A Solution to Drowsy Driving via real-time Image Analysis

Bruno Peynetti, Michael Nowakowski,
Yannan Xu, Baiyu Yang

March 18, 2016

Abstract

Drowsy Driving is a widespread problem in the United States. It is a known cause of crashes and fatal accidents every year. Nevertheless, identifying drowsiness and acting on it is a challenging task. This paper presents a novel mobile solution in order to identify drowsy drivers and encourage them to take preventative safety measures. The system aggregates data from a camera mounted on the rear-view mirror which constantly collects information about the driver. We propose an algorithm that analyses the data and determines a threshold of drowsiness based on a variety of factors. The camera-mounted system then transmits that information to a server, from which a smartphone pulls information and acts on it in order to notify the driver of the danger of falling asleep.

1 Motivation and Background

Drowsy Driving is a major problem in the United States. An estimated 1 in 25 adults report having fallen asleep while driving in the previous 30 days. The National Highway Traffic Safety Administration "conservatively estimates that 100,000 police-reported crashes are a direct result of driver fatigue each year." These accidents cause at least 1,550 deaths and $12.5 billion in monetary losses. It is worth noting that these are just the accidents directly attributed to falling asleep at the wheel - the number of accidents where drowsy driving is an unreported factor is likely to be much larger.
A target market for drowsy driving detection would be drivers who drive for long periods of time in straight, traffic free areas. For example, truck drivers who travel accross the country need to drive for long hours, and risk falling asleep during the drive. Equally likely are passenger bus drivers. There is inherently a larger incentive to keep bus drivers awake since a simple nod could cause a catastrophic crash involved dozens of people.
We are also motivated to create a system that is light-weight, portable, and easy to adapt to any situation. To this purpose, we considered using the Raspberry Pi 2 mobile computer with an attached camera, as well as an Android smartphone, which is the most popular type of smartphone and easy to develop for.

2 Related Work

Several studies have proposed a range of methods to detect drowsiness and distractability while driving. Some of these results have been applied to commercial products and have started appearing in the market. For example, a study [**?**] had developed a pilot drowsy driving detection system in the car to capture the images of driver's face by a camera and to collect the driver photoplethysmograph by a bio-signal sensor installed on the steering wheel. The system was developed using Android-based smartphone technology to receive sensory data from a video sensor and a bio-signal sensor. Then, it processed the data to determine the driver's state of fatigue. Another study developed a five-layer sensing architecture to collect information related to the driver in real-time. [**?**]. Other studies [**?**] [**?**] looked into classification of driving maneuvers to build driver behavior models and determine drowsiness levels. Not all solutions have focused exclusively on monitoring individual drivers. For instance, a study by Pascale [**?**] proposes traffic monitoring sensors deployed along the road in order to improve quality and safety of car mobility and enable short-term traffic predictions, which might in turn be able to pre-

dict driver behavior. Many of these studies do not include an alert system but rather look into identifying the drowsy driving. One study by Sun, Zhang, et.al. [**?**] studied a context-aware smart car which collected and evaluated contextual information about the driving, vehicle, and driving environment. A software platform was built for the context model and its applications.

Figure 1: Raspberry Pi Camera

3 System Architecture

In this section, we focus on describing the layout of the system architecture and the functionality of the system.

3.1 Mounted Camera and Image Analysis Unit

In order to successfully identify driver fatigue using image analysis, a camera has to be placed in a position that enables it to monitor the face of the driver from a head-on vantage point. A smartphone camera will not always be in the best position, since different users may change the location of their smartphone on the dashboard, and many place it at an angle such that a forward-facing camera would not point at the driver.
Since our solution relies on being able to detect drowsy driving as quickly and reliably as possible, it is of outmost importance to have the correct camera placement. The camera is to be placed in the bottom left corner of the rearview mirror, providing the system with an almost direct view of the user without obstructing their view out the front windshield. Figure **??** shows the positioning of the camera next to the rear-view mirror. The small size of the camera minimizes obstruction of the driver's view.

