Driver Alertness Detection

Chameera De Silva

Our proposed solution for the problem statement LB9 by Chameera De Silva Organization has 5 major aspects:

- Drunk driver detection System

- Heart pulse detection System

- Road Speed Alert System

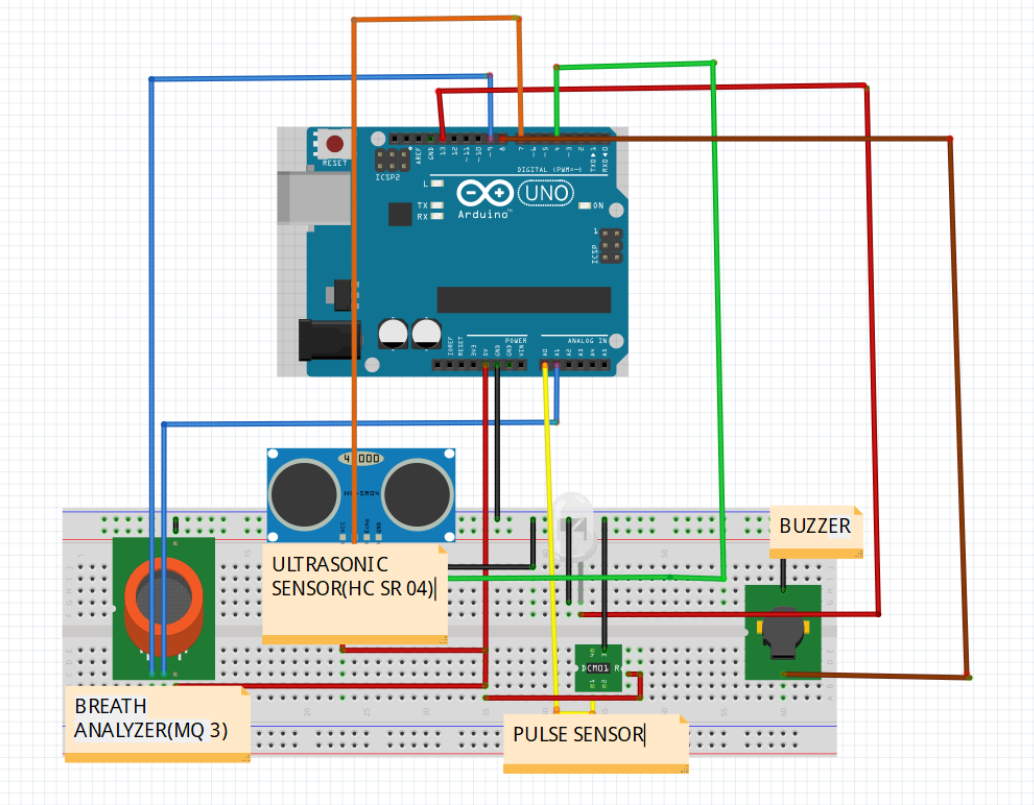
- Driver Drowsiness and Distraction Detection

- Collision Detection System

These Solutions are integrated together on a *Raspberry Pi 3* and an *Arduino Uno microcontroller*.

Components Used:

1. HCSR-04 Ultrasonic Sensor
2. Arduino Uno
3. Buzzer
4. MQ3 Grove Gas Sensor
5. Pulse Sensor
6. Android Phone with camera
7. Mac Book Pro



The following applications were implemented on an Arduino Uno Microcontroller board, using with the appropriate sensors:

* Alcohol Detection using the Grove Gas Sensor (MQ3)
* Collision Detection System using and SR04 ultrasonic receiver-transmitter
* Heart Pulse Detection using the Pulse Sensor

Alcohol Detection System:

An alcohol detector was developed using the MQ3 Grove Gas Sensor, which has been interfaced with the Arduino board. The driver is required to blow into the sensor’s receptacle. The gas sensor gauges the amount of alcohol present in their breath by measuring the change in voltage brought about by the incidence of ethanol on the SnO2 layer. In order to ensure that the driver has blown into the alcohol sensor, a carbon dioxide sensor will be used to detect the carbon dioxide in their breath. If the amount of alcohol in the driver’s breath exceeds 200 mg/l, the engine will not start, thus being instrumental in the prevention of drunken driving.

