EyeGuardian: A Framework of Eye Tracking and Blink Detection for Mobile Device Users

Seongwon Han, Sungwon Yang, Jihyoung Kim and Mario Gerla
Dept. of Computer Science, UCLA
Los Angeles, CA, USA
{swhan, swyang, jhkim, gerla}@cs.ucla.edu

ABSTRACT

Computer Vision Syndrome (CVS) is a common problem in the "Information Age", and it is becoming more serious as mobile devices (e.g. smartphones and tablet PCs) with small, low-resolution screens are outnumbering the home computers. The simplest way to avoid CVS is to blink frequently. However, most people do not realize that they blink less and some do not blink at all in front of the screen. In this paper, we present a mobile application that keeps track of the reader's blink rate and prods the user to blink if an exceptionally low blink rate is detected. The proposed eye detection and tracking algorithm is designed for mobile devices and can keep track of the eyes in spite of camera motion. The main idea is to predict the eye position in the camera frame using the feedback from the built-in accelerometer. The eye tracking system was built on a commercial Tablet PC. The experimental results consistently show that the scheme can withstand very aggressive mobility scenarios.

1. INTRODUCTION

"The information age has taken a toll on our eyesight" says Jeffrey Anshel, an optometrist in Carlsbad, California, and president of Corporate Vision Consulting, which advises employers on vision issues [1]. According to the American Optometric Association (AOA), 90% of employees who use computers at least three hours a day experience vision problems.

Normal vision requires a moist ocular surface, and blinking is essential for this reason [14]. However, reading information on a computer screen often increases burden on the human eyes. To adapt to this screen-saturated viewing situation, people tend to blink less than usual. Tears covering the eyes evaporate more rapidly during long non-blinking phases, resulting in dry eyes. The smaller size and lower resolution the display, the more burden on the eyes. Computer Vision Syndrome (CVS) is a set of eye and vision related problems that results from prolonged use of computers. Staring at a computer screen, smartphone, or tablet PC leads to a significant reduction of spontaneous eye blink rate due to the high visual demand of the screen and mental concentration on computer work [16]. The symptoms of CVS include eye irritation, such as dry eye; red, itchy, and watery eyes; fatigue, including heaviness of the eyelids or forehead; and difficulty in focusing the eyes. Other symptoms of CVS are headaches, neck aches, backaches and muscle spasms.

Mobile devices such as smartphones or tablet PCs have already become ubiquitous. Recently, it has been increasingly common for people to check e-mail, browse on the Internet, watch movies, and even read books on their portable devices. People are indeed exposed to the CVS not only in the office but also elsewhere. To protect people who spend much time on mobile devices from this problem, we propose a non-intrusive application that keeps track of the eye blink rate of the mobile device users. If the blink rate is less than desired, the application nudges the user to blink often by vibrating, or modulating the brightness of the screen. To the best of our knowledge, this is the first study that tries to alleviate the CVS in a mobile computing environment. To achieve this goal, a light-weight yet accurate eye detecting and tracking technique specialized for mobile devices is essential.

Eye tracking and blink detection algorithms using a general video camera have been extensively studied in the literature. However, most of them assumed that the user is always gazing at the screen, and the eyes of the user are always in the video camera frames. The existing techniques are inadequate for a mobile device since frequent changes in device position due to the user movements lead to frequent absence or position changes of the eyes in the video frame.

EyeGuardian is a simple yet effective technique using a built-in 3-axis accelerometer to help efficiently track the user's eyes in mobile environment. The eye location is predictable in the continuous video images by fusing the accelerometer and orientation readings from the device. By eliminating unnecessary region where the eyes rarely exist in the images, we achieve a real-time eye tracking and blink detection on the mobile device even under a dynamic environment. We have implemented the proposed algorithm on a Tablet PC and evaluated its performance. Extensive real experiments in various usage scenarios confirm that our technique is adequate for detecting and tracking eyes in the mobile environment.

The remainder of this paper is organized as follows: Section 2 summarizes previous studies on eye tracking and blink detection. The proposed technique that predicts the eye po-

HotMobile'12 February 28–29, 2012, San Diego, CA, USA.

sition is introduced in Section 3 and is evaluated in Section 4. Finally, the paper's conclusions are summarized in Section 5.

