EyeKeys: A Real-time Vision Interface Based on Gaze Detection from a Low-grade Video Camera

John J. Magee, Matthew R. Scott, Benjamin N. Waber and Margrit Betke

Computer Science Department, Boston University
111 Cummington St., Boston, MA 02215
{mageejo, mrscott, bwabes, betke}@cs.bu.edu
http://www.cs.bu.edu/fac/betke/

Abstract

There are people that are so severely paralyzed that they only have the ability to control the muscles in their eyes. Communication is limited to the interpretation of eye movements. Currently available human-computer interface systems are often intrusive, require special hardware, or use active infrared illumination. We present a system that runs on an average PC with video input from an inexpensive USB camera. The face is tracked using multi-scale template correlation. Symmetry between left and right eyes is exploited to detect if the computer user is looking at the camera, or off to the left or right side. The detected eye direction can then be used to control applications such as spelling programs or games. We developed the game "BlockEscape" to gather quantitative results to evaluate our interface system with test subjects. We also compared our system to a mouse substitution interface.

1 Introduction

The ability to control eye muscles is sometimes the only remaining voluntary movement paralyzed people have. Communication abilities are severly limited, often to *yes* and *no* responses using eye movements or blinks. Future computer assistive technologies will someday stop a mind from being trapped in a body. As progress toward that goal, we present an interface called EyeKeys based on gaze detection that exploits the symmetry between left and right eyes.

There has been much previous work in computer assistive technologies. Most of these methods, though successful and useful, also have drawbacks. Currently available systems are often intrusive, and use specialized hardware. For example, the EagleEyes system [4] uses electrodes placed on the face to detect the movements of the muscles around the eye. It has been used with disabled adults and children to navigate a computer mouse. One problem is that the sensors must touch the user's face, which may be the only place the person has feeling, thus making the sensors intrusive. Another approach [2] uses head mounted cameras to look at eye movements. It takes advantage of the fact that the face will always be in the same location in the video image if the head moves around. However, large headgear is not suited for all users, especially small children. Given the issues with systems that require the interface to be attached to the user, is it one of our goals to design a non-intrusive system that does not need attachments.

Another successful system is the Camera Mouse [3]. Disabled people control a mouse pointer by moving their head, finger, or other limbs, while the system uses video to track the motion. This is successful for those who can move their heads or limbs; however, people who can only move their eyes are unable to use it. These are the people for whom we aim to provide a communication device. A second goal of our system is therefore to use only information from the eyes.

Many systems that analyze eye information use specialized hardware. The use of active infrared illumination is one example [5, 7, 8, 9, 10]. Usually, the infrared light reflects off the back of the eye to create a distinct "bright eye" effect in the image. If the light is synchronized with the camera, the eyes are located by differencing the bright eye image with the image without infrared illumination. One technique to find the relative gaze direction is to find the difference between the center of the bright eye pixel area, and the reflection off the surface of the eye from the light source. There are concerns about the safety of prolonged exposure to infrared lighting. Another issue is that some of these systems require a complicated calibration procedure that is difficult for small children to follow.

Avoiding specialized hardware is an important goal of our system. This means that our system must run on a consumer grade computer. In addition to avoiding infrared light sources and cameras, we decided to build the system around an inexpensive USB camera. The system can therefore be

0-7695-2158-4/04 $20.00 (C) 2004 IEEE

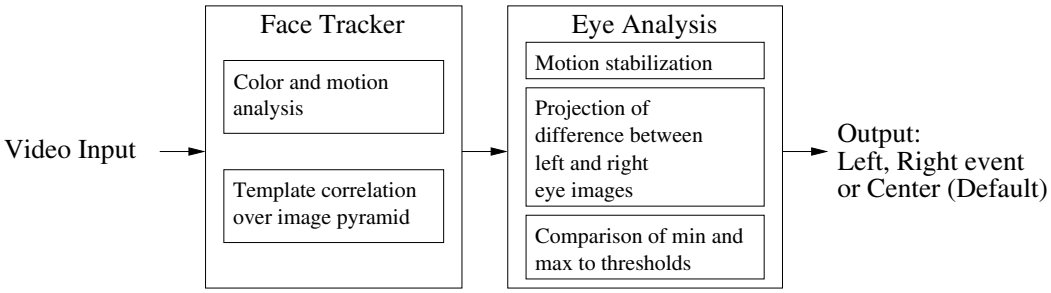


Figure 1: Major functions of the face tracking and eye analysis system.

run on any computer without the need for an expensive frame grabber or pan/tilt/zoom camera. Our system must be able to work with images that have a lower resolution than the images used in previous approaches [13, 14].

