ShellHacks 2020

Drowsy Detector - a lifesaver

Team:

Darshan Solanki

Marshall Mason

Ryan Mitchell

Shayan Riyaz

Overview

According to a study conducted by Virginia Tech Transportation Institute in 2013, 20 percent of car crashes are caused by fatigue, with young drivers practically vulnerable to driving while fatigued. The was a significant jump from collected statistics in the preceding years as the attribute to only 2 or 3 percent crashes[1]. In 2020, according to the National Sleep foundation about half of U.S. adult drivers admit to consistently getting behind the wheel while feeling drowsy. About 20% admit to falling asleep behind the wheel at some point in the past year – with more than 40% admitting this has happened at least once in their driving careers[2]. With numerous other resources to support this claim, without a question, it can be stated that drowsiness/tired driving is a prominent contributor towards the number of fatal car crashes annually (AAA Foundation for Traffic Safety)[3].

At the same time, the development of drowsiness detection technologies is both an industrial and academic challenge. Some modern examples are Volvos Driver Alert Control. Which warns drivers suspected of drowsy driving by using a vehicle-mounted camera connected to its lane departure warning system (LDWS). Mercedes-Benz collects data drawn from drivers driving patterns through its Attention Assist System. While these solutions are effective they have two major drawbacks, they are costly and are limited to very few cars at this point.

Our project introduces a concept for preventing accidents from the drowsiness that is both affordable, reliable, and simple to implement. By utilizing Machine Learning, this project encourages safer driving by providing the user with viable driving and travel solutions. Our software leverages the use of the prominent machine learning libraries for computer vision along with the Cloud services such as Google Cloud Platform. The application allows users to safety over recklessness by helping find ways to rest in the middle of a tiring journey.

We hope this project highlights the importance of road safety and teaches people to be mindful and cautious about their fatigue especially when on the road. The goal of this app is to create a safer world for everyone.

Demo Walkthrough:

The complete codebase can be found at
https://github.com/ShayanRiyaz/Drowsy-Detector

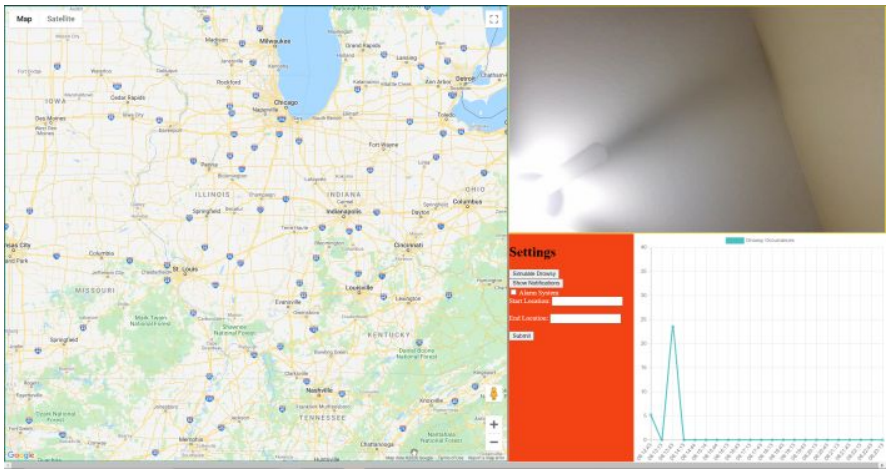


Figure 2.1

Stage 1:

Shown in fig 2.1, we have 4 screens. Screen notifications allow the driver to know about the alerting messages when drowsiness happens. Settings are responsible for multi-use case:
Simulate Drowsiness: To track the start of the journey with live feeds on the webcam
View Graph: Give out real-time analysis of # of time drivers have snapped in a given interval of 30 secs and keeps on updating.
Alarm Button: Responsible for setting up video alarm if drivers want to notify him/her with sound
Start loc: Driver's start location address which automatically maps into lat, long using google place API
Des loc: The driver's end location also in the form of string.

One the left-hand right corner, we have a webcam where we detect the driver's face and trigger alerts using the open-cv and dlib library. On the right, we have Google Maps API integration which allows us to track the route path and custom-made makers for simulation of the driver's journey along the way.

