AN AUTOMATIC EYE WINK INTERPRETATION SYSTEM FOR THE DISABLE

Che Wei-Gang[1], Chung-Lin Huang[1,2], and Wen-Liang Hwang[3]
1. E E Dept. National Tsing-Hua University, Hsin-Chu, Taiwan
2. Informatics Department, Fo-Guang University, I-Lan, Taiwan
3. Institute of Information Science, Academic Sinica, Taipei, Taiwan
clhuang@ee.nthu.,edu.tw, wlhwang@iis.sinica.edu.tw

ABSTRACT

This paper proposes an automatic eye wink interpretation system for the severely handicapped people. First, we apply the support vector machine (SVM) and template matching algorithm to detect the eyes and then track the eye winks. Then, we convert the sequence of eye winks into code (binary digit) sequence. Finally, we use the dynamic programming to translate the eye wink sequence to a certain command for human-machine interface. In the experiments, our system demonstrates very good performance and high accuracy.

1. INTRODUCTION

Eye detection is a crucial aspect in many useful applications [1~6] such as driver behavior analysis and eye gaze estimation etc. Here, we propose an eye-wink control interface for helping the severely handicapped people to manipulate the household devices. In [1], an eye wink control interface is proposed to provide the severely disabled with increased flexibility and comfort. Systems for analyzing human driver's alertness have been proposed [2, 3]. They rely on optical flow and color predicates to robustly track a person's head and facial features. In [2], the system classifies rotation of all viewing directions, detects eye/mouth occlusion, detects eye blinking, and recovers the 3D gaze of the eyes. In [3], an eye detection algorithm works on the whole image, looking for regions that have the edges with a geometrical configuration like the expected one of the iris. The algorithm uses mean absolute error measurement for eye tracking and a neural network for eyes validation.

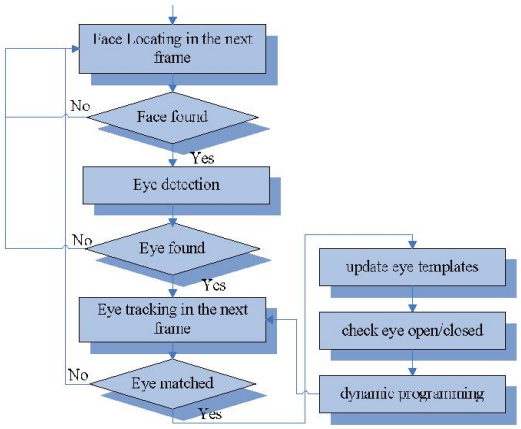

Figure 1 Flowchart of the proposed system.

Most of the methods rely on the motion in the scene, image color, and shape information. Our eye wink control interface aims to provide the ability for user to control the computer-based devices by using their eye winks. In the beginning, we use the skin color information to find the possible face region. The face region has a convex region larger than certain size. If the face region is found then we define the possible eye region in which we apply SVM to localize the precise location of eyes and create the eye template from the identified eye image. In the next frame, we apply the template matching to track the eyes based on the eye template in the previous frame. The eye template is updated every time the eye is successfully tracked. Then we apply SVM to verify the tracked region is eye or non-eye, if it is eye then apply SVM again to check whether the eye is open or closed and then convert the eye winks to 1 or 0 codes. Finally, we apply the dynamic programming to validate the code sequence and convert the code sequence into a certain command. The flow chart of the proposed system is illustrated in Figure 1.

2. EYE DETECTION AND TRACKING

Our eye detection and tracking method consists of three stages: (1) face region detection (2) eye localization based on SVM, (3) eye tracking using template matching. The three steps are illustrated in the following sections.

2.1 Face Region Detection

To reduce the eye search region, we first locate the possible face region. Human skin color distribution analysis has been widely used for face detection. However, the face region in RGB color space may be affected by the brightness, we convert RGB color space to HSI color space to decrease the influence. The hue component is applied to locate faces since human face colors have approximate distribution range on the hue component of the HSI model. From the testing face images, we observe that hue value of skin color falls in the range of [0.175, 0.285] radian.

