

Hand tracking for human-computer interaction with *Graylevel VisualGlove*: Turning back to the simple way

Giancarlo Iannizzotto
Department of Mathematics
University of Messina
C.da Papardo, Salita Sperone,
98166, Messina - Italy.

ianni@ingegneria.unime.it

Massimo Villari
Department of Mathematics
University of Messina
C.da Papardo, Salita Sperone,
98166, Messina - Italy.

mvillari@ingegneria.unime.it

Lorenzo Vita
DIIT
University of Catania
Viale A. Doria 6
95025 Catania - Italy

vita@diit.unict.it

ABSTRACT

Recent developments in the manufacturing and marketing of low power-consumption computers, small enough to be “worn” by users and remain almost invisible, have reintroduced the problem of overcoming the outdated paradigm of human-computer interaction based on use of a keyboard and a mouse. Approaches based on visual tracking seem to be the most promising, as they do not require any additional devices (gloves, etc.) and can be implemented with off-the-shelf devices such as webcams. Unfortunately, extremely variable lighting conditions and the high degree of computational complexity of most of the algorithms available make these techniques hard to use in systems where CPU power consumption is a major issue (e.g. wearable computers) and in situations where lighting conditions are critical (outdoors, in the dark, etc.). This paper describes the work carried out at VisiLAB at the University of Messina as part of the VisualGlove Project to develop a real-time, vision-based device able to operate as a substitute for the mouse and other similar input devices. It is able to operate in a wide range of lighting conditions, using a low-cost webcam and running on an entry-level PC. As explained in detail below, particular care has been taken to reduce computational complexity, in the attempt to reduce the amount of resources needed for the whole system to work.

1. INTRODUCTION

The use of real-time computer vision techniques to implement devices for human-computer interaction (HCI) as an alternatives to the customary keyboard-mouse paradigm has aroused wide interest throughout the last decade. In this situation, it appears that *Wearable Computers* [15] [16], currently the state of the art in portable computers, on which the most vital and technologically advanced research is cur-

rently focusing, are still going to base user input on speech recognition (see a recent IBM advertising campaign) or even tiny, wireless joysticks and small wrist-mounted keyboards [5]. Experience with palmtops confirms that in most cases user-computer interaction does not require a keyboard: all that is needed is a stylus, the movements of which on a surface (in this case, the palmtop screen) can be followed with sufficient accuracy to reconstruct the drawn shape. With the exception of word processing, in which speech processing is as efficient and reliable an input system as a keyboard, for most applications a device to replace the mouse is quite sufficient to meet all user-computer interaction requirements. The market availability of tiny, almost invisible display systems that transform an ordinary pair of spectacles into an efficient, comfortable see-through display [5] suggests the use of small cameras mounted on the glasses to detect and track the movement of at least one of the user’s hands, thus allowing him to interact with the processor without the need for other pointing or keying in devices.

Most research into visual gesture recognition focuses on recognising the high-level, complex gestures that humans use in real life to communicate with each other. This is an ambitious aim and although considerable results have been achieved in this direction (see, for example, the reviews: [4] [9] [10] and among other papers: [7], [14]) it is not the aim of this paper. The computing power required to recognise, in real-time, complex gestures by no means corresponds to that of the CPUs currently available on portable systems, and the expressive potential of human gestures is by far greater than that needed to implement a pointer device. We therefore decided to investigate a vision system which, making use of very stable and low-complexity algorithms, could track the hand (actually, just two fingers) of the user, in order to recognize some simple and intuitive gestures thus substituting the mouse in almost any user operation. To be able to deal with a wide range of illumination conditions, we developed the Graylevel VisualGlove system, which we present in this paper. The following two sections give a detailed description of our approach and present some experimental results. The final section draws our conclusions and outlines further research developments.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PUI 2001 Orlando FL, USA

Copyright 2001 ACM 1-58113-448-7-11/14/01 ...\$5.00.

