A REAL-TIME HUMAN-COMPUTER INTERFACE BASED ON GAZE DETECTION FROM A LOW-GRADE VIDEO CAMERA

by

JOHN J. MAGEE IV
B.A., Boston College, 2001

Submitted in partial fulfillment of the requirements for the degree of Master of Arts 2004
Acknowledgments

This thesis is based on joint work with Margrit Betke at Boston University. Her efforts in both the development of this document and the methods described within are greatly appreciated. Additionally, thanks to Stan Sclaroff and George Kollios for providing crucial feedback on this thesis. Thanks to Matthew Scott and Benjamin Waber for developing and testing the BlockEscape game. Thank you to James Gips for providing motivating experience with EagleEyes, as well as providing some of the pictures used in this thesis. Thanks to Rick Hoyt for his time and experience in testing the system. Lastly, the support of the Image and Video Computing Group at Boston University, especially Jingbin Wang and William Mullally, is appreciated in helping make this thesis possible. Funding was provided by The Whitaker Foundation, the National Science Foundation (IIS-0308213, IIS-039009, IIS-0093367, P200A01031, and EIA-0202067), and the Office of Naval Research (N000140110444).
A REAL-TIME HUMAN-COMPUTER INTERFACE BASED ON GAZE DETECTION FROM A LOW-GRADE VIDEO CAMERA

JOHN J. MAGEE IV

Abstract

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 control applications such as spelling programs or games. The game “BlockEscape” was developed to gather quantitative results to evaluate EyeKeys with test subjects. The system is compared to a mouse substitution interface.
3.1.1 Skin Color Analysis .................................................................................. 22
3.1.2 Motion Detection .................................................................................... 26
3.1.3 Detection and Tracking via Correlation .................................................. 27
3.2 Eye Analysis ............................................................................................... 30
  3.2.1 Motion Analysis and Stabilization .......................................................... 32
  3.2.2 Left–Right Eye Comparisons ................................................................. 32
3.3 Interface Output .......................................................................................... 36
3.4 BlockEscape Game ..................................................................................... 37
  3.4.1 Methods for Gathering Statistics ........................................................... 38

4 Experiments and Results ................................................................................. 41
  4.1 EyeKeys Classification Experiment ............................................................. 41
  4.2 BlockEscape Experiment ......................................................................... 43
  4.3 Initial Experience: A User with Severe Disabilities .................................. 44
  4.4 Real-Time Performance of System ............................................................ 46

5 Discussions and Future Work ....................................................................... 47
  5.1 Real-Time Performance and Algorithm Complexity ............................... 47
  5.2 Design Motivations ................................................................................... 49
  5.3 Testing Experience and Comparisons ....................................................... 50
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>Applications of the Gaze Detection System</td>
<td>51</td>
</tr>
<tr>
<td>5.4.1</td>
<td>On-Screen Keyboard</td>
<td>51</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Web Browser Navigation</td>
<td>52</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Perceptual User Interfaces</td>
<td>53</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Linguistic Communications Research</td>
<td>53</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Vehicle Driver Monitoring</td>
<td>54</td>
</tr>
<tr>
<td>5.4.6</td>
<td>Store Kiosk Attention Monitoring</td>
<td>55</td>
</tr>
<tr>
<td>5.5</td>
<td>Future Work and Improvements</td>
<td>55</td>
</tr>
<tr>
<td>5.6</td>
<td>Conclusions</td>
<td>57</td>
</tr>
<tr>
<td>6</td>
<td>Appendix</td>
<td>58</td>
</tr>
<tr>
<td>6.1</td>
<td>Camera Noise Measurement</td>
<td>58</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Image pyramid resolutions ........................................... 23

4.1 Results of classification experiment .............................. 42

4.2 Results of BlockEscape experiment ............................... 44

5.1 Time complexity of face tracking algorithm .................... 48

5.2 Time complexity of eye analysis algorithm ..................... 48

5.3 Interface mappings for some applications ....................... 51

6.1 Camera noise measurement: low contrast, low resolution .... 59

6.2 Camera noise measurement: low contrast, high resolution ... 59

6.3 Camera noise measurement: high contrast, low resolution ... 60

6.4 Camera noise measurement: high contrast, high resolution ... 60
# List of Figures

1.1 EagleEyes system in use. ................................................................. 2
1.2 Camera Mouse system in use. ......................................................... 2
1.3 Configuration diagram of the proposed EyeKeys system ....................... 4
1.4 On-screen text entry applications .................................................. 5

3.1 System flowchart ........................................................................... 19
3.2 Image pyramids used by the face tracking algorithm ............................. 20
3.3 Skin color analysis ......................................................................... 24
3.4 Skin color histogram plot ................................................................. 25
3.5 Color and template training interfaces ............................................. 25
3.6 Motion analysis ................................................................................ 27
3.7 Face templates ................................................................................ 30
3.8 Eye motion analysis ......................................................................... 32
3.9 Extracted eye region images ............................................................ 34
3.10 Left and right eye image difference ............................................... 34
3.11 Eye image difference projections ................................................... 34
3.12 Screenshot of the BlockEscape game .............................................. 38

4.1 Rick Hoyt testing the EyeKeys system ............................................. 45
Chapter 1

Introduction

The ability to control eye muscles is occasionally the only voluntary movement existing for people with paralysis. Communication abilities are severely limited, often to yes and no responses using eye movements or blinks. Future computer assistive technologies may someday stop an active mind from being trapped in an inactive body. These people can be helped with the creation of an interface that allows them use of a computer. Further, such an interface may prove to be a communication-enabling technology. As progress toward that goal, a video-based interface called EyeKeys is presented which makes communication possible by detecting the eye gaze direction and simulating computer keyboard input.

Two specific human-computer interface systems are important prior work for this thesis: the EagleEyes project [9] and the Camera Mouse [5] are both interfaces for severely disabled users that enable communication. EagleEyes detects eye movements through the attachment of electrodes to the face. The Camera Mouse employs feature tracking, and is able to track a nose, chin, thumb, foot, or any distinguishable body feature. A new version of the Camera Mouse can track the face movement without initialization to control the mouse. Both of these system are used to control a mouse on the computer screen. Special programs are used to create art, play games, or enter messages as in Figure 1.4. Firsthand observation of the use of these system resulted in a profound understanding: lack of communication ability does not equal lack of intelligence.
Figure 1.1: EagleEyes system in use. Photograph credit: Boston College.

Figure 1.2: Camera Mouse system in use. Photograph credit: Boston College.
The major contribution of this thesis is the method of detecting the eye gaze direction. In order to analyze the eyes, they must first be located in the video images. Existing computer vision techniques are combined in a new way to quickly track the face using multi-scale template correlation with the aid of skin color and motion analysis. The left and right eyes are compared to determine if the user is looking center, or to the left or right side. The new approach exploits the symmetry of the face when comparing the eyes. The right eye image is subtracted from a mirrored left eye image. This difference image is then projected to the $x$-axis. The determination of the eye direction is made by analyzing this projection. The output of the system can be used to control applications such as spelling programs or games. This system is one of the few approaches to using computer vision as an assistive technology so far.

Eye gaze estimation is a difficult problem for many reasons. First, the eyes must be located in a video image by some method. Although face and head tracking has been extensively researched, it cannot be said that it is a solved problem in computer vision. Various existing approaches are discussed in the next chapter. Once the eyes are located, various methods have been used to determine the gaze direction. Unfortunately, many of these approaches require high resolution images of the eyes in order to find features such as the eye corners or the edges of the iris. The popular approach of using active infrared illumination, discussed in the next chapter, was rejected here in order to avoid special hardware.

In the traditional human-computer interface configuration, the computer user sits in
front of a keyboard, mouse, and monitor. This system is designed to mimic that setup, with a camera substituting for the input devices. Whereas hands control the keyboard and mouse, the movement of the eyes effectively controls the computer in this system through the use of the camera directed at the user’s face while they look at the monitor. The system could also be configured to use a large projected display instead of a monitor. In the case of a severely disabled user, a dual display is useful in that an assistant can monitor the computer vision interface on one display while the user’s application is displayed on the other screen. A triple monitor configuration could also be used so that the user moves their eyes between the three displays. Other possible uses and configurations of the system are discussed in Chapter 5. The proposed system configuration diagram is shown in Figure 1.3.

![Figure 1.3](image_url)

Figure 1.3: Configuration layout of the proposed EyeKeys system. The user faces both the camera and the computer monitor. The computer uses the camera to detect if the user’s eyes move to the right or to the left. The dashed lines indicate possible view directions of the person’s eyes.
Figure 1.4: Spelling interface programs developed at Boston College for use with the EagleEyes and Camera Mouse interface systems. (a) The spelling method, developed by Rick Hoyt, groups letters of the alphabet into groups A, E, I, O, and U. The program was developed by Jonathan Gips. (b) The on-screen keyboard can be changed into different layouts to facilitate text entry for different users.

