Eurographics Italian Chapter Conference (2008)
V. Scarano, R. De Chiara, and U. Erra (Editors)

Low cost finger tracking on flat surfaces

G. M. Farinella, E. Rustico

Department of Mathematics and Computer Science
University of Catania
Viale A. Doria 6, 95125, Catania, Italy
gfarinella@dmi.unict.it
eugenio.rustico@galileo.dmi.unict.it

Abstract

We present a flexible system to track the movements of a bare finger on a flat surface. The proposed system is able to discriminate whether the user is touching or just pointing at the surface. The system works using two webcams and a fast scanline-based algorithm. The initial setup of the two webcams is easy and fast. No markers, gloves, or other hand-held devices are required. Since the system is independent from the nature of the pointing surface, it is possible to use a screen or a projected wall as a virtual touchscreen. The complexity of the algorithms used by the system grows less than linearly with resolution, making the software layer very lightweight and suitable also for low-powered devices like embedded controllers.

Categories and Subject Descriptors (according to ACM CCS):
H.5.2 [**User Interfaces**]: Input Devices and Strategies; **General Terms**: Low Cost Input Devices, Human Computer Interaction, Computer Vision

1. Introduction

To make the Human-Computer Interaction as natural as possible is a central problem in Computer Science; however, we're still a long way from enabling the users with a natural and immediate interface between their thoughts and the machines.

There are two main kinds of data that humans input to the computers: text and graphic data. They are represented respectively by the two most common computer input devices : keyboard and mouse. Speech and handwriting recognition systems offer a "natural" alternative to keyboards, while these devices seem to be still necessary in every modern computer and communication device. To point a cursor, we have instead several alternatives to mice: trackballs, touchpads, touchscreens, joysticks, graphic tablets and so on. Touchscreens, despite their low flexibility, are probably the preferred ones by most users. The reason is that they reflect, as no other device does, the way we use to get in touch and interact with the reality around us: we use to point and touch directly what we see around us with our hands. Touchscreens allow to do the same with our fingers on computer interfaces. Unfortunately, touchscreen flexibility is low: finger tracking is impossible without physical contact; it is not possibile to use sharp objects on them; large touch-sensible displays are expensive because of their manifacturing cost anddamage-proneness.

In this paper we present a tracking system capable of turning any static surface in a tablet, and any kind of display - even very large ones, like projected walls - in a touchscreen. The system that we propose is made of low cost devices, without the use of any kind of equipment that is not possible to find in any computer shop with less than 100€.

2. Related works

In the era of augmented reality and wearable computing, the research in computer interfaces is turning back to the human body, trying to adapt the way we communicate with computers to our natural way of move and behave. Speech-driven interfaces, gesture-recognition softwares and facial expression interpreters are just some examples of this growing trend, and those that are going to design a system

based on such technologies will find a rich soil in literature. Among the technologies studied in this branch of research, there is a growing interest in the ones which involve real-time body tracking. Some systems track particular parts of the body or use expensive or original devices [GMR02,Wil05,Rak06,Mor05,Lee07], but most approaches require just low cost cameras to track eyes, head and/or hands. We focus on finger tracking systems which do not require special markers, gloves, hand-held devices or skin detection.

While eyes and head tracking need to direct the camera towards the user, finger tracking systems have a wider range of choices. A first possibility to track user's hands is to use the same person-directed view point as for head tracking; this is used in [CT06] to estimate the position of the fingertip in the view frustum of the user, but with strong limits on the maximum tracking resolution; in [IVV01], instead, the absolute position of the arms directly translates into screen coordinates, whereas mouse clicks are represented by hand gestures. The main disadvantage of this kind of approach is that the background of the tracked arms in the view point of the camera has to be static, and the user is forced to do unnatural movements in the air.