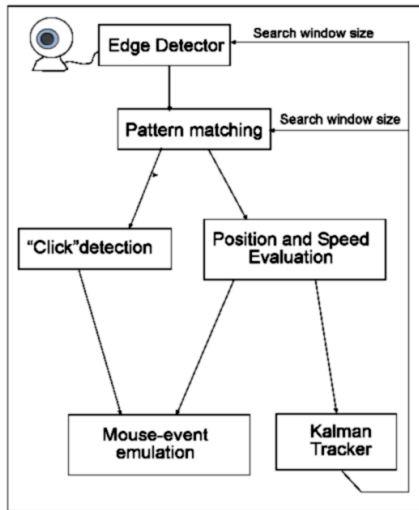


Figure 1: A general scheme of GrayLevel VisualGlove

2. THE ARCHITECTURE OF GRAYLEVEL VISUALGLOVE

The VisualGlove project focuses on both solutions that use colour to detect the user’s hand in the scene and colour-independent techniques (gray-level based). In this paper we will focus on a gray-level approach as it can be used in a wider range of environmental situations including almost total darkness, thanks to the use of a small, economical infrared illuminator and an ordinary infrared filter like those used in television remote control receivers. The Graylevel VisualGlove architecture is illustrated in Fig. 1.

The typical usage situation should feature the user wearing a lightweight, head-mounted, see-through display: a small camera should be head-mounted too, in order to image approximately the same scene seen by the user. The user would then interact with the system in an extremely intuitive way, by using only two fingers. Unlike most systems described in the literature, indeed, Graylevel VisualGlove does not track the operator’s whole hand: it only tracks the fingertips of the thumb and index finger positioned (see Fig. 2) as if the user wanted to clasp something between the two fingers.

Given the lack of a wearable computer prototype in our lab, during the experimental tests we used a webcam connected to a PC to test the system (a more detailed description of the testbed is given in Section 3). In Fig. 2 a webcam is clearly visible over the monitor. Given the generality of our approach, the results shown in this paper will be directly applied to a wearable computer as soon as one will be available.

The two fingertips are tracked separately but in a co-operative fashion, in such a way that the information from one of the two trackers combines with and validates the information from the other tracker. This considerably reduces the effect of background noise, increasing the probability of correct detection and thus the reliability of the system (see Section 3 for the experimental results). Contact between the two fingertips corresponds to the “mouse.click” event, which can easily become “double.click” via fast repetition



Figure 2: An user testing GrayLevel VisualGlove

of the joining of the fingertips. By keeping the index finger and thumb joined and moving the hand at the same time it is quite simple to simulate a “drag” or “draw” operation, while by separating the two fingers after a “drag” operation it is possible to simulate “drag-and-drop”. The user gestures detected by VisualGlove are then translated into messages and inserted into the mouse event queue, thus simulating, for application purposes, the presence of an ordinary mouse.

Graylevel VisualGlove is essentially based on the use of a stable, robust pattern matching algorithm applied to an edge-ness image obtained in real time from each frame. Thanks to the use of a Kalman Tracker [13] the position, orientation and size of the fingertips are predicted in each frame, thus drastically reducing the search area for pattern matching. A scheme of the algorithm is given in Fig. 3.

```

Init:   repeat
        Check for the presence of both
        fingertips in the predefined area
      until
        Both patterns are detected;

Track:  while
        the validation gate is not crossed
      do {
        Predict new status vector;
        Measure new status vector;
        Apply predict/assimilate (see fig.6);
        Produce a mouse.event;
      }

Loop:   goto Init
  
```

Figure 3: Main algorithm outline

The *initialisation* phase (Init) is first activated when VisualGlove is executed and then only if the Validation Gate threshold is crossed. In this case the tracker is no longer able to track the target reliably and needs to be re-initialised. Initialisation consists of applying the pattern matching algorithm in a predefined area of the image (the top right hand corner for right-handed users and the top left hand

corner for left-handed users), i.e. where the user is assumed initially to place his fingertips. The initial positioning of the fingertips in a predefined place corresponds to a precise operational requirement: just as an ordinary desktop PC user does not constantly hold the mouse but only uses it when he wants to interact with the pointer system, so is it necessary to identify a “protocol” to start up a session of interaction with VisualGlove or to interrupt one. It is not, in fact, practical for the user’s gestures to be tracked constantly, with the risk of activating undesired operations by mistake. The interaction session is interrupted simply by clenching the fist so as to remove the two fingertips from the scene. The protocol recalls a simple, intuitive metaphor, i.e. the act of “wearing” a virtual glove (i.e. VisualGlove) only for as long as is needed to interact with the computer.