3.2 Back-end Server

We chose to use a back-end server to store data and act as the intermediary between the smartphone and the Raspberry Pi (the image analysis unit). After the Raspberry Pi executes the drowsiness detection algorithm, the data is sent to a back-end server where it is stored in a MySQL database. When a smartphone sends a GET request to the server, the server responds with a JSON file containing the latest data and a variable that determines whether the driver is drowsy or not. This implementation allows for multiple users to save and request information simultaneously. In addition, all data is being stored and can be analyzed later to improve the drowsiness detection algorithm's accuracy and performance. The server of choice was a LAMP stack (Apache, PHP and MySQL) in the Cloud9 (c9.io) hosting website. This website has a Virtual Machine enabled with terminal access, as well as text editors and debuggers, which allows for easy and efficient and development.

3.3 Smartphone

The smartphone portion of the app involves an information display feature and an alert feature. A simple Android Application was created in order to interact with the user and propose a quick, safe solution in case drowsiness is detected. We used a variety of Android phones to test the app (including the Google Nexus 5 and the Motorola Moto X 2013) in order to ensure broad compatibility with the myriad models of the Android in-

stall base. For testing purposes, we used Wi-fi to connect to the database, but a cellular connection should work just as well. Figure **??** shows the landing page of the app. In a full development version, the user would have the ability to log in and enable user-specific features and preferences (such as destination, prompt alert message, etc).

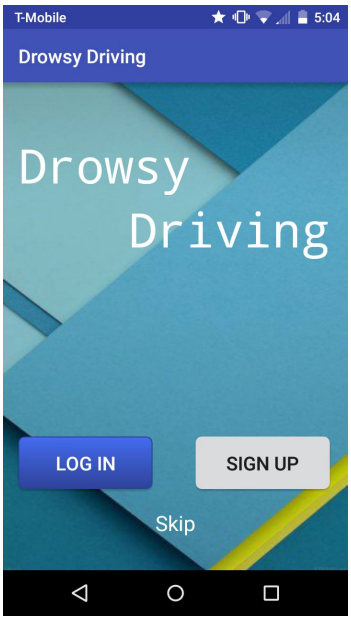

Figure 2: App login page

4 Drowsiness Detection Algorithm

The following section proposes a drowsiness detection algorithm by making use of a variety of classification techniques that enable the creation of a robust and accurate classifier. Some previous work [**?**] does not solve the problem by identifying drowsiness, but rather by identifying one of the features of drowsiness such as a closed eye. We chose to focus on 3 features and built a classification algorithm based on these features:

- Blink rate (blinks per minute)
- Yawn rate (yawns per minute)
- Blink length (moving average length of previous yawns)

In order to calculate these features, we designed an algorithm that uses computer vision techniques to detect the face, eyes, and mouth in every frame. Information from subsequent frames is fed into the algorithm in order to extract the required features.
The flowchart in Figure **??** describes the process to detect drowsiness. The algorithm is performed on each camera frame, and a different algorithm (discussed later in the section) determines the level of drowsiness.

4.1 Face Detection

The first step of the drowsiness detection algorithm is face detection. Since our approach requires image analysis and recognition in each frame, Since our algorithm is run on each frame, a performant solution was of great importance. After some evaluation of possible techniques, Viola and Jones's [**?**] algorithm for real-time face detection was deemed appropriate to the task. The algorithm works on the basis of classification based on simple features rather than pixels directly. Viola introduces the concept of an **Integral image** that contains at location x, y the sum of the pixels above and left of x, y. Figure **??** shows how each rectangular mask and filter compares different intensities of sections of an image in order to create a specific feature. Different combinations of rectangles create tens of thousands of different features. Nevertheless, Viola demonstrated that the correct feature selection (about 200 features) yields a detection rate of about 95%. In order to decrease the time to process the image, a cascade of weak Haar classifiers is built. Each weak classifier is not able to individually classify the image, but the combination of classifiers is able to discern the correct object.

