Counting Fingers in Real Time: A Webcam-Based Human-Computer Interface with Game Applications

Stephen C. Crampton and Margrit Betke

Department of Computer Science, Boston University
111 Cummington Street, Boston, MA 02215, USA
{stevec,betke}@cs.bu.edu

Abstract

Finger Counter is a simple and universal human-computer interface. Using a webcam, it interprets specific hand gestures as input to a computer system in real time. *Finger Counter* employs two novel computer-vision techniques: (1) a background-differencing method adaptive to changing lighting conditions and camera movement and (2) a new procedure to analyze hand contours. The *Finger Counter* interface runs under Linux with multiple threads of execution. *Finger Counter* applications include a game designed to teach children to count with their fingers and a program that allows you to "finger paint" on a computer screen. Test subjects compared the familiar keyboard with *Finger Counter* under difficult lighting and background conditions and found *Finger Counter* a viable substitute.

1 Introduction

Real-time computer-vision systems using hand-gesture analysis have been explored by, for example, Freeman et al., 1998, and Bretzner, Laptev, Lindeberg, Lenman & Sundblad, 2001. *Finger Counter* introduces new techniques for adapting to changing lighting and unsteady cameras, as well as two game applications.

2 Method

Figure 1 shows the system architecture at the highest level. *Finger Counter* processes incoming images, extracts features from the processed images, and stores feature information in a circular buffer. As each new frame arrives, *Finger Counter*'s analysis module inspects the buffer and updates the state, *i.e.,* how many fingers the user is holding up. Application programs, such as those described below, examine the state and respond accordingly.

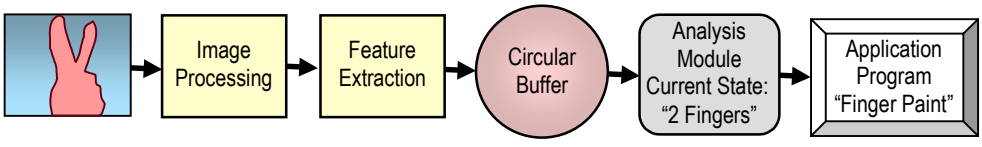

Figure 1: System overview

2.1 Image Processing

In the image-processing stage, *Finger Counter* uses a new background-differencing technique to segment a user's hand from the background. *Finger Counter* thus requires an initialization step, when the user is not in the camera's view. During initialization, which lasts less than a second, the camera adjusts to the lighting conditions and acquires a background image. An audio cue tells the user that initialization is happening and informs him or her once it is completed.

2.1.1 Adaptive Background Differencing

Finger Counter's background-differencing algorithm adapts to changes in scene brightness and camera position. It minimizes the squared error of the brightness difference between pixels in an initial background image B_0 and background pixels in sub-images of subsequent frames. The result of the minimization is an offset $\mathbf{u}$ that aligns a $M' \times N'$ sub-image I with the $M \times N$ background image B_0, where $M' \leq M$ and $N' \leq N$. Initially, $u = (u_i, u_j) = \frac{1}{2}(M - M', N - N')$.

As each new frame comes in, the system aligns, crops, and brightness-adjusts the background image to $B(i,j) = \frac{\lambda}{\lambda_0} B_0(u_i + i, u_j + i)$, where λ and λ_0 are the camera's current and initial automatic-gain-control (AGC) levels, respectively. The AGC adjusts dynamically in response to changed lighting conditions. From the input image I, B is subtracted to get $D(i,j) = \{ I(i,j)$ if $I(i,j) - B(i,j) > \tau_D$, 0 otherwise. Threshold $\tau_D = \kappa \max_{(i,j)}(I(i,j) - B(i,j))$ was chosen empirically. Values for κ and other parameters are given in section 3 below.

After foreground and background pixels are segmented, the background-image offset $\mathbf{u}$ is recomputed by template-matching the background image B_0 with background pixels in I, taking into account the current AGC level. Let $P = \{(i,j) \mid D(i,j) = 0\}$. Updated $\mathbf{u}$ is computed as $\arg\min_{\mathbf{u}} \sum_{(i,j) \in P} \left(I(i,j) - \frac{\lambda}{\lambda_0} B_0(u_i + i, u_j + i) \right)^2$. *Finger Counter*'s background-differencing method allows the camera to be perturbed by as many as $\frac{1}{2}(N - N')$ pixels vertically and $\frac{1}{2}(M - M')$ pixels horizontally.

