Low cost finger tracking for a virtual blackboard

Eugenio Rustico
Dipartimento di Matematica e Informatica
Image Processing Laboratory
Università di Catania
Viale Andrea Doria, 6
95125, Catania, Italy
rustico@dmi.unict.it

Abstract

This paper presents a complete and inexpensive system to track the movements of a physical pointer on a flat surface. Any opaque object can be used as a pointer (fingers, pens, etc.) and it is possible to discriminate whether the surface is being touched or just pointed at. The system relies on two entry-level webcams and it uses a fast scanline-based algorithm. An automatic wizard helps the user during the initial setup of the two webcams. 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.

Keywords: Finger, pointer, tracking, optical, video, webcam, camera, scanline, virtual touchscreen, Human-Computer Interaction.

1 INTRODUCTION

Among the existing graphical input devices, computer users love especially touchscreens. 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 with our hands what we see around us; touchscreens allow to do the same with our fingers on computer interfaces. This preference is confirmed by a strong trend in the industry of high-end platforms (e.g. Surface and Touchwall from Microsoft) and in the market of mobile devices: Apple, LG and Nokia, to cite only a few examples, finally chose a touch-sensible display for their leading products, while the interest for this technology is growing also for design studios, industrial environments and public information points like museum kiosks and ATMs. Unfortunately, touchscreen flexibility is low: finger tracking is impossible without physical contact; it is not possible to use sharp objects on them; large touch-sensible displays are expensive because of their manufacturing cost and damage-proneness.

In [FR08] we presented a low-cost 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. That paper focused its attention mostly on the mapping algorithm and provided only a description of an early stage of the system reported here. In this paper, instead, we introduce a more efficient and mature system, exploiting an improved pointer detection but computationally and economically cheap as the previous one. Among the improvements we made:

- two proximity constraints in the pointer detection help to reduce the number of false positives;
- a convolution-based algorithm is used to locate the presence of a pointer;
- the gap from the reference backgrounds is kept under control to detect camera movements;
- the calibration phase is faster, and the system graphically shows the points to touch;
- iterative algorithms are used to solve the linear systems instead of direct formulas.

2 RELATED WORK

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 recent trend. There is a growing interest in the ones that involve real-time body tracking, especially if no expensive hardware is required and the user does not need to wear any special equipment. The simplest and cheapest choice is to use optical devices to track a specific part of the body (head, eyes, hands or even the nose [GMR02]); we focus on finger tracking systems

that do not require lasers, markers, gloves or hand-held devices [SP98, DUS01, Lee07].

The main application of finger tracking is to move a digital pointer over a screen, enabling the user to replace the pointing device (e.g. the mouse) with his hands. While for eye or head tracking we have to direct the camera(s) towards the users's body, finger tracking let us a wider range of choices.

The first possibility is to direct the camera towards the user's body, as for head tracking, and to translate the absolute or relative position of the user's finger to screen coordinates. In [WSL00] an empty background is needed; in [IVV01] the whole arm position is reconstructed, and in [Jen99] a combination of depth and color analysis helps to robustly locate the finger. Some works tried to estimate the position of the fingertip relatively to the view frustum of the user; this was done in [CT06] with one camera and in [pHYssCIb98] with stereovision, but both had strong limits in the accuracy of the estimation.

The second possibility is to direct the camera towards the pointing surface, which may be static or dynamic. Some works require a simple black pad as pointing surface, making it easy to locate the user's finger with only one camera [LB04]; however, we may need additional hardware [Mos06] or stereovision [ML04] to distinguish if the user is just hovering the finger on it or if there is a physical contact between the finger and the surface. A physical desktop is an interesting surface to track a pointer on. Some works are based on the *DigitalDesk* setup [Wel93], where a overhead projector and one or more cameras are directed downwards on a desk and virtual objects can interact with physical documents [Ber03, Wil05]; others use a similar approach to integrate physical and virtual drawings on vertical or horizontal whiteboards [Wil05, vHB01, ST05], and one integrates visual informations with an acoustic triangulation to achieve better accuracy [GOSC00]. These works use differencing algorithms to segment the user's hands from the background, and then shape analysis or finger templates matching to locate the fingertips; they rely on the assumption that the background surface is white, or in general of a color different than skin. Other approaches work also on highly dynamic surfaces. It is possible to robustly suppress the background by analyzing the screen colorspace [Zha03] or by applying polarizing filters to the cameras [AA07]; in the first the mouse click has to be simulated with a keystroke, while in the latter a sophisticated mathematical finger model allow to detect the physical contact with stereovision. Unfortunately, these two techniques cannot be applied to a projected wall. Directing the camera towards the pointing surface implies, in general, the use of computationally expensive algorithms, especially when we have to deal with dynamic surfaces.