The following variables have been used in the program:

1. value: reads the input from the analogy pin of the Arduino board, stores the reading of the sensor
2. inp1: input given to the system that indicates the presence of carbon dioxide
3. co2: converts inp1 into a 1 digit input, thus making comparisons simpler

If the contents of variable value are greater than 200, the LED and buzzer will be triggered, and the ignition cannot be turned on.

Heart Rate Detector:

A heart rate detector has been implemented using a pulse sensor along with the Arduino board. This sensor is used to ensure that the driver is medically fit to drive. It constantly monitors the heart rate of the driver by the means of photoplethysmography using a sensor that can be embedded in either the steering wheel or the seatbelt. If the heart rate does not fall within 60 and 100 BPM, the engine will not start, as the driver is at the risk of having something happen to them while driving. The Pulse Sensor has an LED and a phototransistor which is used to detect blood flow. The LED blinks at an interval of one second and the phototransistor receives the reflected light. Depending on the intensity of the received light, the amount of blood in that artery can be gauged. This process is called Photoplethysmography and is used to detect the pulse.

Road Speed Alert System

An Android app was developed that takes the current location that the driver is driving in and retrieves the speed limit for that road from the Maps Api Service. For demo purpose, a webpage was created that allows the tester to enter the speed of the car and the speed limit for the car.