The binary image to which the pattern matching algorithm is applied is obtained by applying the Canny Edge Detector [3] to each frame, only in the portion of the image detected, instant by instant, by the Kalman Tracker. As this portion is very small in size (generally no more than 60×100 pixel), the real-time application of the algorithm is not excessively onerous for the system.

The pattern matching algorithm used is a very simple version of the Huttenlocker, Klanderma and Rucklidge matching algorithm based on the Hausdorff Distance, originally described in [8] and subsequently optimised in [11]. At this stage in our research we have not made a particular effort to search for the pattern matching algorithm that would give the best performance but preferred to choose an approach that was sure to be robust and had been dealt with in detail in the literature.

The **Hausdorff Distance** between two binary images (i.e. ones whose points can only take values of 0 or 1) A and B is defined as follows:

$$H(A, B) = \max(h(A, B), h(B, A)) \quad (1)$$

where

$$h(A, B) = \max_{\substack{a \in A \\ a \neq 0}} \min_{\substack{b \in B \\ b \neq 0}} \|a - b\| \quad (2)$$

and the symbol $\|\cdot\|$ represents a norm L_p , usually the norm L_2 . The function $h(A, B)$ is said to be the **Directed Hausdorff Distance** from A to B .

To simplify implementation, it may be useful to define the Hausdorff Distance as a function of the **Distance Transform** of a binary image. If A is a binary image we first define a distance $D(P_1, P_2)$ between any two of its points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$. By way of example, we can consider the Euclidean distance:

$$D(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (3)$$

We then define the Distance Transform of the image A as follows: for each point $P_i \in A$, the Distance Transform in it is equal to the distance from its “nearest non-zero neighbour”. This is expressed by the formula:

$$DT(P_i) = \min_{\substack{P_j \in A \\ P_j \neq 0}} (D(P_i, P_j)) \quad (4)$$

Note that $P_i \neq 0 \Rightarrow DT(P_i) = 0$.

Let us now assume that we want to compare two binary images, A and B . We will assume that they are of the same size. This time we define the Directed Hausdorff Distance

from A to B as follows: for each non-zero point of A we calculate the value of the Distance Transform of B at the point in B with the same coordinates and save this value. We then pick the maximum value collected. This number is the value of the Directed Hausdorff Distance from A to B , expressed as:

$$h(A, B) = \max_{A(x,y) \neq 0} (DT(B(x, y))) \quad (5)$$

The Undirected Hausdorff Distance, briefly known as the Hausdorff Distance, can thus be obtained by substituting (5) in (1).

The properties of this distance are described in literature cited above: it is possible to construct efficient pattern matching algorithms by exploiting its numerous characteristics. Of particular significance are the extensions of the basic algorithm to the case in which one of the two images being compared is smaller in size than the other (the aim thus becoming that of finding all the occurrences of the smaller image (mask) in the larger one) and the case in which the aim is *partial matching*, i.e. presenting an occurrence of only a part of the mask in the image (e.g. on account of occlusions or imperfections in the mask) [11]. For our purposes we used both extensions. It is recalled that the image being considered is in reality a sub-image B the size and position of which with respect to the whole image are as predefined in the case of initialisation or those supplied by the Kalman tracker during tracking. By scaling, rotating and translating the mask with respect to B it is possible to determine all the occurrences of the mask in the image, at the same time assigning each of them a value indicating the degree of confidence of the occurrence (see scheme in fig.4). The occurrence with the highest degree of confidence (i.e. with the lowest Hausdorff distance) is chosen as a possible detected object, but will only be considered as detected if the Hausdorff distance is below a certain threshold, $MaxDist_j$, experimentally set for each of the two patterns.

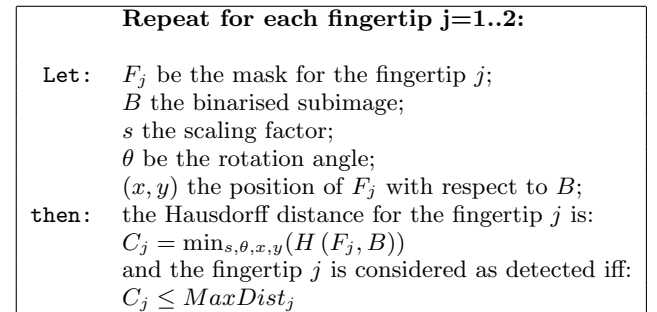


Figure 4: Detection algorithm outline

In our case, therefore, we have two masks, each of which is a binary image whose initial size is 10×20 pixel and which reproduces the pattern corresponding to the edges of only one of the two fingertips. It should be pointed out that it is not necessary to consider all the possible rotations and zooming of the mask: on the basis of biometric and ergonomic considerations, we set a maximum of 20 rotations with respect to the horizontal position (10 positive values and 10 negative ones), at a distance of about 4 degrees from each other. Likewise, we fixed a maximum of 5 possible scaling values. Each rotation and scaling value corresponds to a