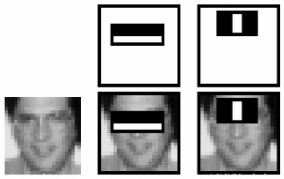

Figure 5: Haar Cascade Classification

The face detection classifier was implemented using the Python OpenCV 2.4 library, which is optimized to work as a straightforward wrapper

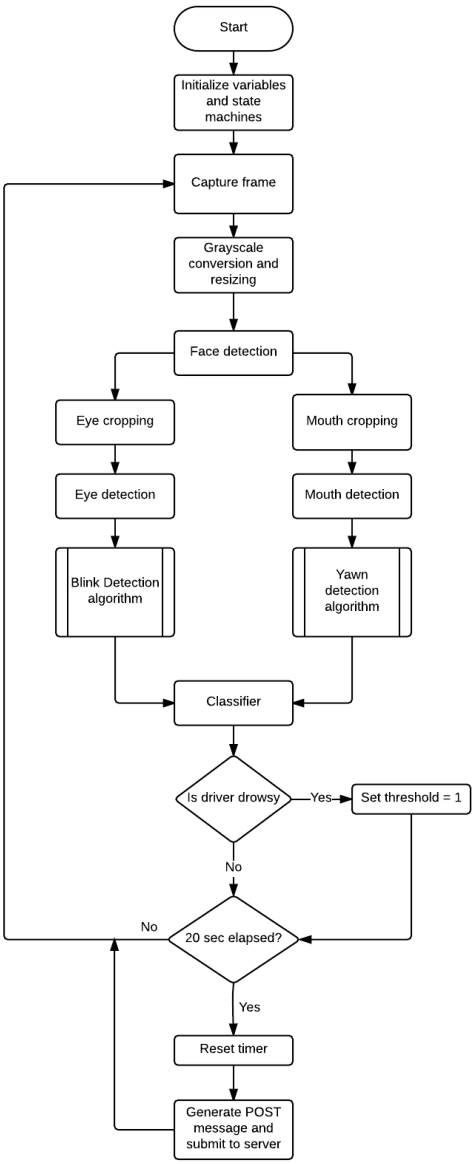

Figure 3: Drowsiness Detection Algorithm

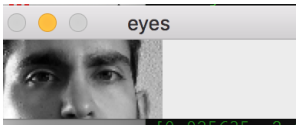

Figure 4: Eye recognition bounding box

for the highly optimized OpenCV C++ functions. Thanks to well-managed Python bindings, there is minimal performance loss from working in Python instead of C++. We decided to use the Python version for simplicity of code and development speed.

4.2 Eye Detection

The eye detection algorithm uses a very similar approach to the cascade-based classification that identifies a face. Nevertheless, an eye is harder to recognize and thus easier to miss-classify. As a result, the search area for eye detection was limited to a small area of the bounding window for the face in order to improve performance while reducing erroneous classifications.
For an identified face starting at coordinates (x, y) and with a width of w and a height of h, the face image can be represented as:

$$face = Image[y : y + h, x : x + w]$$

, where $Image$ is the captured frame with a camera. The bounding box for the eye is as following:

$$eye = Image[(y + h/5) : (y + h/2), x : (x + w)]$$

The resulting cropped image is shown in Figure **??** and requires computation over only 30% of the face-space rather than 100% of the face image or the entire frame. This optimization improved eye detection performance by a factor of approximately 3x.
Additionally, it should be noted that the cascade classifier can and will find one or two eyes depending on the lighting conditions, camera positioning, and other factors. Since blinking mostly occurs with both eyes simultaneously, we considered only the cases when we **1) detected open eyes** and **2) did not detect eyes**

4.3 Mouth Detection

Very similarly to the eye detection, the bounding box was modified to process the image much faster and perform mouth recognition in a negligible time. The bounding box for a mouth image classifier was determined to be as follows:

$$mouth = Image[(y + h/2) : (y + 8 * y/9) \\ , (x + w/5), (x + 4 * w/5)]$$

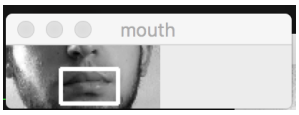

Figure 6: Yawn detection: closed mouth

The resulting bounding box is shown in Figure **??**. As with eye and face detection, we made use of built-in functionality of OpenCV's classification algorithms to identify the user's mouth. The OpenCV classification takes a wide array of parameters as inputs. A few of these parameters are listed:

- Min Size: minimum size of detected object (w,h). used = (20,20)
- Min Neighbors: Parameter specifying how many neighbors each candidate rectangle should have to retain it. used = 4
- Scale: Parameter identifying how much the image can be scaled for classification at each image frame (low values closer to 0 result in higher number of false positives). used=1.3

In order to correctly identify a yawn, we limited the bounding box to a very small area of the face bounding box, which is in line with our test subjects. This area limitation created a miss in mouth detection when an individual was yawning. This missed detection can be seen in Figure **??**, where the mouth goes past the limit of the bounding box and thus is not recognized.

