EYE-TRACKING FOR DETECTION OF DRIVER FATIGUE

Martin Eriksson

Nikolaos P. Papanikolopoulos

Artificial Intelligence, Robotics, and Vision Laboratory

Department of Computer Science, University of Minnesota,

Minneapolis, MN 55455

E-mail: {eriksson, npapas}@cs.umn.edu

Keywords: driver fatigue, eye-tracking, template matching.

Abstract

In this paper, we describe a system that locates and tracks the eyes of a driver. The purpose of such a system is to perform detection of driver fatigue. By mounting a small camera inside the car, we can monitor the face of the driver and look for eye-movements which indicate that the driver is no longer in condition to drive. In such a case, a warning signal should be issued. This paper describes how to find and track the eyes. We also describe a method that can determine if the eyes are open or closed. The primary criterion for the successful implementation of this system is that it must be highly non-intrusive. The system should start when the ignition is turned on without having the driver initiate the system. Nor should the driver be responsible for providing any feedback to the system. The system must also operate regardless of the texture and the color of the face. It must also be able to handle diverse conditions, such as changes in light, shadows, reflections, etc.

INTRODUCTION

Driver fatigue is an important factor in a large number of accidents. Lowering the number of fatigue-related accidents would not only save society a significant amount financially, but also reduce personal suffering. We believe that by monitoring the eyes, the symptoms of driver fatigue in our proposed system can be detected early enough to avoid several of these accidents. Detection of fatigue involves a sequence of images of a face, and observation of eye-movements and blink patterns.

The analysis of face images is a popular research area with applications such as face recognition, virtual tools and handicap aids [9,14], human identification and database retrieval [3]. There are also many real-time systems, being developed in order to track face features [15,13,17]. These kinds of real-time systems generally consist of three components:

a) Localization of the eyes (in the first frame),
b) Tracking the eyes in the subsequent frames,
c) Detection of failure in tracking.

Localization of the eyes involves looking at the entire image of the face and determining the eye-envelopes (the areas around the eyes). During tracking in subsequent frames, the search-space is reduced to the area corresponding to the eye-envelopes in the current frame. This tracking can be done at relatively low computational effort, since the search-space is significantly reduced. In order to detect failure in the tracking, general constraints such as distance between the eyes and horizontal alignment of the two eyes can be used.

This paper is organized as follows: In the next section, we describe some of the previous work in this area. Afterwards, we describe the experimental setup, and how the system operates. Then, we proceed to the description of the algorithm for the detection of fatigue. Finally, we present results and future work.

PREVIOUS WORK

Many methods have been proposed for localizing facial features in images [2,4,5,6,8,10,11,12,19]. These methods can roughly be divided into two categories: *Template-based matching* and *Feature-based matching*. These techniques are compared by Poggio and Brunelli [1]. One popular template-matching technique for extraction of face features is to use *deformable templates* [5,16], which are similar to the active snakes introduced by Kass [7], in the sense that they apply energy minimization based on the computation of image-forces. In feature-based matching, the system uses knowledge about some geometrical constraints. For example, a face has two eyes, one mouth and one nose in specific relative locations.

One interesting application for face recognition was developed by Stringa [12]. He used the observation that the eyes are regions of rapidly changing intensity. We use a similar approach on a reduced version of the image. Another approach, developed by Steifelhagen et al. [13] uses connected regions in order to extract the dark disks corresponding to the pupils. Rather than looking for the pupils, we used the fact that the entire eye-regions are darker than their surroundings, again allowing us to use the reduced image in order to extract these rough regions at a reduced computational cost. The systems described in [15] and [13] use color information in order to extract the head from the background. In order to avoid dependence on a fairly colorless background, we decided to again use the reduced image and localize the symmetry axis [18]. Since the driver will be looking almost straight ahead, there will be a well defined vertical symmetry line between the eyes.