2. RELATED WORK

This section briefly reviews previous work on human eye blink detection and its applications. Eye tracking and blink detection using a video camera has been extensively studied in the literature, particularly in the filed of Human-Computer Interaction (HCI). The main goal was providing an effective way to use computer systems for those who have disabilities or limited motor skills. A thorough survey on initial work on eye tracking and blink detection methods is well summarized in [9,11]. Based on the detection and the tracking algorithms, applications for general desktop computers and mobile devices have been presented.

Magee *et al.* presented EyeKeys [11], a gaze detection system that runs on a general computer with video input from an inexpensive USB camera while previously proposed systems required specialized hardware such as electrodes placed on the face, an infrared illuminator, or an expensive frame grabber.

Batista [4] presented a framework for drowsiness and point of attention monitoring system. The proposed solution focused on detection of head rotation and eye blinks, which were the two important cues for determining driver visual attention, to measure the driver's vigilance level. For the feasibility evaluation of applications that detects the drowsiness of drivers, Friedrichs *et al.* compared the performance of algorithms that detect eye blinks from video images with techniques using Electrooculogram (EOG) that can monitor electrical muscles activities around eyes, thus it can be used as a ground truth [13].

Miluzzo *et al.* developed a smartphone application, called Eyephone [12], which was capable of tracking a user's eye and mapping its position on the smartphone display using the phone's front-facing camera. EyePhone enables the user to activate smartphone applications by "blinking at the app", emulating a mouse click.

3. APPLICATION DESIGN

In this section, we review existing eye detection algorithms and explain why they are inadequate for measuring the eye blink rate on a tablet PC. We then propose a technique that efficiently detects and tracks eyes via built-in accelerometer readings.

3.1 Static vs Dynamic Environment

Previously proposed eye detection applications using a general video camera assumed static usage environment where the camera is fixed and the user is rarely out of center of the camera frame. Moreover, they were developed as a form of HCI, therefore, they did not account for the situation when the user's eyes are off the camera. Under this use case, for efficiency, most of the implementations have tried to detect the user's face first because it is a much easier problem than detecting the eyes. Once the face is detected, then, the region where the two eyes are expected is defined in order to reduce the processing overhead. We call this first phase as the "Detection Phase". After the eye is detected in the previously defined region, another region based on the detected eye location is defined for efficient eye tracking. The new region, called "Search Region", is bigger than the detected eye region but much smaller than the entire frame, and from this moment on, the Search Region is regarded as the new image frame. This frame size reduction leads to a speed boost because it searches the eye only inside of small region instead of searching in the entire image frame. We call this the "Tracking Phase".

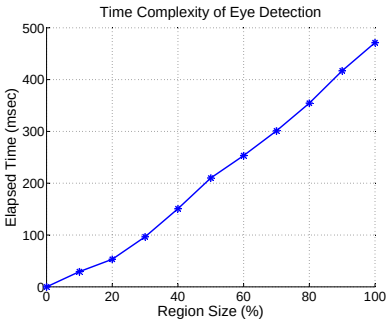


Figure 1: Processing time for eye detection using Haar Cascade Classifiers

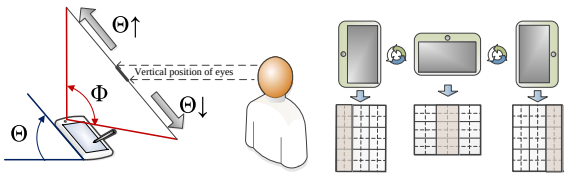
This two-phase approach works well in the static environment. However, mobile devices are used in "mobile environment" where both the user and the device are moving continuously, leading to the following two problems. Firstly, the user's face and both eyes cannot be guaranteed to be included in the entire camera frame at the same time. Secondly, in dynamic environment, the eye location in the next frame cannot be guaranteed to be found in the expected Search Region of the current frame. When this happens, the existing algorithms search for eyes in the entire image frame again. This leads to many misses of blink detection due to the severe processing delay to detect the eyes. Since the user does not intentionally try to be included in the camera frame while non-camera applications are used, the two problems indeed matter. For this reason, the two-phase technique cannot be directly applied to our case.