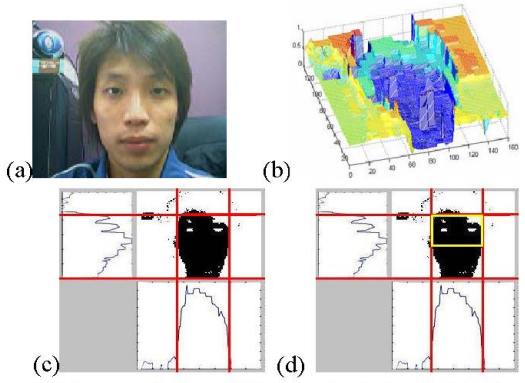

Figure 2 (a) original image. (b) hue distribution of the image. (c) skin color projection. (d) possible eye region.

As shown in Figure 2, we can see that the skin color is distributed in a specified range (the blue color), and the z-axis is the hue value which was normalized between 0 and 1. Thus, the

skin area can be extracted by selecting a specific range of hue values as shown in Figure 2(c). Performing the vertical projection on skin pixels, the right and left boundaries can be determined when the projecting value exceeds a given threshold. We define the threshold as the half of average projection value of face area and the face area is the set of pixels of which the hue is within the corresponding range. Similarly, the up and down boundaries of face area can also be determined. After the skin-color block searching procedure, there may be more than one face-like region in the block image. We select the maximum region as the face region. We assume that eyes should be located in the upper half face area as shown in the yellow rectangle in Figure 2(d). Thus eyes are searched within the yellow rectangle area only.

2.2 Eye Localization using Support Vector Machine

The linear support vector machines. SVM [7] is a general classification scheme that has been successfully applied to find a separating hyper-plane by maximizing the margin between two classes, where the margin is defined as the distance of the closest point in each class to the separating hyper-plane.

Given a data set $\{x_i, y_i\}$ of N examples x_i with labels $y_i \in \{-1, +1\}$, we find the optimal hyper-plane by solving a constrained optimization problem and using quadratic programming, where the optimization criterion is the width of the margin between the classes. The separating hyper-plane can be represented as a linear combination of the training examples and classifying a new test pattern $\boldsymbol{x}$ that is done by using the following expression:

$$f(x) = \sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b \qquad (1)$$

where $k(\boldsymbol{x}, \boldsymbol{x_i})$ is a kernel function and the sign of $f(\boldsymbol{x})$ determines the class membership of $\boldsymbol{x}$. Constructing the optimal hyper-plane is equivalent to finding the nonzero α_i. Any data point x_i corresponding to nonzero α_i is termed "support vector." Support vectors are the training patterns closest to the separating hyper-plane.

First, the training data are required to obtain the optimal hyper-plane of the SVM. The efficiency of SVM classification is determined by the selected features. Here, we select the Sobel edge of eye as feature vector. An eye image is represented by a feature vector consisting of the edged pixel values. We manually select the two classes: positive set (eye) and negative set (non-eye). The eye images are processed by using histogram equalization and their image size are normalized to 20×20. Figure 3 shows some training data consisting of open eye images, closed eye images, and no-eye images.

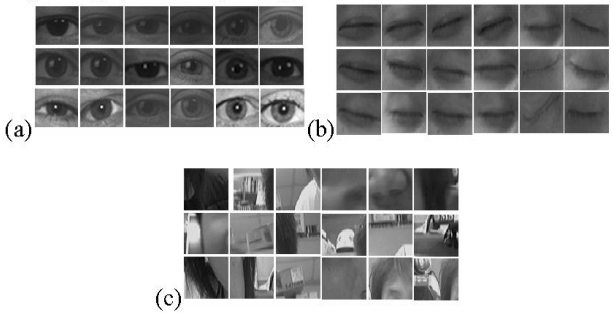

Figure 3. (a) open eye images. (b) closed eye images. (c) non-eye images.

We have tested one thousands of unlabeled data and pick up the miss-labeled data, then put them into the correct training sets and retrain the classifier. After performing this procedure on the unlabeled data obtained from different conditions several times, we can boost the accuracy of the learning machine. Through the retraining process, we can significantly boost the accuracy of the linear SVM.

The eye detection algorithm will search the every candidate image block inside the possible eye region to locate the eyes. Each image block is processed by Sobel edge detector and converted to a feature extraction. With the feature vector, the image block will be classified by the SVM as an eye block or non-eye block.