A third possible approach, which may drastically reduce the above problems, is to have the cameras watching sidewise - i.e. laying on the same plane of the surface; using this point of view we do not have any problem with dynamic backgrounds both behind the user or on the pointing surface, and this enables us to set up the system also in environments otherwise problematic (e.g. large displays, outdoor, and so on). Among the very few works using this approach, in [QMZ95] the webcam is on the top of 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. Also in [WC05] we find a camera on the top of a laptop display directed towards the keyboard, but the mouse pointer is moved accordingly to the motion vectors detected in the grayscale video flow; a capacitive touch sensor enables and disables the tracking, while the mouse button has to be pressed with the other hand. In [Mor05], finally, the "lateral" approach is used to embed four *smart cameras* into a plastic frame that is possible to overlap on a traditional display.

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 is possible to use smart region finding algorithms, these would not resolve the problem entirely. In [FR08] we proposed a different way to track user movements keeping the complexity low. We drastically decreased the scanning area to a discrete number of pixel lines of two uncalibrated cameras. Our system requires a simple calibration phase that is easy to perform also for non-experienced users.

The proposed technique only regards the tracking of a pointer, and it is not about gesture recognition. The output of the system, at present, is directly translated into mouse movements, but may be instead interpreted by a gesture recognition software.

3 SYSTEM DESCRIPTION

We propose to use two off-the-shelf 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 clicks; 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 opaque object 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.

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 to grab a frame before any pointer enters in the tracking area; these reference frames (one per webcam) will be stored as *reference backgrounds*, and will be used to look for runs of consecutive pixels different from the reference background. We will see later how we detect such *scanline interruptions* (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 scanline is occluded in at least one of the two views.

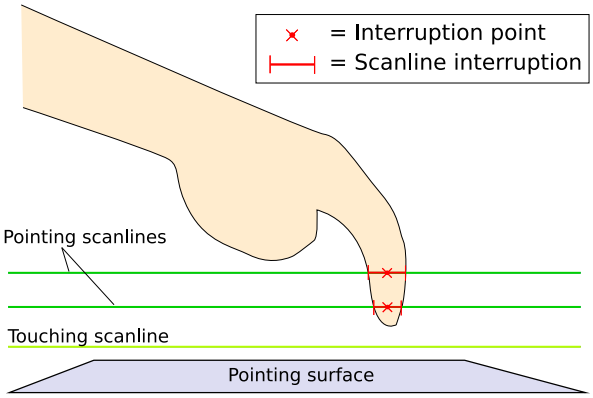

Figure 1: Visual representation of scanlines within the view field of each camera.

During the calibration phase the number of scanlines of interest may vary from a couple to tens; during the tracking, three or four scanlines will suffice for an excellent accuracy. A detailed description of the calibration will be given later.

Noise reduction

We detect the presence of a physical pointer in the view frustum of a webcam by comparing the current frame with the reference background. This is simple in absence of noise; unfortunately, the video flow captured from a CMOS sensor (the most common type of sensor in low cost video devices) is definitely not ideal and presents a bias of white noise, salt and pepper noise and motion jpeg artifacts. This makes pointer detection more difficult, especially when the pointer is not very close to the camera and its silhouette is therefore only a few pixels wide. To keep the overall complexity low we avoid to apply any post-elaboration filter on each of the grabbed frames and adopt two simple strategies in order to reduce the impact of noise on our algorithm.

The first strategy is to store, as a reference background, not just the first frame but the average of the first b frames captured (in current implementation, $b = 4$). The average root mean square deviation of a frame from the reference background, after this simple operation, decreases from ~1.52 to ~1.26 (about -17%).

The second strategy is to apply a simple convolution to the scanlines we focus on. The matrix we use is

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

with divisor 3. This is equivalent to say that we replace each pixel with the average of a 1 pixel neighborhood on the same row; it is not worth increasing the neighborhood of interest because by increasing it we decrease the tracking accuracy.