4.4 Blink Detection

As previously stated, we propose an algorithm that uses computer vision techniques to detect features that enable classification of the level of drowsiness that should generate an alert. The algorithm proposed to detect the blink rate and blink length is an implementation of a finite state machine which feeds its result to a historic data array, a portion of which is then analyzed and sent to the back-end storage. A discussion of the finite state machine follows.

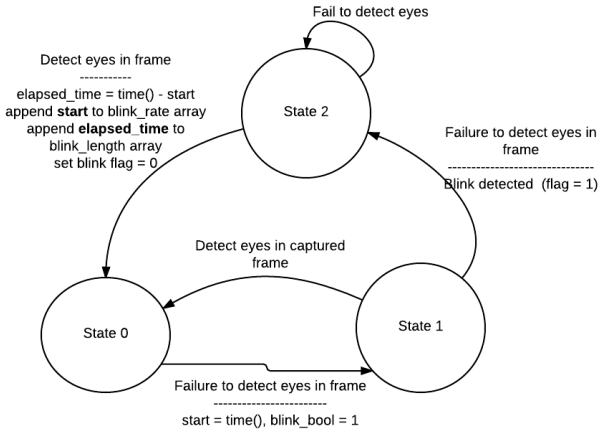

Figure 7: Blink Finite State Machine

The Finite state machine can be seen in Figure **??**. Its operation spans a minimum of 3 frames, 2 to detect a blink and 3 to also detect the length of the blink. Using the Python time library allows reliable measurement of blink length without the need to account for different fps performance in the algorithm. After a blink is detected and its time and length calculated, that information is saved in two separate arrays. The array containing information about the start time of a blink is used to determine blink ratio (blinks per minute) whereas the blink length array is used to determine the average blink length.
The features used in the classification algorithm are as follows:

- Blink start time $\implies$ Append to array of blink start times $[b_0, b_1, b_2 m \cdots]$
- Blink length $\implies$ Append to array of blink lengths $[l_0, l_1, l_2, \cdots]$

4.5 Yawn Detection

The Yawn detection algorithm is very similar as the blink detection algorithm. The mouth detection, however, proved to be less robust and more prone to misses, and thus we decided to have a threshold of minimum 5 continuous non-mouth frames to trigger detection of a yawn. Other considerations were taken into account to determine a yawn since there were many false positives while testing. False positives were triggered for anything from a person looking away, talking loudly, or placing a hand in front of the mouth. Further

work on yawn detection could provide a more reliable drowsiness detection algorithm.

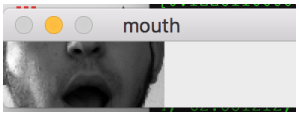

Figure 8: Yawn detection: yawning

4.6 Threshold Calculation

In order to determine whether the driver is sleepy or not, we collected samples from 20 individuals. To establish ground truth, we used a small questionnaire previous to the study and, in addition, considered the time of day.
A classifier was trained using 169 negative data points and 123 positive data points from 20 individuals. The machine-learning tool Weka allowed us to compare multiple classification algorithms with the given data without having to code the algorithms. We tested variants of J48 decision trees, Random Forest, and Naive Bayes. A discussion on the results of this classification task can be found in the results section.
Every 20 seconds, a drowsiness detection classification decision is made. The system uses the resulting J48 classification tree and performs a quick classification. Then, it forwards the data (current blink and yawn rate, as well as avg blink length and the threshold variable, which is set to 1 if the user is drowsy and 0 otherwise) via an HTTP Post request to the back-end server, where the data is stored for future reference.

5 Driver Alert Mechanism

As data from the driver's face is processed and classifications are made, the results are sent to the back-end server. The back-end server stores the information and saves it for retrieval by our mobile Android application.

