# A Real-Time Vision Interface Based on Gaze Detection – EyeKeys

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

Computer Science Department
Boston University
mageejo@cs.bu.edu
betke@cs.bu.edu
mrscott@cs.bu.edu
bwabes@cs.bu.edu

There are cases of paralysis so severe the ability to control movement is limited to the muscles around the eyes. In these cases, eye movements or blinks are the only way to communicate. Current computer interface systems are often intrusive, require special hardware, or use active infrared illumination. An interface system called EyeKeys is presented. EyeKeys runs on a consumer grade computer 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 to the left or right side. The detected eye direction can then be used to work with applications that can be controlled with only two inputs. The game "BlockEscape" was developed to gather quantitative results to evaluate EyeKeys with test subjects.

## 1 Introduction

Some people may be so severely paralyzed that their voluntary movements are limited to movements of the eyes. To communicate with family, friends, and care givers, they look in a certain direction or blink for "yes" and "no" responses. Innovative assistive technologies are needed to enable them to access the computer for communication, education, and entertainment. As progress toward that goal, we present an interface called EyeKeys that simulates computer keyboard input and is based on gaze detection that exploits the symmetry between left and right eyes.

There has been much previous work in computer assistive technologies, e.g., [2, 3, 6, 16, 17, 23, 29]. Most of these methods, though successful and useful, also have drawbacks. Many currently available or early systems are often intrusive, or use specialized hardware [29]. For example, the EagleEyes system [6] uses electrodes placed on the face to detect the movements of the eyes and has been used by disabled adults and children to navigate a computer mouse. 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. Large headgear is not suited for all users, especially small children. One of our goals is to design a non-intrusive system that does not need attachments.

Another successful system is the Camera Mouse [3]. People with disabilities can 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 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 [8, 12, 13, 14, 18, 28]. The infrared light reflects off the back of the eye to create a distinct "bright eye" effect in the image. If switching the infrared light on and off is synchronized with the camera, the pupils can be located by differencing the bright eye image obtained with infrared illumination from the subsequent image without infrared illumination. The illumination also creates a "glint," a reflection off the surface of the eye. One technique to find the gaze direction is to analyze the difference vector between pupil center and glint. 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 another 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 run on any computer without the need for an expensive frame grabber or pan/tilt/zoom cameras as required in some previous work [5]. Our system must be able to work with images that have a lower resolution than the images used in previous approaches [15, 22, 25].

To be a useful human-computer interface, the system must run in real-time. This excludes 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. This is accomplished by exploiting the symmetry between the left and the right eyes. If one eye image is mirrored and subtracted from the other, the large differences will be due to the difference in pupil location.

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. We can use the game to gather statistics on how well the interface works for various situations that we create.

This chapter is organized in the following manner: 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 Sect. 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 EyeKeys system performs two main tasks: (1) face detection and tracking, and (2) eye analysis. Throughout the system, efficient processing techniques are used to enable real-time performance. Major components of the system are presented in Fig. 1.

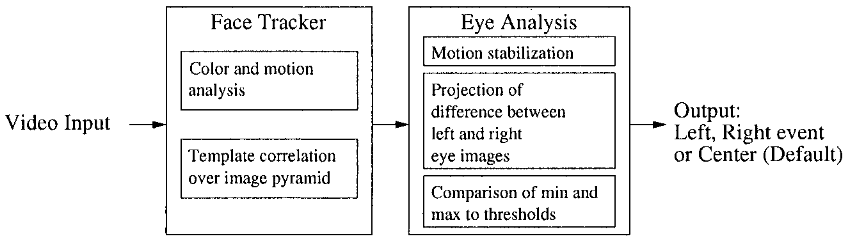
Video Input
Face Tracker
Color and motion analysis
Template correlation over image pyramid
Eye Analysis
Motion stabilization
Projection of difference between left and right eye images
Comparison of min and max to thresholds
Output: Left, Right event or Center (Default)

**Fig. 1.** System Diagram for EyeKeys

In order to facilitate working with the eyes, we developed a fast two-dimensional (2D) face tracker. From the scale and location of the face located by this tracker, regions of interest for the eye analysis are obtained. The eye analysis algorithm then determines if the eyes are looking toward the center, or have moved to the left, or to the right of the camera.