To be a useful human-computer interface, the system must run in real time. This excludes many existing approaches that do not run in real time. In addition, the system can not use all of the processing power of the computer because the same computer will have to run both the vision based interface as well as user programs such as web browsers or games.

EyeKeys tracks the face using multi-scale template correlation. The left and right eyes are compared to determine if the user is looking center, or to the left or right side. The output of our system can be used to control applications such as spelling programs or games.

We tested EyeKeys on the BlockEscape game. This game was developed specifically as an engaging way to test our interface system while reporting quantitative results. This is important because it motivates users and test subjects to try the system. Since we designed the game, we can use it to gather statistics on how well the interface works for various situations that we create.

This paper is organized in the following fashion: Section 2 discusses the methods employed in the EyeKeys system itself, including a thorough description of the EyeKeys' modules and the BlockEscape game. Section 3 details our experiments and results, while section 4 presents an in-depth discussion of our results, comparisons to other HCI systems, and plans for future extensions to our system.

2 Method

The system consists of two main modules: (1) the face detector and tracker, and (2) the eyes analysis module. Throughout the system, efficient processing techniques must be used to avoid wasting computing power. Major components of the system are presented in Figure 1.

In order to facilitate working with the eyes, we developed a simple but fast 2D face tracker. From the scale and location of the found face, a region of interest for the eye analysis module is obtained. The eye analysis module then determines if the eyes are looking towards the center, to the left, or to the right of the camera.

The output from the eye module can be the input to a simple computer control interface. Usually, looking center means "do nothing." The interface system can then map the left and right outputs to events such as mouse movements, left and right arrow keys, or tab and enter ("next link" and "follow link" for web browsing). Text can be entered in a variety of ways. An on–screen keyboard can scan to the correct letter, or letters can be selected by following a binary search approach. Some of this type of software is already in use with current interfaces for people with disabilities [3, 4, 6].

2.1 Face tracker

The face tracker itself consists of various modules, some of which are parts of previous face tracking approaches, e.g. [16]. Color and motion information is combined to create a mask to exclude areas of the search space for the correlation template matching. To deal with different scales, the system uses image pyramids [1] along each step. The pyramid consists of the input image at 640×480 with 8 stages to the lowest resolution of 32×24. The combined output from the color and motion modules is used as a mask to exclude regions of the image at every pyramid level from the search. Normalized correlation is used to match a small 12×16 template along each stage of the pyramid.

Color module. Skin color has been used to track faces previously, e.g. [11]. Here, it is used as a preprocessing mask. The color input image is converted into the YUV color space. A binary image is created with a 2D histogram lookup in UV space. If a pixel's lookup on the histogram for the specified UV value is over a threshold, then the pixel is marked as skin, otherwise not. The binary image is then decimated into the other levels using Gaussian blurring. A box filter is applied to the whole pyramid to make each level represent the color information for the appropriate scale of the face to search for.

The color histogram was trained on 15 face images. The images were marked by hand with a rectangle covering most of the facial regions. In cases where the color segmentation fails to provide good results, the histogram can be retrained on the fly simply by clicking on areas of skin in the live video. The histogram can be saved and loaded so that it can be used again for the same user or lighting conditions without retraining.

Motion module. A simple assumption is behind the motion module: if the person moves, then motion detection should find where they are moving, otherwise the person should be found in nearly the same location as in the previous frame. Frame differencing creates a motion image, which is then thresholded to create a binary image. The binary image is then decimated into a pyramid as in the color module, and a box filter is applied to account for scale.

