Section I - A short overview of solutions proposed by other companies

Introduction

According to available statistical data, over 1.3 million people die each year on the road and 20-50M people suffer non-fatal injuries due to road accidents[1]. Statistics show that one of the main causes of road accidents are driver drowsiness[2] and cell phone distractions[3].

One solution for that could be recognizing whether the driver is looking straight and concentrating on the road or not by taking livestream video of the driver and judge the driver's condition on real time. However, there are some difficulties with this solution. First, it contains detecting the driver's eyes, a small feature in the face. Second, we should analyze if the driver is falling asleep, looking down or just blinking. In addition, the light through the day changes and at night it is harder to fetch the driver's facial features.

In the commercial market, there are several solutions to alert the driver in case he is not concentrated on the road. One of them is 'CarSafe'[4] – a mobile app developed by researches from Mexico and Italy. CarSafe uses dual-cameras functionality on smartphones by switching the usage of the frontal and back camera. The app analyses data from both cameras (under the assumption of correct camera location), detects any dangerous driving conditions inside and outside the car, and alerts the driver in real time if any dangerous situation has been identified.

Additional solution, which is also the most similar to our product, is the DrowsyDriver[5] mobile App. Developed as a master thesis by an Indian computer science researcher, the app's goal is to solve a somewhat identical problem as we do, although the suggested solution is based on slightly different approaches.

We built a system which alerts the driver if he's not concentrated on the roads, based on the criterions we mentioned above. In this article, we are going to discuss its methods and explain its algorithm.

Section II -

System Description

Challenges the System Had to Face

[1] Global Status Report on Road Safety 2009; World Health Organisation (WHO): Geneva, Switzerland, 2009.
[2] "Drowsy driving is the cause of at least 100,000 auto crashes each year", UCLA Sleep Disorders Center.
[3] "Cell phone use while driving leads to 1.6M crashes each year", The National Safety Council report
[4] Description of CarSafe's driver safety app and working process
[5] https://play.google.com/store/apps/details?id=mano.facialprocessing

During the learning process on the project, we have encountered with few main challenges. First, we had to handle the differences in lighting conditions and contrast throughout the day. While it was easy to detect the driver's face in day time when there were no clouds in the sky, it was match harder to recognize him in the night.

Second, we have come across with finding the optimal location for the camera. We noticed that incorrect positioning could easily lead us to false positive matchings in high rates and potentially false negatives incidents. Using a mobile holder, when the camera is mounted on the car dash board directly facing the driver, solved this challenge since it turned out to be the most efficient solution, while not disturbing the driver's awareness. The stability and direct angle of a well mounted mobile holder ensures successful, duel eye detection in most cases, and for that reason we base on accurate positioning of the camera, as described in the 'Assumptions' section.

Third, acknowledging of closed eyes is not enough and the system need to distinguish blinking and sleeping**.** An average person would blink 16 times per minute. The blinking frequency enhances the computation calculations and challenges the detection process. Our mission is to differentiate harmless blinking from dangerous sleep pauses while driving. The duration of a blink is on average 100-150 milliseconds according to a research[6] conducted in London College University. Due to the short temper and mostly harmless blinks, we have solved that challenge by determining a "closed eyes" threshold - the ratio of closed eyes in a certain time. If the ratio is greater than the threshold above, we refer to the driver as unaware/sleep, as explained in the article ahead.

Another implementation challenge we have faced with is finding the optimal ROI. The ROI – region of interest, is the area we focus on and the area that our calculations and methods are based upon since it is the area where we search for the relevant features to obtain the necessary conclusions. Finding that ROI was a major challenge (few different approaches are based on different regions of interest – the driver's face, both of his eyes as one ROI or each of the driver's eyes separately). Our solution is based on the face as a primarily ROI, and once a face has been detected, we search for another ROI – the driver's eyes. More on that is being elaborated in the article.