The output from the eye module can be the input to a 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 other key combinations. This allows the system to be configured for a variety of applications such as playing games, entering text, or navigating a web site.

## 2.1 Face Detection and Tracking

The face detection and tracking method consists of various parts, some of which were used in previous face tracking approaches, e.g., [11, 27]. Color and motion information is combined to create a mask to exclude areas of the search space for the correlation-based matching of a 12×16-pixel face template. To enable detection of faces that differ in size (for example, a user may have a large head or sit close to the camera), the system uses image pyramids [1] along each step of the face detection. To avoid the large size difference between traditional pyramid levels, where the image at each successive level is half the size of the previous image, the pyramid structure has been modified to include images with intermediate resolutions. This allows the system to find face scales at smaller discrete steps. The resolutions of the images of the pyramids are listed in Table 1.

**Table 1.** Resolutions used by the image pyramids. Coordinates in any level can be transformed into coordinates in the 640×480 input frame by multiplying by the scale factor. Levels 2 through 7 are used to find the face

| Level | Width | Height | Scale Factor |
|---|---|---|---|
| 0 | 640 | 480 | 1 |
| 1 | 320 | 240 | 2 |
| 2 | 160 | 120 | 4 |
| 3 | 128 | 96 | 5 |
| 4 | 80 | 60 | 8 |
| 5 | 64 | 48 | 10 |
| 6 | 40 | 30 | 16 |
| 7 | 32 | 24 | 20 |

**Color analysis.** Skin color has been used to track faces previously, e.g., [19]. Here, it is used as a preprocessing mask. The color input image is converted into the YUV color space [24]. YUV was chosen because the camera can be configured to provide images in that format, and the color information is contained within two dimensions. 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 [1]. A box filter that smoothes an image by averaging with a support of 12×16 pixels is applied to each image in the pyramid so that each pyramid level represents the color information for the appropriate scale of the face to search for. Thresholding then produces a binary pyramid mask $P_{\text{color}}$ (Fig. 2).

The color histogram was trained on 15 face images which 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

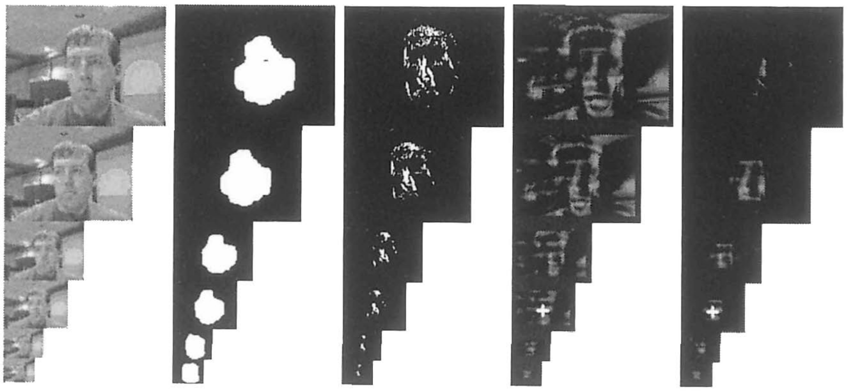

**Fig. 2.** Pyramids $P_{\mathrm{input}}$, $P_{\mathrm{color}}$, $P_{\mathrm{motion}}$ before application of box filter, $P_{\mathrm{correlation}}$, and $P_{\mathrm{masked}}$ computed by the face detection and tracking algorithm. The cross indicates the maximum correlation peak in the pyramid and after applying the appropriate scale factor in Table 1, yields the location and scale of the face

retrained during system operation by clicking on areas of skin in the live video. The histogram can be saved and reloaded so that it can be used again for the same user or lighting conditions without retraining.