Many different templates have been described for finding the shape of an eye. Xie et al. [16] developed a deformable template consisting of 10 cost equations, based on image intensity, image gradient and internal forces of the template. Since we are greatly concerned about computational speed, we decided to use only the two cost equations dealing with image intensity. Once the eyes are found, the search-space in the subsequent frames is limited to the area surrounding the found eye-regions. In the system by Stiefelhagen et al., the darkest pixel (which is likely to be a pixel inside the pupil) is used for tracking, allowing high computational speed. Another approach [5] is to perform edge-detection on the region of interest and then track the region with a high concentration of edges.

THE SYSTEM

When the system starts, frames are continuously fed from the camera to the computer. We use the initial frame in order to localize the eye-positions. Once the eyes are localized, we start the tracking process by using information in previous frames in order to achieve localization in subsequent frames. During tracking, error-detection is performed in order to recover from possible tracking failure. When a tracking failure is detected, the eyes are relocalized. During tracking, we also perform the detection of fatigue. At this point, we count consecutive frames during which the eyes are closed. If this number gets too large, we issue a warning signal.

Experimental setup

The final system will consist of a camera pointing at the driver. The camera is to be mounted on the dashboard inside the vehicle. For the system we are developing, the camera is stationary and will not adjust its position or zoom during operation. For experimentation, we are using a JVC color video camera, sending the frames to a Silicon Graphics Indigo. The grabbed frames are represented in RGB-space with 8-bit pixels (256 colors). We do not use any specialized hardware for image processing.

Localization of the eyes

We localize the eyes in a top-down manner, reducing the search-space at each step. The steps are:

1. Localization of the face.
2. Computation of the vertical location of the eyes.
3. Computation of the exact location of the eyes.
4. Estimation of the position of the iris.

Localization of the face. Since the face of a driver is symmetric, we use a symmetry-based approach, similar to [18]. We found that in order for this method to work, it is enough to use a subsampled, gray-scale version of the image. A symmetry-value is then computed for every

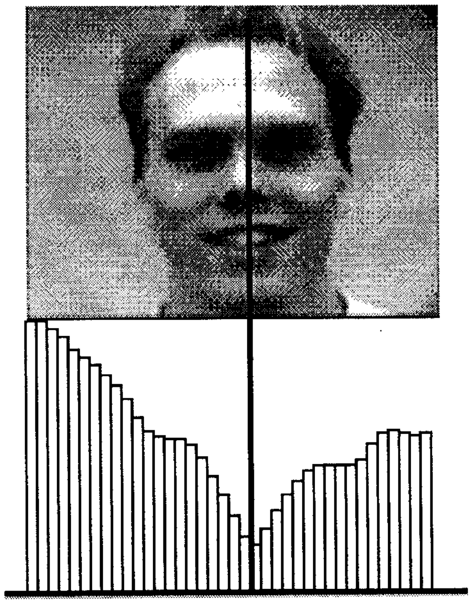

Figure 1. The symmetry histogram.

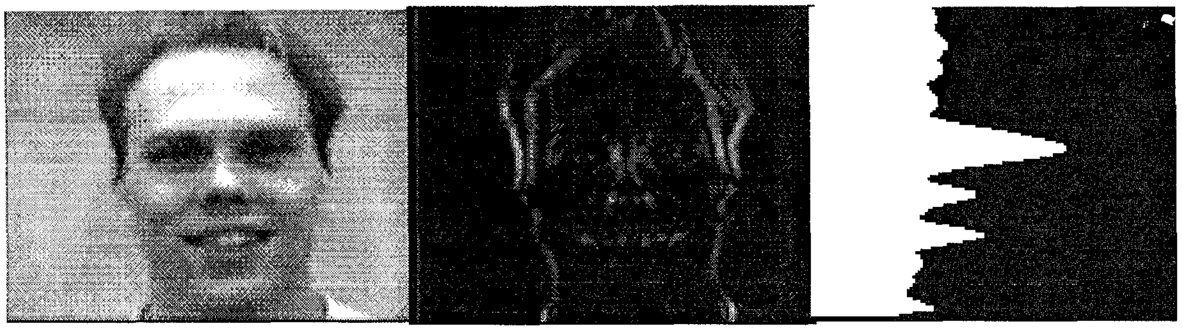

Figure 2. The original image, the edges and the histogram of projected edges.