A second possibility is to direct the camera towards the pointing surface. We can assume that this surface is static [LB04, MRB05, ML04], making relatively simple to locate and interpret finger positions. However, a tracking system working with dynamic backgrounds like projected surfaces would be desiderable. Image differencing can be a good solution [vHB01], but it requires the algorithm to "know" what is currently projected; thus, if the surface is highly dynamic, the pointing device has to be in real time communication with the projecting device. Cross-correlation differencing has been used [CBC95] and a robust color-based segmentation alghoritm has been developed [DKS01]. Pointing the camera towards a dynamic surface implies in general the use of computationally expensive algorithms.

A third possible approach, that may drastically reduce all the above problems, is to have the cameras watching sidewise; using this point of view we do not have any problem with dynamic backgrounds both behind the user and on the pointing surface. Among the very few works using this approach, in [QMZ95] the webcam is above the monitor looking towards the keyboard, and the finger is located with a color segmentation algorithm. The movement of the hand along the axis perpendicular to the screen is mapped to the vertical movement of the cursor, and a keyboard button press simulates the mouse click. However, the position of the webcam has to be calibrated and the vertical movement is mapped in an unnatural way.

All of the above approaches need to process the entire image as it is captured by the webcam. Thus, every of the above algorithms is at least quadratic with respect to resolution (or linear with respect to image area). Although it's possible to use smart region finding algorithms, these would not resolve the problem entirely. We propose a different way to track user movements keeping the complexity low. To this aim we use two cameras drastically decreasing the scanning area to a discrete number of pixel lines. Furthermore our system makes even non-experienced users able to set up the cameras in a few seconds from any uncalibrated position.

3. System description

We use two low cost webcams positioned sidewise so that the lateral silhouette of the hand is captured into an image like figure 1. After a quick auto-calibration, the software layer will be able to interpret the image flow and translate it into absolute screen coordinates and mouse button pressures; the corresponding mouse events will be simulated on the operative system in a completely transparent way for the application level. We call *pointing surface* the rectangle of surface to be tracked; as pointing surface we can choose a desk, a lcd panel, a projected wall, etc.. An automatic region stretching is done to map the coordinates of the pointing surface to the target display. Any type of "device" can be used to point or touch the surface. The system will track a finger as well as a pencil, a chalk or a wooden stick.

3.1. Scanlines

We focus the processing only on a small number of pixel lines from the whole image provided by each webcam; we call these lines *scanlines*. Each scanline is horizontal and ideally parallel with the pointing surface; we call *touching scanline* the lowest scanline (the nearest to the pointing surface), and *pointing scanline* every other one. The calibration phase requires that we first grab a frame before any pointer enters in the tracking area; these reference frames (one per webcam) will be used to look for scanline interruptions (e.g. presence of fingers) through a simple image differencing algorithm (fig.1). The detection of a finger only in pointing scanlines will mean that the surface is only being pointed, while a detection in all the scanlines will mean that the user is currently touching the surface. To determine if a mouse button pressure has to be simulated, we can just look at the touching scanline: we assume that the user is clicking if the touching scaline is interrupted in at least one of the two views. We must have at least two scanlines for each view (a pointing and a touching one); we could increase the number of scanlines up to tens, but three or four will suffice for an excellent accuracy. During the calibration phase the system decides the vertical position of each scanline depending on the position in the image of the finger while the user touches the surface.

3.2. Smart finger detection

We call a pixel *different* if the difference between its color and the reference color of the same pixel is higher than a predetermined threshold; a *scanline interruption* occurs when

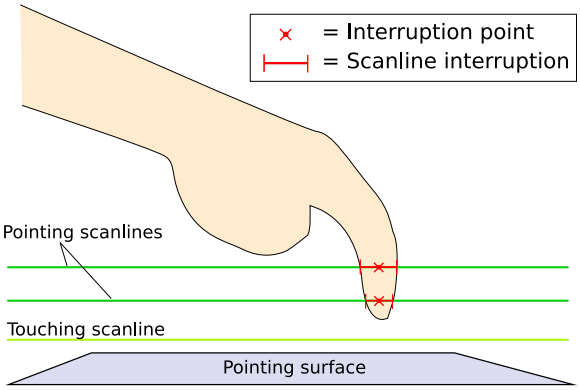

Figure 1: *Visual representation of scanlines within the view field of each camera.*

a run of k *different* pixels is detected. We call *interruption point* the middle point of a scanline interruption, and *touching point* the interruption point of a touching scanline. Pixel comparison can be made either in RGB or YUV; we obtained better results with the latter color model.