Stage 2:

Next, we input the driver's location, trace the map route on google maps. Set the custom markers that would be the path locations the driver would cross along its journey shown in fig 2.2. Since, GPS is not enabled due to time-constraint, for the prototype we had shown a simulation of the coordinates which are near to the destination shown in fig 2.3. Otherwise, given any current driver's location, we should be able to recommend nearby rest places.

We can see the contours are getting generated by the model and when the driver blinks the eye or yawns at times, Status Notifications will have Drowsy messages flowing in. We can also see the graph functionality of dynamic update on drowsiness occurrences over the period of time.

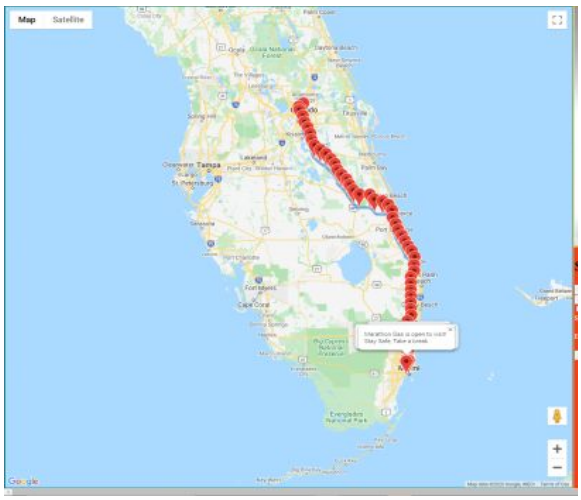


Figure 2.2

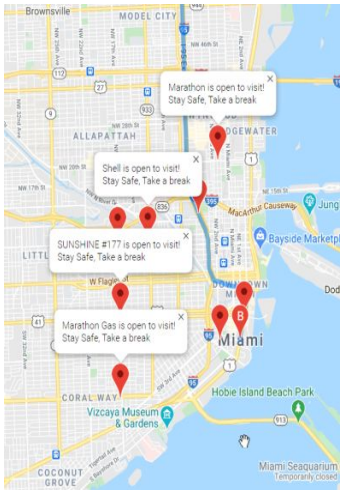


Figure 2.3

Stage 3:

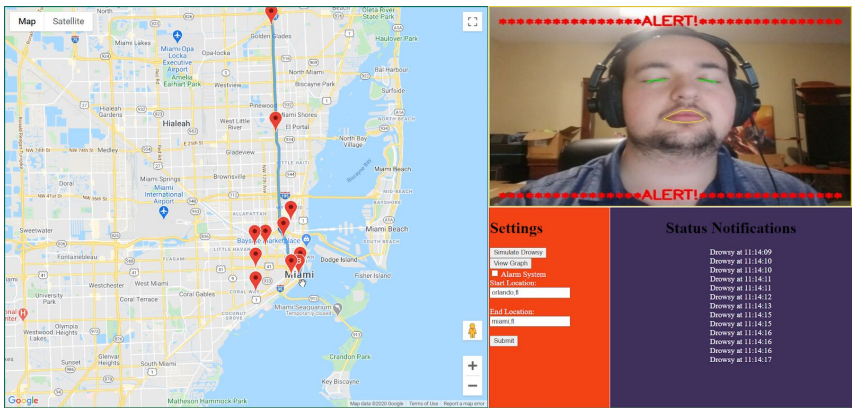


Figure 2.4

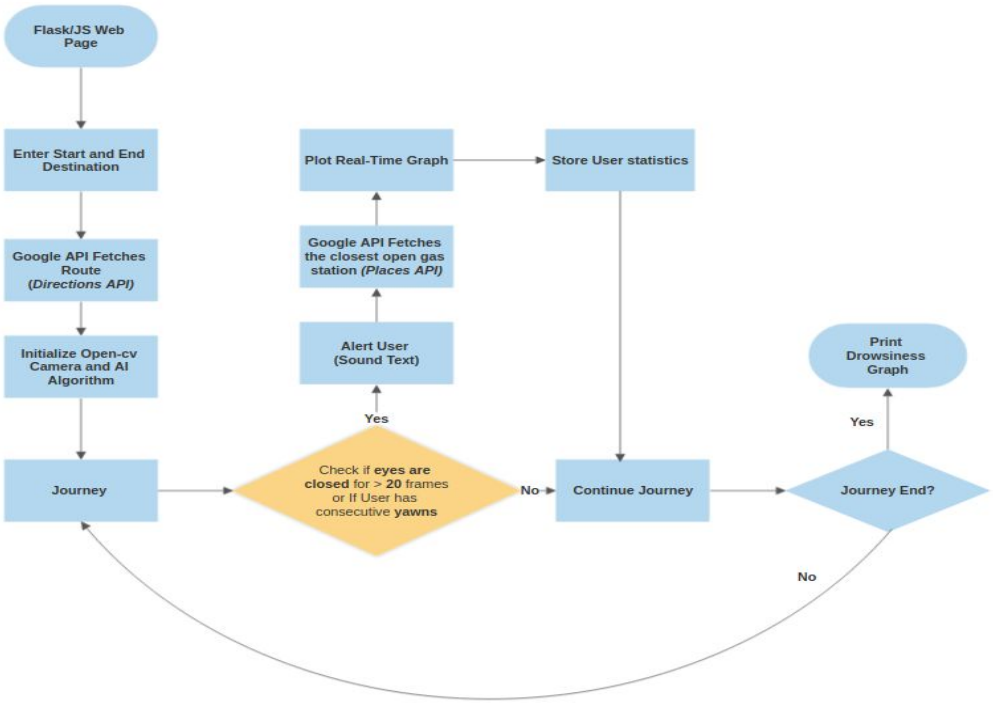
These alerts trigger the place recommendations given the current driver location. As shown in figure 2.4, the recommendations are given where we display things like:

“Name of the Place”
“Place Open or Not”
“Short Description Message”

So the algorithms work computing the distance haversine formula and filter the places according to the type and proximity. This way our demo helps mitigate the accident failure.

Workflow and Technical Implementations

FlowChart



Flow chart structure

1. The Flask/JavaScript web page is loaded. The dashboard displays
    - Google Maps
    - Camera interface
    - Menu with settings for drowsiness tracking.
    - The real-time graph to show the drowsiness instances over the period of the journey.

2. The driver inputs their start and end destination. Our program uses the Google **Directions API** to fetch the complete route map along with latitude-longitude coordinate markers throughout the route.
3. The camera is initialized and uses a pre-trained machine learning model in order to detect the 68 landmark points on the face of the driver.
4. During the journey, the user is warned visually and with an audio cue on whether or not they are perceived as drowsy.
5. If the drowsiness persists the user is informed about the closest gas stations (fetched using the Google **Places API**)
6. If the computer vision algorithm detects any drowsiness, the instance is plotted on the drowsy instances vs time graph.
7. If the journey is finished. The user Graph is plotted showing the frequency of the instances.

Environment Requirements

Software Stack

Languages and Frameworks: Python, JavaScript, HTML, CSS
Computer Vision: Opencv-python, Scipy, Dlib, Imutils
API-Requests: Google Cloud Platform, Google-maps *(Py/Js)*
Data Extraction/Mapping: Numpy, Json, Chart.js
Web-server: SocketIO *(Py/Js)*, Flask,
Formatting: Prettyprint

Software Compatibility

Our project mainly involved the use of Windows and Linux OS. The IDEs and supporting software used for this project include Visual Studio Code, PyCharm IDE, Postman API, Python Idle, Google Colab, Google Cloud Platform.

Computer Vision

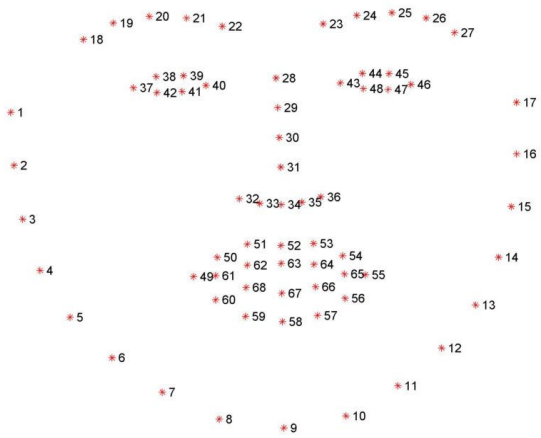


Figure 3.1: Facial Features being tracked.