5.1 Motivation

To add greater novelty and usability to our system, we chose to have a driver alert mechanism that offered a tangible solution to the user, as many available systems attempt to detect drowsy driving but do not provide proper feedback or

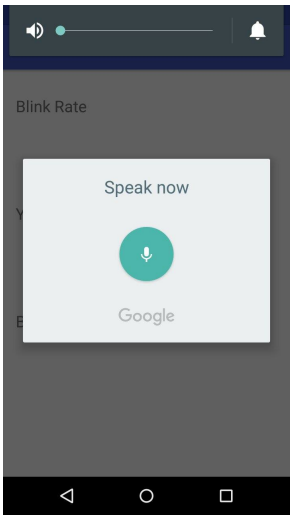

Figure 9: Google voice recognition

an effective solution. Other systems ask for direct user interaction with the smartphone's touchscreen, which is dangerous since it requires the user to look away from the road while their driving abilities are already impaired. Most solutions found online included interaction with the phone in ways that can be dangerous and increase the probability of a crash. We propose a hands-free, voice-activated alert system.

5.2 Alert

A smartphone submits a request directly to the back-end server. When the back-end server obtains the request, the response is sent in the form of a JSON file. Along with the most recent data on blink rate, blink length, and yawn rate, the data includes a threshold value which determines if the driver is sleepy or not. Upon the receipt of the proper threshold value indicating that the driver is drowsy, the smartphone activates the alert message. The alert message consists of a loud, easy to understand notification that could in the future be customized to consider the user's preferences. Google's Text-To-Speech API was used in order to generate the alert. A typical alert would sound the following message: "Hey John, you seem sleepy. Do you want to get a coffee?" After a 5-second waiting period, voice recognition is triggered to ask for a response from the user.

5.3 Voice Recognition

The Voice Recognition system is shown in Figure **??**. In the voice recognition phase, the system listens to the user's response to an alert message. A reply that is not strictly 'yes' will force the App to return to its normal state of near-real-time data collection. After a designated waiting time (5 seconds) it will request data again from the server and as long as the threshold is still active, it will prompt the user again with the same alert message and option to reply. A reply of 'yes' will trigger the Google Maps API. In order to recognize the human response, we made use of the Google Voice Recognition API, which returns a string of concatenated replies. As an example, saying "yes" would return a string similar to "yesyeahyeYESYesyepjees" Thus, we need only look if the desired feedback information 'yes' is contained within the resulting recognized string.

5.4 Maps

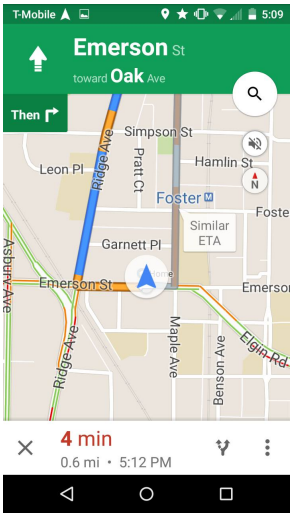

Figure 10: Maps integration

A reply of 'yes' triggers an event within the Android phone that connects directly with the Google Maps app, as seen in Figure **??**. The motivation to use the Google Maps app was to provide a real and effective solution that is familiar to the user, so that they will be more likely to get off the road and reduce their drowsiness. One of the related studies [**?**] does not take action once drowsiness has been detected, and thus successfully identifies the problem but does not provide a feasible and safe solution.

6 Experimental Set Up and Testing

Because of the inherent danger of running a clinical trial on real drowsy subjects driving a car, testing was done in a safe and quiet environment. The first step to build the classifier was to gather data from individuals and create a classification algorithm. Data from 20 individuals was captured using various devices, including both a Macbook Pro and a Raspberry Pi with an attached camera module. A Macbook Pro was used in order to provide faster, more detailed time-series data than the Raspberry Pi, which offers limited classification capability owing to its lower-power CPU.
In order to test how robust the system is to different factors, some tests were made by people wearing clear glasses and some by people without glasses. Additionally, some tests were made with improper lighting conditions: a shaded area and a quickly changing shadows that covered the face at irregular periods of time. The results determined that clear glasses with thin or non-existent frames did not affect the eye identification, while thick frames increased the error rate. Needless to say, this system would not work when a driver is wearing sunglasses. It should be noted that blink detection requires at least 2 subsequent frames of failed eye detection. Considering that the average human blinks for $200 - 400$ milliseconds, there is a minimum performance requirement. The minimum FPS calculation is as follows:

$$FPS_{min} = 2 * (1/.2) = 10FPS$$

This measurement is for the lower end of the average human blink. When testing the design, we noticed that this average was in fact true, and thus any implementation that did not manage to achieve a minimum of 10 FPS performance would render incorrect results due to missed frames and insufficient data.