There are various situations when the UV-histogram might need to be retrained. Certain changes in lighting conditions can result in changes of the UV values of skin. A histogram trained on one person might not work well with a person with a different skin tone. Pixels corresponding to objects such as wooden doors or tan carpets can often have similar pixel values as skin. The default histogram will represent a wider range of skin tones, while a histogram trained on one person will represent that person's skin more exclusively. Since skin color segmentation may not yield accurate segmentation results due to the difficulties described above, UV-based segmentation is used only as a preprocessing mask for face localization.

**Motion analysis.** Frame differencing creates a motion image that is decimated into a pyramid (Fig. 2). Pixels in the face with large brightness gradients also have large values in the motion image if the face is moving. The box filter is applied again to each motion image in the pyramid to account for the appropriate scale of the face to search for. This yields, after thresholding, a binary pyramid mask $P_{\mathrm{motion}}$. The pyramid $P_{\mathrm{motion}}$ computed from the scene shown in Fig. 2 looks similar to the color pyramid mask $P_{\mathrm{color}}$.

In cases when there is little or no motion, the motion mask must be prevented from excluding the previously found face location from the correlation search. Locations near the previous face location are therefore set to one in the binary motion image. The other motion pyramid levels are also modified in this way to account for movements toward 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 matching.** Template matching based on the normalized correlation coefficient [4] is used to find the location of the face. A small, 12×16 face template is correlated over all levels of the grayscale input pyramid $P_{\text{input}}$ (Y channel from the YUV color image), which allows for fast processing. The resulting correlation values yield the pyramid $P_{\text{correlation}}$ (Fig. 2). 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. To eliminate possible ambiguous correlation peaks in the background, the color and motion information masks are applied to $P_{\text{correlation}}$. An efficient implementation of the correlation function can also use the mask to save processing time by skipping background locations excluded by the mask.

The face template is created by averaging the brightness values of 8 face images. This ensures that the relevant information that it represents a face is preserved, while specific features of a particular person are smoothed, and thus allows the correlation method to find a "general" face in the image.

## 2.2 Eye Analysis

Given the estimate of face location provided by the face tracker, the approximate location and scale of the eyes can be inferred from simple anthropomorphic properties: The eyes must be located in a region above the center of the face, the left eye must be on the right side of this image region and the right eye on the left. Taking advantage of these properties, the eye analysis module crops out two subimages containing the eyes from the highest resolution image. The size of the subimages depends on the scale at which the face was found. To simplify the eye analysis, the system produces eye images of a fixed size of 60×80 pixels by linear interpolation.

**Motion analysis and stabilization.** Ideally, the two eyes would be centered in the respective eye images 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 eye images for comparison. The method chosen here to locate the center of the eyes is frame differencing to create binary motion images (Fig. 3), followed by computing the first-order moments. These "centroid" points are used to adjust the estimates of the eye locations in the face image. Using this method, the eye images do not need to have as high a resolution as required by many feature-based eye localization methods, e.g., [25].

**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 is small. On the other hand, if the eyes are looking left, then the mirrored left eye image appears to be looking right as shown in Fig. 4.

The signed difference between the two images shows distinct pixel areas where the pupils are in different locations in each image. The unsigned dif-

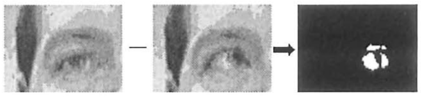

**Fig. 3.** Motion detected by frame differencing is thresholded and used as a mask for the differencing of left-right eye images, and for finding the centroids for motion stabilization

ference can be seen in Fig. 5. To reduce extra information from the image areas outside of the eyes, the images are masked by their thresholded motion images (Fig. 3). To determine the direction of the eyes, the signed differences are projected onto the $x-$axis (Fig. 6). The signed difference creates peaks in the projection because eye sclera pixels are lighter than pupil pixels.