3.2 Region of Interest (ROI) Prediction

Due to the first aforementioned problem, we remove the face detection step. Our application only searches eyes in the camera frame. In addition, the probability that the Tracking Phase fails to locate the eye position is higher in the mobile environment. Thus, the increased processing time becomes the primary problem in our application. Therefore, a technique that can reduce the size of the region in which we search for the eyes is necessary. Figure 1 shows the processing time for eye detection using Haar Cascade Classifiers method [7, 10] with QVGA (320X240) resolution in respect of the frame size. The processing time for eye detection is solely dependent on the image size, thus, the delay can be dramatically reduced if a precise region of interest can be defined. This paper proposes a simple yet effective technique that predicts the region of interest based on the information from the built-in accelerometer. For eye tracking and blink detection itself, we employ previous performance-proven methods.

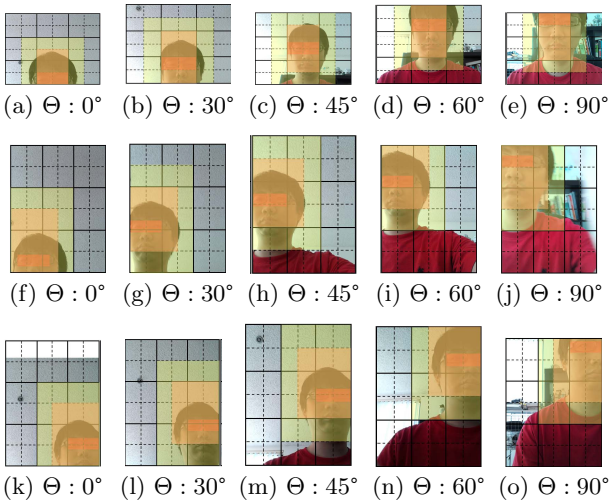
3.3 Prediction during Detection Phase

After observing the behavior of tablet PC users, we found the following hints:



(a) Vertical prediction (b) Horizontal prediction

Figure 2: Predicting ROI during detection phase



(a) Θ : 0° (b) Θ : 30° (c) Θ : 45° (d) Θ : 60° (e) Θ : 90°

(f) Θ : 0° (g) Θ : 30° (h) Θ : 45° (i) Θ : 60° (j) Θ : 90°

(k) Θ : 0° (l) Θ : 30° (m) Θ : 45° (n) Θ : 60° (o) Θ : 90°

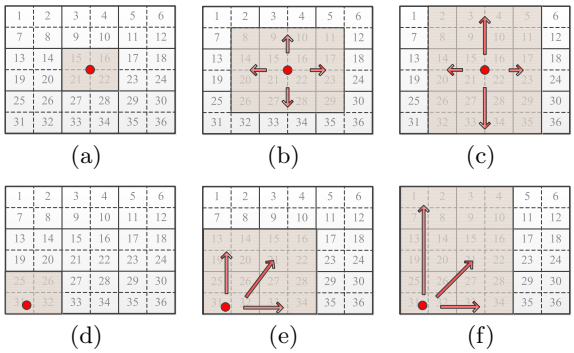
Figure 3: Examples of eye locations in the camera frame by position of a mobile device (camera position: 1^{st} row: top-middle, 2^{nd} row: left-middle, 3^{rd} row: right-middle)

1. The vertical tilt angle of the device remains between 0°and 90°in most cases.

2. The user maintains a certain distance to the device, thus the size of one eye does not exceed 80% of the captured camera frame

3. Depending on the position of the camera, the location of the eyes changes.

Our method that initially finds eyes in the video frame is devised based on the observations above. Figure 2(a) shows the relation between the vertical tilt angle of the device and the vertical location of the user's eyes. Let Θ denote the tilt angle of the device and Φ denote the view angle of the device's front-facing camera. Since Φ is fixed, Θ decides the vertical location of the eyes. As the tablet PC leans forward (i.e., Θ increases), the eye location moves towards the top of the camera frame, and as Θ decreases the eye location moves to the bottom of the frame. The location of the front camera also affects the prediction of the eye location. Tablet PCs can be used either portrait or landscape modes by applications. When the camera is located on the left (right) side, the eye is also likely to be located on the left (right) side in the frame. Figure 3 is an example that shows how the eye position varies depending on the vertical tilt angle and the location of the camera. Based on the information from the in-built accelerometer, our method calculates both



(a) (b) (c) (d) (e) (f)