Finally, we keep track of the Root Mean Square Error (RMSE) with respect to the reference frames; if the RMSE gets higher than a threshold, this is probably due to a disturbing entity in the video or to a movement of the camera rather than to systematic noise. In this case, the system automatically stops tracking and informs the user that a new reference background is about to be grabbed.

Fast pointer detection

Although some noise has been reduced, we cannot rely only a binary differencing algorithm. A set of pixels different from the reference frame is *meaningful* if they are *close* to each other; we apply this *spatial contiguity* principle both horizontally and vertically. This approach imitates the so called Helmholtz principle[1] for human perception.

The first goal is to find a run of consecutive pixels significantly different from the reference; what we care is the X coordinate of the center of such interruption. We initialize to zero a buffer of the same size of one row, then we start scanning the selected line (say l). For each pixel $p = (p_x, p_l)$, we compute the absolute difference δ_p from the correspondent reference value; then, for each pixel $q = (q_x, q_l)$ in a neighborhood long n, we add this δ_p multiplied by a factor m inversely proportional to $|p_x - q_x|$. Finally we read in the buffer a peak

[1] The Helmholtz principle states that an observed geometric structure is perceptually *meaningful* if its number of occurrences would be very small in a random situation (see [MmM01]).

 ISBN 978-80-86943-93-0

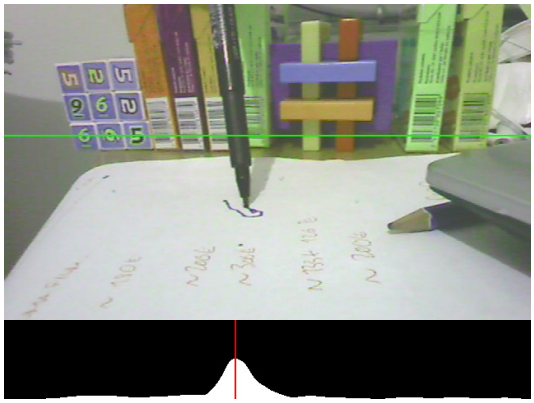

Figure 2: The buffer used for the analysis of the green row shows a clear peak

value correspondent to the X coordinate of the center of the interruption (fig. 2); if no interruption occurred in the row (i.e. pixels different from the reference were not close to each other), we will have only "low" peaks in the buffer. To distinguish between a "high" and a "low" peak we can use a fixed or a relative threshold; in our tests, a safe threshold was about 20 times greater than the neighborhood length.

Now we have a horizontal proximity check, but not a vertical one yet. As section 3 explains, each webcam sees the pointer always breaking into the view frustum by the upper side. The pointer silhouette may be straight (like a stick) or curved (e.g. a finger); in both cases, the interruptions found on scanlines close to each other should not differ more than a given threshold. This vertical proximity constraint gives a linear upper bound to the curvature of the pointer, and helps discarding interruptions caused by noise or other objects entering in the view frustum; in other words, the system detects only pointers coming from above, and keeps working correctly if other objects appear in the view frustum from a different direction (e.g. the black pen in fig. 3).

Figure 3: The system correctly detects only the pointer coming from above.

These two simple *proximity checks* make the recognition of the pointer an easier task. Fig.4 shows the correct detection of the pointer (a hand holding a pen) over a challenging background. The lower end of the vertical sequence of interruptions is marked with a little red cross.

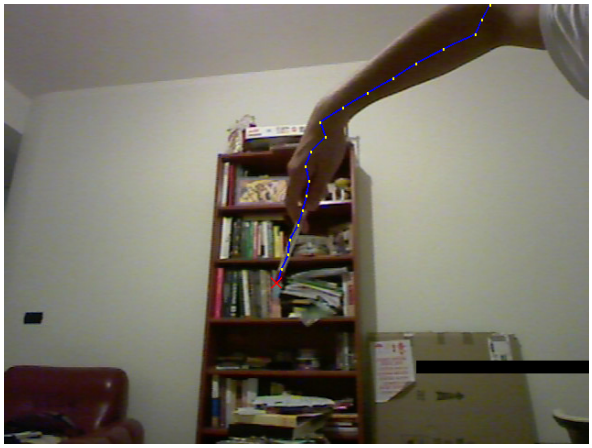

Figure 4: The vertical contiguity constraint of a hand holding a pen.

Positioning the cameras

The proposed technique requires the positioning of two webcams relatively to the pointing surface. The simplest choice is to 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.