The locations within 5 pixels of the previously found face location are set to one in the binary motion image. This prevents the motion mask from excluding the previous face location from the correlation search in cases when there is little or no motion. The two adjacent motion pyramid levels are also modified in this way to account for movements towards or away from the camera that are not caught by the motion segmentation. The area modified is proportional to the scale represented by the respective pyramid level.

Correlation module. Normalized correlation template matching is used to find the exact location of the face. A 12×16 face template is then correlated over all levels of the greyscale input pyramid (Y channel from the YUV color image).

The small face template allows the correlation to find a general face in the image, which is created by averaging the brightness values of 8 face images. Matching this template over the pyramid allows for faster correlation while preserving the relevant information in each level. The maximum correlation peak among all of the levels indicates the location of the face. The scale of the face is known by the level of the pyramid at which the maximum is found.

The template can be updated from the current video feed by clicking on the nose and then selecting the correct scale of the face from a slide bar.

2.2 Eye analysis

From the output of the face tracker, approximate location and scale of the eyes are known based on common anthropomorphic properties. Two images around the region of interest are cropped out from the highest resolution level in the pyramid. Movement of the head in the image plane or into different scale levels of the pyramid are accounted for to keep eyes near the center of the cropped images. The images are scaled using linear interpolation to 80×60 pixels.

Motion analysis and stabilization. Ideally, the two eye images would stay perfectly still even as the head moves. However, slight movements of the head by a few pixels may not be accurately tracked by the face tracker. A method must be used to stabilize the images for comparison, since motion will occur during blinks and eye movements. The method chosen here to locate the center of the eyes is frame differencing to make a motion image as in Figure 3, and then computing the first order moments. These "center" points are used to stabilize the image by adjusting location of the region of interest in the face image so that the eyes appear in the *same* locations in the eye images. Using this method, the eye images do not need to have as high resolution as required by many feature based location methods.

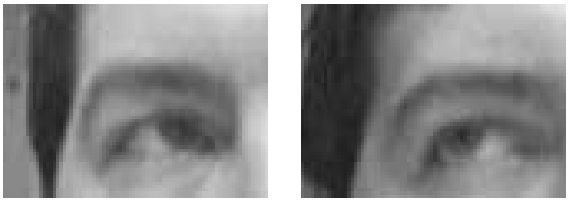
Figure 2: Motion detected by frame differencing is thresholded and used as a mask for the left-right image differencing.

Left–right eye comparisons. The left and right eyes are compared to determine where the user is looking. The left eye image is mirrored and subtracted from the right eye image. If the user is looking straight at the camera, the difference should be very small and the system determines that the user is looking straight. On the other hand, if the eyes are looking left, then the mirrored left eye image will be looking right as shown in Figure 3.

The signed difference between the two images show distinct pixel areas where the pupils are in different locations in each image. The unsigned difference can be seen in Figure 4. To further reduce extra information from the image areas outside of the eyes, the images are masked by their thresholded motion images. To determine the direction, the signed differences are projected onto the x-axis. The results of these projections can be seen in Figure 5. The signed difference will create peaks in the projection because the eye sclera pixels are lighter than pupil pixels. When the eyes look straight, these values nearly cancel themselves out creating low differences. When the person looks left, there will be an area of sclera minus pupil pixels followed by an area of pupil minus sclera pixels along a horizontal axis. This computation is thus a light area minus dark area resulting in an area with large positive values, followed by dark area minus light area, resulting in an area with large negative values. The opposite order holds true for right–looking eyes.

A strong positive peak followed by a negative peak in the projection indicates left direction, while a strong negative peak followed by a positive peak indicates right di-

rection. If the positive or negative peaks do not exceed a certain threshold, then the default (looking center) value is the output.



(a) Right eye looking left (b) Mirrored left eye looking left

Figure 3: Eye images automatically extracted from input video by face tracker and aligned by motion analysis.

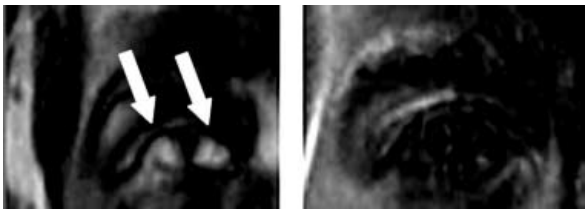


Figure 4: Difference between right and mirrored left eyes. Left: Eyes are looking to the left; arrows highlight large brightness differences. Right: Eyes are looking straight ahead.