Figure 4: Examples of initial eye detection algorithm

the vertical tilt angle and the position of the front camera to predict a region where an eye exist. We split the image frame into 9 sub-regions, thus, the initially predicted region falls into one of the 9 sub-regions. For example, the initial region becomes left-middle one if the camera is located on the left side of the device and the device is tilted about 45 degrees. Now, we try to find an eye inside the predicted one sub-region (Figure 4(a) and (d)). If an eye is not found there, we expand the region of interest to 4 sub-regions (Figure 4(b) and (e)) and try to find eyes again. If an eye is not found, we lastly expand the region to 7 sub-regions (Figure 4(c) and (f)). Our method stops expanding the region at the third attempt because there is almost no chance to find an eye in the remaining $\frac{2}{9}$ areas.

3.4 Prediction during Tracking Phase

Figure 5 illustrates our eye detection algorithm during the Tracking Phase. Due to the movement of user's hand that holds the mobile device, the eye frequently falls out of the Search Region. In this case, existing schemes discard the current video frame and try to find eyes in the Detection Phase, resulting in severe delays. If the eye is not found in the Search Region during the Tracking Phase, our algorithm tries to find the eye in the current frame once more based on the information of vertical and horizontal tilt angle difference. Suppose that the device is tilted up from the vertical tilt angle of Θ_m to Θ_n. Similarly, suppose that the horizontal tilt angle changed from Ψ_m to Ψ_n. Then the position of the eye moves towards the upper left area in the frame. One of the four areas in which our scheme tries to find the eye is decided based on the angle difference of the device, which is defined in Figure 5(c). The four areas are split by V_m and H_m on the image frame. V_m is a vertical line that is drawn on the location of the detected eye when the vertical tilt angle is Θ_m, and H_m is a horizontal line that is drawn on the location of the detected eye when the horizontal tilt angle is Ψ_m.

Figure 6 describes our combined detection and tracking algorithm. Namely, it tries to find eyes in the first region of interest. If at least one eye is found in the first region, it goes to the Tracking Phase. If it fails, the region increases in size in the second and the third attempts. If it fails in the third region, the current image frame is discarded and it starts over with the next frame. Once it enters the Tracking Phase, the region of interest is predicted based on the vertical and the horizontal tilt angles. Our algorithm looks for the eye

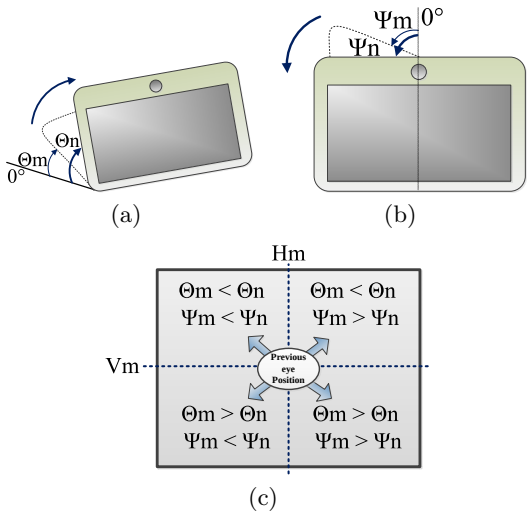


Figure 5: Predicting ROI during tracking phase

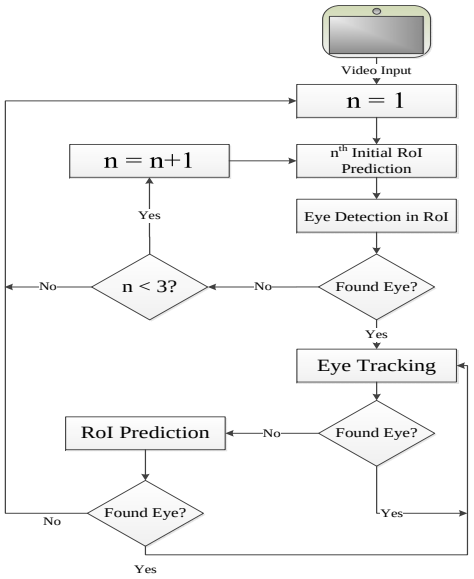


Figure 6: Proposed eye detection and tracking algorithm

in the predicted region of interest when it fails to find the eye in the Search Region.