7 Results and Discussion

7.1 Data Analysis

We analyze the data collection results and the performance of our algorithm. Additionally, we tested the functionality of the system as a whole.

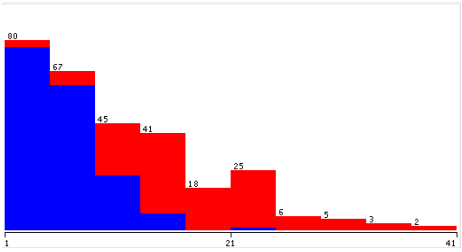

Figure 11: Blink Rate Histogram

Figure **??** shows a histogram of the blink rate (average blinks per minute) from the collected data. Note that red represents the positive state (drowsy) and blue represents the negative state (not drowsy). When performing feature selection, it was easy to see that blink rate was the most important feature in the classification task.

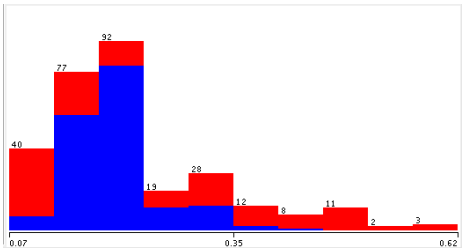

Figure 12: Blink Length Histogram

Figure **??** shows the blink length histogram with seconds in the x axis. It should be noted that this is not as robust of a feature, since there is a large portion of blinks of a minimum length where the classification is drowsy. Finally, Figure **??** shows the distribution of the yawn rate in our tests. Even though this histogram seems skewed towards 0 yawns per minute, it is a reasonable measurement since most non-drowsy people will not yawn in a relatively long period of time, which can easily be a few minutes.

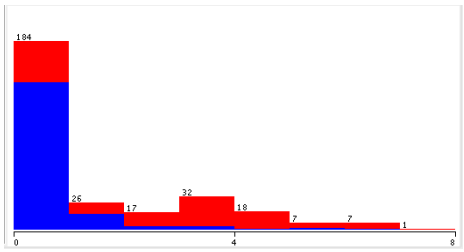

Figure 13: Yawn Rate Histogram

7.2 Algorithm Performance

Because of the computationally heavy nature of the algorithm, we tested our system on different hardware to determine impact of processing capabilities. We tested different implementations in which we varied the image size and the hardware used. Table **??** shows the performance in Frames Per Second for each of the implementations. As mentioned previously, a frame rate lower than 10 FPS renders a very weak blink detection algorithm and thus cannot be considered reliable for drowsiness detection. This renders our Raspberry Pi 2 implementation too slow to be useful. Further optimization of the algorithm, changes in design, multi-threading, and the newer version of the Raspberry Pi - the Raspberry Pi 3 - are all factors that could help increase efficiency and create a working mobile solution.
Another consideration of this algorithm is its effectiveness to perform under different lighting conditions. Upon testing, we noticed that as soon as a large shadow covered the eye region and not the rest of the frame, blink detection began to fail. Since eyes were detected fewer times than under normal lighting conditions, the blink ratio increased. Attempts at eliminating this issue includes minimum blink detection, where a blink would not only needs 2 frames but also 100 milliseconds in order to be counted as a valid blink.

Implementation	Image Capture Size	FPS
MacBook Pro 2014	(640,800)	18
MacBook Pro 2012	(640,800)	17
Raspberry Pi	(640,800)	4
Raspberry Pi	(300,400)	6

Table 1: Classification algorithms

7.3 Machine Learning - Classification

The classification algorithm was created with the aid of Weka. Table **??** shows the different results of each algorithm after performing 10-fold cross validation on the dataset. The dataset had 292 total instances.

Classifier	% Correct	Precision	Recall
J48	90.06%	0.901	0.901
Naive Bayes	93.15%	0.932	0.932
Random Forest	92.46%	0.925	0.925

Table 2: Performance results

As shown in Table **??**, all algorithms showed at least 90% correct classification. For simplicity, the J48 decision tree was implemented in the Python script that determined the drowsiness classification. The confusion matrix for the J48 classification is shown in Table **??**.