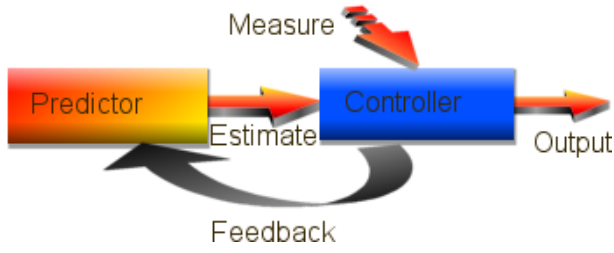


Figure 5: The prediction-measure-assimilation scheme

mask calculated offline: this obviates the need to recalculate the masks in each frame.

Detection of the fingertips is joint: only if both fingertips have been found does the procedure give a positive result. When the pair of patterns has been detected, the information about its position, orientation and size is passed to the tracker module. As shown in fig.5, this module comprises an estimator, a controller and a measure module connected in the conventional closed-loop fashion commonly adopted for visual object tracking [1]. At each frame the Kalman Tracker, on the basis of the previous observations (measures), produces an estimate of the new status of the fingertips, the accuracy of which tends to improve at each iteration (in the ideal case, the error tends to zero) thanks to the information provided by each new measurement.

Let us now define the *status vector* representing the status of the system to be tracked. Although there are two fingertips, and they feature a different orientation and different movements that may cause variations in the distance between them, it is still possible to identify a common scaling factor for the two patterns, taking as a reasonable hypothesis that the two fingers will approach or depart from the camera together. All the other parameters have to be tracked separately: the co-ordinates of each fingertip (to be more precise, the centre of mass of the pattern), their angles of orientation (one per fingertip), and the Boolean value indicating whether a “click” is being made, i.e. whether the two fingertips are currently in contact with each other. Finally, it is necessary to take into account the values taken at the previous time instant by the position of the two fingertips, their orientation and the common scaling factor. The status vector should therefore have a total of 15 elements. For reasons of symmetry and uniformity, we decided to separate the Boolean variable from the status vector. As expressed by the equation 6, it therefore comprises 7 independent variables considered in the two (consecutive) time instants i and $i - 1$.

$$X_i = \begin{pmatrix} x_{i-1}^1 & y_{i-1}^1 & x_{i-1}^2 & y_{i-1}^2 & \theta_{i-1}^1 & \theta_{i-1}^2 & f_{i-1} \\ x_i^1 & y_i^1 & x_i^2 & y_i^2 & \theta_i^1 & \theta_i^2 & f_i \end{pmatrix} \quad (6)$$

In eqn.(6), (x_i^1, y_i^1) and (x_i^2, y_i^2) are the positions of the two fingertips (in image co-ordinates); θ_i^1 and θ_i^2 are the orientations of the two patterns and f_i is the common scaling factor, i.e. the relative size with respect to the size returned by the initialisation step.

The Prediction-Assimilation algorithm is outlined in Figure 6: Z is the vector of our measures, so it has the same composition as X_i in equation (6). The matrix G_i represents

the linear relation between the measure and the status: in our case, $G_i = I$ (I is the Identity matrix). w_i and v_i represent the noise associated with the status and the observation process. We assume that they both have a Gaussian probability distribution, zero mean and variances, respectively: B_i and R_i . The variance of X_i is P_i . The model adopted for prediction is a linear, second-order Autoregressive Process (AR), and its parameters were determined experimentally.