The analysis of the scanline is made faster adopting two "tricks". First, we do not need to scan all the pixels in a scanline: candidate locations for *interruption points* can be discovered comparing every k^{th} pixel on the line. We then proceed to look for a k-run of *different* pixels only in the areas around the candidates discovered insofar (see figure 2). If no interruptions are detected on a scanline, we do not need to continue our frame analysis; otherwise, we scan the next scanline starting from the same x coordinate where we detected an interruption point in the previous line (figure 2).

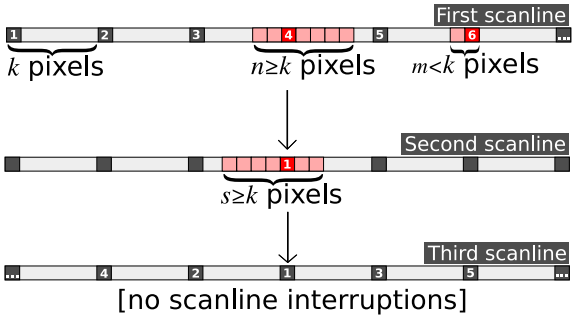

Figure 2: *Example of smart finger detection. In the highest scanline, we first scan only a pixel every k ones; then, the neighbourhood of candidate pixels is analyzed. If any interruption is detected in a scanline, we start scanning the next one at the same x coordinate of the last detected interruption.*

3.3. Positioning the cameras

The proposed technique requires the positioning of two webcams relatively to the pointing surface. We could just put them so that one detects only movements along the X axis, while the other one detects Y axis changes. This solution is the simplest to implement, but requires the webcams to have their optical axes perfectly aligned along the sides of the pointing surface. Moreover, the wider is the view field of a webcam, the more we loose accuracy on the opposite side of the surface. On the other hand, the narrower is the view field of the webcams, the farther we have to put them to capture the entire surface. For instance, for a 2×1.5 m projected wall and a 45° view field, we have to put the webcam ~5.2 meters away to catch the whole horizontal size. (figure 3)

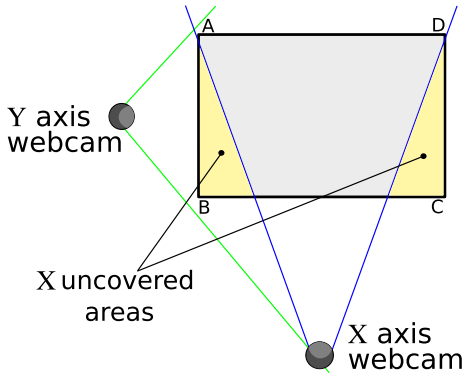

Figure 3: *The webcam along Y axis has a wide view field, but this brings resolution loss on segment $\overline{DC}$; on the other side, the webcam along X axis has a narrow view field, but it has to be positioned far from the pointing surface to cover the whole area.*

A really usable system should not bother the final user about webcam calibration, view angles and so on. A way to minimize the calibration effort is to position the webcams near two non-opposite corners of the pointing surface, far enough to catch it whole and oriented as the surface diagonals were about bisectors of the respective view fields (figure 4). With this configuration there is no need to put the webcams far away from the surface; this reduces the accuracy loss on the "far" sides.

In the rest of this paper we will assume, for sake of clarity, that the webcams are in the same locations and orientations as in figure 4. However, the proposed tracking algorithm works with a variety of configurations without changes in the calibration phase.