The final challenge we faced with is real-time analysis **–** the *Driver Attention Alert* is intended to alert drivers in real time, and by that hopefully rescue the car passengers from life threating situations which can occur in a matter of seconds. To face this enormous responsibility, computations should work efficiently, in a maximal speed and accuracy as possible. Therefore, we aim to achieve the minimal necessary computations with respect to all safety factors in order to gain the best trade-off solution. The real-time analysis challenges the amount and range of calculations and should be taken into consideration when implementing our algorithm.

[6] http://www.ucl.ac.uk/media/library/blinking

Assumptions

While developing our product, we have relied on few assumptions to assist us provide the best possible solution to the problem we have been aspiring to solve. Our first assumption is that the lighting conditions outside the car is not completely dark. Since our solution is based on couple of algorithms and methods that were meant to recognize facial features of the driver, we ought to be able to successfully identify the driver. Hence, we must satisfy the assumptions of those algorithms. To solve that problem, we could have been using night-vision cameras, and by that being able to detect the driver's attention at complete darkness as well.

Our main assumption is that the positioning of the camera is optimal to our needs (with consideration to the safety of the driver). This assumption is necessary since we need to successfully recognize the drive's face on one hand, and not damaging his/her field of vision on the other hand. We rely on the fact that camera is mounted on the car dash board facing the driver in a direct angle to the driver's face, and that the camera captures the driver's face only. Under that assumption – our mission is to continuously monitor the driver's eyes and alert if needed.

The last assumption is somewhat obvious, but is still crucial to a successful identification of our system. We assume that the driver begins his driving awake, since our solution is based on mathematical calculations that detect when the driver's eyes are gradually closing.

The vision problems we solved

First we identified the driver's face with the 'Haar-Cascade' algorithm. This algorithm is an object-detection algorithm used to locate faces, objects and facial expressions in an image and it is mainly used for face detection. In general, the 'Haar-cascade' approach revolves through machine learning of positive images and negative ones which conclude the wanted pattern. We have used a pre-trained classifier to detect the driver’s eyes and face quickly and efficiently.

After the stage of identifying the driver’s face, the system will identify the driver’s eyes and determent if they are open or closed. We tried several methods for detecting the driver’s eyes:

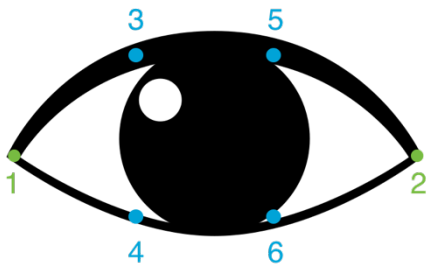
- Using Haar Cascade feature to detect open and closed eyes – this method failed because the algorithm doesn’t analyze each eye separately. Moreover,it had a high rate of false negative detection of eyes. Therefore we couldn’t compare the “eyes” state in each frame – whether they are open or closed.
- Using a combination of Haar Cascade and histogram combination so we could detect the eye by the white color of the eye ball. However we noticed

that for this method the lightening needs to be very good and also if it occurs, the reflection from the eye ball interrupts with the analyzing.

- Using a combination of Haar Cascade and edge detector (such as Canny). As a result, we got a binary map of all the eyes edges and then we could see if the eye is open or closed. However when we tried this method we got a high rate of false negative errors (60%). In addition it took the algorithm a long time to finish all the calculations and therefore if there was a delay of two minutes in detecting closed eyes. Moreover, the edge detector was sensitive in different conditions of light and didn't work well in the evening or when there was shade on the driver's face

Finally we found a method that worked well in detecting the eyes – using Tal Hassner's algorithm for detecting 68 facial landmarks.