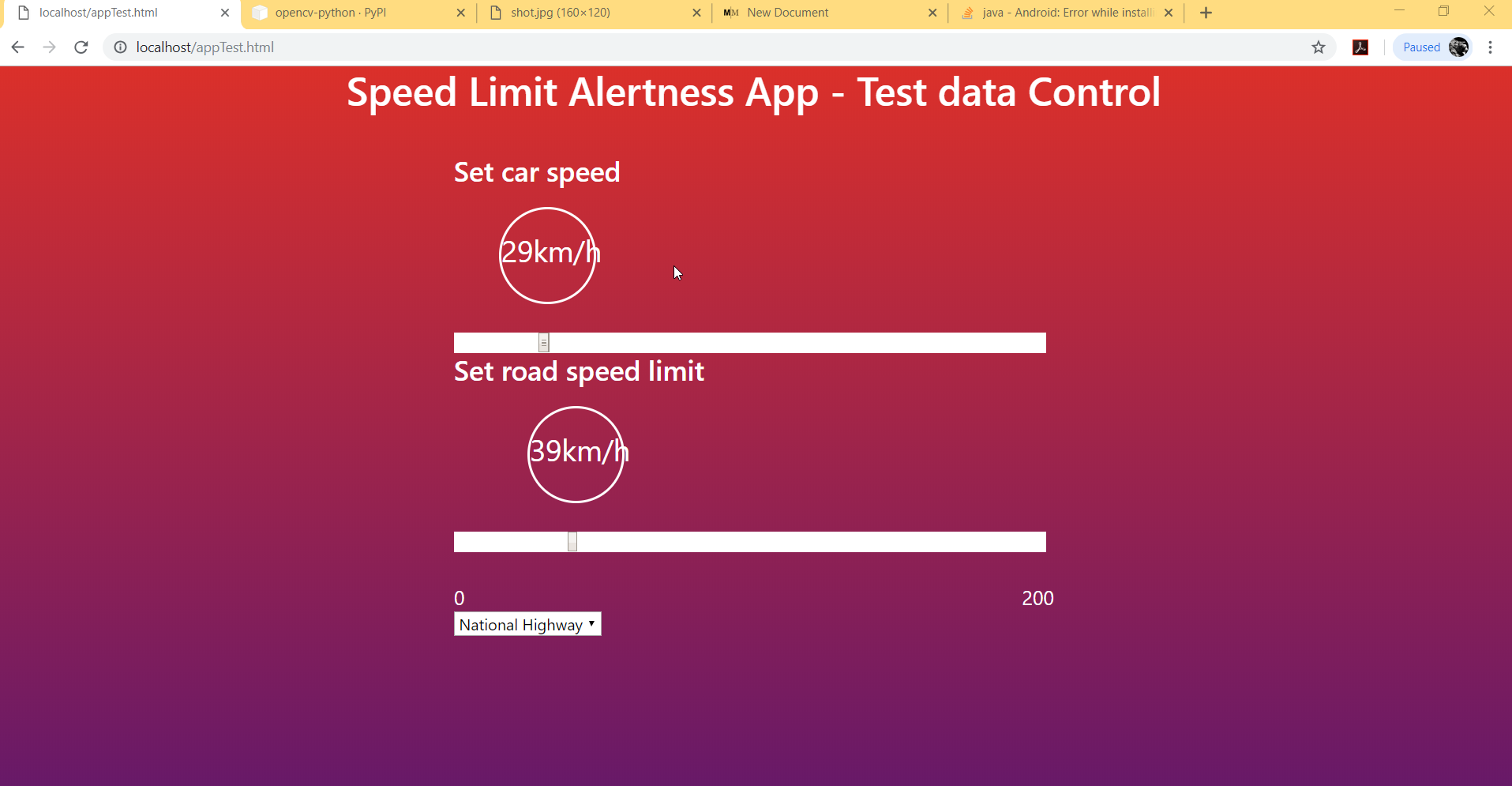


Figure Test inputs webpage

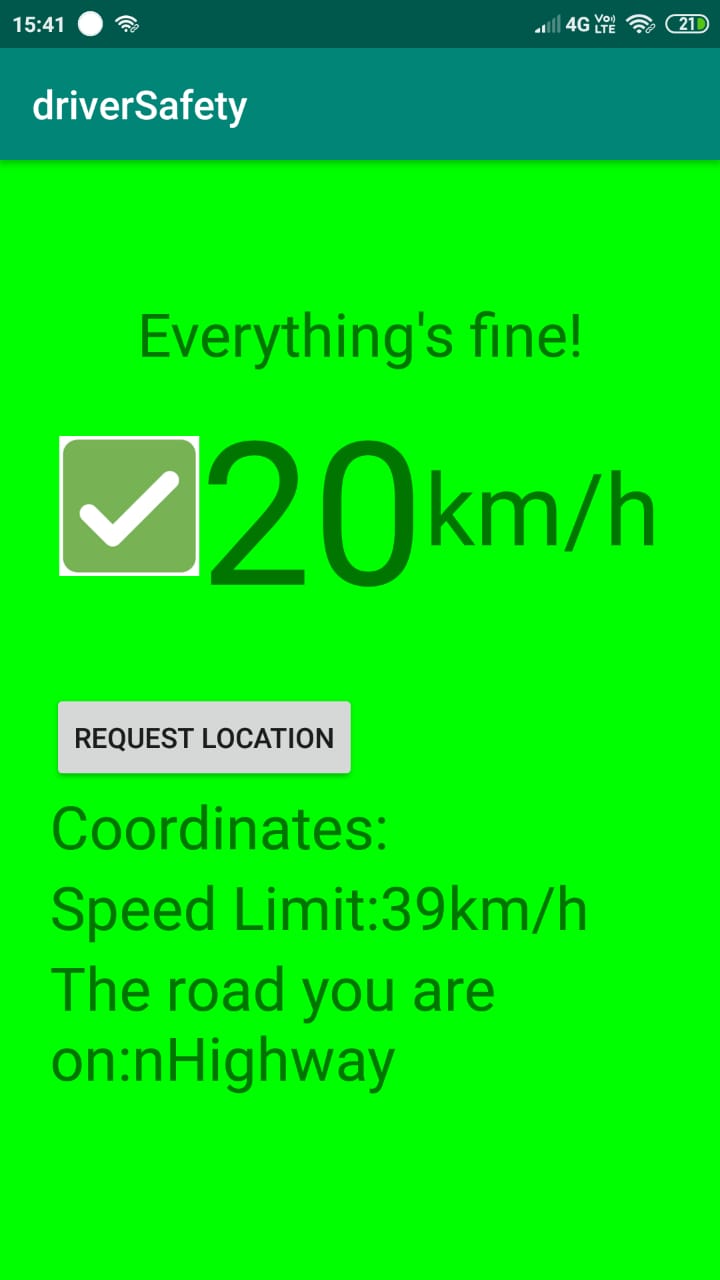
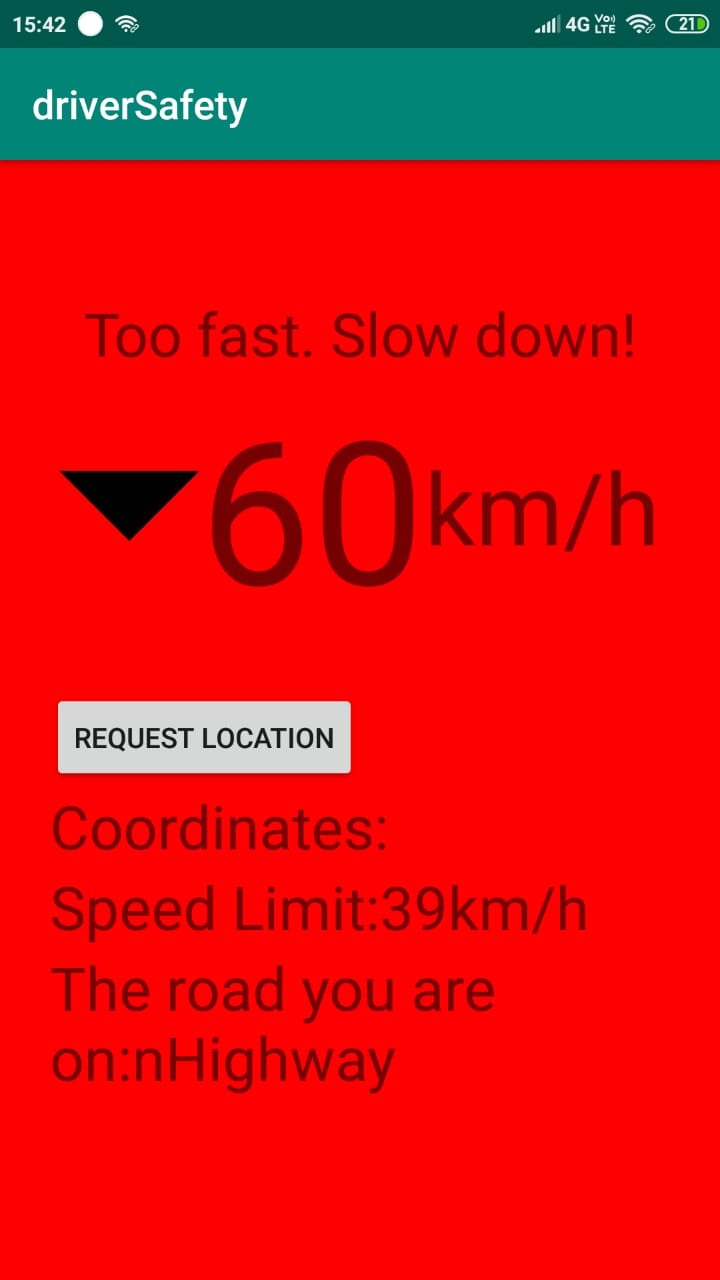


Figure 2 Screengrabs from the app

The app does a GET request to the server and retrieves the uSpeed, speedLimit, road values. The app triggers a voice alarm to notify the driver of overspeeding if the uSpeed is greater than the specified speed limit for the road. It uses the Android Text-to-Speech library to notify the driver.

The test server is in the folder /sih\_mindtree\_lb9\_inferno123/SpeedLimitApp/testServer

The appTest.html file is the launch file which sends a GET request with the values in sliders Car Speed and Speed Limit to the server.php file.