Let I_ℓ and I_r be the $m \times n$ left and right eye images masked by motion information. The projection of the signed difference onto vector $\mathbf{a} = a_1, \ldots, a_m$ is computed by:

$$a_i = \sum_{j=1}^{n} (I_r(i,j) - I_\ell(m-i,j)). \quad (1)$$

Two thresholds T_p and T_d, are used to evaluate whether a motion occurred to the right, left, or not at all. The thresholds can be adjusted to change the sensitivity of the system. First, the maximum and minimum components of the projection vector $\mathbf{a}$ and their respective indices are computed:

$$a_{\min} = \min_{i=\{1,\ldots,m\}} (a_i) \text{ and } a_{\max} = \max_{i=\{1,\ldots,m\}} (a_i) \quad (2)$$

$$i_{\min} = \arg\min_{i=\{1,\ldots,m\}} (a_i) \text{ and } i_{\max} = \arg\max_{i=\{1,\ldots,m\}} (a_i). \quad (3)$$

The minimum and maximum values are then compared to the projection threshold T_p:

$$a_{\min} < -T_p \quad \text{and} \quad a_{\max} > T_p. \quad (4)$$

This threshold assures that there is a sufficient brightness difference to indicate a left or right motion. The second

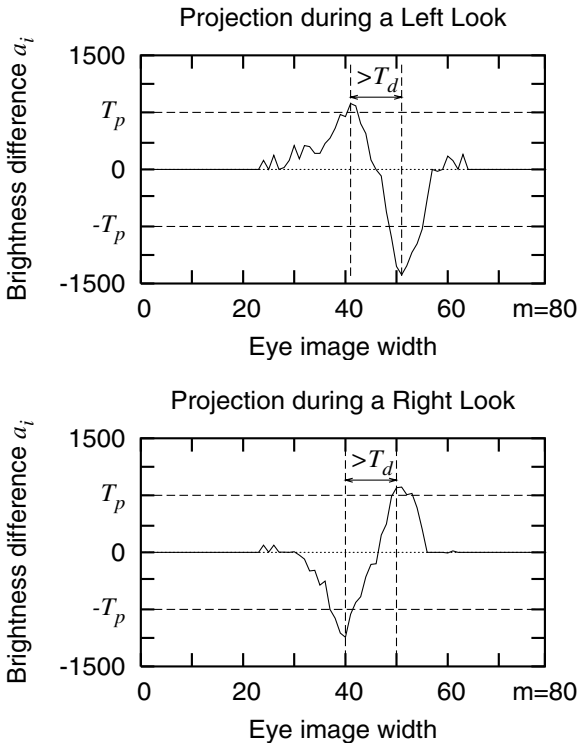


Figure 5: Results of projecting the signed difference between right and mirrored left eyes onto the x–axis. The top image is the result of left–looking eyes. The bottom image is the result of right–looking eyes.

threshold T_d is used to guarantee a minimal spatial difference between the minimum and maximum projection values when motion is detected. The direction of motion is determined as follows:

$$i_{\max} - i_{\min} > T_d \Rightarrow \text{'right motion'} \quad (5)$$

$$i_{\max} - i_{\min} < -T_d \Rightarrow \text{'left motion'}. \quad (6)$$

2.3 Classification

Information from both the motion and comparison analysis are combined to determine if there was an intentional look to the left or right. The system searches for motion to the left, followed by eye direction to the left in order to trigger the "user has looked left" event. The corresponding right event is similarly triggered.