If the user is looking left, the signed difference operation creates large values in the projection because the dark-gray iris and pupil pixels in the left image are subtracted from the light-gray eye sclera pixels in the right image. This is followed by small values in the projection because light-gray eye sclera pixels in the left image are subtracted from dark-gray iris and pupil pixels in the right image. Vice versa, if the user is looking right, there will be a valley in the projection, followed by a peak (Fig. 6). If the peaks and valleys in the projection do not exceed a certain threshold, then the eye analysis method outputs the default value "looking center."

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

**Fig. 4.** Eye images automatically extracted from input video by face tracker

**Fig. 5.** Absolute difference between right and mirrored left eyes. Left: Eyes are looking to the left; arrows indicate large brightness differences due to pupil location. Right: Eyes are looking straight ahead

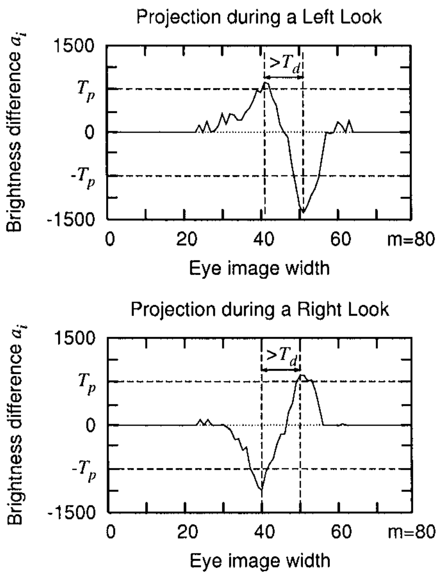

**Fig. 6.** Results of projecting the signed difference between right and mirrored left eyes onto the $x$−axis. The top graph is the result of left-looking eyes. The bottom graph is the result of right-looking eyes

Let $I_\ell$ 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:

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

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:

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

$$i_{\min} = \operatorname*{arg\,min}_{i=\{1,\ldots,m\}} (a_i) \quad \text{and} \quad i_{\max} = \operatorname*{arg\,max}_{i=\{1,\ldots,m\}} (a_i) \tag{3}$$

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

$$a_{\min} < -T_p \qquad \text{and} \qquad a_{\max} > T_p \tag{4}$$

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

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

$$i_{\max} - i_{\min} < -T_d \quad \Rightarrow \quad \text{`left motion'} \tag{6}$$

### 2.3 Classification

Information from both the motion and eye comparison analysis are combined to determine if there was an intentional look to the left or right. The system detects motion 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. The limit was set experimentally at one event every 0.5 seconds. The user must move his or her eyes back to the center position before attempting to trigger another left or right event. 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 a key. Audio feedback or multiple monitors would be needed to let the user know when events are triggered.

### 2.4 BlockEscape Game

The game BlockEscape was developed as a tool to test the performance of EyeKeys as an interface. It is a game that is easy to learn and provides an interactive and engaging user experience, which is particularly important for users with severe disabilities who have difficulty remaining physically active for long periods of time. Providing an enjoyable game as a statistics gathering device may encourage subjects to play for longer periods of time. Figure 7 shows a screenshot of BlockEscape.

The rules of the game are as follows. The walls, which are the black rectangles in Fig. 7, are fixed objects that move upward at a constant rate. The user, who controls a white block, must lead it into the holes between these walls, where it "falls through" to the next wall. The user is restricted to move the white block horizontally 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 continues to move in that direction until it falls through a hole or the user issues a new direction command. If the block reaches the bottom of the screen, the user wins. If the block is pushed to the top of the screen by the walls, the user loses.