Some assumptions needed to be made in order to facilitate the design of the system. The face detection and tracking algorithm assumes the face is in an upright position. This assumption allows for fast operation since the face template matching algorithm does not need to search for multiple orientations. In addition, a vertical head creates a vertical line of symmetry across the face. This symmetry is exploited by the eye analysis algorithm. The user is assumed to be facing towards the camera for the same reasons, as other methods would be required to detect the location of a turned head in the images. To make detection and tracking of the intended interface user easier, there should be exactly one face in the field of view of the camera.

The user’s eyes are assumed to be symmetrical across the center of the face. This allows the eye direction to be detected by analyzing the difference between the left and
right eye images. The user should be able to hold their head in position while moving their eyes quickly to the left or right side. This is necessary as the system uses the apparent motion of the eyes in the images to determine when the user looks left or right.

These assumptions about the user’s location, appearance, and abilities should be reasonable for people without any disabilities. However, the aim of this system is to provide an interface for severely disabled people. An individual may have disabilities that violate any of these assumptions, possibly limiting their use of the current version of this system. Future modifications of this system may be able to overcome some of these limitations.

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, e.g. [29]. In addition, the system cannot use all of the processing power of the computer as the same computer will have to run both the vision-based interface as well as user programs, such as web browsers or games.

Evaluation of the EyeKeys system performance is first presented with a classification experiment. Test subjects were asked to use the system, and to look left or right when instructed by an observer. The observer then recorded if the system classified the eye motion correctly, incorrectly, or if it was missed.

EyeKeys was also tested as an interface for the BlockEscape game. The goal of the game is to navigate the block left and right through holes in moving walls. When it is used with EyeKeys, the user can look left to make the block move left, and look right to make the block move right. It can also be used with a keyboard, mouse, or the Camera Mouse.
This game was developed specifically as an engaging way to test the interface system while reporting quantitative results, which is important because it encourages test subjects to use the system. Since the game was designed as a testing tool, it can be used to gather statistics on how well the interface works for various situations and users. A screenshot of the game is shown in Figure 3.12.

This thesis is organized in the following fashion. Chapter 2 discusses related work. Chapter 3 discusses the methods employed in the EyeKeys system itself, including a thorough analysis of all of EyeKeys’ modules. Also in this chapter is a description of the BlockEscape game. Chapter 4 details the experiments and results, while Chapter 5 presents an in-depth discussion of the results, comparisons to other HCI systems, and plans for future extensions to the system.
Chapter 2

Related Work

The methods used in the EyeKeys system draw upon a wide variety of computer vision, image processing, and computer science related work. Some of the standard vision methods can be found in a computer vision textbook, e.g. [20]. This chapter highlights some of that work in the areas of computer assistive technologies, face detection and tracking, and gaze estimation.

2.1 Human-Computer Interfaces for the Disabled

There has been much previous work in computer assistive technologies. Many early systems require contact with the user [43]. Most of these methods, though successful and useful, also have drawbacks.

A switch placed in the right position allows a paralyzed person with some motor control to operate programs that take binary input. For example, Figure 4.1 shows a button mounted near the user’s head, which is activated by leaning the head into it. An example of a useful program is a text entry interface that automatically moves between letters or groups of letters. When the intended group or letter is highlighted, the switch is pressed to select it. Using this method, entire sentences can be slowly constructed.

Currently available systems are often intrusive, i.e. the interface must be attached to
the user. For example, the EagleEyes system [9] 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. This approach, called electrooculography, has also been explored in other systems, e.g. [4]. One problem is that the sensors must touch the users face, which may be the only place the person has feeling, thus making the sensors intrusive.

Another problem is with using head mounted cameras to look at eye movements, e.g. [2]. This approach takes advantage of the fact that the face will always be in the same location in the video image if the head moves around. Therefore, the requirement of a complicated face tracking algorithm is removed. 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, one of the goals was to design a non-intrusive system that does not need attachments.

One successful computer vision interface system for people with severe disabilities is the Camera Mouse [5]. Disabled people control a mouse pointer by moving their head, finger, or other limbs, while the system uses video to track the motion. A similar system is used by Aribnia [3]. This approach 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 the system is therefore to use only information from the eyes.
2.2 Face Detection and Tracking

The system first tracks the face in order to locate the eyes. Face detection and tracking has been researched with a variety of methods [16, 41]. The approach often is motivated by the specific application of the face detector or tracker. Such methods can use features, textures, color, or templates for example, which are tracked in either two or three dimensions.

Previous work at Boston University have produced good results, as presented by LaCascia et al. [24]. Here, the head is tracked in three dimensions by registering texture-mapped 3D models. This allows the system to accurately track head tilts and rotations. In addition, this work focuses on the issue of varying illumination, which can create problems for many other existing face trackers. One advantage of this work is that it produces a “rectified” face image as output: even if the head is tilted or rotated, the facial features in the output image should be in the same location.

A 3D head position estimate from tracking facial features is used by Stiefelhagen et al. [36]. Eyes are located with an iterative thresholding algorithm combined with geometric constraints. Mouth corners are located by the horizontal projection in a search area, followed by a horizontal edge finder.

Skin color analysis is often used as part of a face detection system, e.g. [17, 32, 40]. Various methods and color spaces can be used to segment pixels that are likely skin from pixels that may belong to the background. This is a difficult problem because skin tones are widely varied over the population. In addition, background colors may be similar to skin
colors by some measure. Having a red carpet or a wooden door in the image can cause such systems to fail. An assumption that the background is a constant non-skin color would not work well for an application that is intended to be used in real world situations.

A mean face has been used previously in face detection. Liu [25] created a mean face with images from the FERET database. The face was not used as a template for matching in that case. The statistical method of Rikert et al. [30] uses multiple head orientations in training to detect faces that are both frontal and turned.

The system presented in this thesis uses color and motion information as a preprocessing mask. A similar approach has been used previously to segment the background and foreground, e.g. [11].

The Carnegie Mellon face detection system [31] is important prior work. This is just one example of the use of Neural Networks in face detection.

### 2.3 Gaze Estimation

Many systems that analyze eye information use specialized hardware. The use of active infrared illumination is one example [13, 18, 21, 22, 26, 42]. In most cases, the infrared light reflects off the back of the eye to create a distinct “bright pupil” effect in the image. The light is synchronized with the camera to cycle on and off for every other frame. The eyes are located by differencing a frame with the light on and a frame with the light off, creating the bright pupil image. The 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 and infrared light sources are important goals of more recent efforts in computer vision, e.g. [5, 14]. The system proposed here was therefore designed around an inexpensive visible light camera. The system can be run without the need for an expensive image capture board or pan/tilt/zoom camera, which has been used previously to obtain high resolution eye images, e.g. [7]. The system must be able to work with images that have a lower resolution than the images used in previous approaches, e.g. [6, 19, 23, 35, 38, 45].

One method to locate the eyes is to use a series of intentional blinks [14, 32, 37]. Some users may not be able to correctly initialize such systems, for instance, if they cannot understand the procedure, or lack the ability to perform it correctly. This system does not require an initialization or calibration, and can therefore be more easily used by all users.

2.3.1 Betke and Kawai

Betke and Kawai [6] use self organizing grey-scale units to find the pupil position relative to the center of the eye. The inputs are single images of an eye. The output shows the location of the pupil in the eye. This technique requires training on each person, or a similar eye. This approach is not suitable for human-computer interaction because it is not currently real time.
2.3.2 Essa et al.

A facial features tracking system is presented by Essa et al. [10] for expression recognition and analysis. Features are tracked with small 2D templates and normalized correlation. The small feature points are constrained with physically-based model parameters and interpolated. The system uses many small points over the whole face, and a speed of 5 to 10 frames per second is achieved.

This technique may be applicable to eye feature tracking if there can be enough resolution in the input images for features to track the eye parameters.

2.3.3 Herpers et al.

In the work of Herpers et al. [15], several key points on the face to track are identified. These include: corners of the mouth and eyes, as well as points on the eyebrows, nose, and ears.

A model of a the eye with important features is described, including eye corners, iris, and eyelids. Their feature tracker begins by looking for the sharp vertical contrast between the eye whites and the darker iris. The iris is segmented by finding the corresponding right edge of the iris. The inner eye corners are detected by looking at strong edges that are strongly curved. Results are reported of the detection of the eye corners and the iris location. This method looks somewhat promising at high resolutions. At lower resolutions, however, there may be problems finding sharp edges to track the pupil.
2.3.4 Hutchinson et al.