A limit was set on how frequently events can be triggered in order to avoid the system from becoming confused and triggering many events in quick succession. In the future however, it may be preferable to let the user keep looking to one side in order to trigger many events in a row to simulate holding down an arrow key. Audio feedback or multiple

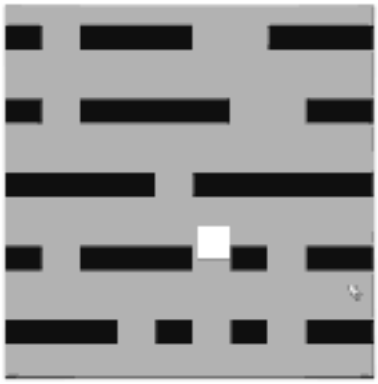


Figure 6: Screenshot of the game BlockEscape. The player navigates the block through the holes by moving the mouse left or right as the block falls towards the bottom of the screen.

monitors would be needed to let the user know when events are triggered.

2.4 BlockEscape game

BlockEscape is an easy to learn game which allows for an interactive and engaging user experience while concurrently providing a useful framework for testing HCI techniques. Its development was motivated by the fact that many applications whose primary goal is testing an HCI system ignore that the test subject cannot be expected to remain attentive for long periods of time. Providing an enjoyable game as a statistics gathering device allows subjects to play for long stretches of time, and thus allows for us to retrieve a large amount of data while tests subjects use EyeKeys. Figure 6 shows a screenshot of BlockEscape.

The rules of the game are as follows. The wall levels, which are the black rectangles in Figure 6, are fixed objects that move upwards at a constant rate. The user, who only controls the white block, must lead it into the holes between these walls, where it will "fall through" to the next wall level. The user is restricted to move the white block horizontally in two directions: left and right. The block movement is *triggered* by issuing a 'left motion' or 'right motion' command. The command can be issued using the EyeKeys interface, the mouse, or the left/right keys on the keyboard. The block will continue to move in that direction until the wall level is exited or a key in the opposite direction is pressed. When the block reaches the bottom of the screen, the user wins. Conversely, if the block ever reaches the top of the screen, the game ends.

There are numerous ways to configure game play. The significant configurations are game speed and distance between walls. The game speed specifies the how often the game state is updated: by increasing this setting, the game is made slower and therefore more easy to play. The game difficulty can be similarly be modified by setting the distance between successive walls. These settings allow the game to be configured appropriately for the abilities of the user with a chosen interface method.

Methods for gathering statistics. Incorporated within BlockEscape is the functionality to gather detailed usage statistics that are generated and recorded during game play. These statistics offer a detailed view of the blocks movements throughout the game, including a score that gauges the users movement "mistakes" compared to "good" movements.

Suppose that the block is on the rightmost side of the screen, and that there is one hole on the leftmost side of the screen. It would be a mistake, then, if the user moved to the right at any time, since there is clearly only one possible movement that will lead to success. In cases with multiple holes on a particular wall level, if there is a clear choice which direction to choose, then statistics can still be reported. The following equations are used to determine these player deviations:

$$d_{\mathrm{ij}} = |x_{\mathrm{ij}} - h_{\mathrm{i}}| \tag{7}$$

$$D_{\mathrm{ij}} = \begin{cases} 0 & \text{if } d_{\mathrm{ij}} < d_{\mathrm{i\,j-1}}, \text{ or } j = 0 \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

where h_{i} is the hole's position on wall level i and x_{ij} is the block's position on wall level i at time j. Distance d_{ij} is defined as the distance from the block's current position to the hole and D_{ij} determines whether the block is closer or farther away from the nearest hole. We also define the normalized deviation for wall level i as:

$$\sigma_{\mathrm{i}} = \frac{1}{sg} \sum_{j=1}^{W_i} D_{\mathrm{ij}} \tag{9}$$

where s is the current block speed in pixels, g represents the width of the game board in pixels, and W_i is the number of cycles the block is on wall level i. The standard deviation σ_{avg}, averaged over all wall levels, was approximately zero in our tests with users employing a keyboard. Therefore, we can assume that all movement errors encountered during testing are not due to user error resulting from difficulty of the game itself, but are instead due to the interface system being employed. These errors are represented by a deviation score in the final statistics report, which displays the number of deviations for each individual wall level, and a coordinate-pair listing denoting the pixel extents of each individual wall in the wall level.

This information may then be used to reconstruct the exact wall level sequence as was seen in a previous game, allowing the user to play the same game multiple times. This

is useful in that it we can now use the same wall level sequence on multiple users, and thus get results that are comparable.