There are numerous ways to configure game play. The significant configuration variables are game speed and distance between walls. The game speed specifies how often the game state is updated: by increasing this setting, the game is made slower and therefore easier to play. The settings allow the game

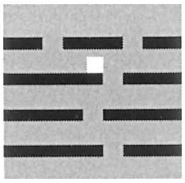

**Fig. 7.** Screenshot of the BlockEscape game. The player navigates the block through the holes by moving the mouse left or right or pressing keys as the block falls toward the bottom of the screen

to be configured appropriately for the abilities of the user with a chosen interface method.

**Methods for gathering statistics.** During playing, usage statistics, in particular, the departure of the user-controlled block from an optimal path, were computed based on the positions of the block, walls, and holes and compiled into XML (Extensible Markup Language) documents. If the block is on the rightmost side of the screen, and there is one hole on the leftmost side of the screen, the user should obviously move the block left. In cases with multiple holes on a particular wall, the user should move the block in the direction to the closest hole. The following equations are used to determine the player deviations:

$$d_{ij} = |x_{ij} - h_i| \tag{7}$$

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

where $h_i$ is the hole's position on wall $i$ and $x_{ij}$ is the block's position on wall $i$ 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 define the deviation for wall $i$ as:

$$\sigma_i = \sum_{j=1}^{W_i} D_{ij} \tag{9}$$

where $W_i$ is the number of game-update cycles during which the block is on wall $i$. The deviation $\sigma_{\text{avg}}$, averaged over all walls, 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.

The XML document includes a coordinate-pair listing denoting the configuration of each individual wall during a game play. This information may then be used to reconstruct the exact wall sequence that was seen in a previous game, allowing the user to play the same game multiple times. This is also useful for playing the same sequence with multiple users.

## 3 Experiments and Results

This section describes experiments to evaluate the performance of EyeKeys.

### 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.

The EyeKeys system was tested by 8 able-bodied people. 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 face detection and tracking method was first updated with a template representing the face of the test subject. Testers were told to look at the computer monitor. When asked to look left, the tester should quickly 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 move his or her eyes 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 recognize the look was recorded. We also recorded when the system misinterpreted a left or right look, and the test proceeded to the next look in the sequence.

**Results.** The faces of all test subjects were correctly tracked in both location and scale while they were 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 2.

**Table 2.** Number of actual and detected left and right looks in testing the EyeKeys system

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

EyeKeys was more successful with some of the test subjects than others. For example, one subject had all 20 looks correctly identified while only making 24 actual look attempts. Cases where an incorrect recognition occurred were likely due to a problem with alignment of the right and mirrored-left eyes. The number of extra look attempts is due to high thresholds that were chosen to avoid false detection of looks, since it was decided that 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 measure 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 3.

**Table 3.** Results of four users employing three devices to play BlockEscape. Units are percentage of game playing area

| | Device | | |
|---|---|---|---|
| | EyeKeys | Camera Mouse | Keyboard |
| $\sigma_{\text{avg}}$ | 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.** The win percentage of EyeKeys compared to the Camera Mouse was the same, although EyeKeys had a higher $\sigma_{\text{avg}}$, median, and standard deviation. We also noted that a Camera Mouse failure requires manual intervention to correct, while an EyeKeys user could frequently make another look in the appropriate direction to correct a mistake. However, the median deviation for the Camera Mouse system indicates that errors were quickly corrected by the user in most instances. The median deviation for EyeKeys is due to the time restriction limit between detections. The keyboard control is obviously the most accurate way to play the game for those that are able, however, the results demonstrate that EyeKeys works well enough as an interface to play this game, and that it is comparable in performance to an existing assistive-technology interface that is in current use.

Users had different levels of success playing BlockEscape with EyeKeys. One user mastered EyeKeys quickly, winning all three games, but had trouble with the Camera Mouse. With EyeKeys, all the other users improved their performance on succeeding games. This did not hold true for the Camera Mouse experiments.