The system by Hutchinson et al. [18] is an early human computer interface system using computer vision to detect eye gaze. The system uses active infrared illumination to work, and must be calibrated each time it is used. The computer screen is divided into a three by three grid for inputs of different type using eye gaze.

This system is not well suited for the proposed application because of the following major problems: active infrared illumination, a calibration routine is required for each use, and there are potential problems with large head movements.

2.3.5 Kapoor and Picard

Another system that uses active IR to find and track the pupils is proposed by Kapoor and Picard [22]. In this case, a camera under a monitor is synchronized with IR lights to produce a “bright eye” and a “dark eye” image. The image difference clearly shows the location of the pupils. From these locations two regions of interest (ROI) are cropped out for further eye model analysis.

Principal Component Analysis (PCA) is used to recover the model parameters, which consists of points outlining the eyes and eyebrows. The system is trained on a number of hand marked examples. From there, the input image is expressed as a linear combination of the examples. The same linear combination is applied to the control points to recover the model control points. In comparison to other systems, because this approach is non-iterative, it is ideal for real-time systems.
The infrared illumination portion of this approach seems to be only used for the detection of an appropriate region of interest. From there, the model is recovered with PCA based on the examples. The results look good, but the eyes are not tracked through various states of closure. The system is able to achieve 30fps even with a large number of examples.

2.3.6 Radeva and Marti

Model-based snakes for feature segmentation is presented by Radeva and Marti [28]. Since the snakes need to be started near the features, the system employs constrained and restricted template matching to find the locations of the features. The nose crest is located first, as it is usually not occluded by beards or glasses. The “bright crest” is easy to find. “Pits” in the horizontal and vertical image projections are used to hypothesize about the locations of the eyes and eyebrows. Simple templates looking for iris and eyelash edges are used to localize the eyes.

Snakes used for eye segmentation are first started with the irises due to their circular nature. Then the other eye features are analyzed. The rubber snake is attracted to the valley of the eye contour, due to the darkness of the eyelashes (this is as opposed to attracting them to edges). In addition, the detected pupil adds a term to the energy function to ensure that the snake deforms correctly. When the lower eyelash fails to give a large valley, edge potentials are used instead. The deformations are compared across the symmetry of the nose crest to check the results. The upper and lower snakes are divided into sections to
account for differences in gradients between where the eyelash is over eye whites or over
the darker iris region.

Since the required resolution of the eye area is unknown, low resolution images may be
difficult for this process. In addition, an estimate of the computational time for this method
is unknown. Further, once the snakes have found the correct deformations, additional
processing would be required to determine possible gaze directions.

2.3.7 Sirohey et al.

In the work of Sirohey et al. [35], the iris and eyelid are detected and tracked. To detect the
iris, the system searches for the lower half of the iris boundary. This is done by searching
for the curved edge. The parameters are constrained so that the system only searches in
valid parameter spaces, i.e. the diameter of iris relative to eye size is reasonable.

Detection of blinks can be supplied to feature detection algorithms to improve their
reliability. As the lower eyelid moves very little, it is sufficient to only use the motion of
the upper eyelid. Information about the iris center and eye corners are used in estimating
eyelid position. The shape of edges of the eyelid vary depending on the degree of opening.
By looking upwards from the iris center for a horizontal edge, the eyelid should be found.

Three facts are exploited: the eyelid edge is above the center of the iris when the eye
is open, the eyelid edge is an arc joining the two corners of the eye, and the more open the
eye, the greater the curvature of the arc.

A third-degree polynomial is used as it was found that a second-degree polynomial
sometimes did not follow the eyelid curvature correctly. All edge segments that are less than four pixels long are eliminated. Then the remaining edge pixels are used to fit the polynomial with a RMS error measure. Different segments are combined if the segments have similar curvatures and do not overlap. Up to triples were considered. This leaves three lists of possible eyelid candidates: single segments, double segments, and triple segments. The best candidate is selected by the edge segment above the iris that had the most number of pixels. An additional requirement is that segments curve downwards and must not have large slopes.

Tracking during a blink is done by linear interpolation between positions before and after a blink. Blink detection is fed back into the system to help make the results better.

2.3.8 Tian et al.

Tian et al [38] propose a dual state parametric eye tracker. While most related work focuses on open eyes, in this approach eyes are detected as either open or closed.

The system is initialized in the first frame. From there, inner corners of the eyes are tracked using a modified Lucas-Kanade algorithm. Outer corners are searched outwards along the line connecting the inner two corners. This gives some of the eye parameters.

Eyes are open if the iris can be detected. This is done by looking searching for the circle of a certain radius threshold with the most edge pixels. If the iris can not be detected, the eye state is closed and the model only consists of the corner points.

If the iris is detected, the upper and lower eyelid are searched along a path perpendic-
ular to the line connecting the eye corners. The tracking of the iris produces good results when high resolution images are used. The iris detection may be problematic at lower resolutions.

2.3.9 Yuille et al.

The system proposed by Yuille et al. [44] uses a deformable template method for recovering eye parameters. The open eye template from Tian et al. [38] was developed from an earlier version of this work.

The deformable template method has a problem for use in human-computer interfaces in that it is too slow for video processing. In 1992, the method took minutes for one eye. In addition, the template must be started somewhere near and below the eye, otherwise it tends to attract itself and deform to the eyebrow contours.
Chapter 3

Methods

The system is broken down into 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 3.1.

![Figure 3.1: Major functions of the face tracking and eye analysis system.](image)

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 [14, 5, 9].

3.1 Face Detection and Tracking

Figure 3.2: Pyramids used by the face detection and tracking algorithm. From left to right: greyscale input, color, motion, correlation, masked correlation. The face is detected in the third smallest resolution image.

The system uses color, motion, and correlation-based template matching to detect and track faces. It can detect and track different size faces at various distances to the camera. This allows users to move closer or farther from the camera and still be detected automatically. To achieve this, the system uses image pyramids [1], as shown in Figure 3.2. Each pyramid consists of eight levels. The highest resolution image in the pyramid is the $640 \times 480$ video frame and the lowest resolution image contains $32 \times 24$ pixels. All
resolutions are shown in Table 3.1. In each level of the pyramid, the system searches for a face of size $12 \times 16$. This approach allows the system to operate in real time. The level of the pyramid at which the face is detected can then be used to infer the “true” size and location of the face, which is the size and location of the face in the original video frame. In particular, when the person is far from the camera and appears relatively small in the video frame, the system can efficiently detect the face in a high-resolution level of the pyramid. In the case where the person is close to the camera and therefore appears relatively large in the original video frame, the face is detected efficiently in a lower resolution level of the pyramid. The following algorithm describes the face tracking algorithm, where the $P$ variables refer to an entire pyramid, $C_t$ is the result of the color histogram lookup, and $M_t$ is the current motion image.
input : Color video 640 × 480
output : Location and scale of face: (x, y, scale).

for Each video frame $I_t$ do
    /* Color image analysis */
    begin
        $C_t$ ← histogram-lookup($I_t$)
        $P_{Color}$ ← pyramid-decimate($C_t$)
        $P_{Color}$ ← average-filter($P_{Color}$, 12 × 16)
        $P_{Color}$ ← threshold($P_{Color}$, 10)
    end

    /* Motion analysis */
    begin
        $M_t$ ← $|I_t - I_{t-1}|$
        $P_{Motion}$ ← pyramid-decimate($M_t$)
        $P_{Motion}$ ← average-filter($P_{Motion}$, 12 × 16)
        $P_{Motion}$ ← threshold($P_{Motion}$, 10)
        $P_{Motion}$ ← add-prior-location($P_{Motion}$, x, y, scale)
    end

    /* Template matching */
    begin
        $P_{Input}$ ← pyramid-decimate($I_t$)
        $P_{Masked}$ ← $P_{Input} \& P_{Color} \& P_{Motion}$
        $P_{Correlation}$ ← normalized-correlation-coefficient($P_{Masked}$, face template)
    end

    /* Find location and scale of the face */
    (x, y, scale) ← arg-max-value($P_{Correlation}$)
end

Algorithm 1: Face tracking algorithm

3.1.1 Skin Color Analysis

Skin color has been used previously to track faces, e.g., [17, 32, 40]. It is a difficult problem because there is a wide variety of skin tones, and many background colors may be misinterpreted as skin. In this system, it is used as a preprocessing mask, as in [11]. The color input image is converted into the YUV color space. The YUV color space was
Table 3.1: Resolutions used by the image pyramids. Coordinates in any index scale can be transformed into coordinates in the $640 \times 480$ input frame by multiplying by the scale factor.