2.3 Eye Tracking

Eye tracking find the eye in each frame by using template matching. Given the detected eyes in the previous frame, the eyes in subsequent frame can be tracked from frame to frame. After eye localization, we get the eye templates (gray level image) for the next frame's eye tracking. The search regions of next frame extend 50% length in four directions in the region of original eye bounding box. We individually normalize eye template width and height into 20 pixels and 10 pixels because we catch different size of candidate eye image. The candidate eye image also needs to be normalized. Consider an eye template $t(x, y)$ locating at (a, b) of the image frame $f(x, y)$. To compute the similarity between the candidate image blocks and the eye template, we use

$$M(p,q) = \min\left[\sum_{x=0}^{w}\sum_{y=0}^{h} |f_n(x+p, y+q) - t_n(x,y)|\right] \qquad (2)$$

where f_n and t_n are normalized image, and w, h are the width and height of the eye template $t_n(x, y)$, and p, q are offsets of the x-axis and y-axis in which $a-0.5*w<p<a+0.5*w$ and $b-0.5*h<q<b+0.5*h$. If M(p', q') is the minimum value within the search area, the point (p', q') is defined as the best matched position of the eye, and (a, b) is updated by the new position (p', q'), *i.e.*, (a, b)=(p', q'), Then, the new eye template is applied for the eye tracking in the next frame.

Because people may blink their eyes unintentionally, it may cause error propagation and thus make the eye tracking fail. To avoid the error propagation, we process the binarized eye images using Otsu algorithm [8]. For eye gray-level image, we can see that the intensity of the pupil and iris pixels is darker than skin and sclera colors. Using Otsu algorithm, we can segment iris and pupil from eye image as shown in Figure 4. We may find that the centroid of the region of iris and pupil should be at the center of bounding box. Similarly, for the closed eye, the centroid is at the center of eyelashes instead of the center of bounding box. So, we can shift the bounding box center to the centroid of binary eye image after tracking.

To reduce the errors, we apply the SVM again to classify the tracked image to eye class (open or closed eye) or non-eye class. If the tracked image is a non-eye region, the system will restart the face and eye localization procedures.

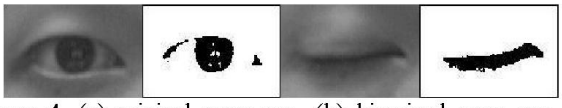

Figure 4. (a) original open eye. (b) binarized open eye. (c)

original closed eye. (d) binarized closed eye.

3. THE COMMAND INTERPRETER

After eye tracking, we continue using SVM to distinguish between the open eye and the closed eye. If eyes are open exceeding a fixed duration, it represents a digit "1". Similarly, the closed eye represents a digit "0". So we can convert the sequence of eye winks to a sequence of 0 and 1. The command interpreter validates the set of code sequences, and issues their respective output actions. Each action is represented by the sequence of codes (binary digits). The user issues a command by a sequence of eye winks, starting from the base state. The base state is defined as open eye for a long time without intentionally closing eye. The input sequence of codes is then matched with the predefined sets of codes by the command interpreter, and a valid command may be issued.

First to avoid an unintentional and very short eye wink, we determine a duration threshold θ_{il}. If the time interval of the continuously opened or closed eye is longer than θ_{il}, then it can be converted to a valid code, *i.e.*, "1" or "0". However, we may allow two contiguous "1" or "0", so we define another threshold $\theta_{ih} \approx 2\theta_{il}$. If the time interval of the continuously opened or closed eye is longer than θ_{ih}, then we may consider it as code 00 or 11. The threshold θ_{il} is user-dependent, and the user may select the best suitable threshold for himself. Here, we may pre-define some valid code sequences of which each one corresponds to a certain command. Once the code sequence has been issued, we need to validate the code sequence. To find a valid code sequence, we need to calculate the similarity (or alignment) score between the issued code sequence and each one of the predefined code sequences. Because the code lengths are all different, we need to align the two sequences and maximize their similarity. We use dynamic programming to compare the two code sequences [9]. Our pre-defined codes are shown in **Table 1**.

Table 1. Code sequences

Code Length	1	3	4	5
	0	010	0010	00100
			0100	00110
			0110	01010
				01100

A dynamic programming algorithm consists of four parts: a recursive definition of the optimal score; a dynamic programming matrix for remembering optimal scores, a bottom-up approach of filling the matrix, and a trace back of the matrix to recover the structure of the optimal solution that gave the optimal score. For pair-wise alignment, these four steps are explained in the following:

1) Recursive definition of the optimal alignment score. There are only three conditions that the alignment can possibly be: (i) residues x_M and y_N are aligned with each other; (ii) residue x_M is aligned to a gap character, and y_N appears somewhere earlier in the alignment; or (iii) residue y_N is aligned to a gap character and x_M appears earlier in the alignment. The optimal alignment will be the most preferred of these three cases. The optimal alignment score of the prefix of sequence $\{x_1, \ldots x_M\}$ to the prefix of $\{y_1, \ldots y_N\}$ is $S(i, j)$ defined as:

$$S(i,j) = \max \begin{cases} S(i-1,j-1) + \sigma(x_i, y_j) \\ S(i-1,j) + \gamma \\ S(i,j-1) + \gamma \end{cases} \quad (3)$$

Where $i \le M$ and $j \le N$. The score for case (i) is the score $\sigma(x_M, y_N)$ for aligning x_M to y_N plus the score $S(M-1, N-1)$ for an optimal alignment of everything else up to this point. Case (ii) is the gap penalty γ plus the score $S(M-1, N)$; whereas case (iii) is the gap penalty γ plus the score $S(M, N-1)$. This works because the problem breaks into independently optimized pieces, as the scoring system is strictly local to one aligned column at a time. For instance, the optimal alignment of $\{x_1, \ldots x_{M-1}\}$ to $\{y_1, \ldots y_{N-1}\}$ is unaffected by adding on the aligned residue pair x_M and y_N. The initial score $S(0, 0)$ for aligning nothing to nothing is zero.

2) The dynamic programming matrix. For the pair-wise sequence alignment algorithm, the optimal scores $S(i, j)$ are tabulated in a two-dimensional matrix, with i running from 0...M and j running from 0...N, as shown in Figure 5. As we calculate solutions to sub-problems $S(i, j)$, their optimal alignment scores are stored in the appropriate (i, j) cell of the matrix.

3) A bottom-up calculation to get the optimal score. Once the dynamic matrix programming matrix $S(i, j)$ is laid out, it is easy to fill it in a 'bottom-up' way, from the smallest problems to progressively bigger problems. We know the boundary conditions in the leftmost column and the topmost row (*i.e.*, $S(0, 0) = 0$; $S(i, 0) = \gamma * i$; $S(0, j) = \gamma * j$). For example, the optimum alignment of the first i residues of sequence x to nothing in sequence y has only one possible solution, which is to align to gap characters and pay i gap penalties. Once we've initialized the top row and left column, we can fill in the rest of the matrix by using the recursive definition of $S(i, j)$ to calculate any cell where we already know the values we need for the three adjoining cells to the upper left $(i-1, j-1)$, above $(i-1, j)$ and to the left $(i, j-1)$. There are several different ways we can do this; one is to iterate two nested loops, $i = 1 \ldots M$ and $j = 1 \ldots N$, so we're filling in the matrix left to right, top to bottom.

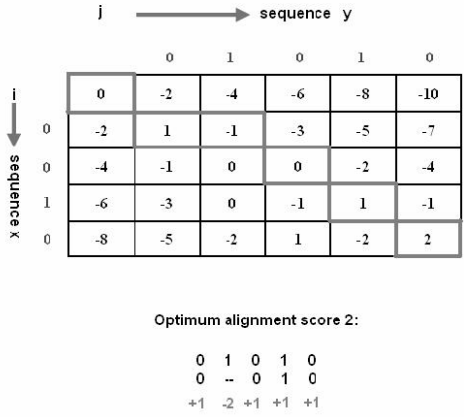

Figure 5. An example of the dynamic programming matrix.

4) A trace back to get the optimal alignment. Once we have done filling in the matrix, the score of the optimal alignment of the complete sequences is the last score, we calculate, $S(M, N)$. We still do not know the optimal alignment itself, however. We recover this by a recursive 'trace back' of the matrix. We start from cell (M, N), determine which of the three cases we used to get here (*e.g.*, by repeating the same three calculations), record that choice as part of the alignment, and then follow the appropriate path for that case back into the previous cell on the optimum path. We keep doing that, one cell in the optimal path at a time, until we reach cell (0, 0), at which point the optimal alignment is fully reconstructed. Figure 5 shows the dynamic

programming matrix for two code sequences, x = 0010 and y = 01010 (0: closed eye, 1: open eye). The scores of match, mismatch, and insertion or deletion are +1, -1, and -2 respectively. The optimum path consists of the cells marked by red rectangles.

4. EXPERIMENT RESULTS

We use a Logitech QuickCam Pro3000 camera to capture users video sequence, and the image resolution is 320x240. The eye wink control system can achieve the speed of 13 frames per second. We have tested 5131 frames of four people under normal indoor lighting conditions.

