Research Article

LoCoBoard: Low-Cost Interactive Whiteboard Using Computer Vision Algorithms

Christophe Soares,^{1,2} Rui S. Moreira,^{1,2} José M. Torres,¹ and Pedro Sobral¹

¹ ISUS Group, FCT, University Fernando Pessoa, Porto, Portugal

² INESC TEC, FEUP, University of Porto, Porto, Portugal

Correspondence should be addressed to Christophe Soares; csoares@ufp.edu.pt

Received 13 December 2012; Accepted 10 January 2013

Academic Editors: M. A. Rahman and T. Zhang

Copyright © 2013 Christophe Soares et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In the current digital age, the adoption of natural interfaces between humans and machines is increasingly important. This trend is particularly significant in the education sector where interactive tools and applications can ease the presentation and comprehension of complex concepts, stimulate collaborative work, and improve teaching practices. An important step towards this vision, interactive whiteboards are gaining widespread adoption in various levels of education. Nevertheless, these solutions are usually expensive, making their acceptance slow, especially in countries with more fragile economies. In this context, we present the low-cost interactive whiteboard (LoCoBoard) project, an open-source interactive whiteboard with low-cost hardware requirements, usually accessible in our daily lives, for an easy installation: a webcam-equipped computer, a video projector, and an infrared pointing device. The detection software framework offers five different Pointer Location algorithms with support for the Tangible User Interface Object protocol and also adapts to support multiple operating systems. We discuss the detailed physical and logical structure of LoCoBoard and compare its performance with that of similar systems. We believe that the proposed solution may represent a valuable contribution to ease the access to interactive whiteboards and increase widespread use with obvious benefits.

1. Introduction

Over the past decades, the computing power evolution has been remarkable, but human computer interaction (HCI) is still being realized, in most cases, through the traditional keyboard and mouse. The future of HCI should rely more on the use of natural interfaces, such as haptic, speech, and gestures. In particular, the use of natural interfaces through Interactive Whiteboards (IW) in education environments can ease the presentation and comprehension of complex concepts, allow collaborative work between teachers and students and improve pedagogical practices. Although there are wide ranges of commercial IW solutions, they are generally expensive and difficult to afford and implement by a large number of education institutions.

The main goal of this project consists in developing an open source software-based IW solution, based on usually accessible hardware in our daily lives, that is, a video projector, a laptop with a webcam, and an Infra-Red (IR)

pointing device. The LoCoBoard [1] prototype uses computer vision algorithms for processing captured images through the webcam and interpret the user interactions. Each developed algorithm tries to isolate sets of pixels with common characteristics (cf. binary large objects-blob) from each image and is tested in its performance, accuracy, and CPU load. The application is proactive, adapting itself to the environment light conditions (cf. Background Subtraction techniques [2]). Coordinates of detected user interaction points are then used to control the cursor/interaction on the computer or distributed to other applications, through Tangible User Interface Objects (TUIO) protocol [3]. The developed prototype reveals that with this software running on a common computer, it is possible to obtain a generic, low-cost, easy to install, and useful IW system for any classroom.

The rest of the paper is organized as follows. In Section 2, we briefly discuss the current state, mostly open-source, for interactive surfaces in general and interactive whiteboards in particular. In Section 3, we present the physical and logical

structure of the LoCoBoard system. Section 4 is devoted to presenting the quantitative and qualitative evaluation carried out. We introduce a comparative evaluation between the five Pointer Location (PL) detection algorithms implemented within the LoCoBoard system and also a comparison with other available systems. We conclude the paper in Section 5.

2. State of the Art

In the last years, there has been quite active research in developing solutions for interactive surfaces, in general, ranging from small sized displays to large interactive whiteboards [4]. The LoCoBoard system, described in this paper, has been conceived, in first hand, for working as an interactive whiteboard system, targeted to be used with existing whiteboard surfaces, universally present in class and meeting rooms, although it can easily be adapted to setup an interactive table using multitouch gestures (cf. Algorithm A5, see Algorithm 6).

Interactive whiteboards can be a valuable aid in the process of teaching although some problems may arise when teachers fail to appreciate that interactivity requires a new approach to pedagogy [5]. Also, the use of pen and touch, in interactive whiteboards, for tasks such as data exploration or others, constitutes a challenge for rethinking conventional application solutions [6]. Nevertheless, one of the major problems of typical commercially available whiteboards solutions continues to be their excessive prices, avoiding the proliferation of this valuable educative tool as a way to enhance educative practices.