<table>
<thead>
<tr>
<th>Index</th>
<th>Width</th>
<th>Height</th>
<th>Scale Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>640</td>
<td>480</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>320</td>
<td>240</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>160</td>
<td>120</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>96</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td>60</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>48</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>30</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>32</td>
<td>24</td>
<td>20</td>
</tr>
</tbody>
</table>

chosen because the camera can be configured to provide images natively in that format. A new image is created with a 2D histogram lookup in UV space. The histogram is shown in Figure 3.4. If a color pixel’s lookup on the histogram for the specified UV value is greater than zero, then the pixel is marked as skin with the brightness value from the lookup, otherwise it is marked as not skin, with a brightness value of zero. An example result from the histogram lookup is shown in Figure 3.3(a). The image is blurred using Gaussian smoothing, with a $5 \times 5$ approximation to a Gaussian filter [20], then decimated into the other pyramid levels by removing rows and columns. We are only interested in groups of pixels that are marked as skin and have the same shape and scale of the face that we are trying to detect. Therefore small areas that are incorrectly detected as skin may be removed as follows. A low pass filter is applied along each level of the pyramid. This filter averages the pixels over a neighborhood of $12 \times 16$ in each pyramid level. This is the size of the face that we aim to detect. The size of the filter devaluates groups of skin-detected pixels that are smaller than the scale of the face at the appropriate pyramid level, while maintaining
groups of pixels at the correct scale. An intermediate result is presented in Figure 3.3(b). A second threshold is used to mark the groups of pixels at the target scale and remove the other pixels. In the implementation, a value of 10 is used for this threshold. The filter and threshold also serve to fill in holes that were not detected as skin but could be part of the face, as seen in Figure 3.3(c). The pyramid of the resulting binary images are shown Figure 3.2(b).

(a) Result of histogram lookup.  (b) Result after low-pass filter.  (c) Result after threshold.

Figure 3.3: Color Analysis: Intermediate images from the 160 × 120 resolution pyramid level. Notice that the small group of pixels to the left of the face in (a) are removed in the result (c), while the eyes are filled in. Note: (a) and (c) were computed from the same input frame.

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 during operation simply by clicking on areas of skin in the live video, as in Figure 3.5(a). The histogram can be saved and loaded so that it can be used again for the same user or lighting conditions without retraining.
Figure 3.4: 2D Histogram lookup table that was trained on skin color.

Figure 3.5: Training Interfaces. These windows can be opened during the operation of the system to train the color histogram or replace the template used by the correlation algorithm. (a) The user repeatedly clicks on the face skin areas to train the histogram. (b) The user first clicks on the nose to select the location of the face, then moves the slider to select the appropriate scale.
3.1.2 Motion Detection

As in the skin color analysis, apparent motion in the image is used as a preprocessing mask. A simple assumption is behind the motion analysis: if the person moves, then frame differencing should find the pixels where motion occurs in the image. The frame differencing creates a motion image. The frame differencing can be defined as:

\[ M(x, y) = \text{abs}(I_t(x, y) - I_{t-1}(x, y)), \]  

(3.1)

where the pixels of the motion image \( M(x, y) \) are the absolute difference between the pixels of the current image frame \( I_t \) and the previous image frame \( I_{t-1} \). Since the current frame rate is 15 frames per second, those frames will represent images taken approximately 67 milliseconds apart. A higher frame rate may require the algorithm to be modified to maintain a similar temporal separation between frames. An example motion image is shown in Figure 3.6(a).

The motion image is then decimated into a pyramid. As in the color analysis, we are interested in finding a face of size 12 x 16. A low pass averaging filter of support 12 x 16 and subsequent thresholding are therefore applied to remove motion that cannot be due to the motion of the face. Figures 3.6(b) and 3.6(c) demonstrate these steps. Figure 3.2(c) shows a motion image pyramid with the low pass filter disabled to better highlight pixels that contribute to the motion image. A threshold value of 10 is used in the implementation.

In the case where there is little or no motion, we assume that the face should be
Figure 3.6: Motion Analysis: Intermediate images from the $160 \times 120$ resolution pyramid level. Note: (a) and (c) were computed from the same input frame.

found near the same location and scale as found in a previous frame. When the system initializes and there is no previously detected face location, any location is set as the prior location. In subsequent frames, 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 are 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.

3.1.3 Detection and Tracking via Correlation

Normalized correlation template matching is used to find the exact location of the face. A $12 \times 16$ face template was created by averaging the brightness values of 8 face images, and modified with a paint program. The greyscale input pyramid (Y channel from the YUV color image) is masked by the color and motion information pyramids. The face template
is then correlated over all levels of this masked pyramid.

The template matching is computed as follows for a single input image. Let the template dimensions be \( w \times h \) pixels, and the input image dimensions be \( W \times H \) pixels. The resulting image then has dimensions \((W - w + 1) \times (H - h + 1)\) pixels. Each location \((x, y)\) in the result image represents the similarity between the template and the image region \((x, y)\) to \((x + w - 1, y + h - 1)\). The normalized correlation coefficient is computed:

\[
\tilde{R}(x, y) = \frac{\sum_{y'=0}^{h-1} \sum_{x'=0}^{w-1} (T(x', y') - \overline{T}) \cdot (I(x + x', y + y') - \overline{I}(x, y))}{\sqrt{\sum_{y'=0}^{h-1} \sum_{x'=0}^{w-1} (T(x', y') - \overline{T})^2 \sum_{y'=0}^{h-1} \sum_{x'=0}^{w-1} (I(x + x', y + y') - \overline{I}(x, y))^2}},
\]

where \(T(x, y)\) and \(I(x, y)\) are the pixel values at location \((x, y)\) in the template and input image respectively, \(\overline{T}\) is the mean pixel value of the template, and \(\overline{I}(x, y)\) is the mean pixel value of the input image region \((x, y)\) to \((x + w - 1, y + h - 1)\). Probable locations of the template in the image can be found in a local or global maximum of the resulting image values [8]. The resulting image from the normalized correlation coefficient is shifted by \((\frac{w}{2}, \frac{h}{2})\) so that the value at \((x, y)\) corresponds to the similarity of the center of the template.

The correlation described above is computed for the same face template over all images in the input pyramid. The resulting unmasked and masked pyramids are shown in Figures 3.2(d) and 3.2(e). The maximum correlation peak among all of the levels indicates the location of the face. The scale of the face is inferred by the level of the pyramid at
which the maximum is found.

Masking the input image with information from color and motion analysis serves two purposes. By removing the background pixels, there is less of a chance that the face template will correlate well with something that is not a face. Secondly, by eliminating those pixels as possible locations of the face, the search space for the correlation function is reduced. Therefore, a smart implementation of the correlation function would be able to skip computations for those locations and increase overall efficiency.

A “mean face” has been used previously in face detection, e.g. [25]. That the face template is created with the average brightness of multiple people, and that it is relatively small so that it is a coarse human face, allows the system to find a face in the image that was not used in the creating the template. Matching this template over the pyramid allows for fast correlation while preserving the relevant information, that it is a face, in any particular level.

Instead of using the average face template provided by the system, the operator has the option of initializing the template with the face of the current user. This can be done by clicking on the nose of the face in the image on the screen, and then selecting the correct scale. Examples of a mean face template and a user specific template are shown in Figure 3.7.
Figure 3.7: Face templates used by the correlation algorithm to find the scale and location of the face. The user specific face template was created when the user was more brightly lit from one side.