Prediction-Assimilation paradigm

$$\begin{aligned} \tilde{X}_i &= A_{i-1}X_{i-1} + w_{i-1} && \text{Prediction} \\ Z_{i-1} &= G_{i-1}X_{i-1} + v_{i-1} && \text{Observation model} \\ w_i \text{ and } v_i &\text{ have zero mean} && \\ &\text{and variances: } B_i \text{ and } R_i && \\ X_i &\text{ has a variance of: } P_i && \\ \tilde{P}_i &= AP_{i-1}A^T + BB^T && \text{Riccati eqn.} \\ K_i &= \tilde{P}_{i-1}G_i^T \left(G_i\tilde{P}_iG_i^T + R_i \right)^{-1} && \text{Kalman Gain} \\ \hat{X}_i &= \tilde{X}_i + K_i \left(Z_i - G_i\tilde{X}_i \right) && \text{Assimilation} \\ P_i &= (I - K_iG_i)\tilde{P}_i && \end{aligned}$$

Figure 6: Prediction - Assimilation algorithm

The performance of the Kalman controller [13] described above is closely related to the hypothesis that both the noise vectors and the status vector have a Gaussian distribution. At this stage we will not address this issue, since the performance of the Kalman tracker is reasonable for our purposes; several different solutions do, however, exist for this problem [1].

The measure module basically uses the same detection algorithm as is used for the initialisation phase, described in Fig.4. In this case, however, it is applied to a rectangular area B for each fingertip, centred in the position predicted by the tracker as shown in Fig.6 and of a size calculated on the basis of the current scaling factor, increased in accordance with the variance currently associated with the status vector. We consider, in fact, the vector V_i , comprising only 4 elements of the status vector X_i : the co-ordinates of the centres of the two fingertips. If we use U_i to indicate the variance matrix relating to the vector V_i , the exact increase in the size of the two rectangles (Regions Of Interest: ROIs) is determined according to the norm $\|\tilde{U}_i\|$ of the status variance U_i , as predicted by the algorithm in Figure 6. We exploited the following formula:

$$S_i^2 = \kappa \cdot \text{tr} \left(\tilde{U}_i \right) \quad (7)$$

where κ was determined experimentally and the norm of the matrix \tilde{U}_i was calculated as its **trace**. He same approach was used to bound the number of rotations and the zooming of the mask during the matching phase. We note once again that detection of the two fingertips is collaborative, in the sense that only simultaneous detection of both fingertips produces a positive response to the detection function.

Calculation of the variance R associated with the measure Z requires evaluation, frame by frame, of the measurement error. There are several sources of error and not all of them are Gaussian; background texture and a cluttered background in particular can introduce noise, the distribution of which is basically unknown. Similar problems may be

caused by bad lighting conditions, which may generate shadows and areas of excessive intensity. Finally, interference with artificial light sources (e.g. fluorescent lamps) may generate spotty or anyway non-Gaussian noise. Camera non-linearity or that introduced by the software driver may add further uncertainty to the observation. In this situation we opted, mainly on the basis of experimental observation, to use a “reference model” called a “Median Model”, obtained from median filtering of a sufficient number of consecutive observations, with respect to which the variance was calculated frame by frame. The Median Model M_i is obtained by applying to the measure vector Z_i a median filter with a width of m (see equation 8).

$$M_i[j] = \underset{\nu=i-m+1\dots i}{median} (Z_\nu[j]) \quad (8)$$

The median filter [6] is a non-linear filter commonly used to reduce the effect of spotty noise in digital signals. Given a 1-D signal represented by its sample vector V of length n , a window of size m is set to slide on V , centring on each of the elements of V , so that for every element of V , a “neighbourhood” W of size $m < n$ is identified. The elements of W are then sorted, and the element of V , the centre of the neighbourhood, is replaced with the median value of W .

As is well known, the median M of a set of values is such that half the values in the set are less than M and half are greater than M .

At every discrete time instant we maintain a “buffer” with the last m measure vectors Z_ν [$\nu = i - m + 1, \dots, i$], arranged as a circular queue. At each iteration the Median Model is updated replacing each of its elements with the median of the corresponding elements in the vectors the buffer contains. This basically means that each element in the Median Model is the median of the values that the corresponding element in the measure vector Z_ν has assumed in the last m discrete time intervals. The size of the window m is chosen as a trade-off between the need to maintain adequate tracker tolerance towards variations in the motion dynamic of the patterns and the necessary rejection of the effects of noise. It should, however, be observed that small variations in this value do not significantly affect the overall performance of the algorithm: it is therefore possible to choose it “a priori”, taking into account the frame rate obtainable with the available hardware. In our case, we set the filter window to 15 frames.