4. EVALUATION

We implemented EyeGuardian on a Samsung Galaxy Tab 10.1. Samsung Galaxy Tab runs Android 3.1 version, and we adopted OpenCV [2] library for the purpose of image processing. In the next subsections, we present the eye detection schemes in the Detection and the Tracking Phases and blink detection algorithm we used.

4.1 Eye Detection

We used Haar Cascade Classifiers for detecting the eyes during the Detection Phase, which was first developed by Paul Viola and Michael Jones [15] and later extended by Rainer *et al.* [10] to use diagonal features. Integral images enable an algorithm to rapidly detect any object using AdaBoost Classifier Cascades [7] that are based on Haar-like features. This technique is very commonly used for the real time face detection and also works fairly well with eye detection [5].

After locating the eye, the online template of the user's eye is extracted and used in the Tracking Phase. For the eye tracking, we used a simple algorithm to update the Search Region centered in the eyes. At each frame, centered eye location in the Search Region is updated by the normalized correlation coefficient proposed by Grauman *et al.* [8].

4.2 Blink Detection

Blink detection is purely based on the correlation scores generated by matching templates between opened eyes and closed eyes in the Tracking Phase. The blink detection method proposed in [8] provides two advantages when it is applied to EyeGuardian. Our goal is to provide a lightweight app with a reasonable accuracy. EyeGuardian is lightweight since no extra computation is needed due to the pre-computed correlation scores at each frame during the Track Phase. Additionally, the blink detection method proposed in [8] improves the blink detection accuracy. According to the experiment results in [6], the number of false negatives and the false positives of the aforementioned blink detection method was almost the same. Most of the previous eye blink detection applications have targeted the HCI function. For example, the eye blink has the function of the computer mouse click, and the accuracy of detecting the blinks is critical to the performance of the application. Hence, minimizing the number of false positives and false negatives was important in the previous eye blink detection applications. In contrast, in our case if the ratio of false positive and false negative is close to one, the number of false detections is not an issue. Namely, our approach focuses on recording the number of eye blinks during a certain period of time, and the blink miss is compensated by a false positive. Hence, the number of blinks can still be estimated accurately.

4.3 Blinking Criterion for Users

In order to prevent CVS, 20-20-20 rule [3] recommends people to look at a different place 20 feet away the computer screen for 20 seconds, every 20 minutes during the use of computers. The average human eye blink rate is 22.4±8.9 times per minute under relaxed condition [14]. However, it decreases to 10.5±6.5 times per minute while reading a book and 7.6±6.7 times per minute while doing near work such as using the computer [14]. Hence, EyeGuardian monitors the eye blink rate of the user for 20 minutes, and recommends the users to rest their eyes if the number of eye blinks is lower than 13.5 (22.4-8.9) times per minute.

4.4 Detection and Tracking Accuracy

In this section, we present the performance improvement of our approach via experiments. Five participants used the tablet PC not being aware of the eye detection function during the experiments. The most crucial function of the EyeGuardian is to detect the user's eyes successfully. Since the eyes are in the Region of Interest, we tested how accurately EyeGuardian detected the Region of Interest during the Detection Phase. According to the experiment results, the accuracy of detecting the region of interest was about

User	Total Frames	No Prediction	← Percentage	Prediction	← Percentage
1	11773	9516	79.13 (%)	10458	88.83 (%)
2	10032	9368	93.38 (%)	9813	97.81 (%)
3	10053	7676	76.35 (%)	9126	90.78 (%)
4	10119	8964	88.58 (%)	9513	90.11 (%)
5	10039	9637	95.99 (%)	9993	99.54 (%)

Table 1: Improvements of eye detection during tracking phase

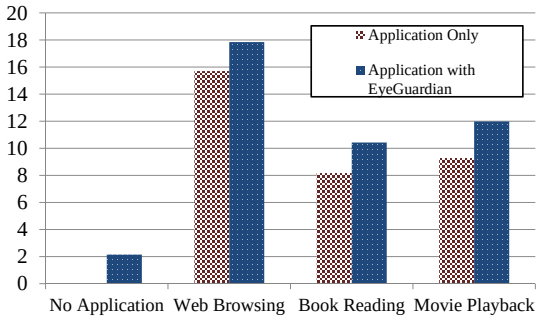


Figure 7: Energy consumption per hour (%)

50%, 70%, and 90% in the first, second, and the third step, respectively.