### 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 frequently able to move the window in the direction that we asked him. Sometimes, involuntary head motion would cause the system to detect an unintentional eye event. Since he has used the Camera Mouse on numerous occasions, he would often move his head in a motion that would work with the Camera Mouse, but caused problems with EyeKeys. Adjusting the thresholds in future tests may allow the system to work better with these head motions. The system could also be configured to ignore eye movements when head movements are detected.

### 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. The performance of EyeKeys easily enables it to run concurrently with other applications such as spelling programs and web browsers.

## 4 Discussion and Future Work

**Real-time performance.** Correlation-based face tracking is the most computationally expensive procedure in our system. The face tracker employs multi-scale 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 could be 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 proportional to the size of the eye image.

**Design motivations.** The approach of EyeKeys to exploit symmetry works well with eye images of low resolution. Other approaches to gaze detection that model eye features require higher resolution eye images, e.g., [25]. If such images cannot be obtained, and therefore 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 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.

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. This is useful in cases when a person's face does not correlate well with the default template. Detection methods based on the normalized correlation coefficient can work well with uniform changes in brightness [4], however, problems may occur if the user becomes more brightly lit from one side. In addition, the template-based detection method works well if the template face and the user's face remain in the same orientation. If the default template is applied, the user should face the camera and hold his or her head straight. An updated template can work with specific head tilts and lighting conditions.

**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.

EyeKeys performed well in comparison to the Camera Mouse. When the Camera Mouse loses track, the performance decreases dramatically. In our system, a false detection can be 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. Since this system was designed as an HCI application, it was expected that the user would be cooperative and *try* to make it work. Future tests will determine the limitations for EyeKeys to detect head tilts or rotations.

**Future work and improvements.** EyeKeys has the potential to become an integral part of a complete HCI system, e.g., perceptual interface systems described in references [20, 26]. 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 part of 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.

EyeKeys can be adapted for specific applications such as text entering. Text can be entered in a variety of ways, for example, an on-screen keyboard can scan to the intended letter, or letters can be selected by following a binary search of the alphabet. Some of this type of software is already in use with current interfaces for people with disabilities [3, 6, 7, 9, 10, 21].

Another important application for EyeKeys is navigating a web browser. The two commands, left and right looks, could map to the *Tab* and *Enter* keys of the keyboard. This allows the user to tab through the links on a page, and then select a link to follow. If the user starts on a web page with a hierarchical structure of the web, such as Yahoo, then information can be retrieved by following a few links. This would allow access to news, weather, sports, entertainment, and educational material. A current issue is that following an incorrect link by mistake results in the user on the wrong page. A possible solution would be to detect other events, such as blinks [10], to serve as an undo command. Alternatively, a confirmation step could be built into the interface before a link was followed to add one level of protection against this kind of problem.

The EyeKeys system could be improved with an algorithm to more precisely locate the eyes. The current method relies on eye motion for position refinement. 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. Another extension would be an analysis of the difference projection by fitting a polynomial function instead of thresholding. The current system assumes that the head is held vertically and faces toward the camera. When the user's head tilts, the eyes are no longer symmetrical across a vertical axis, which causes problems in detecting the gaze. Extending the system to find the amount of head tilt would improve the detection rate. This could be done by rotating the template, or by finding the rotated line of symmetry of the face or between the eyes.

Future possibilities for extending this system include the addition of a blink analysis module [10], which would give the interface three events to work with. Unfortunately, some 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 and low-resolution eye images used here.