We focused on twelve specific points for our calculations. These points are around the eyes region - six for each eye: 2 horizontal points and 4 vertical points (two upper and lower points) as described in the picture below:



After this region was detected, the program measures two distances for each eye individually:

1. v_1: horizontal (absolute) distance between the eye's edges, where $v_1 = |x_1 - x_2|$
2. v_2, v_3: two vertical (absolute) distances between the top and bottom eyelashes where $v_2 = |x_3 - x_4|, v_3 = |x_5 - x_6|$

Then the program calculates the following average distance: $v = \frac{v_2 + v_3}{v_1}$

When we developed our algorithm, we have encountered with a very interesting problem. Although the location of the driver inside the car is relatively static, he/she is still prone to minimal movements, which implicate the vertical eyelids distance we measure at that exact moment. That is a problem since the distance is being measured in pixels, and the number of pixels alters when the driver moves towards the camera, even if the eyelids distance from each other remains identical. This

situation damages our ability to scale and measure the correct vertical distance to distinguish closed eyes. To solve that problem, we used the horizontal distance calculated for each eye to normalize the vertical distance between the eyelids, disregard to any movement of the driver in the direction of the camera inside the car.

At that point, we also assign the maximal vertical distance calculated so far from the beginning of the drive. If, at a certain time of the drive, the ratio between the vertical eyelids distance to that maximal distance is smaller than a certain threshold (we refer it as accuracy threshold) than we have a trigger for closed eyes. In order to avoid any 'white noise' (all it takes is 1 wrong calculation to damage our maximal distance) we check for each frame the maximal distance calculated above. If this distance is unproportionable big, then a false calculation error has occurred, and we set the maximal distance to be the last distance calculated. That way – we solve the 'white noise' problem by O(1) for each frame, and by that we do not damage our real-time analysis.

If the ratio between the vertical eyelid distance to the maximal distance calculated above is smaller than the accuracy threshold for both eyes, we approach the time threshold and beginning with a new set of examines and calculations. If in that time the system recognized closed eyes for more than the 'closed eyes threshold' (to avoid "white noise" errors, that threshold can't be 100%, but is accurate enough to block false detections) – than the system alerts the driver. In any situation where the system recognized false recognition, all the appropriate parameters are being set to their default values, and the system restarts itself at the next frame.

Technical Details

We chose to write the application in Python 2.7 which has a moderate learning curve, and doesn't require complex syntax understanding. We found that language to be the best fit for our needs in a short-time project as we had. Python 2.7 has a variety of machine learning and computer vision libraries which are considered top notch among them the Open-CV2 library and D-lib which are also used in our project and benefit us with a range of functionalities.

The '*Driver Attention Detector'* App processes frame by frame with a constant number of calculations preformed on each frame, which concludes in a running time complexity of O(1) per frame.