In order to evaluate the performance of the Tracking Phase, we recorded the number of frames that did not have the user's eyes in the search region. We evaluated our approach by checking the overall reduced processing time achieved from discarding the aforementioned frames. Each experiment was conducted 10 times and the average values of the experiment results are presented in Table 1. The first and second column of Table 1 represent the user number and the total number of the frames used for detecting each user's eye blinks, respectively. The third and fourth column represent the number of frames that detected eye blinks during the tracking phase without using our approach and its percentage, respectively. The fifth and sixth column represent the number of frames that detected eye blinks during the tracking phase using our approach and its percentage, respectively. Even though the amount of performance improvement depends on the user's usage behavior and the number of concurrently running applications, the 86.82% of the total image frames detected the eyes inside the Search Region during the tracking phase without using our approach. The remaining 13.18% of the total image frames failed to detect the eyes in the Search Region and returned back to Tracking Phase. With our approach, the 94.15% of the total image frames detected the eyes inside the Search Region during the Tracking Phase. In the case of the remaining 5.85% of the image frames, the eyes were out of the image or the image was too blurry for the eyes to be detected. Hence, our prediction method reduces the processing time by only using 7.33% of the total image frame during the Detection Phase, and the reduced processing time reduces power consumption.

4.5 Energy Consumption

Since EyeGuardian is always running in the background on the tablet PC, the energy efficiency is a crucial factor to be considered. We read the remaining battery level via the Android 3.1(Honeycomb) default battery level indicator. It is true that the battery level indicated by the default battery app is not as accurate as measuring the battery level via the battery meter. Since we did not have such a device, we had to measure the battery level based on the default battery app. In order to compensate the inaccuracy of the battery app, we fully charged Galaxy Tab before conducting each experiment. Since we used a new Galaxy Tab, the battery status was reliable. Additionally, we conducted all the experiments on the same device. We measured the battery power consumption of EyeGuardian by running it concurrently with one of the following three apps – a web browser with wifi connection, an ebook reader, and a movie player. Each measurement was conducted for three hours and we repeated the same measurement three times. Hence, we measured EyeGuardian's power consumption nine times. After each measurement, the tablet PC was fully charged, and the display brightness was always set to 80%. As shown in Figure 7, it is interesting to see that playing a movie clip consumes far less energy than browsing the web. The CPU usage meter showed that playing a movie clip only consumed about 25% of CPU resource. On the other hand, there was "user interaction" every time we measured the energy consumption of web browsing and book reading. We kept accessing different websites while browsing the web. In addition, we intentionally browsed web pages that contain a lot of images and flash movie clips rather than only browsing the text based web pages to see how performance changes on a relatively harsh condition. This may have resulted in more power consumption of the web browser than the other two applications. The measurement results that appear in Figure 7 indicate that EyeGuardian consumes 2.15% of the total battery capacity per hour alone. Considering the case that EyeGuardian and another app were running concurrently, EyeGuardian consumed similar amount of power when it was operated alone. EyeGuardian did not cause significant battery drainage even under the multitasking environment. Our approach is thus feasible from the energy budget standpoint, since it only consumes about 2% of the total battery capacity per hour alone or under multitasking environment. However, battery consumption may be an issue running EyeGuardian on a smartphone. Galaxy Tab adopts a 7000 mAh battery that is 4-5 times larger than the average smartphone batteries. EyeGuardian consumes a small amount of energy on Galaxy Tab that has a large battery. We can port EyeGuardian on a smartphone, however, it may worsen the battery drain. Developing an energy efficient eye tracking and blink detection application for smartphones will be a part of our future work.

4.6 Frame Rate

There is a correlation between the frame rate and the blink detection. If the frame rate is too low, the probability missing the blinks increases while extremely high frame rate does not improve the blink detection accuracy but only consumes power. Since the average length of a blink is 300-400 ms, in order to detect the blinks the frame rate should be at least 10 fps. It is crucial to check whether or not the frame rate is above 10 fps under multitasking environment. We ran EyeGuardian concurrently with one of the following three apps-a web browser with wifi connection, an ebook reader,

Application	Average	Max	Min	Standard Deviation
Web Browseing	12.46	14.43	9.77	1.98
Book Reading	14.17	14.57	12.91	0.87
Movie Playback	13.69	14.53	11.92	1.16

Table 2: Frame per second by applications

and a movie player. The maximum recordable frame rate of the front camera is 15 fps. The average frame rate of the front camera during reading an ebook was 14.17±0.87 fps, while the maximum frame rate was close to 15 fps (Table 2). The average frame rates during watching a movie and using the web-browser were 13.69±1.16 fps, and 12.46±1.98 fps, respectively. Playing a movie on a tablet PC consumes resources continuously until the movie ends or the user stops the movie. However, web browser uses relatively more resources loading web pages on the web browser than playing a movie. The more resources consumed, the less frame rate.