The Android application can be built by cloning the folder at /sih\_mindtree\_lb9\_inferno123/SpeedLimitApp/AndroidApp.

The url for the GET request can be can be set by modifying the requestURL variable in MainActivity.java file.

For demo purpose, the API is expected to return a uSpeed,speedLimit, road values.

In the real-life scenario, the Raspberry Pi will upload the current speed as uSpeed to the local server, which the mobile app will call. Given current location co-ordinates, the actual API is expected to return the speedLimit for that location.

**Drowsiness and Distraction Detection**

The driver drowsiness system uses a HOG based image detection algorithm to detect 68 points on a face. The system uses a pre-trained dataset at /sih\_mindtree\_lb9\_inferno123/shape\_predictor\_68\_face\_landmarks.dat file. The file is uploaded as zip parts due to size restriction on github. Download them and extract together using WinRAR. The system also uses HAAR cascades to detect the ears in a non-frontal face. The HAAR cascade XML file is at sih\_mindtree\_lb9\_inferno123/ haarcascade\_mcs\_leftear\_3rdparty.xml

The face detection code is at sih\_mindtree\_lb9\_inferno123/detect.py

The list of packages that need to be installed are

1. Numpy
2. Scipy
3. Threading
4. Imutils
5. Dlib
6. openCV
7. urllib

The dlib library might need some work to be installed since the package needs to be compiled from source. Follow this for information about installing the dlib library on a Windows Machine <https://www.learnopencv.com/install-dlib-on-windows/>

The present code uses an Android App IP Webcam to stream the camera source to the computer. The app can be configured and the IP Address shown should be entered at the url variable in detect.py file.

The code is well commented with what each function does. The variables eye\_thresh, drowsyFrames.

The marking regions thresholds are on line 29 to 37. These tell the part of the image in which the detection of face is allowed.

Once the dependencies are installed, the path to the HOG model can be defined at line 139 in dlib.shape\_predictor(...) and the Haar Cascade for ear detection can be set at line 53 in l\_ear\_cascade variable.

The driver face detection program can be started by calling python detect.py at the terminal.

Successfully taken care conditions under bright light conditions:

1. Yawning
2. Closed eyes
3. Both Yawn and Closed eyes
4. Non frontal face detection
5. Face detection within specified boundary region

Failed cases under bright light condition

1. More than one face cannot be detected accurately simultaneously
2. Haar cascade gives false postitives at times
3. Doesn’t work in low-light conditions
4. Eye width to height ratio is different for different people and is not generalized.

Collision Detection System :

A collision detection system has been designed using an SR04 ultrasonic receiver transmitter.

The SR04 is a range finder that is used to measure distances by means of echolocation. When connected to the Arduino board, the distance can be determined by multiplying the speed of sound in air with the time interval within which the ultrasonic pulse is sent and the reflection is received. By the same method, the distance between two cars, ***x****,* is determined. The system will have access to the speedometer of the car, and thus the velocity **‘*v’***will be known. Taking the average braking deceleration as 6.56 m/s2 , we can calculate the distance travelled by the car before it comes to a complete stop once the brakes have been applied.

The formula is as follows:

where ***s***is the necessary stopping distance, ***v*** is the speed of the car and ***d*** is the average braking deceleration, which is **6.56 m/s2**.

If the stopping distance, s is greater than the true distance, x, then the car will not stop in time upon application of brakes and thus the message “Collision is imminent” will be displayed.

The speed of sound in air is defined as a function of the ambient temperature in **°**C, as ***u = 331 + 0.6\*temp.***

The following variables have been used in this program:

1. currdistance: the distance between our car and the car in front of us
2. truedist: the distance that will be travelled by our car before it comes to a complete stop when the brakes are applied
3. decel : average deceleration of the car when brakes are applied
4. speedometer: stores the speed of the car at that instant in km/h
5. spd: speed of the car in m/s
6. temp: ambient temperature in **°**C

Thus the designed system informs the driver when the brakes must be applied.