Figure 6 shows that the eye classifier based on SVM correctly identifies the real eye regions as marked. Pupil verification with SVM works reasonably well. SVM can work under different illumination conditions due to the intensity normalization for the training images via histogram equalization. Before detecting the eye, we separate the face region to the two parts to detect left eye and right eye separately.

Figure 6. Some images with eyes detected correctly.

The results of eye tracking from four test videos are shown in Table 2. "Total frames" indicates the total number of frames in each video. "Tracking Failure" counts the number of eye tracking failure frames. The correct rate of eye tracking is defined as

$$\text{Correct Rate}=\frac{\text{Total Frames-Tracking Failure}}{\text{Total Frames}}$$

Table 2 shows that the proposed system achieves 98 % correct eye identification. Table 3 shows the result of user eye signals detection on the four test videos. Our system contains nine signals. Each signal is composed of a sequence of eye winks. Each fraction represents the correct rate of each signal.

Table 2. Result of eye tracking

	Video 1	Video 2	Video 3	Video 4
# of Frames	1763	1544	583	1241
Failure	17	19	6	15
Correct Rate	99 %	98.7 %	98.9 %	98.7 %
Average Correct Rate			98.8 %	

Table 3. Result of eye signal detection

	Video 1	Video 2	Video 3	Video 4
Signal 1	29/30	25/27	18/18	35/38
Signal 2	15/15	13/13	5/7	15/17
Signal 3	13/13	15/18	12/13	18/20
Signal 4	14/15	10/12	6/7	14/17
Signal 5	13/15	16/17	10/11	18/20
Signal 6	17/17	14/19	8/8	6/8
Signal 7	17/19	17/19	10/12	10/12
Signal 8	18/21	18/21	13/13	21/25
Signal 9	16/17	8/10	6/8	17/18
Correct Rate	95 %	90.7%	90.6%	88%
Average Correct Rate			90.1 %	

Figure 7 shows the eye wink control interface of our system. The red solid circle indicates that the eyes are open. Similarly, the green solid circle indicates that the eyes are closed. There are nine blocks at the right portion, where each number in the small block represents a command. In the base mode, we design eight categories: medical treatments, diet, TV, radio, air conditioning, fan, lamp and telephone. There are two layer in the command mode, so we can create at most 9*9(81) commands. Here, we only have 8*8 + 1 (65) commands because each layer we have a "Return" command. We illustrate layer 1 and one of the layer 2.

Figure 7. Program interface of layer 1 and layer2.

5. CONCLUSIONS

We propose an effective algorithm for eyewink control interface. By integrating support vector machine and dynamic programming, the eye wink control interface will enable user to control household devices by using a sequence of eye winks. Experimental results have illustrated the encouraging performance of the current methods in both accuracy and speed.

REFERENCES

[1] R. Shaw, E. Crisman, A. Loomsi, and Z. Laszewski, "The Eye Wink Control Interface: Using the Computer to Provide the Severely Disabled with Increased Flexibility and Comfort", 3rd *IEEE Symposium on Computer-Based Medical Systems*, 1990

[2] P. Smith, M. Shah, and N. da Vitoria Lobo, "Monitoring Head/Eye Motion for Driver Alertness with One Camera", *The Fifteenth IEEE ICPR*. Nov. 2000.

[3] T. D'Orazio, M. Leo, P. Spagnolo, C. Guaragnella, "A neural system for eye detection in a driver vigilance application", *IEEE Conference on ITS*, Washington DC, October 2004.

[4] P. W. Hallinan, "Recognizing human eyes", *Geometric Methods Comput. Vision*, vol. 1570, pp. 214-226, 1991.

[5] S. Amamag, R. S. Kumaran and J. N. Gowdy, "Real Time Eye Tracking For Human Computer Interfaces", *ICME 2003*.

[6] Z. Zhu and Q. Ji, "Robust real-time eye detection and tracking under variable lighting conditions and various face orientations", Computer Vision and Image Understanding 98 (2005) 124–154.

[7] V. Vapnik, *The Nature of Statistical Learning Theory*, New York: Springer, 1995.

[8] N Otsu, "A threshold selection method from gray-level histograms" *IEEE Transactions on Systems, Man, and Cybernetics*. 1979

[9]S. R Eddy, "What is dynamic programming", *Nature Biotechnology* Volume22 Number 7 July 2004.