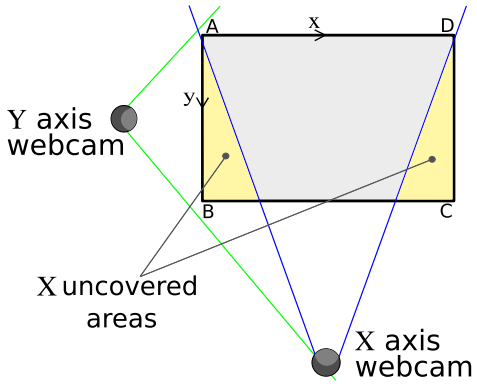

Figure 5: Example of a simple but inefficient configuration.

In figure 5, for example, the webcam along Y axis of the surface has a wide view field, but this brings resolution loss on segment $\overline{DC}$; on the other side, the webcam

along X axis of the surface has a narrow view field, but it has to be positioned far from the pointing surface to cover the whole area. If the surface is a 2×1.5 m projected wall and the webcam has a 45° view field, we have to put the camera ~5.2 meters away to catch the whole horizontal size.

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 6). 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.

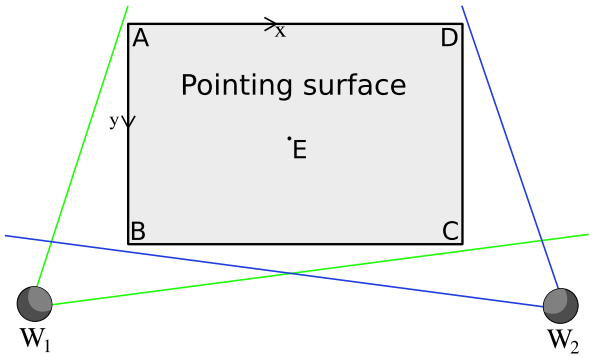

Figure 6: Suggested configuration to optimize the use of view frustum of the cameras.

In the rest of this paper we will assume, for the sake of clarity, that the webcams are in the same locations and orientations as in figure 6. However, the proposed tracking algorithm works with a variety of configurations without changes in the calibration phase: the cameras may be positioned anywhere around the surface, and we only need that they do not face each other.

Calibration phase

When the system is loaded, the calibration phase starts. In this phase, after grabbing the reference backgrounds, we ask the user to touch the vertices of the pointing surface and its center. When a pointer is detected in both views, we track the position of its lower end (the red cross in fig. 4 and 3); if this position holds with a low variance for a couple of seconds, the correspondent X coordinate is stored. After we grabbed the position of all the five points, we compute the Y coordinate of a "special" scanline as the lowest row not intercepting the pointing surface: during the tracking we will focus only on this row to grab the position of the pointer, so that the overall complexity will be linear with the horizontal resolution.

Tracking algorithm

During the calibration phase we stored the X coordinate of each vertex as seen by the webcams. The basic idea is to calculate the perspective transformation that translates the absolute screen coordinates to absolute coordinates in the viewed image. We store vertices in homogeneous coordinates and use a $3x3$ transformation matrix M:

$$\begin{pmatrix} l_{11} & l_{12} & l_{13} \\ l_{21} & l_{22} & l_{23} \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \cdot V = P \cdot \alpha$$

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 $l_{33} = 1$. 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 l_{21},l_{22},l_{23} of M and we only have to determine the values of l_{11},l_{12},l_{13},l_{31},l_{32}.

The number of unknown matrix elements has been decreased to five, so we only need the x coordinate of five points (instead of the x and y of four 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.6) 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 the 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 to solve is:

$$\begin{pmatrix} 0 & H & 0 & -HB_{xp} \\ W & H & -WC_{xp} & -HC_{xp} \\ W & 0 & -WD_{xp} & 0 \\ E_x & E_y & -E_xE_{xp} & -E_yE_{xp} \end{pmatrix} \cdot \begin{pmatrix} l_{11} \\ l_{12} \\ l_{31} \\ l_{32} \end{pmatrix} = \begin{pmatrix} B_{xp} - A_{xp} \\ C_{xp} - A_{xp} \\ D_{xp} - A_{xp} \\ E_{xp} - A_{xp} \end{pmatrix}$$

which makes easy to obtain l_{11}, l_{12}, l_{13}, l_{31}, l_{32} for each camera.