Implementation with OpenCV and dlib

So, we have used dlib's 68_shape_predictor pre-trained models[5,6,7,8] to predict the driver's live video feed into 68 point landmarks as shown in fig 3.1.
We used OpenCV to work with video capturing, did initial preprocessing of converting video into streams of image frames, with customized image size along with internal grayscale conversion for reducing model complexity.

We take into account two features for drowsiness detection:
EAR - Eye Aspect Ratio
MAR - Mount Aspect Ratio

EAR - computing the ratio of the eye's height to width. Assumption: when we feel drowsy, our eyes tend to get smaller and the ratio of height to width decreases.

Formula for Computation: (leftEyeEAR + rightEyeEAR) / 2
For instance leftEAR cal: (|p38 - p42|| + |p39 - p41|) / |p37 - p40| (ref fig 3.1), it's taking the euclidean distance of extreme points on width and height and computing the EAR score.

MAR - works in an identical fashion, computing the ratio of the distance between height to width.
Logical Rationale - when we yawn, the mount size increases in height and lowers in width so that makes the MAR ratio increase while yawning. This ratio determines the state of a driver and is therefore a good indicator of drowsiness detection.

This makes our detector system robust as we track down live feeds of drivers and give the danger signals in terms of notifications and Alarms.

Also, we have provided further mitigation steps on how to avoid accidents or any disaster.

Records Driver's History in Real Time:

So, our system is capable of handling real-time track on how many times in given time-intervals of 30 secs the driver has snapped his eyes. This helps in keeping track of the driver's history and traits which can be further used to extrapolate predictions intelligently on a driver's past record and whether he is fit for the journey.

Server and Web Sockets

Flask

The Flask library on Python was used to implement the backend of our server. Flask allows for many operations such as delivering an HTML page with the according to CSS and javascript files, redirecting pages, or supplying other files such as images or audio clips.

SocketIO

In addition to Flask for supporting our connection of backend and frontend, we implemented SocketIO to handle communications such as drowsiness detection or longitude and latitude coordinates to be processed on the backend.

Google Cloud Platform

Directions API

The Directions API is a service that calculates directions between locations using an HTTP request. The API returns the most efficient routes when calculating directions. Travel time is the primary factor optimized, but the API may also take into account other factors such

as distance, number of turns, and many more when deciding which route is the most efficient. In our case, we set our Directions API mode to '*driving'.*

By using the directions API from the google cloud, users are able to fetch a route from destinations of their choice. A path is plotted and marked by using GPS latitude and longitude coordinates. This allows us to simulate our user's movement through the planned route.

Places API

Utilization of the Places API allowed us to retrieve data from google to plot points along or nearby the travel route and suggest places for the driver (user), to pull over and rest. By employing certain tags, optimal rest stop locations were suggested within the minimum and maximum viable distance. Another important criterion was that if the rest station was open at that point in time.

Data Extraction

Latitude and Longitude Scraping

Due to unavailability and lack of time for a live GPS tracking option for this project. We implemented a script for latitude and longitude extracting between two points. The reason we chose to use a custom extractor on top of the present google-maps directions API is that the directions API strictly gives us latitude and longitude on turns and route directions. This limits full coverage of our route and limits finding locations for the user to rest in case they are fatigued and need a rest stop.