3.4. Tracking algorithm

When the system is loaded, the calibration phase starts. In this phase, after grabbing a view of the background, we ask the user to touch the vertices of the pointing surface and its

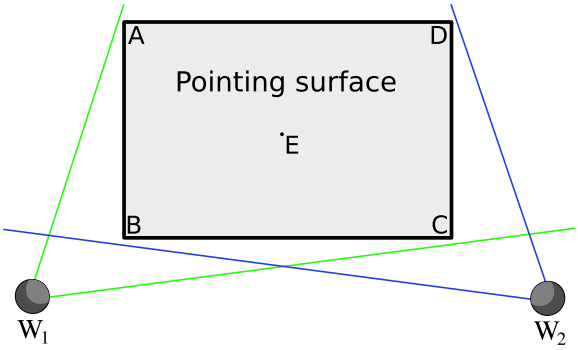

Figure 4: *Suggested configuration to optimize the use of view frustum of the cameras.*

center; for each vertex the system stores the x coordinate of the corresponding interruption point in the touching scanline. In a couple of seconds, the calibration is complete and the system is ready to start tracking.

During calibration phase we calculate the perspective transformation which translates the absolute screen coordinates to absolute coordinates in the viewed image. Aiming to store vertices in homogeneous coordinates, we use a transformation defined by a 3x3 matrix M:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \cdot V = P \cdot \alpha$$

Since P is determined up to a proportional factor α there is no loss of generality in setting one of the elements of M to an arbitrary non-zero value. In the following we set the element $i = 1$. To obtain all the other elements of M, in principle the correspondence between four pairs of points must be given. The proposed application only needs to look at horizontal scanlines; for this reason there is no need to know the coefficients d,e,f of M and we only have to determine the values of a,b,c,g,h.

The number of unknown matrix elements has been decreased to five, so we only need the x coordinate of five points. During the calibration phase, we ask the user to touch the four vertices of the pointing surface and its center. This setup greatly simplifies the computation of the unknown coefficients. Indeed points A,B,C,D and the center E (see fig.4) have screen coordinates respectively:

$$\begin{aligned} A &= (0,0) \\ B &= (0,H) \\ C &= (W,H) \\ D &= (W,0) \\ E &= (W/2,H/2) \end{aligned}$$

when display resolution is $W \times H$.

If Q is a point on the surface, let Q_{xp} be the x coordinate of the corresponding projected point. The final linear system ot solve is:

$$\begin{pmatrix} 0 & H & 0 & -HB_{xp} \\ W & H & -WC_{xp} & -HC_{xp} \\ W & 0 & -WD_{xp} & 0 \\ E_x & E_y & -E_xE_{xp} & -E_yE_{xp} \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ g \\ h \end{pmatrix} = \begin{pmatrix} B_{xp} - A_{xp} \\ C_{xp} - A_{xp} \\ D_{xp} - A_{xp} \\ E_{xp} - A_{xp} \end{pmatrix}$$

which makes easy to obtain a, b, c, g, h for each camera.

During the tracking phase, we have the opposite aim: we know the projected x coordinate in each view, and from these values (let them be X_l and X_r) we would like to compute the x and y coordinates of the correspondent unprojected point (that is, the point the user is touching). Let a_l, b_l, c_l, g_l, h_l be the transformation values for the first camera, and a_r, b_r, c_r, g_r, h_r for the second one; the linear system we have to solve in this case is

$$\begin{cases} a_l x_l + b_l y_l + c_l z_l & = X_l \\ g_l x_l + h_l y_l + z_l & = 1 \\ a_r x_r + b_r y_r + c_r z_r & = X_r \\ g_r x_r + h_r y_r + z_r & = 1 \end{cases}$$

It is convenient to divide the first two equations by z_l and the latter two by z_r, and rename the unknown variables as follows:

$$\begin{aligned} x &= \frac{x_l}{z_l} = \frac{x_r}{z_r} \\ y &= \frac{y_l}{z_l} = \frac{y_r}{z_r} \\ z'_l &= \frac{1}{z_l} \\ z'_r &= \frac{1}{z_l} \end{aligned}$$

so that the final system is

$$\begin{pmatrix} a_l & b_l & -X_l & 0 \\ g_l & h_l & -1 & 0 \\ a_r & b_r & 0 & -X_r \\ g_r & h_r & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z'_l \\ z'_r \end{pmatrix} = \begin{pmatrix} -c_l \\ -1 \\ -c_r \\ -1 \end{pmatrix}$$

This is a determined linear system, and it is possible to demonstrate that in the setting above there is always one and only one solution. Solving this system in x and y we find the absolute coordinates of the point that the user is pointing/touching on the surface.

