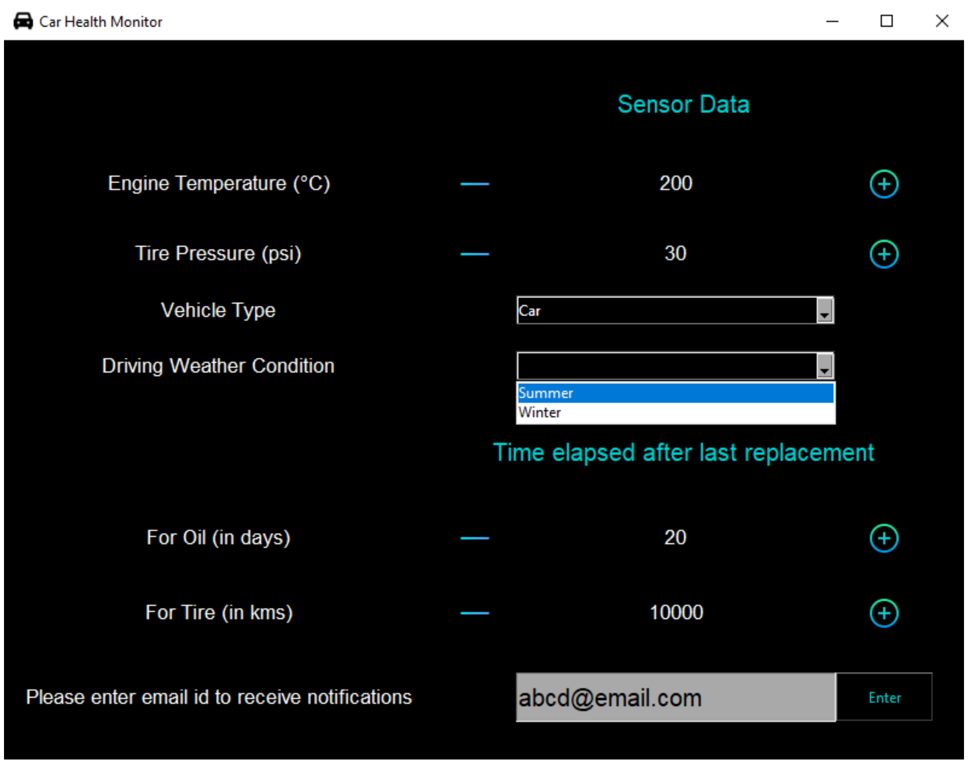
The main function of alertloop.py is to update the thresholds based on the selected categories and set the email. After which we continuously poll the checkcond function every 5 seconds from alertsys.py until the flag is triggered.

The body of this file consists of two things, a function select thresh() which reads the file parameters.txt as a json dictionary and retrieves the categories and the email set by the user in the simulator. Based on these selections the threshold value is selected. The criterion for threshold selection is shown in the table below:

	Car	Truck
Summer	• Engine temperature threshold = 106 C • Tire pressure threshold = 30 psi	• Engine temperature threshold = 115 C • Tire pressure threshold = 34psi
Winter	• Engine temperature threshold = 100 C • Tire pressure threshold = 33 psi	• Engine temperature threshold = 110 C • Tire pressure threshold = 37 psi

- Tire max distance threshold = 60000 km
- Oil replacement threshold = 5000hrs

The second part of this code consists of a loop that infinitely polls the *checkcond()* function from alertsys.py file every 5 seconds with the updated thresholds and email address.



Engine Temperature (°C) 200

Tire Pressure (psi) 30

Vehicle Type Car

Driving Weather Condition

For Oil (in days) 20

For Tire (in kms) 10000

Please enter email id to receive notifications abcd@email.com

Figure 6 GUI 1

- The sensor data is the live data that is received from the sensor. Taking live data is out of scope of the project.
- The push buttons are used to increase or decrease the data which represents the live data.
- The sensor data changes based on the make of thee car and the region where the car is driven
- Accordingly, there is a drop-down menu which is used to select the model of the car and another one is used to select the region where the car is driven.
- The time elapsed is recorded after the last replacement and this data is processed and sent to the .CSV file through AMQP
- The email text box can be used to set the email to which the user would like to receive the notification to. (default: rain.cloud.bot@gmail.com)

The main features of the GUI are described above. The code for the GUI is written using the Tkinter module in python which involves creating various sub objects of the module like a label, button, combobox, etc., followed by linking them to their specific functions and finally placing them in a grid.

After having created an interface which updates values for Engine temperature, tire pressure, the selected category 1 & 2 and allows the user to simulate time for oil replacement in days and time replacement in months, all this data is saved in a dictionary and sent to the reciever.py file using AMQP every 10th update (each update = 1sec) or 10th second.

Obstacles and Solution:

1. **GUI (obstacles):**
    a. Background image could not be inserted easily in GUI2 since a matplotlib graph was getting plot simultaneously.
    b. The placement of buttons, Labels is very tedious if. grid or. place commands are used.
    c. Not many design features were available to make the GUI more attractive.
    d. Tried using the current window_info for screen_height and screen_width in order to set the size of GUI frames, but somehow it did not work the way it was supposed to.
2. **AMQP:**
    a. RabbitMQ requires a server that needs to be running in order to publish and consume the data. A RabbitMQ server had to be created in order to keep it running.
    b. Data was not consumed by the consumer even after publishing the data. Multiple tests had to be conducted on a sample file and after debugging the error the system was implemented in the application.
3. **Other:**
    a. Running multiple .exe files which were dependent on multiple .py files was challenging.
    But using the package of auto-py-to-exe it was easy to convert the .py files into .exe bundling them up with their dependent files.
    Also, a new .py file was created (carHealthMonitor.exe) which is the final executable file for running all the other executable files in the folder.

Conclusions:

The project was successfully completed with all the decided parameters.

- ✓ The GUI design is implemented successfully.
- ✓ Sending the data from GUI to store it in the .CSV file is achieved.
- ✓ Based on the data received the stored in the .CSV file the graphs are plotted.
- ✓ The data from the sensor is used to predict the Failure Probability.
- ✓ An email notification system is set up to notify the user in advance about the part failure.

Limitations:

- 👎 The Car Health Monitor predicts failure of the just two parameters.
- 👎 The sensor data was not based on real sensor data and had to be simulated.
- 👎 Tkinter was a basic package to develop GUI giving restrictions in certain designs.
- 👎 Receiver and notification system open in command windows.

Future Scope:

- The number of parameters can be increased.
- A better package can be used for developing GUI (e.g. Kivy)
- More number of performance graphs can be displayed for different components (e.g. Weibull distribution, Bathtub curve, etc.)
- Receiver and notification system should open as background process