In order to map points along our route, we simulate a car moving between two points. This procedure plots points along the route. To set our space to a latitude and longitude, we use the **Haversine formula.** The Haversine formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation. The haversine can be expressed in trigonometric function as:

$$\text{haversine}\left(\frac{d}{r}\right) = \text{haversine}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\text{haversine}(\lambda_2 - \lambda_1)$$

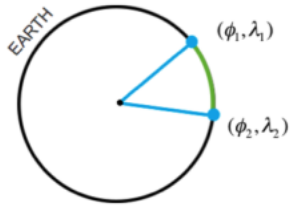


Figure 6.1: Spherical triangle solved by the law of haversines

First, the script generates an *OrderDict* of points. In this, our timestamp is the key while our value is the (latitude, longitude) coordinates in a tuple form. The coordinates represent the location of the user at that point in time. ***Note: Time is relative, with t=0 indicating the time when the journey begins.***

The script calls the google maps directions API to get the directions from a given starting address to a destination address. Another issue that arises with using google maps other than lack of coordinates is that depending on routes google maps does not give a smooth set of points. Therefore for some parts of our route, we may have a clustered number of (latitude, longitude) pairs while for others it may just be a single point. Overall, this script interpolates points and smoothens it so that you get a (latitude, longitude) pair ~5 seconds.

Mitigation Strategy

To avoid accidents and disaster, we have come up with 3 mitigation strategies

- The driver's route path has been traced on the google maps.
- We have bifurcated the path into coordinates of latitude and longitude to simulate the driver pathway.
- Then in the middle of the journey, when the driver snaps, the alert is generated on the Status Notification stating: "Drowsiness Alerts", we also provide Alarm Feature when he/she enables the alarm, and when the alerts come in the alarm would ring up aloud to distract the sleep.

Next Mitigation is to give out the Recommendation of nearby places according to the current driver location. As GPS is not used, we are prototyping on giving out for one of the coordinates of the driver's trajectory. Google's API was used to fetch 20 places given a certain radius and lat-long.
Custom filtering was done on the JSON responses of places on conditions like:

Is the place Open to Visit?
Is the place of the category "gas_stations", "rest_area", "food court"?

This certainly helps the driver to navigate to the nearest suitable location to take a break and then continue with the journey.

Future Work

We can expand our work by adding more functionalities to make application Robust:

- Produce Neural-Net based facial recognition models as the model used has a drawback of not capturing the side face images as well as when the image size is small. In that case, producing DL models using CNN for object detection and extraction would give more flexibility and achieve significant performance reliant.
- Enable GPS to live track the user's location and compute onDemand Nearby places, due to time constraint we simulated for one of the coordinate locations.
- Recorded history travel would help us determine how one performs in which type of journey. This analysis would help car renting companies to assign drivers based on their expertise.

References

[1] Walter, Laura. "**Wake Up and Drive: Fatigue Causes 20 Percent of Crashes.**" *EHS Today*, 5 June 2013, www.ehstoday.com/safety/article/21917988/wake-up-and-drive-fatigue-causes-20-percent-of-crashes.

[2] "**Drivers Are Falling Asleep Behind the Wheel.**" *National Safety Council*, www.nsc.org/road-safety/safety-topics/fatigued-driving.

[3] **Drowsy Driving NHTSA reports**. (2018, January 08). Retrieved from https://www.nhtsa.gov/risky-driving/drowsy-driving

[4] Jabbar, Rateb, et al. "**Real-Time Driver Drowsiness Detection for Android Application Using Deep Neural Networks Techniques.**" *Procedia Computer Science*, Elsevier, 24 Apr. 2018, www.sciencedirect.com/science/article/pii/S1877050918304137.

[5] C. Sagonas, E. Antonakos, G, Tzimiropoulos, S. Zafeiriou, M. Pantic. **300 faces In-the-wild challenge: Database and results**. Image and Vision Computing (IMAVIS), Special Issue on Facial Landmark Localisation "In-The-Wild". 2016.

[6] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, M. Pantic. **A semi-automatic methodology for facial landmark annotation**. Proceedings of IEEE Int'l Conf. Computer Vision and Pattern Recognition (CVPR-W), 5th Workshop on Analysis and Modeling of Faces and Gestures (AMFG 2013). Oregon, USA, June 2013.

[7] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, M. Pantic. **300 Faces in-the-Wild Challenge: The first facial landmark localization Challenge**. Proceedings of IEEE Int'l Conf. on Computer Vision (ICCV-W), 300 Faces in-the-Wild Challenge (300-W). Sydney, Australia, December 2013.

[8] One Millisecond Face Alignment with an Ensemble of Regression Trees DOI:10.1109/CVPR.2014.241