We can solve this system in a very fast way by computing once a LU factorization of the coefficient matrix, and by using it to compute x and y for each pair of frames; we can also use numerical methods, such as Single Value Decomposition, or direct formulas. We chose the latter method because of the relatively small size of the matrix.

3.5. Resolution accuracy

Let's consider now how accurate is the tracking system depending on display and webcam physical characteristics. Let $t = (x_t, y_t)$ be a point on the pointing surface, $X_D \times Y_D$ the display resolution (i.e. the resolution of the projector for a projected wall) and $X_{W_1} \times Y_{W_1}$ the resolution of a webcam W_1; let β_{W_1} be the bisector of the view frustum of W_1, and let the upper left corner of the surface be the origin of our coordinate system (with Y pointing downwards, like in fig.5). We will assume for simplicity that the view frustum of the camera is centered on the bisector of the coordinate system, but the following considerations keep their validity also in slightly different configurations.

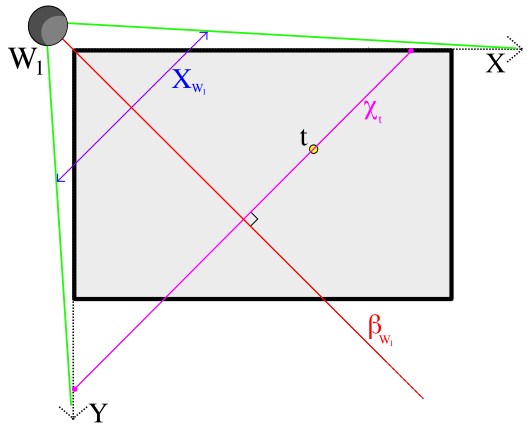

Figure 5: *We define "resolution accuracy of W_1 in t" the ratio between the length of χ_t and the number of pixels detected by W_1; from this figure it's clear that we only care about the horizontal resolution of W_1, which is constant in the whole view frustum.*

The higher is the number of pixels detected by the webcam for each real pixel of the display, the more accurate will be the tracking; thus, if we want to know how accurate is the detection of a point in the pointing surface, we could consider the ratio between the length in pixels of the segment χ_t, passing by t and perpendicular to β_{W_1}, and the number of pixels detected by the webcam W_1 (see fig. 5). We define *resolution accuracy of W_1 in t* and we call $\sigma(W_1, t)$ this ratio. Because pixels are approximatively squares, the number of pixels along the diagonal of a square is equal to the number of pixels along an edge of the square; thus, the length of χ_t will be equal to the distance from the origin of one of the two points that χ_t intercepts on the X and Y axes. For every point $p \in \chi_t$ is $x_p + y_p = k$; then, its length will be equal to the y-intercept of the line passing by t and perpendicular to β_{W_1}. So we have $|\chi_t| = x_t + y_t$; hence, the resolution accuracy of W_1 in t is

$$\sigma(W_1, t) = \frac{X_w}{x_t + y_t}$$

One of the most commonly used display resolutions for projected walls is nowadays 1024×768 pixels, while one of the maximum resolutions that recent low-cost webcams support is 1280×1024 pixels at 15 frames per second. In this configuration, the resolution accuracy in $t = (1024, 768)$ is

$$\sigma(W_1, t) = \frac{1280}{1024 + 768} \approx 0.71$$

This is the lowest resolution accuracy we have with W_1 in the worst orientation; if we invert the Y axis to get the accuracy for W_2 (supposing that W_2 is placed on the upper right corner of the surface), $\sigma(W_2, t) \approx 1.7$. In the central point $u = (512, 384)$ of the display we have $\sigma(W_1, u) = \sigma(W_2, u) \approx 1.4$; it's immediate that, in the above configuration, the average resolution accuracy is higher than 1:1 (sub-pixel).