3 Experiments and results

3.1 EyeKeys performance evaluation

Experimental setup. EyeKeys is designed to be used by a person sitting in front of a computer display. The camera is mounted on the end of an articulated arm, which allows the camera to be optimally positioned in front of a computer monitor. The USB camera we used is a Logitech Quickcam Pro 4000, with a retail price of $79.99. The tests were run on an Athlon 2100.

Tests were created to determine if the system can detect when a user intentionally looks to the left or to the right. The average face template used by the tracker was first updated to include the test subject. Testers were told to look at the computer monitor. When asked to look left, the tester should move their eyes to look at a target point to the left of the monitor. A similar target was to the right side of the monitor. After the look was completed, the user should look back at the monitor.

We created a random ordered sequence of twenty looks: ten to the left and ten to the right. The same sequence was used for all the test subjects. If the system did not recognize a look, the user was asked to repeat it. The number of tries required to make a recognition was recorded. If the system made an incorrect recognition, that fact was recorded and the test proceeded to the next look in the sequence.

Results. Our system was tested by 8 people. All of the faces of the test subjects were correctly tracked in both location and scale while moving between 2 and 5 feet from the camera. Our system correctly identified 140 out of 160 intentional looks to the left or right. This corresponds to an 87.5% success rate. For the system to detect and classify 160 looks, the users had to make 248 attempts. On average, 1.55 actual looks are made for each correctly identified look event. The results are summarized in Table 1.

Table 1: Results of testing the user interface system on a sequence of left and right looks.

		Actual		
		Left	Right	% Correct
Observed	Left	72	12	90.0%
	Right	8	68	85.0%
	Missed	40	48	

Some of the test subjects were more successful than others. One subject had all 20 looks correctly identified while making 24 actual looks. Cases where an incorrect recognition occured were due to a problem with alignment of the right and mirrored–left eyes. The number of extra look attempts is probably due to high thresholds that were chosen to avoid false detection of looks, since it is better to miss a look than to misclassify a look. Other incorrect recognitions were due to the system missing a look in one direction, but detecting eye movement back to the center position as a move in the opposite direction.

3.2 BlockEscape experiment

Experimental setup. Four test subjects participating in this experiment were read the rules of BlockEscape, followed by two demonstrations of the game using a mouse. We chose to test the Camera Mouse in this experiment in order to gauge the effectiveness of EyeKeys against a previously developed HCI system for people with disabilities. The keyboard was chosen as a control against the HCI systems. All subjects were unfamiliar with BlockEscape, EyeKeys, and the Camera Mouse.

In the "practice" phase, the subjects were allowed to become familiar with the game and the interfaces. They played up to three trial games, or for up to three minutes, on the keyboard, Camera Mouse and EyeKeys. They were then asked to play at least one game for 30 seconds with each device.

For the "trial" phase, the test subjects played three games on each input device, the results are shown in Table 2.

Table 2: Results of four users employing three devices to play BlockEscape. Units are percetage of game playing area.

	Device		
	EyeKeys	Camera Mouse	Keyboard
σ_{avg}	2.9	2.27	0
Median	2.54	0	0
Std. Dev.	4.01	2.68	0
Wins	$\frac{10}{12}$ (83%)	$\frac{10}{12}$ (83%)	$\frac{12}{12}$ (100%)

Results. Notice that the win percentage of EyeKeys compared to the Camera Mouse was the same, although EyeKeys had a higher σ_{avg}, median, and standard deviation. It is also of interest that a Camera Mouse failure requires manual intervention to correct, while an EyeKeys user could merely look in the appropriate direction to correct a mistake. The keyboard control is obviously the most accurate way to play the game for those that are able, however, the results demonstrate that our system works well enough as an interface to play this game, and that it is comparable to an existing interface that is in actual use.

Users had different levels of success with EyeKeys. One user mastered EyeKeys quickly, winning all three games,

but had trouble with the Camera Mouse: losing one game and performing poorly on another. With EyeKeys, all the other users improved their performance on succeeding games. This did not hold true for the Camera Mouse.