During the tracking phase we face a somehow inverse problem: 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 l_{ij} be the transformation values for the first camera, and r_{ij} for the second one; the linear system we have to solve in this case is

$$\begin{cases} l_{11}x_l + l_{12}y_l + l_{13}z_l & = X_l \\ l_{31}x_l + l_{32}y_l + z_l & = 1 \\ r_{11}x_r + r_{12}y_r + r_{13}z_r & = X_r \\ r_{31}x_r + r_{32}y_r + z_r & = 1 \end{cases}$$

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

$$\begin{pmatrix} l_{11} & l_{12} & -X_l & 0 \\ l_{31} & l_{32} & -1 & 0 \\ r_{11} & r_{12} & 0 & -X_r \\ r_{31} & r_{32} & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z'_l \\ z'_r \end{pmatrix} = \begin{pmatrix} -l_{13} \\ -1 \\ -r_{13} \\ -1 \end{pmatrix}$$

This is a determined linear system, and it is possible to prove that in the setting above there is always one and only one solution. By 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. In the previous version of the system direct formulas were used, while now a LU factorization is implemented.

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.7). We 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.

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. We define *resolution accuracy of* W_1 *in* t and we call $\sigma(W_1, t)$ this ratio. It is clear that we only care about the horizontal resolution of W_1, which is constant in the whole view frustum of the camera (fig. 7)

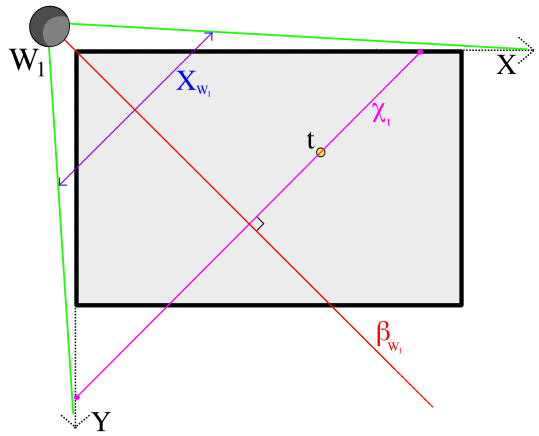

Figure 7: We define "resolution accuracy of W_1 in t" the ratio between the length of χ_t and the number of pixels detected by W_1.

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 interesting applications of the system is to projected walls, so that they become virtual blackboards. A very common projector resolution 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 X 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 is immediate that, in the above configuration, the average resolution accuracy is higher than 1:1 (sub-pixel).

Algorithm complexity

The number of scanlines is constant and in the tracking phase it is not useful to use more than 3 or 4 of them. For each scanline we do a noise reduction (in linear time), we apply a linear convolution filter (in linear time too) and then we do a linear search for a peak. Finally, we solve the system (in constant time). The total complexity is therefore linear with the horizontal resolution of the webcams.

4 EXPERIMENTAL SETTINGS AND SYSTEM PERFORMANCE

The webcams we used for testing are two Philips SPC1000NC, with a native SXGA video sensor; their 2008 price has been of about 40€each, and they are capable of producing a SXGA video at about 15fps, and a VGA one at 60fps. There is a mature Video4Linux2 compliant driver (uvcvideo) available for GNU/Linux.

Our prototype has good resolution accuracy and excellent time performances: less than 10 milliseconds are needed to elaborate a new frame and compute the pointer coordinates. 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 (less than 4000 lines) all software layers are strictly separated, so that it is possible to port the whole system to different platforms with very little changes in the source.

A demonstrational video is available for download at `http://svg.dmi.unict.it/iplab/download/FingerTracking/`. At the same URL is available a demonstrative video of a previous version of the presented system.

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 requires a simple calibration phase.

Future works will be devoted to improve the robustness of the calibration and the pointer-detection subsystems; moreover, suitable evaluation procedures to test the empirical accuracy of tracking will be addressed. Adding multitouch support will also be considered.

ACKNOWLEDGEMENTS

Thanks to prof. G.Gallo for the invaluable guidance offered during this research and dr. G.M.Farinella for his support and assistance in the set up of the previous version of the system. We also thank `it.scienza.matematica` newsgroup for their precious tips.

REFERENCES

[AA07] Chandraker M. Blake A. Agarwal A., Shahram Izadi S. High precision multitouch sensing on surfaces using overhead cameras. In *Horizontal Interactive Human-Computer Systems, 2007. TABLETOP '07. Second Annual IEEE International Workshop on*, pages 197–200, 2007.