### 3.2 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 in the eye *region of interest* are cropped out from the highest resolution image. This region consists of the pixels above the center of the face, and to the right and left of center for the right and left eyes respectively. Since their size depends on the scale at which the face was found, these images are then scaled to $60 \times 80$ pixels using linear interpolation. 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 scaling of the images maintains the size of the eyes in the images when the face is found in different scales.
**input**: Video $640 \times 480$, face location and scale (x, y, scale)

**output**: Gaze classification: left, center (default), right

**for Each video frame** $I_t$ do

/* Compute eye region of interests (ROI) */

$ROI_{left} \leftarrow$ compute-ROI($x$, $y$, scale, left)

$ROI_{right} \leftarrow$ compute-ROI($x$, $y$, scale, right)

/* Crop and rescale eye images */

$I_{left} \leftarrow$ crop-resize-ROI($I_t$, $ROI_{left}$, scale)

$I_{right} \leftarrow$ crop-resize-ROI($I_t$, $ROI_{right}$, scale)

/* Motion stabilization */

begin

$M_{left} \leftarrow |I_{left} - I_{left-1}|$

$M_{left} \leftarrow$ threshold($M_{left}$, 30)

$M_{right} \leftarrow |I_{right} - I_{right-1}|$

$M_{right} \leftarrow$ threshold($M_{right}$, 30)

$Moment_{left} \leftarrow$ first-order-moment($M_{left}$)

$Moment_{right} \leftarrow$ first-order-moment($M_{right}$)

$I_{left} \leftarrow$ recenter-eye($I_{left}$, $Moment_{left}$)

$I_{right} \leftarrow$ recenter-eye($I_{right}$, $Moment_{right}$)

end

/* Left-right eye comparison */

begin

/* $M_{left}$ and $M_{right}$ are recentered motion masks */

$I_{left} \leftarrow I_{left} \& M_{left}$

$I_{right} \leftarrow I_{right} \& M_{right}$

$a \leftarrow$ compute-difference-projection($I_{left}$, $I_{right}$)

$a_{min} \leftarrow$ min($a$)

$a_{max} \leftarrow$ max($a$)

$i_{min} \leftarrow$ argmin($a$)

$i_{max} \leftarrow$ argmax($a$)

if $a_{min} < -T_p$ and $a_{max} > T_p$ then

if $i_{max} - i_{min} > T_d$ then

Output: right

end

if $i_{max} - i_{min} < -T_d$ then

Output: left

end

end

Default Output: center

end

Algorithm 2: Gaze analysis algorithm
3.2.1 Motion Analysis and Stabilization

Ideally, the two eye images would stay perfectly still in the cropped eye images 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. Frame differencing, as described in Equation 3.1, is used create eye motion images as shown in Figure 3.8. A threshold value of 30 is used in the implementation. The first-order moments [20], sometimes considered the “center of mass” for a binary image, are calculated for these images. These 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.

Figure 3.8: Motion detected by frame differencing is thresholded and used as a mask for the left-right image differencing.

3.2.2 Left–Right Eye Comparisons

The following method is the most important contribution of this thesis. The left and right eyes are compared to determine where the user is looking. To accomplish this, 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 appear to be looking right as shown in Figure 3.9(b).

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 3.10. To further reduce extra information from the image areas outside of the eyes, the images are masked by their thresholded motion images, obtained as described in the previous section. To determine the direction, the signed differences are projected onto the \( x \)-axis. The results of these projections can be seen in Figure 3.11. 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 direction. If the positive or negative peaks do not exceed a certain threshold, then the default (looking center) value is the output.

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 \( a = a_1, \ldots, a_m \) is computed by:
Figure 3.9: Eye images automatically extracted from input video by face tracker and aligned by motion analysis.

Figure 3.10: Difference between right and mirrored left eyes. (a) Eyes are looking to the left; arrows highlight large brightness differences. Computed by differencing Figures 3.9(a) and 3.9(b). (b) Eyes are looking straight ahead.

Figure 3.11: Results of projecting the signed difference between right and mirrored left eyes onto the x-axis. Left: the result of left-looking eyes. Right: the result of right-looking eyes.
\[ a_i = \sum_{j=1}^{n} (I_r(i, j) - I_e(m - i, j)). \]  

(3.3)

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 \( a \) and their respective indices are computed:

\[ a_{\text{min}} = \min_i (a_i) \quad \text{and} \quad a_{\text{max}} = \max_i (a_i), \]  

(3.4)

\[ i_{\text{min}} = \arg \min_i (a_i) \quad \text{and} \quad i_{\text{max}} = \arg \max_i (a_i). \]  

(3.5)

Therefore,

\[ a_{\text{min}} = a_{i_{\text{min}}} \quad \text{and} \quad a_{\text{max}} = a_{i_{\text{max}}}. \]  

(3.6)

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

\[ a_{\text{min}} < -T_p \quad \text{and} \quad a_{\text{max}} > T_p. \]  

(3.7)

This threshold assures that there is a sufficient brightness difference to indicate a left or right motion. The suggested range for this threshold is \( 400 \leq T_p \leq 750 \).

The second threshold \( T_d \) is used to guarantee a minimal spatial difference between the minimum and maximum projection values when motion is detected. This helps prevent
motion that is not an intentional eye movement from being detected. The implementation uses a value of $T_d = 8$. The direction of motion is determined as follows:

$$i_{\text{max}} - i_{\text{min}} > T_d \Rightarrow \text{‘right motion’}$$ \hspace{1cm} (3.8)

$$i_{\text{max}} - i_{\text{min}} < -T_d \Rightarrow \text{‘left motion’}.$$ \hspace{1cm} (3.9)

3.3 Interface Output

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. Use of the system found that setting this limit to once every 0.5 seconds works well. 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 monitors would be needed to let the user know when events are triggered.

The detected eye motion classification described in the previous section is used to determine when an event should be sent. Since the system is implemented in Microsoft Windows, the implementation creates a “windows message” that instructs the operating system to simulate a keyboard press. When “right motion” is detected in Equation 3.8, a left arrow key press is simulated. Similarly, the “left motion” detected by Equation 3.9 causes a right arrow key press to be simulated. The simulated keys can be changed depending on the application the user wishes to control. Although the current system does not control a mouse, it could be modified to move the mouse right or left when the corresponding
motions are detected.

3.4 BlockEscape Game

BlockEscape is a game that is easy to learn 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 3.12 shows a screenshot of BlockEscape.

The rules of the game are as follows. The wall levels, which are the black rectangles in Figure 3.12, 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. To lead the block, the user is restricted to move in a plane perpendicular to the orientation of the board.

By default, the board is oriented downward; therefore the user is restricted to move horizontally in two directions, left and right. In our case, the block 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
Figure 3.12: 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.

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 redraw speed and wall levels per screen. The redraw speed specifies the redraw interval of the game, the higher the setting, the slower and therefore more manageable the game play. The wall levels per screen allows the user to configure how close the walls come to each other, the higher the setting, the more difficult the game play. These settings allow the game to be configured appropriately for the abilities of the user with a chosen interface method.

3.4.1 Methods for Gathering Statistics

Incorporated within BlockEscape are detailed usage statistics that are generated during game play and compiled into XML (Extensible Markup Language) documents. 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:

\[ d_{ij} = |x_{ij} - h_i| \quad (3.10) \]

\[ D_{ij} = \begin{cases} 
 0 & \text{if } d_{ij} < d_{i-1,j}, \text{ or } j = 0 \\
 1 & \text{otherwise}
\end{cases} \quad (3.11) \]

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.

The normalized deviation for wall level \( i \) is defined as:

\[ \sigma_i = \frac{1}{sg} \sum_{j=1}^{W_i} D_{ij} \quad (3.12) \]

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 average standard deviation is:

$$\sigma_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} \sigma_i$$

where $\sigma_{\text{avg}}$ is the average player deviation over all wall levels, and $n$ is the number of wall levels.

Several users employing a keyboard to play this game exhibited $\sigma_{\text{avg}} \approx 0$, and thus it is assumed that all movement errors lie with the HCI system. 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 the game can now use the same wall level sequence on multiple users, and thus get results that are comparable.
Chapter 4

Experiments and Results

Two experiments were conducted to evaluate the EyeKeys system. The first experiment provides results on the performance of the classification system. In this test, the user is asked to “look left” or “look right”, and the classification of the system is recorded. The second experiment evaluates the use of EyeKeys as an interface to a possible real-world application. The BlockEscape game uses left and right commands to move a block on the screen. Other games or applications that can be modified to take only two inputs may be able to be controlled in a similar fashion. In addition, the BlockEscape experiment measures the deviation of the user controlled block from an optimal path.

4.1 EyeKeys Classification Experiment

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 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. Test subjects 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.

A random ordered sequence of twenty looks was created: 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.** The 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. The 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 4.1.

**Table 4.1: Results of testing the user interface system on a sequence of left and right looks.**

<table>
<thead>
<tr>
<th>Observed</th>
<th>Actual</th>
<th>% Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td>72</td>
<td>12</td>
</tr>
<tr>
<td>Right</td>
<td>8</td>
<td>68</td>
</tr>
<tr>
<td>Missed</td>
<td>40</td>
<td>48</td>
</tr>
</tbody>
</table>

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 recog-
nition occurred were due to a problem with alignment of the right and mirrored–left eyes. The method of aligning these eyes using the first-order moment of eye difference images described previously is not entirely robust. Possible solutions to this problem are discussed in Chapter 5. 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.

4.2 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 4.2.

Table 4.2: Results of four users employing three devices to play BlockEscape. Units are percentage of game playing area.