3.3 Initial experience: A test user with severe disabilities

We were able to hold a preliminary test of the EyeKeys system with a user with cerebral palsy. This user can control his eyes and has some control over head movements. However, he also has involuntary head movements.

We asked him to use the EyeKeys system to move a window left and right across the screen. We observed that he was able to move the block in the direction that we asked him to in most cases. Sometimes, involuntary head motion would cause the system to detect an unintentional eye event. Adjusting the thresholds in future tests may allow the system to work better with these unpredictable changes in the location of the head.

3.4 Real-time performance of system

Our system achieves real–time performance at 15 frames per second, which is the limit of the USB camera at 640×480 resolution. The BlockEscape game had no problem running concurrently with the real-time vision interface system. Our experience indicates that the performance of our system easily enables it to run concurrently with other applications such as spelling programs and web browsers.

4 Discussions and future work

Real-time performance. The correlation module of the face tracker is the most computationally expensive function required in our system. The face tracker employs multiscale techniques in order to improve real-time performance. The template correlation over the image pyramid is more efficient than performing multiple correlations with a scaled template. In addition to improving accuracy, the color and motion information is used to reduce the search space of the template correlation, further improving efficiency.

The eye analysis is relatively computationally inexpensive. The eye direction is computed in time linear to the size of the eye image.

Design motivations. The ability to update the average face template is important for the correlation tracker. This can help fix two problems. The average face template allows most people to use the system without manual initialization. However, if the user's face does not correlate well with the current template, the updated template will be more specific to the user and will work better. A template from one person generally works well in finding another person since the information contained in the template is non-specific. Another significant benefit of being able to change the template is that an updated template will allow the correlation to work better under different lighting conditions. The template can be saved and loaded so that it does not have to be retrained for the same user or lighting conditions. While normalized correlation can work with uniform intensity changes, it has problems if the user becomes more brightly lit from one side. Updating the template solves this.

The approach of EyeKeys to exploit symmetry works well even though the eye images are of low resolution. Other more sophisticated approaches of gaze detection that model the eye features require higher resolution eye images. In the cases where eye features such as corners of the eyes or curve of the iris cannot be used, the difference mirroring approach allows eye direction classification to still be successful.

The two thresholds that determine when the user looks right or left are adjustable. Increasing T_p makes the system more likely to miss an intentional look, but less likely to misclassify a look. Increasing T_d has the effect of requiring that the looks be faster and more deliberate. While this can decrease false detections, it also makes the system difficult and uncomfortable to use.

Testing experience and comparisons. Our test subjects had little difficulty learning the EyeKeys interface. After only a minute of practice, users were able to play BlockEscape. In addition, most subjects improved after each game, leading us to believe that EyeKeys users will become as proficient as Camera Mouse users over time. With further testing we may determine if experienced users of EyeKeys outperform those users of the Camera Mouse.

In comparison to the Camera Mouse, our system performed well. When the mouse tracker gets lost in the Camera Mouse, the performance decreases dramatically. In our system, a false detection can be easily rectified by a correct detection. This, however, is specific to certain applications. For instance, if our system caused a web browser to follow a hyperlink in error, then it would be difficult to return to the original page without manual intervention. A possible solution would be to detect other events, such as blinks, to serve as an undo command. Another solution would be to add a "confirm" step.

Since this system was designed as an HCI application, it was expected that the user would be cooperative and *try* to make it work. This is in contrast to a surveillance application where the subject may not know about the system. The face in the images is thus assumed to be frontal to the camera. Head turns or rotations can cause the face tracker or eye direction classification to break. Future tests will determine the limitations of head orientations and the use of eye glasses with this approach.

Future work and improvements. EyeKeys has the potential to become an integral part of a complete HCI system [12, 15]. Combining EyeKeys with other HCI applications would give the user greater control over the computer, and if utilized with other facial processing techniques, could prove to be an all-purpose command interface. While the current research is focused on creating an interface system for people with severe disabilities, gaze detection systems such as EyeKeys can be useful in other areas such as linguistic and communication research, or monitoring a vehicle driver's attention.