pixel-column in the reduced image. If the image is represented as $I(x,y)$ then the symmetry-value for a pixel-column is given by

$$S(x) = \sum_{w=1}^{k} \sum_{y=1}^{ysize} \left[abs\left(I(x,y-w) - I(x,y+w) \right) \right].$$

$S(x)$ is computed for $x \in [k, xsize-k]$ where k is the maximum distance from the pixel-column that symmetry is measured, and *xsize* is the width of the image. The x corresponding to the lowest value of $S(x)$ is the center of the face. The result from this process is shown in Figure 1. The search-space is now limited to the area around this line, which reduces the probability of having distracting features in the background.

Computation of the vertical location of the eyes. As suggested by Stringa [12], we use the observation that eye-regions correspond to regions of high spatial frequency. Again we are working with the reduced image. We create the gradient-map, *G(x,y)*, by applying an edge detection algorithm on the reduced image. Any edge-detection method could be used. We choose to use a very simple and fast method called pixel-differentiation, that assigns *G(x,y)* = *I(x,y)* - *I(x-1,y)*. We selected this method since it does not involve any convolution. *G(x,y)* will now reveal areas of high spatial frequency. By projecting *G(x,y)* onto its vertical axis, we get a histogram *H(y)*:

$$H(y) = \sum_{i=1}^{xsize} G(i,y).$$

Since both eyes are likely to be positioned at the same row, *H(y)* will have a strong peak on that row. However, in order to reduce the risk of error, we consider the best three peaks in *H(y)* for further search rather than just the maximum. This process is illustrated in Figure 2.

Find the exact location of the eyes. In order to find the eye-regions given the proceeding processing, we rely on the fact that the eyes correspond to intensity-valleys in the image. Given that, we can threshold the image and then extract the connected regions. We used a raster-scan algorithm on the reduced image in order to extract these regions. In general, our raster-scan algorithm found 4-5 regions. In order to resolve which of these regions correspond to the eyes, we use the information in *H(y)*. We try to find a peak corresponding to a row in the image with two connected regions on. The three best peaks in *H(y)* are considered. We also use general constraints, such that both eyes must be located "fairly close" to the center of the face.

The difficulty with this method is to find a threshold that will generate the correct eye-regions. We used a method called adaptive thresholding [13] that starts out with a low threshold. If two good eye-regions are found, that threshold is stored, and used the next time the eyes have to be localized. If no good eye-regions are found, the system automatically attempts with a higher threshold, until the regions are found.

Estimation of the position of the iris. Once the eye-regions are localized, we can apply a very simple template in order to localize the iris.

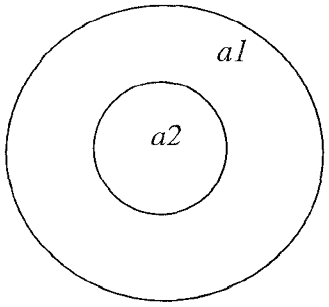

Figure 3. The eye-template.

We constructed a template consisting of two circles, one inside the other. A good match