[Ber03] F. Berard. The magic table: Computer vision based augmentation of a whiteboard for creative meetings. *IEEE International Conference in Computer Vision*, 2003.

[CT06] Kelvin Cheng and Masahiro Takatsuka. 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*, pages 397–400, New York, NY, USA, 2006. ACM.

[DUS01] Klaus Dorfmüller-Ulhaas and Dieter Schmalstieg. Finger tracking for interaction in augmented environments. *Augmented Reality, International Symposium on*, 0:55, 2001.

[FR08] G.M. Farinella and E. Rustico. Low cost finger tracking on flat surfaces. In *Eurographics Italian chapter 2008*, 2008.

[GMR02] D. Gorodnichy, S. Malik, and G. Roth. Nouse 'use your nose as a mouse' - a new technology for hands-free games and interfaces, 2002.

[GOSC00] Christophe Le Gal, Ali Erdem Ozcan, Karl Schwerdt, and James L. Crowley. A sound magicboard. In *ICMI '00: Proceedings of the Third International Conference on Advances in Multimodal*

Interfaces, pages 65–71, London, UK, 2000. Springer-Verlag.

[IVV01] Giancarlo Iannizzotto, Massimo Villari, and Lorenzo Vita. 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*, pages 1–7, New York, NY, USA, 2001. ACM.

[Jen99] Cullen Jennings. Robust finger tracking with multiple cameras. In *In Proc. of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, pages 152–160, 1999.

[LB04] Julien Letessier and François Bérard. Visual tracking of bare fingers for interactive surfaces. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 119–122, New York, NY, USA, 2004. ACM.

[Lee07] Johnny Chung Lee. Head tracking for desktop VR displays using the Wii remote. `//www.cs.cmu.edu/~johnny/projects/wii/`. 2007.

[ML04] Shahzad Malik and Joe Laszlo. Visual touchpad: a two-handed gestural input device. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 289–296, New York, NY, USA, 2004. ACM.

[MmM01] Lionel Moisanm and Jean michel Morel. Edge detection by helmholtz principle. *Journal of Mathematical Imaging and Vision*, 14:271–284, 2001.

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

[Mos06] Tomer Moscovich. Multi-finger cursor techniques. In *In GI '06: Proceedings of the 2006 conference on Graphics interface*, pages 1–7, 2006.

[pHYssCIb98] Yi ping Hung, Yang Yao-strong, Yong sheng Chen, and Hsieh Ing-bor. Free-hand pointer by use of an active stereo vision system. In *in Proc. 14th Int. Conf. Pattern Recognition*, pages 1244–1246, 1998.

[QMZ95] F. Quek, T. Mysliwiec, and M. Zhao. Fingermouse: A freehand computer pointing interface, 1995.

[SP98] Joshua Strickon and Joseph Paradiso. Tracking hands above large interactive surfaces with a low-cost scanning laser rangefinder. In *Proceedings of CHI'98*, pages 231–232. Press, 1998.

[ST05] Le Song and Masahiro Takatsuka. Real-time 3d finger pointing for an augmented desk. In *AUIC '05: Proceedings of the Sixth Australasian conference on User interface*, pages 99–108, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[vHB01] Christian von Hardenberg and François Bérard. Bare-hand human-computer interaction. In *PUI '01: Proceedings of the 2001 workshop on Perceptive user interfaces*, pages 1–8, New York, NY, USA, 2001. ACM.

[WC05] Andrew D. Wilson and Edward Cutrell. Flowmouse: A computer vision-based pointing and gesture input device. In *In Interact '05*, 2005.

[Wel93] Pierre Wellner. Interacting with paper on the digitaldesk. *Communications of the ACM*, 36:87–96, 1993.

[Wil05] Andrew D. Wilson. Playanywhere: a compact interactive tabletop projection-vision system. In Patrick Baudisch, Mary Czerwinski, and Dan R. Olsen, editors, *UIST*, pages 83–92. ACM, 2005.

[WSL00] Andrew Wu, Mubarak Shah, and N. Da Vitoria Lobo. A virtual 3d blackboard: 3d finger tracking using a single camera. In *In Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 536–543, 2000.

[Zha03] Zhengyou Zhang. Vision-based interaction with fingers and papers. In *Proc. International Symposium on the CREST Digital Archiving Project*, pages 83–106, 2003.