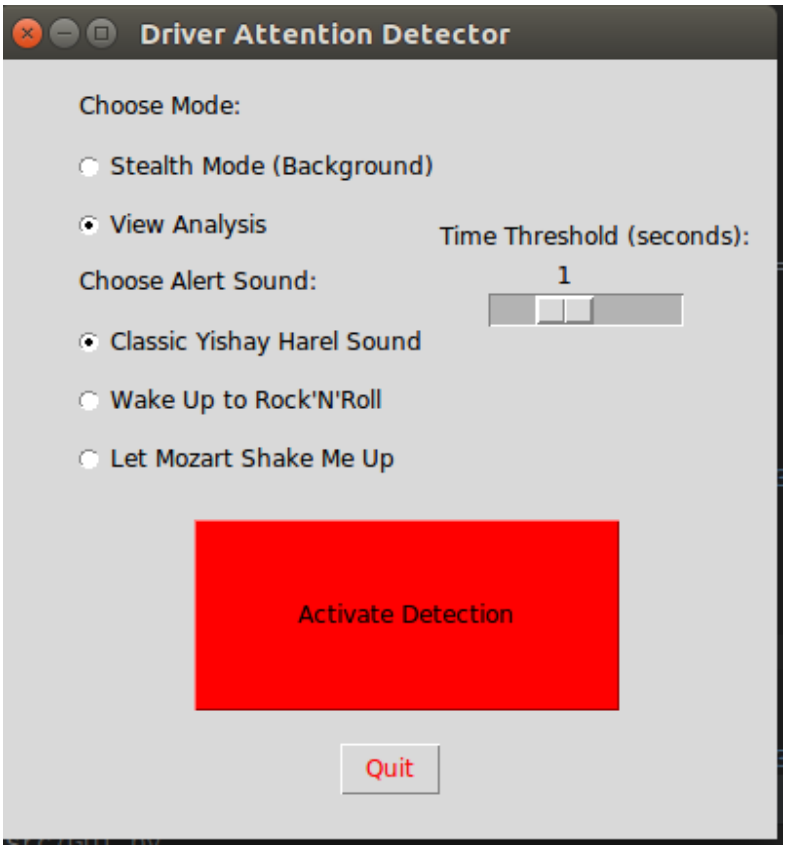
Imported Code Summary

To operate our product, we have used **'OpenCV'** – an open source computer vision and machine learning software library, that enables usage of camera and image processing functions which are highly helpful for our project. As regard to the methods we have used in our project, the first method is **'Haar-cascade'**, which is an object-detection algorithm used to locate faces, objects and facial expressions in an

image and it is mainly used for face detection. In general, the 'Haar-cascade' approach revolves through machine learning of positive images and negative ones which conclude the wanted pattern. We have used a pre-trained classifier to detect our ROI (Region of Interest) quickly and efficiently. The classifier 'haarcascade_frontalface_default.xml' assisted us to verify the presence of the driver, detect the driver's face, and by that gave us the "green light" to start further processing and calculations. After the 'Haar-cascade' classifier's confirmation, we look for facial features inside the region found by the classifier, and it is our face recognition method De-facto. Another important library that we have used is the **'Dlib'** library. The 'D-lib' library contains a wide range of machine learning algorithms. It is considered quick to execute, and simple to use via intuitive Python API. The main algorithm that we have relied on is the '**68 Landmark point**' algorithm by Tal Hassner. Tal Hassner is an Israeli computer science researcher who developed a D-lib dependent algorithm which proved to be a relevant tool for our product. The '68 Landmark point' algorithm analyzes a region which is assumed to contain a face, (thanks to the face we have successfully recognized using the 'Haar-Cascade method) and marks 68 landmark points inside the face. Those landmarks are stored in a data structure with the location of the relevant pixels for each point. The data provided by Tal Hassner's algorithm is crucial to our product since it helps us determine the status, location and whereabouts of the relevant facial features.

Appendix

How to use the Driver Attention Detector



We offer the user a basic GUI with minimal and easy to use functionality.
The user can choose between two modes:

1. View Analysis – presents to the user the output video during run time.
2. Stealth mode – hides the presentation of the process from the user and lets the software work in the background without any visual interference to screen.

Users can also pick the alert type which is played in case the system recognized a drowsy driver. The 3 options include music by Mozart, Rock ’n Roll and a funny bonus track by one of the software creators - Yishay Harel.

The Time threshold argument is presented by a slider which enables the user to decide on the appropriate time for the system to look for a drowsy user. The user can choose the time threshold to be between 1 second (default and secure option) up to 4 seconds.

When the 'Activation Detection' button is pressed, the system begins to analyze the input (from the camera) according to the given arguments, whereas the 'Quit' button lets the user exit the GUI without operating the App. The App is also available via Python console and is activated with default parameters: View analysis, the 'Time threshold' is set to 1 second, and the alert type is 'Yishay Harel sound'. In all cases

of operation – if the user would like to abort the program, they could do so be by pressing the ESC button.

Conclusion and future work

The algorithm take a livestream video of the driver and each frame it detects he's eyes by Tal Hassner's algorithm. Then it checks if the eye is closed. If it finds this id the situation, it will start a timer and after a threshold of time which was set at the beginning, it will start play a song according to the driver's choice.

In the future we hope to improve the running time of the system by inverting the code to other low level language such us C. In addition we would like to improve the accuracy of the system in different camera angles and different times of the day