The measure variance R_i is finally calculated by comparing the current measure vector Z_i with the Median Model. If R_i is greater than a fixed threshold $MaxVar$, then the whole measure is invalidated and a new initialisation is invoked: this threshold is usually called the “Validation Gate”.

2.1 Checking for a “click”

As described above, in our framework it is essential to detect the moment in which the two fingertips touch each other: this event corresponds to a “mouse.click”. When the two fingertips come into contact, the relative pair of patterns disappears from the image and is replaced by a single pattern that is quite different from a simple combination of the two. This is mainly due to application of the Canny algorithm edge tracer [2]. It is therefore necessary to provide for the case where, following a progressive shortening of the distance between them, the two patterns are replaced by a single one, corresponding to the event “mouse.button_down”,

and then the case in which the pattern is subsequently replaced by the pair of patterns again, very close to each other, corresponding to the event “mouse.button_up”. The “mouse.button_down” event is easily detected by activating a procedure, when the two patterns are very close to one another, that checks whether the “single” pattern has a greater degree of confidence than the weighted sum of the two confidence values for the two separate patterns. The weights of the sum were experimentally determined and in our case were 0.65 for the index finger and 0.35 for the thumb. The reverse procedure is applied to detect the “mouse.button_up” event.

We should recall at this point that the state variable for the “click” event has up to now been excluded from the status vector. This means that it is entirely unaffected by the *smoothing* typical of the Kalman Tracker and is thus subject to sharp variations due to noise: spurious signalling of “mouse.button_down” or “mouse.button_up” events may occur, which are a great nuisance to the reader. Application of a suitably sized median filter to the state variable “click” eliminated these spurious variations almost entirely, while maintaining the necessary response speed. The median filter window can be set experimentally or calculated off-line, bearing in mind that, after application of the filter, a variation in the value of the state variable is only detected if it persists for a number of frames equal to at least 50% of the width of the filter. Considering, therefore, that the dynamic typical of the gestures considered (move, click, drag) is about 8 Hz, and considering that the processing frequency typical of the hardware we used is about 25 frames per second, a convenient choice is a window of about 6 frames.

3. EXPERIMENTAL RESULTS

A prototype application was implemented in C++ in a Windows environment and ran on a Pentium II 300 MHz PC equipped with 64 MB of RAM memory. The camera is a low-cost USB webcam, able to acquire 25 frames per second at a resolution of 320×200 pixels. This architecture was conceived in order to test the performances of the system while running on a low-power platform. The tests in infrared (IR) light were conducted in a lab under fluorescent lighting, and illuminating the hand of the user with an illuminator made of 10 IR-LED. An IR filter glass was positioned in front of the webcam lenses.

Figure 7 shows some screenshots from a drag and drop sequence, in which the operator selects an icon on the Windows desktop, drags and finally drops it in a preselected, new position. Graylevel VisualGlove was tested continuously for two months as a pointing device during common-life interaction with the PC, by 4 users differently skilled. A further test was performed during three days, by asking 12 different users who were not accustomed at all to using vision-based devices, to perform some basic operations a number of times (about 120 times). The results of this latter test are shown in Table 1: “drag-drop” stands for drag and drop operation and “d-click” stands for double-click operation. “Draw” stands for a simple straight line draw operation. The “select” operation obviously gave the best performances, since it basically implies only a click on a given icon. As regards the minimum size of the selectable icon, some considerations should be made on the *resolution* of our pointing device. The size of the acquired frames, the acquisition speed and the scale used for the Canny edge detector (namely, the given value