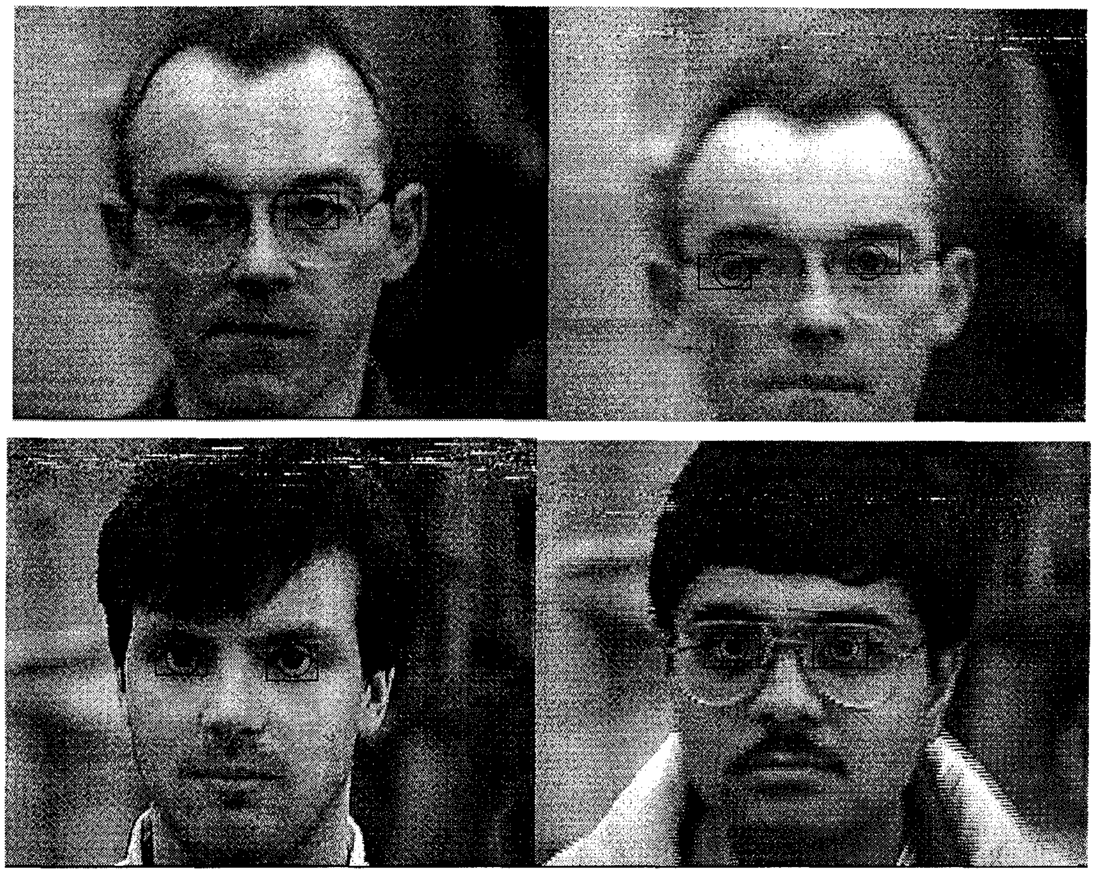

Figure 4. Snapshots from the system during tracking. Note that in the second image, the system missed tracking of one eye.

would result in many dark pixels in the area inside the inner circle, and many bright pixels in the area between the two circles. The template is shown in Figure 3. This match occurs when the inner circle is centered on the iris and the outside circle covers the sclera.

The match $M(a1,a2)$ is computed as

$$M(a1,a2) = \sum_{(p,q)\in a1} I(p,q) - \sum_{(p,q)\in a2} I(p,q).$$

A low value for $M(a1,a2)$ corresponds to a good match. The template is matched across the predicted eye-region, and the best match is reported.

Tracking the eyes

We track the eye by looking for the darkest pixel in the predicted region [13]. In order to recover from tracking errors, we make sure that none of the geometrical constraints are violated. If they are, we relocalize the eyes in the next frame. To find the best match for the eye-template, we initially center it at the darkest pixel, and then perform a gradient descent in order to find a local minimum. In Figure 4, we show a few snapshots during tracking.

DETECTION OF FATIGUE

As the driver becomes more fatigued, we expect the eye-blinks to last longer. We count the number of consecutive frames that the eyes are closed in order to decide the condition of the driver. For this, we need a robust way to determine if the eyes are open or closed; so we developed a method that looks at the horizontal histogram across the pupil.