4. Experimental settings and system performance

The webcams we used for testing are two Philips SPC1000NC, with a native SXGA video sensor; their price was about 40€each, and they are capable of producing a SXGA video at about 15fps. There is a mature Video4Linux2 compliant driver (uvcvideo) available for GNU/Linux.

Our prototype has good resolution accuracy and excellent time performances. Two USB webcameras connected to the same computer can usually send less than 20 frames per second simultaneously, while the software layer could elaborate hundreds more.

We implemented the tracking system in C++ in a GNU/Linux environment; in the relatively small source code (~1000 lines) all software layers are strictly separated, so that it's possible to port the whole system to different platforms with very little changes in the source.

A first demonstrational video is available for download at `http://svg.dmi.unict.it/iplab/download/FingerTracking/`; we are working to produce other videos showing the calibration phase and the finger tracking on different conditions. These lasts videos will be available at the same website at the conference time.

5. Conclusions and future work

We presented a low cost system for bare finger tracking able to turn lcd displays into touchscreens, as well as a desk into a design board, or a wall into an interactive whiteboard. Many application domains can benefit from the proposed solution: designers, teachers, gamers, interface developers. The proposed system require a simple calibration phase.

Future works will be devoted to improve the robustness of the overall system in each involved step. Moreover, suitable evaluation procedures for such kind of system will be addressed.

6. Acknowledgements

We would like to thank the `it.scienza.matematica` newsgroup for their unvaluable tips and suggestions.

References

[CBC95] CROWLEY J., BERARD F., COUTAZ J.: Finger tracking as an input device for augmented reality, 1995.

[CT06] CHENG K., TAKATSUKA M.: Estimating virtual touchscreen for fingertip interaction with large displays. In *OZCHI '06: Proceedings of the 20th conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: design: activities, artefacts and environments* (New York, NY, USA, 2006), ACM, pp. 397–400.

[DKS01] DOMINGUEZ S. M., KEATON T., SAYED A. H.: Robust finger tracking for wearable computer interfacing. In *PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces* (New York, NY, USA, 2001), ACM, pp. 1–5.

[GMR02] GORODNICHY D., MALIK S., ROTH G.: Nouse 'use your nose as a mouse' - a new technology for hands-free games and interfaces, 2002.

[IVV01] IANNIZZOTTO G., VILLARI M., VITA L.: Hand tracking for human-computer interaction with graylevel visualglove: turning back to the simple way. In *PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces* (New York, NY, USA, 2001), ACM, pp. 1–7.

[LB04] LETESSIER J., BÉRARD F.: Visual tracking of bare fingers for interactive surfaces. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2004), ACM, pp. 119–122.

[Lee07] LEE J. C.: Head tracking for desktop VR displays using the Wii remote. `http://www.cs.cmu.edu/~johnny/projects/wii/`. 2007.

[ML04] MALIK S., LASZLO J.: Visual touchpad: a two-handed gestural input device. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces* (New York, NY, USA, 2004), ACM, pp. 289–296.

[Mor05] MORRISON G. D.: A camera-based input device for large interactive displays. *IEEE Computer Graphics and Applications 25*, 4 (2005), 52–57.

[MRB05] MALIK S., RANJAN A., BALAKRISHNAN R.: Interacting with large displays from a distance with vision-tracked multi-finger gestural input. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2005), ACM, pp. 43–52.

[QMZ95] QUEK F., MYSLIWIEC T., ZHAO M.: Fingermouse: A freehand computer pointing interface, 1995.

[Rak06] RAKKOLAINEN I.: Tracking users through a projection screen. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia* (New York, NY, USA, 2006), ACM, pp. 101–104.

[vHB01] VON HARDENBERG C., BÉRARD F.: Bare-hand human-computer interaction. In *PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces* (New York, NY, USA, 2001), ACM, pp. 1–8.

[Wil05] WILSON A. D.: Playanywhere: a compact interactive tabletop projection-vision system. In *UIST* (2005), Baudisch P., Czerwinski M., Olsen D. R., (Eds.), ACM, pp. 83–92.