<table>
<thead>
<tr>
<th>Device</th>
<th>EyeKeys</th>
<th>Camera Mouse</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma_{avg})</td>
<td>2.9</td>
<td>2.27</td>
<td>0</td>
</tr>
<tr>
<td>Median</td>
<td>2.54</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>4.01</td>
<td>2.68</td>
<td>0</td>
</tr>
<tr>
<td>Wins</td>
<td>(\frac{11}{12}) (83%)</td>
<td>(\frac{11}{12}) (83%)</td>
<td>(\frac{12}{12}) (100%)</td>
</tr>
</tbody>
</table>

**Results.** The most noteworthy portion of the results in Table 4.2 is the high win percentage. Notice that the win percentage of EyeKeys compared to the Camera Mouse was the same, although EyeKeys had a higher \(\sigma_{avg}\), 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. EyeKeys also has the advantage that the user does not need to move their head in order to interact with the computer.

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.

### 4.3 Initial Experience: A User with Severe Disabilities

Rick Hoyt was able to hold a preliminary test of the EyeKeys system. Rick has cerebral palsy, and can control his eyes and head movements. However, he also has some
involuntary head and limb movements. Since he is non-verbal, his primary method of communication is to use small head nods to spell out words and sentences with an assistant. He is also able to use a computer by pressing a switch with his head. Previously, he has helped test and evaluate the Camera Mouse system.

The EyeKeys system was first evaluated while tracking his head and eyes. One problem was immediately apparent: he has difficulty holding his head straight up. Since the face tracker assumes that the head will be straight up, it was not immediately able to track the head correctly. Re-initializing the face template solved this initial problem with tracking the face. However, the eye analysis algorithm assumes symmetry across a vertical face as well. Repositioning his body in the chair allowed him to more easily hold his head vertically. The face tracker was then able to track the face more accurately.

Figure 4.1: Rick Hoyt tests the EyeKeys system. Rick’s usual computer interface is the single button mounted to the right behind his head.

The system was evaluated by asking him to look at the computer monitor, and then to make left and right looks. The system was able to correctly identify most of the times when
he looked to one side without moving his head. Many of the attempts he made were not correctly identified because he also moved his head to the side to try to look at the target.

He was asked to use the EyeKeys system to move a window left and right across the screen. In most cases, it was observed that he was able to move the window in the direction that he was asked. Sometimes, involuntary head motion would cause the system to detect an unintentional eye event.

He had more of a problem with the BlockEscape game when using EyeKeys than when using the Camera Mouse. One issue is that he has used the Camera Mouse to control a computer in the past, and is therefore accustomed to controlling the computer with his head. When the game would begin, he could not help moving his head in an attempt to play, while for the most part keeping his eyes fixed on the computer screen. Head movements have also been his only method of communicating, so it may be difficult for him to try to use only eye movements while keeping his head still.

4.4 Real-Time Performance of System

The system achieves real–time performance at 15 frames per second, which is the limit of the USB camera at $640 \times 480$ resolution. The BlockEscape game had no problem running concurrently with the real-time vision interface system. Observation experience indicates that the performance of the system easily enables it to run concurrently with other applications such as spelling programs and web browsers. The time complexity is analyzed in the next chapter.
Chapter 5

Discussions and Future Work

5.1 Real-Time Performance and Algorithm Complexity

The correlation module of the face tracker is the most computationally expensive function required in the 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 is used to reduce the search space of the template correlation, further improving efficiency.

In the face detection, all of the following processes have a computation time linear in the number of pixels $N$ in the image: histogram lookup, pyramid decimation, thresholding, frame differencing, and image masking. The averaging filter and normalized correlation have a fixed filter size, so they remain linear in the number of pixels. For the eye analysis, the following are also $O(N)$ in the size of the input image: resampling, first-order moments, and difference projection. Minimum and maximum peak values and location search is linear in the width $m$ of the image. The size of the input dimensions is $N = w \times h$ pixels, and the size of the face searched for is $p \times q$ pixels. For the eye analysis, $N = m \times n$ is the dimensions of the cropped eye images. Tables 5.1 and 5.2 show the running time complexity of the face tracker and eye analysis algorithms respectively.
Table 5.1: Time complexity of the face tracking algorithm.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Color analysis</strong></td>
<td></td>
</tr>
<tr>
<td>histogram lookup</td>
<td>( w \times h )</td>
</tr>
<tr>
<td>pyramid decimation</td>
<td>( w \times h )</td>
</tr>
<tr>
<td>average filter</td>
<td>( w \times h \times p \times q )</td>
</tr>
<tr>
<td>thresholding</td>
<td>( w \times h )</td>
</tr>
<tr>
<td><strong>Motion analysis</strong></td>
<td></td>
</tr>
<tr>
<td>frame differencing</td>
<td>( w \times h )</td>
</tr>
<tr>
<td>pyramid decimation</td>
<td>( w \times h )</td>
</tr>
<tr>
<td>average filter</td>
<td>( w \times h \times p \times q )</td>
</tr>
<tr>
<td>thresholding</td>
<td>( w \times h )</td>
</tr>
<tr>
<td>add prior location</td>
<td>constant</td>
</tr>
<tr>
<td><strong>Template matching</strong></td>
<td></td>
</tr>
<tr>
<td>pyramid decimation</td>
<td>( w \times h )</td>
</tr>
<tr>
<td>masking input</td>
<td>( w \times h )</td>
</tr>
<tr>
<td>normalized correlation</td>
<td>( w \times h \times p \times q )</td>
</tr>
<tr>
<td><strong>Location and scale output</strong></td>
<td></td>
</tr>
<tr>
<td>max correlation peak search</td>
<td>( w \times h )</td>
</tr>
</tbody>
</table>

Table 5.2: Time complexity of eye analysis algorithm.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cropping eye images</strong></td>
<td></td>
</tr>
<tr>
<td>compute ROI</td>
<td>constant</td>
</tr>
<tr>
<td>rescale images</td>
<td>( m \times n )</td>
</tr>
<tr>
<td><strong>Motion stabilization</strong></td>
<td></td>
</tr>
<tr>
<td>frame differencing</td>
<td>( m \times n )</td>
</tr>
<tr>
<td>thresholding</td>
<td>( m \times n )</td>
</tr>
<tr>
<td>first-order moments</td>
<td>( m \times n )</td>
</tr>
<tr>
<td>recenter eye</td>
<td>( m \times n )</td>
</tr>
<tr>
<td><strong>Right-left eye comparison</strong></td>
<td></td>
</tr>
<tr>
<td>image masking</td>
<td>( m \times n )</td>
</tr>
<tr>
<td>difference projection</td>
<td>( m \times n )</td>
</tr>
<tr>
<td>min and max peaks</td>
<td>( m )</td>
</tr>
<tr>
<td><strong>Classification output output</strong></td>
<td></td>
</tr>
<tr>
<td>threshold comparison</td>
<td>constant</td>
</tr>
</tbody>
</table>
The eye analysis classification is relatively inexpensive to compute. The eye direction is computed in time linear to the size of the eye image. The face tracker is more complex, where the time for the normalized correlation and the averaging filter depends on the size of the input as well as the size of the filter or template.

5.2 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 classical image pyramid consists of images that are half the resolution of the previous pyramid level. The face can be located at non-discrete distance intervals from the camera. The use of a classical pyramid would require the face to be found at discrete intervals that are far apart, i.e. the face appearance in the image would need to halve in size
in order to be found in a higher pyramid level. The pyramid used in this system consists of intermediate levels, so that the face can be found at more discrete intervals. The template correlation can locate a face as it moves close or far from the cameras at these intervals, whereas it would have difficulty with the levels used in a classical image pyramid.

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.

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.

5.3 Testing Experience and Comparisons

The 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, the EyeKeys system performed well. When the mouse tracker gets lost in the Camera Mouse, the performance decreases dramatically.
With EyeKeys, a false detection can be easily rectified by a correct detection. This, however, is specific to certain applications that are forgiving of incorrect inputs.

### 5.4 Applications of the Gaze Detection System

The EyeKeys system would work in controlling certain applications even given the constraints of only two types of possible input. Some of these applications are discussed here, along with the required modifications that would be required to maximize the usefulness of the interface with the application. Table 5.3 gives possible mappings for some applications.