During initialization (the first frames after the driver has settled down), an average match over a number of frames is calculated. When the match in a frame is "significantly" lower than the average, we call that frame a *closed* frame. If the match is close to the average, we call that an *open* frame. After C consecutive closed frames, we issue a warning signal, where C is the number of frames corresponding to approximately 2 to 2.5 seconds (the time when the eyes have been closed for too long).

	0 Degrees Left	30 Degrees Left	45 Degrees Left	30 Degrees Down
Detected Alerts (10)	10	10	10	10
False Alerts	0	1	2	0
Lost track	0	0	0	0

Table 1. Results from the tests.

Horizontal histogram across the pupil

We use the characteristic curve generated by plotting the image-intensities along the line going through the pupil from left to right, as shown in Figure 5. The pupil is always the darkest point. Surrounding the pupil, we have the iris, which is also very dark. To the right and left of the iris is the white sclera. In Figure 5 we show two curves, one corresponding to an open eye, and one corresponding to a closed eye.

Note that the curve corresponding to the closed eye is very flat.

We compute the matching function $M(x,y)$ as

$$M(x,y) = I(x,y) / \min\{I(x-r,y), I(x+r,y)\}$$

where (x,y) is the computed center of the pupil and r is the radius of the iris. $I(x,y)$ is the image intensity at (x,y). When the eye is open, the valley in the intensity-curve corresponding to the pupil will be surrounded by two large peaks corresponding to the sclera. When the eye is closed, this curve is usually very flat in the center. However, in the latter case there is no pupil to center the curve on, which can lead to a very unpredictable shape. In order to minimize the risk of having one big peak nearby (due to noise), we always use the minimum peak at the distance r from the pupil. This will lead to a good match when the eye is open, and very likely to a bad match when the eye is closed.

RESULTS AND FUTURE WORK

We simulated three “test-drives” where we measured the accuracy of the detection of opened/closed eyes. In each test-drive, we simulated 10 long eye-blinks, and recorded how many were computed by the system. In each test-drive, the driver had the head turned in a different angle. The results are shown in Table 1.

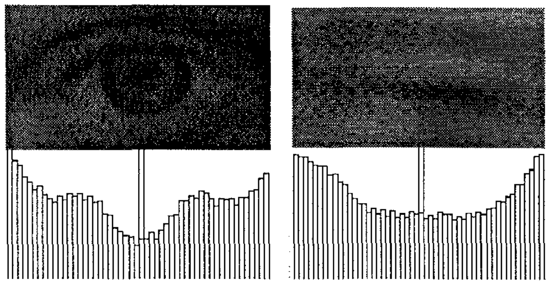

Figure 5. Histograms corresponding to an open and a closed eye, respectively.

For this test, we did not allow for any rapid head movements, since we wanted to simulate the situation when the driver is tired. For small head-movements, the system rarely loses track of the eyes, as we can see from the results. We can also see that when the head is turned too much sideways, we had some false alarms. However, in the case where the head is tilted forward (which is the most likely posture when the driver is tired), the system operated perfectly.

When we perform the detection of driver fatigue, we operate on frames of size 640 by 320. This frame-size allows us to operate at approximately 5 frames per second. In order to track the eyes, without detecting fatigue, it is enough to use frames of size 320 by 160, which allows a frame-rate of approximately 15 frames / second.

At this point, the system has problems localizing eyes when the person is wearing glasses, or has a large amount of facial hair. We believe that by using a small set of face templates, similar to [15], we will be able to avoid this problem, without losing anything in performance. Also, we are not using any color-information in the image. By using techniques described in [13], we can further enhance robustness.

Currently, we do not adjust zoom or direction of the camera during operation. Future work may be to automatically zoom in on the eyes, once they are localized. This would avoid the trade-off between having a wide field of view in order to locate the eyes, and a narrow field of view in order to detect fatigue.

We are only looking at the number of consecutive frames where the eyes are closed. At that point, it may be too late to issue the signal. By study the eye-movement patterns, we are hoping to find a method to generate the alert signal at an earlier stage.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.