The system could be improved with an algorithm to more precisely locate the eyes. This would allow the left–right detection to be more robust during head movements. It would also possibly allow detection of the degree that the eyes are looking to the side. Analysis of the difference projection could be done in a more sophisticated manor: fitting a function to the curve may improve detection accuracy.

Our system should also work better with head motion. One solution could be to not allow eye movement detection when the head is moving. However, that may cause a problem for disabled users that have involuntary head movements.

Future possibilities for extending this system include the addition of a blink analysis module [6], which would give the interface three events to work with. Unfortunately, many subjects with severe cerebral palsy cannot control their eye blinks. Another way to extend the system is with further analysis of the duration that the user looks left or right to allow mapping of more events to additional commands.

Eventually, it would be useful to increase the number of gaze directions that can be detected reliably, but this is a very challenging problem with the low-grade cameras used here. This would, however, allow mouse–like control of a cursor.

One extension of BlockEscape would be the actual saving of an entire game session. It would then be possible to replay, in a video file, the saved game from beginning to end, allowing further analysis of gameplay. This will most likely be available in a future release of BlockEscape.

Acknowledgments

Funding was provided by the National Science Foundation (IIS-0308213, IIS-039009, IIS-0093367, P200A01031, and EIA-0202067).

References

[1] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid methods in image processing. *RCA Engineer*, 29:33–41, 1984.

[2] Applied Science Laboratories, Bedford, MA. http://www.a-s-l.com.

[3] M. Betke, J. Gips, and P. Fleming. The Camera Mouse: Visual tracking of body features to provide computer access for people with severe disabilities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 10(1):1–10, March 2002.

[4] P. DiMattia, F. X. Curran, and J. Gips. *An Eye Control Teaching Device for Students without Language Expressive Capacity – EagleEyes*. The Edwin Mellen Press, 2001. See also http://www.bc.edu/eagleeyes.

[5] A. Gee and R. Cipolla. Determining the gaze of faces in images. *Image and Vision Computing*, 12(18):639–647, 1994.

[6] K. Grauman, M. Betke, J. Lombardi, J. Gips, and G. Bradski. Communication via eye blinks and eyebrow raises: Video-based human-computer interfaces. *Universal Access in the Information Society*, 2(4):359–373, November 2003.

[7] T. Hutchinson, K. P. White JR., W. N. Martin, K. C. Reichert, and L. A. Frey. Human-computer interaction using eye-gaze input. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1527–1533, 1989.

[8] Q. Ji and Z. Zhu. Eye and gaze tracking for interactive graphic display. In *Proceedings of the International Symposium on Smart Graphics*, Hawthorne, NY, June 2002.

[9] A. Kapoor and R. W. Picard. Real-time, fully automatic upper facial feature tracking. In *Proceedings of the Fifth IEEE International Conference on Automatic Face Gesture Recognition*, pages 10–15, Washington, DC, May 2002.

[10] C. H. Morimoto, D. Koons, A. Amit, and M. Flickner. Pupil detection and tracking using multiple light sources. Technical Report RJ-10177, IBM Almaden Research Center, 1998. domino.watson.ibm.com/library/cyberdig.nsf/Home.

[11] K. Schwerdt and J. L. Crowley. Robust face tracking using color. In *Proceedings of the 4th IEEE International Conference on Automatic Face and Gesture Recognition*, pages 90–95, Grenoble, France, March 2000.

[12] R. Sharma, V. I. Pavlovic, and T. S. Huang. Toward multimodal human-computer interfaces. *Proceedings of the IEEE*, 86(5):853–869, May 1998.

[13] S. Sirohey, A. Rosenfeld, and Z. Duric. A method of detecting and tracking irises and eyelids in video. *Pattern Recognition*, 35(5):1389–1401, June 2002.

[14] Y. Tian, T. Kanade, and J. Cohn. Dual-state parametric eye tracking. In *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 110–115, Grenoble, France, March 2000.

[15] M. Turk and G. Robertson. Perceptual user interfaces. *Communications of the ACM*, 43(3):32–34, March 2000.

[16] M. Yang, D. Kriegman, and N. Ahuja. Detecting faces in images: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58, January 2002.