5. CONCLUSION

This paper proposes a novel mobile application to alert a user at risk of CVS and protects their eyes from possible damages. To accomplish this, EyeGuardian monitors the user's eye blink frequency via the front camera of the device in a non intrusive way. EyeGuardian determines whether or not it will recommend the users to rest their eyes based on the user's eye blink rate. The built-in accelerometer of the tablet PC enables EyeGuardian to track the position of the device and estimate the location of the user's eyes. The processing time of detecting the eyes from the images acquired from the front camera is considerably reduced by predicting the region of interest where the eyes could exist in the images rather than scanning the full-size images. Our approach may also be extended to various other applications. One possible application is providing health advice by predicting the user's drowsiness and fatigue via the user's movement and eye blink frequency. As a future work, we will focus on developing an eye detection algorithm that is suitable for mobile environment – e.g., an algorithm that efficiently detects the eyes from multiple viewing angles and in various lighting conditions. Finally, we plan to verify the performance of EyeGuardian in reducing of the vision problems associated with CVS by conducting clinical trials in cooperation with the School of Nursing.

6. REFERENCES

[1] `http://online.wsj.com/article/SB10001424052748704868604575433361436276340.html?mod=WSJ_hpp_MIDDLENexttoWhatsNewsThird`.

[2] `http://opencv.willowgarage.com/wiki/Android`.

[3] `www.osteopathic.org`.

[4] Batista, J. A drowsiness and point of attention monitoring system for driver vigilance. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE* (30 2007-oct. 3 2007), pp. 702 –708.

[5] Bradski, G., and Kaehler, A. *Learning OpenCV*. O'Reilly Media, Inc., 2008.

[6] Chau, M., and Betke, M. Real time eye tracking and blink detection with usb cameras. Tech. rep., 2005.

[7] Freund, Y., and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory* (London, UK, 1995), Springer-Verlag, pp. 23–37.

[8] Grauman, K., Betke, M., Gips, J., and Bradski, G. Communication via eye blinks - detection and duration analysis in real time. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* (2001), vol. 1, pp. I–1010 – I–1017 vol.1.

[9] Grauman, K., Betke, M., Lombardi, J., Gips, J., and Bradski, G. R. Communication via eye blinks and eyebrow raises: Video-based human-computer interfaces. In *UNIVERSAL ACCESS IN THE INFORMATION SOCIETY* (2003), pp. 2–4.

[10] Lienhart, R., and Maydt, J. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on* (2002), vol. 1, pp. I–900 – I–903 vol.1.

[11] Magee, J. J., Scott, M. R., Waber, B. N., and Betke, M. Eyekeys: A real-time vision interface based on gaze detection from a low-grade video camera. In *In Proceedings of the IEEE Workshop on Real-Time Vision for Human-Computer Interaction (RTV4HCI* (2004), pp. 159–166.

[12] Miluzzo, E., Wang, T., and Campbell, A. T. Eyephone: activating mobile phones with your eyes. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds* (New York, NY, USA, 2010), MobiHeld '10, ACM, pp. 15–20.

[13] Picot, A., Caplier, A., and Charbonnier, S. Comparison between eog and high frame rate camera for drowsiness detection. In *Applications of Computer Vision (WACV), 2009 Workshop on* (dec. 2009), pp. 1 –6.

[14] Tsubota, K. Tear dynamics and dry eye. *Progress in Retinal and Eye Research 17*, 4 (1998), 565 – 596.

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

[16] Yan, Z., Hu, L., Chen, H., and Lu, F. Computer vision syndrome: A widely spreading but largely unknown epidemic among computer users. *Comput. Hum. Behav. 24* (September 2008), 2026–2042.