2.1.2 Edge Detection

From the difference image D, the system computes an edge image using standard computer-vision techniques. First, the difference image D is convolved with Prewitt filters. (Prewitt, 1970) Then an iterative edge-following algorithm similar to the one described in Jain, Kasturi & Schunck, 1997, p. 47, identifies distinct connected contours in the image. *Finger Counter* discards all but the largest contour, which is represented by the 1 pixels in a binary image E. Let A be the number of 1 pixels. The centroid $(\bar{x}, \bar{y}) = \left(\frac{1}{A} \sum_{(i,j)} jE(i,j), \frac{1}{A} \sum_{(i,j)} iE(i,j) \right)$ corresponds to the centre of mass of the contour. An adjusted centroid $(\bar{x}, \zeta \bar{y})$ is then computed, with ζ chosen experimentally to move the centroid downward near the centre of an up-facing palm. Finally, the

system defines a rectangular region of interest (ROI) containing all contour pixels level with or above the adjusted centroid $(\bar{x}, \zeta \bar{y})$.

2.2 Feature Extraction

Human fingers protrude in a radial fashion, a trait *Finger Counter* exploits. A pixel in contour image E, referenced by Cartesian coordinates (i, j), is converted to polar coordinates (α, r). The origin of the polar coordinates is the adjusted centroid $(\bar{x}, \zeta \bar{y})$. Figure 2 shows the result of this conversion.

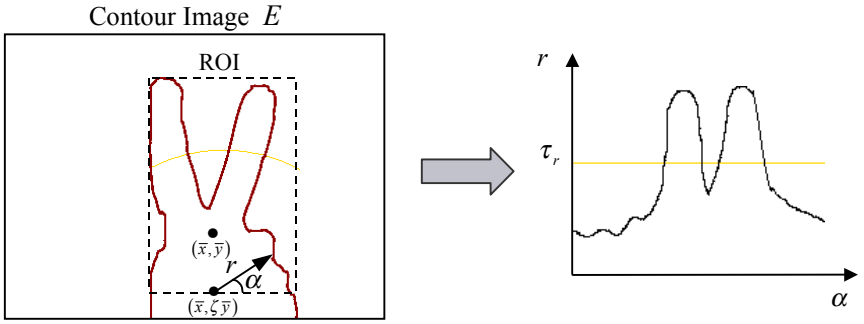

Figure 2: From Cartesian to polar coordinates

To count fingers the system first sets threshold τ_r to be a fraction of the maximum r in the ROI: $\tau_r = c \max_\alpha r(\alpha)$. Then, the system identifies the tip $\boldsymbol{\varphi}_t(k)$ of finger-like protrusion k in frame t as a local maximum in a range of angles for which $r(\alpha)$ exceeds τ_r: $\alpha_{max} = \arg\max_\alpha r(\alpha)$, where $r(\alpha) > \tau_r$. The system stores $\boldsymbol{\varphi}_t$ and v_t, the number of finger-like protrusions found in frame t, in the circular buffer shown in Figure 1.

2.3 State Analysis

The ultimate goal is to reliably determine the state of a user's hand, *e.g.,* "he is holding up three fingers." To that end, the system analyzes sequences of frames at a time. By doing so, *Finger Counter* is less prone to errors caused by noise, camera artefacts, or cluttered images.

Finger Counter's state-analysis module considers the last ρ records from the circular buffer. If (1) they report the same number of finger-like protrusions, $v_{t-k} = v_{t-k-1}\ \forall 0 \le k < \rho$, and (2) the distance in pixels between the same protrusion in successive frames is less than a threshold τ_v, $\left\| \boldsymbol{\varphi}_k(i) - \boldsymbol{\varphi}_{k-1}(i) \right\| < \tau_v$ for $t - \rho + 1 \le t \le k$, $1 \le i \le v_t$, then the state-analysis module concludes that there are v_t fingers held up in frame t. Note that this technique introduces a f/ρ-second delay at a frame rate of f frames per second.

3 Implementation

We implemented *Finger Counter* under Linux kernel 2.4.18 on a laptop computer with a Pentium IV 1.4 GHz processor and 256 MB of RAM. Attached to the computer via USB were a Logitech Quickcam 4000 Pro or a Creative Labs Webcam III, running (with compression) at 30 frames per second. The *Finger Counter* interface processes about 10 frames per second. Most of the delay is due to the adaptive background differencing. Parameters were empirically determined as follows: $M = 320$, $N = 240$, $M' = 300$, $N' = 225$, $\kappa = 0.2$, $\zeta = 0.67$, $c = 0.75$, $\rho = 5$, and $\tau_v = 28.28$.

Multiple threads of execution improve the system's efficiency. (Silberschatz & Galvin, 1997) *Finger Counter* uses two threads to interface with Video4Linux, a Linux module that facilitates communication with cameras attached to the system. Another thread processes incoming frames as described above and implements one of the application programs described below. A final thread handles audio output. Multithreading prevents the system from stalling while waiting for input or output and thus maintains seamless interaction with the user.