<table>
<thead>
<tr>
<th>Application</th>
<th>Left Look</th>
<th>Left Function</th>
<th>Right Look</th>
<th>Right Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlockEscape</td>
<td>Left arrow</td>
<td>Move block left</td>
<td>Right arrow</td>
<td>Move block right</td>
</tr>
<tr>
<td>Driving Game</td>
<td>Left arrow</td>
<td>Drive left</td>
<td>Right arrow</td>
<td>Drive right</td>
</tr>
<tr>
<td>Web browser</td>
<td>Tab</td>
<td>Go to next link</td>
<td>Enter</td>
<td>Follow current link</td>
</tr>
<tr>
<td>Rick Hoyt Speller</td>
<td>Click</td>
<td>Select</td>
<td>Click</td>
<td>Select</td>
</tr>
<tr>
<td>Other text entry</td>
<td>Enter</td>
<td>Select letter</td>
<td>Right arrow</td>
<td>Next letter</td>
</tr>
</tbody>
</table>

### 5.4.1 On-Screen Keyboard

An on-screen keyboard application would possibly be the most useful application to a non-verbal paralyzed user that can successfully use EyeKeys. A number of researchers have explored text entry methods for disabled users, e.g. [12, 34]. These interfaces range from single input switches to gaze point detection.

EyeKeys would fall somewhere in between in terms of usefulness and speed. EyeKeys would allow one input, such as looking left, to move the selection to different letters or
groups of letters. Looking right would select the letter and insert it into the text being composed by the user. The current system would have some problems due to occasional misclassification. Improving the accuracy of the system would greatly improve its usefulness for a text input application that is not as forgiving to extraneous inputs. Another solution would be to require a “confirmation” from the user, such as two successive right looks, in order to input a letter.

5.4.2 Web Browser Navigation

Using EyeKeys as an interface to navigate a web browser would allow disabled users to find information on the internet on their own. Since EyeKeys detects only large eye motions, reading text on a monitor should not trigger any events. In this scenario, looking left would map to the tab key to move to the next link, and looking right would map to the enter key to follow a link. A good default starting page would be yahoo.com or myway.com due to their hierarchical menus that index the world wide web. For example, finding a page with the weather in Boston requires following only four links: Weather → United States → Massachusetts → Boston. As in the text entry application, the usefulness of EyeKeys would be improved with increased accuracy. For instance, if the 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.
5.4.3 Perceptual User Interfaces

EyeKeys has the potential to become an integral part of a complete HCI system, e.g. [33, 39]. 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. Some of the assumptions used in the design of EyeKeys do not make this possible at this time. Specifically, head tilts or out of plane rotation cause problems for the current system. Modifications would require generalized tracking of a person’s body pose and head. Sufficient resolution images of the eyes would need to be acquired and used in an improved method for determining more precisely where eyes are looking.

A perceptual user interface might allow your computer or entertainment center at home to determine what you are looking at, or what you are doing. For example, your computer might turn on your television by detecting that you are sitting on the couch and looking at it. Then when you fall asleep watching television, it will automatically be turned off. The techniques used in EyeKeys may be used in future gaze detection systems are part of such perceptual user interfaces.

5.4.4 Linguistic Communications Research

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. The improvements described in the previous
section would be required to manage head tilts and rotations. In addition, a fine resolution gaze estimation would be more useful than the discrete left-center-right classification of the current approach.

The American Sign Language Linguistic Research Project (ASLLRP) at Boston University studies, among other things, facial feature movements during sign language conversations. Video sequences of these conversations are currently analyzed manually to annotate when events such as eyebrow raises occur, using a system called SignStream [27]. The techniques used in EyeKeys could be combined with other facial feature analysis algorithms to aid in the annotation of such video sequences. More data could be collected and analyzed to be made available to the linguistic researchers with the use of these automatic systems.

5.4.5 Vehicle Driver Monitoring

Many of the major car manufacturing companies have current research in automatic monitoring of vehicle drivers. This research focuses both on what happens inside and outside of the car. A computer vision system, called Lane Departure Warning System, is currently available on some trucks. The car companies are also interested in determining if the driver is alert and monitoring where their attention is. Eye blinks or the amount that eyelids are open are being studied to determine if the driver is falling asleep.

One application where EyeKeys could be useful is in an accident avoidance system. Assuming that other computer vision, radar, or infrared techniques can find pedestrians or
other vehicles, the car could automatically alert the driver if it determines that they are looking the other way. The techniques used in EyeKeys could be adapted to be a part of this system. Again, this would require better head tracking to deal with tilts and rotations, and a fine resolution gaze estimation to be of the most use.

5.4.6 Store Kiosk Attention Monitoring

A store or kiosk in the mall might want to gather information on what products are interesting or eye catching to potential customers. A computer vision system that tracks the user and where the user is looking would be able to gather statistics that could be used as marketing research. This research in turn would allow stores to create more appealing displays. One potential negative side for the consumer is that such systems would also be able to collect research based on gender, age, or race.

5.5 Future Work and Improvements

The system could be improved with an algorithm to more precisely locate the eyes. The current method uses the first-moment of the motion image as the center of the eyes. Higher resolution eye images would allow a feature detector to find the corners of 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. Then, the parameters of the function could
be used as a measure for the degree the eyes are looking away from center.

One improvement could be the use of a 3D head tracker. The current head tracker does not deal well with a head turned to one side. A 3D head tracker, e.g. [24], would be able to rectify the face to a frontal view and allow the eye analysis to work better with various head orientations.

The 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 [14], which would give the interface three events to work with. Further analysis of the duration that the user looks left or right may 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.
5.6 Conclusions

This thesis has presented a new way to use eyes as an input device to the computer. The face tracker combines existing techniques in a way that allows the face to be tracked quickly as a means to locate the eyes. The method of mirroring and projecting the difference between the eyes is a novel approach to detecting which way the eyes look. By developing this system, along with an application for testing its performance, severely disabled users may soon have a new way to communicate.
Chapter 6

Appendix

6.1 Camera Noise Measurement

An experiment was conducted to measure the noise in the sensing device. The Logitec Quickcam Pro 4000 was tested with two different scenes at two different resolutions. The first scene, in Figure 6.1(a), which has low brightness variation, was created by attaching a face image to a wall. The second scene, in Figure 6.1(b), with more brightness variation, included the first picture, as well as a chair with black armrests and a second face. These scenes were recorded by placing the camera on a table approximately five feet away from the wall. To minimize the possibility of noise created by vibration of the table or the scene, no people were moving in the room during the recording. The results for some pixel locations are reported in the following tables.

(a) Low brightness variation       (b) High brightness variation

Figure 6.1: Scenes used in camera noise measurement.
Table 6.1: Camera noise measurement. Statistics for 13 pixel coordinates from a $160 \times 120$ video sequence were gathered over 9000 frames for a scene with small brightness variation.

<table>
<thead>
<tr>
<th>Average ($\mu$)</th>
<th>$x$</th>
<th>$y$</th>
<th>Min</th>
<th>Max</th>
<th>Std. Dev. ($\sigma$)</th>
<th>Variance ($\sigma^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>143.72</td>
<td>84</td>
<td>52</td>
<td>139</td>
<td>146</td>
<td>0.86</td>
<td>0.73</td>
</tr>
<tr>
<td>146.04</td>
<td>80</td>
<td>59</td>
<td>143</td>
<td>150</td>
<td>0.87</td>
<td>0.76</td>
</tr>
<tr>
<td>159.22</td>
<td>82</td>
<td>56</td>
<td>155</td>
<td>164</td>
<td>0.90</td>
<td>0.82</td>
</tr>
<tr>
<td>162.89</td>
<td>84</td>
<td>54</td>
<td>159</td>
<td>166</td>
<td>1.26</td>
<td>1.59</td>
</tr>
<tr>
<td>165.66</td>
<td>85</td>
<td>57</td>
<td>162</td>
<td>168</td>
<td>0.81</td>
<td>0.65</td>
</tr>
<tr>
<td>180.33</td>
<td>86</td>
<td>55</td>
<td>176</td>
<td>183</td>
<td>0.77</td>
<td>0.59</td>
</tr>
<tr>
<td>180.69</td>
<td>83</td>
<td>56</td>
<td>178</td>
<td>185</td>
<td>0.82</td>
<td>0.68</td>
</tr>
<tr>
<td>186.32</td>
<td>87</td>
<td>51</td>
<td>183</td>
<td>190</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>193.17</td>
<td>86</td>
<td>59</td>
<td>189</td>
<td>196</td>
<td>0.89</td>
<td>0.79</td>
</tr>
<tr>
<td>194.57</td>
<td>82</td>
<td>50</td>
<td>192</td>
<td>197</td>
<td>0.67</td>
<td>0.45</td>
</tr>
<tr>
<td>207.53</td>
<td>30</td>
<td>20</td>
<td>204</td>
<td>210</td>
<td>0.65</td>
<td>0.42</td>
</tr>
<tr>
<td>208.99</td>
<td>80</td>
<td>10</td>
<td>207</td>
<td>211</td>
<td>0.73</td>
<td>0.53</td>
</tr>
<tr>
<td>209.37</td>
<td>90</td>
<td>15</td>
<td>207</td>
<td>211</td>
<td>0.73</td>
<td>0.53</td>
</tr>
</tbody>
</table>

Table 6.2: Camera noise measurement. Statistics for 10 pixel coordinates from a $640 \times 480$ video sequence were gathered over 9000 frames for a scene with small brightness variation.