## Acknowledgments

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

## References

1. E H Adelson et al. Pyramid methods in image processing. *RCA Engineer*, pp 33–41, 1984.
2. Applied Science Laboratories. `http://www.a-s-l.com`
3. M Betke et al. The Camera Mouse: Visual tracking of body features to provide computer access for people with severe disabilities. *IEEE Trans Neural Systems and Rehabilitation Engineering*, pp 1–10, 2002.
4. M Betke and N C Makris. Recognition, resolution, and complexity of objects subject to affine transformation. *Int J Computer Vision*, pp 5–40, 2001.
5. M Betke et al. Active detection of eye scleras in real time. *Proc IEEE Workshop on Human Modeling, Analysis, and Synthesis*, 2000.
6. P DiMattia et al. *An Eye Control Teaching Device for Students without Language Expressive Capacity – EagleEyes*. The Edwin Mellen Press, 2001.
7. L A Frey et al. Eye-gaze word processing. *IEEE Trans Systems, Man, and Cybernetics*, pp 944–950, 1990.
8. A Gee and R Cipolla. Determining the gaze of faces in images. *Image and Vision Computing*, pp 639–647, 1994.
9. J Gips and J Gips. A computer program based on Rick Hoyt's spelling method for people with profound special needs. *Proc Int Conf on Computers Helping People with Special Needs*, 2000.
10. K Grauman et al. Communication via eye blinks and eyebrow raises: Video-based human-computer interfaces. *Universal Access in the Information Society*, pp 359–373, 2003.
11. E Hjelmas and B K Low. Face detection: A survey. *Computer Vision and Image Understanding*, pp 236–274, 2001.
12. T Hutchinson et al. Human-computer interaction using eye-gaze input. *IEEE Trans Systems, Man, and Cybernetics*, pp 1527–1533, 1989.
13. Q Ji and Z Zhu. Eye and gaze tracking for interactive graphic display. *Proc Int Symp on Smart Graphics*, 2002.
14. A Kapoor and R W Picard. Real-time, fully automatic upper facial feature tracking. *Proc IEEE Int Conf on Automatic Face Gesture Recognition*, 2002.
15. K-N Kim and R S Ramakrishna. Vision-based eye-gaze tracking for human computer interface. *Proc IEEE Int Conf on Systems, Man, and Cybernetics*, 1999.
16. J J Magee. A real-time human-computer interface based on gaze detection from a low-grade video camera. MS Thesis, Boston University, 2004.
17. J J Magee et al. EyeKeys: A real-time vision interface based on gaze detection from a low-grade video camera. *Proc IEEE Workshop on Real-Time Vision for Human-Computer Interaction*, 2004.
18. C H Morimoto et al. Pupil detection and tracking using multiple light sources. Technical Report RJ-10177, IBM Almaden Research Center, 1998.
19. K Schwerdt and J L Crowley. Robust face tracking using color. *Proc IEEE Int Conf on Automatic Face and Gesture Recognition*, 2000.
20. R Sharma et al. Toward multimodal human-computer interfaces. *Proceedings of the IEEE*, pp 853–869, 1998.
21. R C Simpson and H H Koester. Adaptive one-switch row-column scanning. *IEEE Trans Rehabilitation Engineering*, pp 464–473, 1999.
22. S Sirohey et al. A method of detecting and tracking irises and eyelids in video. *Pattern Recognition*, pp 1389–1401, 2002.

23. O Takami et al. Computer interface to use head and eyeball movement for handicapped people. *Proc IEEE Int Conf on Systems, Man, and Cybernetics*, 1995.
24. J-C Terrillon and S Akamatsu. Comparative performance of different chrominance spaces for color segmentation and detection of human faces in complex scene images. *Proc IEEE Int Conf on Automatic Face and Gesture Recognition*, 2000.
25. Y Tian et al. Dual-state parametric eye tracking. *Proc IEEE Int Conf on Automatic Face and Gesture Recognition*, 2000.
26. M Turk and G Robertson. Perceptual user interfaces. *Comm ACM*, pp 32–34, 2000.
27. M Yang et al. Detecting faces in images: A survey. *IEEE Trans PAMI*, pp 34–58, 2002.
28. D H Yoo et al. Non-contact eye gaze tracking system by mapping of corneal reflections. *Proc IEEE Int Conf on Automatic Face and Gesture Recognition*, 2002.
29. L Young and D Sheena. Survey of eye movement recording methods. *Behavior Research Methods and Instrumentation*, pp 397–429, 1975.