4 HCI Applications

We created two applications for *Finger Counter*. One is a game in which the player is audibly prompted to hold up a certain number of fingers and then the system tells the player how many fingers it sees. A modified version of the game randomly alternates requests to hold up fingers with requests to type a key on a standard keyboard and logs response times. We used the modified game to evaluate the interface, as discussed below.

The second application allows the user to paint on the screen using her fingertips. If she holds up one finger, she paints with one brush. If she holds up two fingers, she paints with a single brush, but can vary the brush size dynamically by spreading or contracting the two fingers. If she holds up three or four fingers, she paints simultaneously with that many brushes. Finally, holding up five fingers erases the entire image.

5 Experiments

To test the *Finger Counter* interface, we performed 18 experiments on test subjects of various ages and with diverse computer experience. Test subjects were shown the finger-painting application and asked to play with it for a few minutes. Then, the subjects played the modified voice-interactive game, and their response times were logged. Table 1 shows the average times it took to respond to an audio request to either hold up a certain number of fingers from one through five or type a key from "1" through "5" on the keyboard.

Table 1: Response Times

	Finger Counter	Keyboard
Average	2.72 seconds	2.16 seconds
Standard Deviation	0.86 seconds	0.53 seconds

We tested the interface under varied lighting and background conditions, from a fluorescent-lighted laboratory to a sunny coffee shop in Santa Barbara, California. With minimal training, test subjects achieved response times with Finger *Counter* comparable to those with the familiar

keyboard. Figure 3 demonstrates how response times were distributed among test subjects and also shows a *Finger Counter* user.

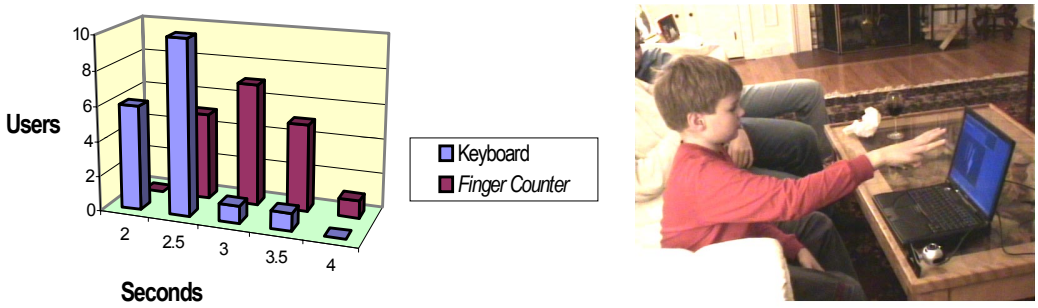

Figure 3: Histogram of response times (left) and a *Finger Counter* user (right)

For applications that require a user to choose between five or fewer options, *Finger Counter* is a viable keyboard replacement. Moreover, because *Finger Counter* does not require the user to touch any equipment, it is well suited to applications where the user's hands are engaged in other activities or are dirty (*e.g.,* an auto mechanic) or sterile (*e.g.,* a medical worker). *Finger Counter* may also be useful for those who, because of physical limitations, find it difficult to use a keyboard.

6 Conclusion

Finger Counter works with an inexpensive webcam to provide a fun and easy human-computer interface. Future research will focus on recognizing additional hand postures and developing further applications. Future experiments will include computer users with physical disabilities. The ultimate goal is an intuitive human-computer interface for everyone, regardless of computer expertise or physical ability.

References

Bretzner, L., Laptev, I., Lindeberg, T., Lenman, S., & Sundblad, Y. (2001). A prototype system for computer vision based human computer interaction. Retrieved February 13, 2003, from http://www.nada.kth.se/cvap/abstracts/cvap251.html

Freeman, W., Anderson, D., Beardsley, P., Dodge, C., Roth, M., Weissman, C., & Yerazunis, W. (May/June 1998). Computer vision for interactive computer graphics. *IEEE Computer Graphics and Applications,* 18(3), 42-53.

Jain, R., Kasturi, R., & Schunck, B. (1995). Machine vision. New York: McGraw-Hill.

Jennings, C. (1999). Robust finger tracking with multiple cameras. Retrieved February 13, 2003, from http://www.cs.ubc.ca/spider/jennings/ratfg-rts99/cj99.pdf

Prewitt, J. (1970). Object enhancement and extraction. In B. Lipkin & A. Rosenfeld (Eds.), *Picture Processing and Psychopictorics* (pp. 75-149). New York: Academic Press.

Silberschatz, A. & Galvin, P. (1997). Operating system concepts (5th ed.). New York: Wiley.

Yang, M.-H. (2002). Extraction of 2D motion trajectories and its application to hand gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 24, 1061-74.