<table>
<thead>
<tr>
<th>Average ($\mu$)</th>
<th>$x$</th>
<th>$y$</th>
<th>Min</th>
<th>Max</th>
<th>Std. Dev. ($\sigma$)</th>
<th>Variance ($\sigma^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>186.77</td>
<td>450</td>
<td>335</td>
<td>181</td>
<td>190</td>
<td>1.67</td>
<td>2.78</td>
</tr>
<tr>
<td>186.77</td>
<td>480</td>
<td>330</td>
<td>183</td>
<td>190</td>
<td>1.12</td>
<td>1.26</td>
</tr>
<tr>
<td>188.97</td>
<td>470</td>
<td>320</td>
<td>185</td>
<td>193</td>
<td>1.07</td>
<td>1.14</td>
</tr>
<tr>
<td>189.84</td>
<td>445</td>
<td>315</td>
<td>185</td>
<td>194</td>
<td>1.55</td>
<td>2.41</td>
</tr>
<tr>
<td>190.07</td>
<td>460</td>
<td>310</td>
<td>186</td>
<td>194</td>
<td>1.56</td>
<td>2.45</td>
</tr>
<tr>
<td>190.09</td>
<td>455</td>
<td>305</td>
<td>185</td>
<td>196</td>
<td>2.25</td>
<td>5.08</td>
</tr>
<tr>
<td>190.58</td>
<td>450</td>
<td>310</td>
<td>186</td>
<td>194</td>
<td>1.62</td>
<td>2.62</td>
</tr>
<tr>
<td>191.72</td>
<td>460</td>
<td>300</td>
<td>187</td>
<td>195</td>
<td>1.48</td>
<td>2.20</td>
</tr>
<tr>
<td>192.82</td>
<td>465</td>
<td>295</td>
<td>187</td>
<td>197</td>
<td>1.63</td>
<td>2.65</td>
</tr>
<tr>
<td>193.93</td>
<td>475</td>
<td>285</td>
<td>189</td>
<td>197</td>
<td>1.15</td>
<td>1.32</td>
</tr>
</tbody>
</table>
Table 6.3: Camera noise measurement. Statistics for 16 pixel coordinates from a $160 \times 120$ video sequence were gathered over 9000 frames for a scene with large brightness variation.

<table>
<thead>
<tr>
<th>Average ($\mu$)</th>
<th>$x$</th>
<th>$y$</th>
<th>Min</th>
<th>Max</th>
<th>Std. Dev. ($\sigma$)</th>
<th>Variance ($\sigma^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.93</td>
<td>117</td>
<td>86</td>
<td>29</td>
<td>36</td>
<td>0.78</td>
<td>0.61</td>
</tr>
<tr>
<td>39.52</td>
<td>118</td>
<td>88</td>
<td>37</td>
<td>43</td>
<td>0.79</td>
<td>0.63</td>
</tr>
<tr>
<td>59.24</td>
<td>116</td>
<td>90</td>
<td>55</td>
<td>62</td>
<td>0.76</td>
<td>0.57</td>
</tr>
<tr>
<td>68.33</td>
<td>76</td>
<td>62</td>
<td>65</td>
<td>73</td>
<td>1.37</td>
<td>1.88</td>
</tr>
<tr>
<td>71.05</td>
<td>81</td>
<td>69</td>
<td>67</td>
<td>74</td>
<td>1.36</td>
<td>1.85</td>
</tr>
<tr>
<td>71.38</td>
<td>79</td>
<td>65</td>
<td>67</td>
<td>75</td>
<td>1.28</td>
<td>1.64</td>
</tr>
<tr>
<td>71.69</td>
<td>80</td>
<td>65</td>
<td>67</td>
<td>75</td>
<td>1.13</td>
<td>1.27</td>
</tr>
<tr>
<td>74.34</td>
<td>79</td>
<td>68</td>
<td>71</td>
<td>79</td>
<td>1.24</td>
<td>1.53</td>
</tr>
<tr>
<td>75.83</td>
<td>84</td>
<td>73</td>
<td>72</td>
<td>81</td>
<td>1.50</td>
<td>2.26</td>
</tr>
<tr>
<td>77.45</td>
<td>81</td>
<td>67</td>
<td>74</td>
<td>82</td>
<td>1.52</td>
<td>2.32</td>
</tr>
<tr>
<td>81.16</td>
<td>82</td>
<td>68</td>
<td>76</td>
<td>86</td>
<td>1.13</td>
<td>1.27</td>
</tr>
<tr>
<td>82.05</td>
<td>77</td>
<td>70</td>
<td>77</td>
<td>87</td>
<td>1.41</td>
<td>2.00</td>
</tr>
<tr>
<td>87.89</td>
<td>77</td>
<td>72</td>
<td>83</td>
<td>93</td>
<td>1.70</td>
<td>2.88</td>
</tr>
<tr>
<td>138.43</td>
<td>83</td>
<td>10</td>
<td>135</td>
<td>143</td>
<td>0.90</td>
<td>0.81</td>
</tr>
<tr>
<td>158.42</td>
<td>82</td>
<td>12</td>
<td>154</td>
<td>161</td>
<td>0.92</td>
<td>0.84</td>
</tr>
<tr>
<td>172.45</td>
<td>82</td>
<td>9</td>
<td>169</td>
<td>175</td>
<td>0.98</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 6.4: Camera noise measurement. Statistics for 16 pixel coordinates from a $640 \times 480$ video sequence were gathered over 9000 frames for a scene with large brightness variation.

<table>
<thead>
<tr>
<th>Average ($\mu$)</th>
<th>$x$</th>
<th>$y$</th>
<th>Min</th>
<th>Max</th>
<th>Std. Dev. ($\sigma$)</th>
<th>Variance ($\sigma^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>108.54</td>
<td>390</td>
<td>250</td>
<td>104</td>
<td>114</td>
<td>1.14</td>
<td>1.30</td>
</tr>
<tr>
<td>124.00</td>
<td>385</td>
<td>265</td>
<td>114</td>
<td>130</td>
<td>1.99</td>
<td>3.97</td>
</tr>
<tr>
<td>145.65</td>
<td>600</td>
<td>375</td>
<td>140</td>
<td>150</td>
<td>1.26</td>
<td>1.59</td>
</tr>
<tr>
<td>149.19</td>
<td>560</td>
<td>370</td>
<td>144</td>
<td>171</td>
<td>1.26</td>
<td>1.58</td>
</tr>
<tr>
<td>150.34</td>
<td>380</td>
<td>220</td>
<td>146</td>
<td>154</td>
<td>1.10</td>
<td>1.21</td>
</tr>
<tr>
<td>152.51</td>
<td>595</td>
<td>365</td>
<td>147</td>
<td>158</td>
<td>1.17</td>
<td>1.36</td>
</tr>
<tr>
<td>159.08</td>
<td>585</td>
<td>355</td>
<td>153</td>
<td>164</td>
<td>1.24</td>
<td>1.53</td>
</tr>
<tr>
<td>161.26</td>
<td>580</td>
<td>350</td>
<td>158</td>
<td>171</td>
<td>1.20</td>
<td>1.45</td>
</tr>
<tr>
<td>163.75</td>
<td>585</td>
<td>345</td>
<td>157</td>
<td>171</td>
<td>1.92</td>
<td>3.69</td>
</tr>
<tr>
<td>163.86</td>
<td>575</td>
<td>345</td>
<td>155</td>
<td>171</td>
<td>2.01</td>
<td>4.03</td>
</tr>
<tr>
<td>166.64</td>
<td>555</td>
<td>325</td>
<td>161</td>
<td>173</td>
<td>1.58</td>
<td>2.49</td>
</tr>
<tr>
<td>166.72</td>
<td>565</td>
<td>335</td>
<td>161</td>
<td>173</td>
<td>1.30</td>
<td>1.70</td>
</tr>
<tr>
<td>169.26</td>
<td>590</td>
<td>330</td>
<td>165</td>
<td>173</td>
<td>1.26</td>
<td>1.60</td>
</tr>
<tr>
<td>236.15</td>
<td>260</td>
<td>45</td>
<td>231</td>
<td>240</td>
<td>1.33</td>
<td>1.78</td>
</tr>
</tbody>
</table>
Bibliography