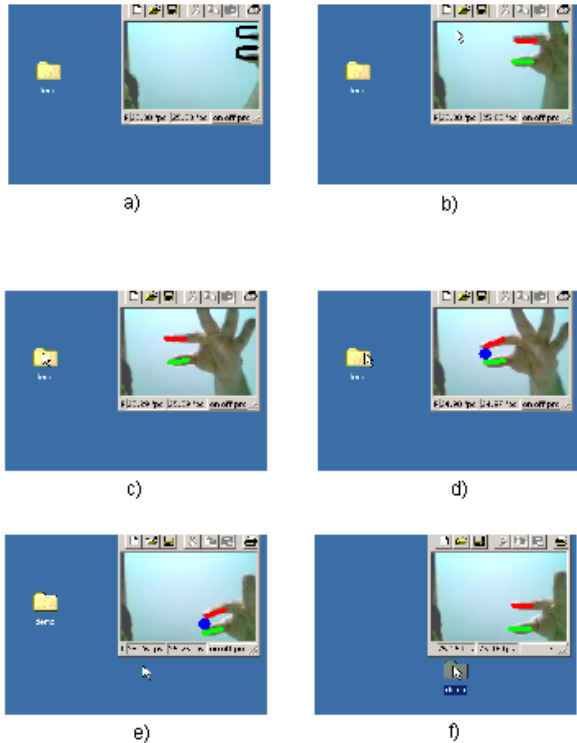


Figure 7: A sequence of screenshots from a drag and drop operation

illumination	select	drag-drop	d-click	draw
common light	100%	97%	85%	92%
IR light	100%	100%	94%	98%
normal light, clutt. bkgnd.	95%	91%	82%	86%

Table 1: Results from the experiments.

for its “window size” parameter) determine the resolution of our system. Since we are dealing with low-performance CPUs and cheap webcams, we decided to operate at a resolution of 320×200 which, in turn, leads to a frame rate of 20 to 25 frames per second. The window size for the Canny operator was experimentally set to 5, as a trade-off between resolution and tolerance to noise. This gave a very good accuracy while selecting icons on the Windows desktop (“normal” icon size, resolution 800×600) as well as clicking on links on a normal-size webpage. Quite hard, indeed, was to accurately select an item from a menu: this operation (actually made of a selection and a very accurate drag) showed to be very annoying for the testers and the results are not given in the table. A possible solution is to adopt different approaches, such as the use of buttons instead of menus (this approach is indeed quite common in palmtops).

The “double-click” operation returned the less accurate results during our experiments. This was mainly due the fact that the users were not accustomed to joining their fingers while maintaining their current position: it happened quite often that, while double-clicking, the users also moved the hand, thus performing a different operation (drag and

drop instead of double click). This phenomenon also appears with unskilled users while using for the first time a common mouse.

Since the size of the screen window, while operating, was 800×600 pixels, it was not possible to keep a “one-to-one” relation between the motion of the pointer in the VisualGlove environment (bounded by the 320×200 pixels resolution) and in the screen coordinates: otherwise, the pointer could cover only a fraction of the screen. We had to set a scaling factor between the displacements in the two coordinate systems. Unfortunately, setting a fixed relation between the screen and the VisualGlove resolutions (for example, stating that a displacement of 2 pixels in the VisualGlove environment corresponds to a displacement of 5 pixels on the screen) could substantially reduce the accuracy of the pointer. We opted for a dynamic solution: the relation between the two resolutions is set according to the current speed of the pointer, as returned by the Kalman Tracker. We experimentally set three different thresholds for the speed, each of them corresponding to a different scaling factor for the resolution (namely, 1.5, 2, 3). As the speed increases, the scaling factor increases. This leads to a total coverage of the screen, without loosing in accuracy while performing fine-grained operations.

Graylevel VisualGlove showed its limitations while working in ambient light, with a heavily cluttered background. In this situation, if the scale of the pattern in the background is comparable with the current size of the tracked patterns, the accuracy of our pointer degrades rapidly (see the last row in Table 1). This phenomenon is mainly due the use of the Canny edge detector, as its edge tracer is easily misled by a noisy background. The effect of a cluttered background on the Canny edge detector is shown in the first three frames of Figure 8. In the first frame, an usual (not heavily cluttered) background is present; in the second frame the background is noisy but the hand of the operator is still distinguishable; in the third frame the Canny edge detector is confused by the noise, thus “loosing contact” with the edge of the fingers and returning a false contour. The last frame shows the same scene after switching to IR illumination: as the background is not covered by the low-power LED illuminator, it does not appear in the image any more. Unfortunately, this solution is only partial, as it obviously does not work in sunlight. A better solution might be the use of a real infrared camera as in [12], which, on the other hand, is currently very expensive, and is thus unsuitable as a part of a pointer device. However, we should note that such a cluttered background had to be created “ad hoc” for this experiment: in fact the low quality of the lenses mounted on webcams has a smoothing effect on far objects, so heavily reducing the importance of background in our application.