.	Non-drowsy	Drowsy
Classified = Non-drowsy	158	11
Classified = drowsy	11	112

Table 3: Confusion Matrix for J48 classification

8 Conclusion and Future Work

We proposed a new efficient drowsiness detection for real-time implementation by making use of blink and yawn detection. Our computer vision techniques provide accurate results when performed under proper lighting conditions and with sufficient processing power. We also successfully tested the software on people wearing glasses with no discernible change.

Future work on developing this application would account for people wearing large headwear such as caps or wide-framed glasses. In addition, the algorithm should be upgraded to correctly analyse and detect drowsiness in people with diseases or physiological properties that prevent an eye or mouth from being recognized, even if this classification is limited to a lower rate of accuracy. In addition, modification and optimization to implement AI technologies and Machine Learning techniques would allow the system to learn individual blinking patterns, driving behavior, etc and build a personalized drowsiness detection system. This personalization could lead to improved accuracy for specific individuals.

Performance improvements would allow this system to be mounted on a small, cheap, and portable computer such as the Raspberry Pi without having to re-engineer the entire system. This would enable the system to be run off the car battery and be kept within the vehicle at all times, making install and regular use by end users a very manageable proposition. The newest iteration of the Pi, the Raspberry Pi 3, is equipped with greater processing power and could potentially be used in this project. Moving some of the code-base to C or C++ could also prove very beneficial for efficiency purposes.

References

[1] E. H. Norman *Japan's emergence as a modern state* 1940: International Secretariat, Institute of Pacific Relations.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, 2001, pp. I-511-I-518 vol.1.

[3] B.-G. Lee and W.-Y. Chung, "Driver Alertness Monitoring Using Fusion of Facial Features and Bio-Signals," IEEE Sensors Journal, vol. 12, no. 7, pp. 2416–2422, Jul. 2012.

[4] S. Al-Sultan, A. Al-Bayatti, and H. Zedan, "Context Aware Driver Behaviour Detection System in Intelligent Transportation Systems (ITS).," IEEE Transactions on Vehicular Technology, no. c, pp. 1–1, 2013.

[5] Chin-Teng Lin, Che-Jui Chang, Bor-Shyh Lin, Shao-Hang Hung, Chih-Feng Chao, and I-Jan Wang, "A real-time wireless braincomputer interface system for drowsiness detection.," IEEE transactions on biomedical circuits and systems, vol. 4, no. 4, pp. 214–22, Aug. 2010.

[6]] C. Lin, Y. Chen, R. Wu, S. Liang, and T. Huang, "Assessment of Driver's Driving Performance and Alertness Using EEG-based

Fuzzy Neural Networks," 2005 IEEE International Symposium on Circuits and Systems, pp. 152–155, 2005.

[7] D. Kim, H. Han, S. Cho, and U. Chong, "Detection of Drowsiness with eyes open using EEG- Based Power Spectrum Analysis," 2012 7th International Forum on Strategic Technology (IFOST), Tomsk, pp. 3–6, 2013.

[8] A. Sathyanarayana, S. O. Sadjadi, and J. H. L. Hansen, "Leveraging sensor information from portable devices towards automatic driving maneuver recognition," 2012 15th International IEEE Conference on Intelligent Transportation Systems, pp. 660–665, Sep. 2012.

[9] L. Tijerina, T. Pilutti, J. F. Coughlin, and E. Feron, "Detection of Driver Fatigue Caused by Sleep Deprivation," IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 39, no. 4, pp. 694–705, Jul. 2009.

[10] A. Pascale, M. Nicoli, F. Deflorio, B. Dalla Chiara, and U. Spagnolini, "Wireless sensor networks for traffic management and road safety," IET Intelligent Transport Systems, vol. 6, no. 1, p. 67, 2012.

[11] J. Sun, Y. Zhang, and K. He, "Providing Context-awareness in the Smart Car Environment," 2010 10th IEEE International Conference on Computer and Information Technology, no. Cit, pp. 13–19, Jun. 2010.

[12] A. Rahman, M. Sirshar and A. Khan, "Real time drowsiness detection using eye blink monitoring," 2015 National Software Engineering Conference (NSEC), Rawalpindi, 2015, pp. 1-7. doi: 10.1109/NSEC.2015.7396336