4. CONCLUSIONS

In this paper a vision-based pointer device for Human-Computer Interaction, named Graylevel VisualGlove, was presented. It is able to reliably and efficiently substitute the common pointer devices (mouse, stylus, etc) in most situations, without requiring high computational resources or costly additional hardware. It only relies on a USB, graylevel webcam and, in case of very dark environment, on a lightweight IR illuminator. Graylevel VisualGlove is mainly intended as a pointer device for wearable computers, as in this case the low computational complexity is a main is-

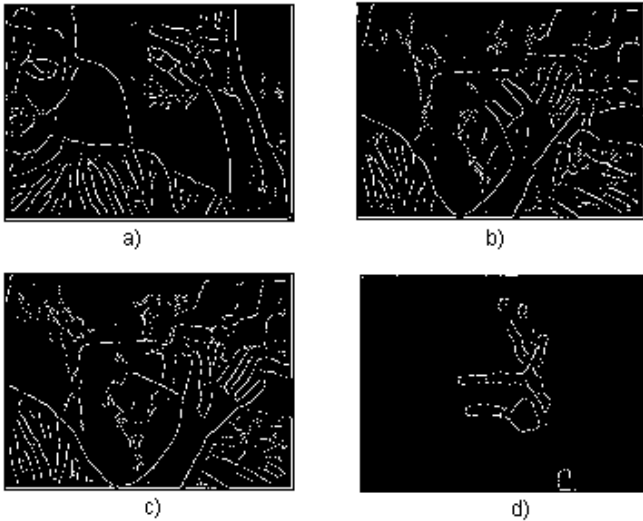


Figure 8: The effect of different cluttering levels on background with common (a to c) and infrared (d) illumination (see text)

sue due to the low computational power of the CPUs. While previous methods are mainly based on colour for image segmentation and human skin detection, our system only relies on graylevel gradient, thus being able to operate even in extreme illumination conditions. Experimental results were presented, which show the encouraging performances of the presented approach in several operational situations.

We are currently investigating the opportunity of exploiting motion analysis in order to reduce the effect of very noisy background. The main issue in this case is, as obvious, the increase in computational complexity. A possible solution might come from the outcome of hardware, yet economically affordable, products to generate in real-time a disparity map of the foreground scene.

5. REFERENCES

- [1] A. Blake and M. Isard. *Active Contours*. Springer-Verlag, 1998.
- [2] J. F. Canny. Finding edges and lines in images. Technical Report Tech. Rep. AI-TR-720, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1983.
- [3] J. F. Canny. A computational approach to edge detection. *IEEE Trans. on PAMI*, 8(6):679-698, June 1986.
- [4] R. Cipolla and A. Pentland. *Computer Vision for Human-Machine Interaction*. Cambridge University Press, 1998.
- [5] S. Ditlea. The pc goes ready-to-wear. *IEEE Spectrum*, 37(10):34-39, October 2000.
- [6] R. Gonzalez and R. Woods. *Digital Image Processing*. Addison-Wesley, 1992.
- [7] R. Grzeszczuk, G. Bradski, M. H. Chu, and J.-Y. Bouguet. Stereo based gesture recognition invariant to 3d pose and lighting. In *ICCV98*, pages 826-833, 2000.
- [8] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15, September 1993.
- [9] M. Kohler and S. Schroter. A survey of video-based gesture recognition - stereo and mono systems. Technical report, Fachbereich Informatik, Dortmund University, 44221 Dortmund, Germany, 1998.
- [10] V. I. Pavlovic, R. Sharma, and T. S. Huang. Visual interpretation of hand gestures for human-computer interaction: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):677-695, July 1997.
- [11] W. Rucklidge. *Efficient Visual Recognition Using the Hausdorff Distance*. Springer-Verlag, 1996.
- [12] Y. Sato, Y. Kobayashi, and H. Koike. Fast tracking of hands and fingertips in infrared images for augmented desk interface. In *Proc. of IEEE Automatic Face and Gesture Recognition (FG2000)*, pages 462-467, 2000.
- [13] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis and Machine Vision*. Chapman & Hall Computing, 1993.
- [14] T. Starner and A. Pentland. Visual recognition of american sign language using hidden markov models. In *Proc. of Int. Workshop on Automatic Face and Gesture Recognition*, pages 189-194, Zurich, Switzerland, June 1995.
- [15] Ibm wearable computers news. http://www.ibm.com/news/ls/1998/09/jp_3.phtml.
- [16] Xybernaut home page. <http://www.xybernaut.com/>.