As mentioned in [7], there are five main techniques, being refined by the open community research efforts that allow for the creation of a stable multitouch hardware systems: frustrated total internal reflection (FTIR); rear diffused illumination (Rear DI); laser light plane (LLP); diffused surface illumination (DSI); and LED light plane (LED-LP). All these five key techniques work in conjunction with image sensing devices (e.g., cameras) and presume the use of computer vision techniques to process the image signal produced. Besides those techniques, there are other sensing hardware devices frequently used, based on sensors such as the resistive, capacitive, motion, and others, or a combination of more than one of the listed. Several solutions are presented in literature, demonstrating the application of these types of technology [8]. In general, multitouch technologies, using more or less complex hardware setups, have been in use for some time, triggering a challenge for user interfaces designers [9].

Two significant requirements that guide the development of the LoCoBoard system were: (i) the system should be simple to setup and (ii) it should be low cost. This led to the decision of using a hardware setup based solely in a camera and an infrared pen, besides the computer and projector.

Other touch systems based on overhead cameras are also presented in the literature. The PlayAnywhere system [10] considers the camera and projector sitting off to the side of the active surface. The solution proposed in [11] uses a pair of cameras mounted above the surface.

Some other popular interactive whiteboard systems based on the Nintendo Wii remote command, the wiimote, are the ones presented by Lee [12] and Schmidt [13]. Both systems provide software libraries to implement the interactive whiteboard. The wiimote also has inspired several works in the interaction area using diverse types of sensing technology, such as accelerometers [14].

All the presented systems based on image sensing (i.e., video cameras), with or without extra sensor devices, need software libraries to process the video input signal and transform it into pointing devices locations and/or commands to be passed to a client application. Among the most used software libraries are Touchlib [15] and Community Core Vision (CCV aka tbeta) [16] from the NUI Group and the reacTIVision framework [17].

Those pointing device commands can be sent by the tracker application to the client application using appropriate protocols. The protocol TUIO [3], based on Open Sound Control (OSC) interface [18], is a popular open framework that defines a common protocol and APL for tangible multitouch surfaces. Using the TUIO, the client application, for instance, an application written in ActionScript 3 language and Adobe Flash, can reside in the same computer or a different computer of the tracker application, for instance, the LoCoBoard system.

3. LoCoBoard System

3.1. Overview. The LoCoBoard system overview is presented in Figure 1. We can see a computer running the LoCoBoard software, a camera which allows us to sense/capture the interactions on the projected image, an infra-red (IR) pointing device to generate the interaction, and a video projector to output the computer application image over the whiteboard. The pointing device, presented in Figure 2, was built using an IR led installed on top of a traditional whiteboard pen. Switching on and off the IR led in the pen, allows the user to interact directly with the computer through the whiteboard surface. The supported kinds of interactions are click and drag, single and double mouse click. The camera is equipped, with an IR band-pass filter, to discard unwanted light wavelengths from the captured images, easing the real-time detection of the IR pointing device. A photographic film can be used as a straightforward IR band-pass filter. Since all the hardware is widely available and is much cheaper than traditional interactive whiteboards, this project could be easily replicated in many education institutions.

Figure 3 presents the software architecture of the LoCoBoard system. On top of the already described hardware layer, we have the computer operating system. The LoCoBoard system was designed to support the most widely used operating systems, that is, Microsoft Windows, Linux, and Mac OS X. Regarding the libraries, OpenCV computer vision framework [19] is used to manage image acquisition and to preprocess those images. Although in the current version OpenCV offers some blob detection capabilities, they are not used in this system. We have developed, implemented, and tested our own blob detection algorithms in

FIGURE 1: Overview of the LoCoBoard system.

FIGURE 2: Components of the Infrared Pointer device.

LoCoBoard. The Ospecpack framework [20] provides interoperability between LoCoBoard and any TUIO [3] compliant client application. The TUIO protocol is also used in applications like the Touchlib or CCV [16]. A detailed description of the four LoCoBoard building blocks is presented in Section 3.2.

The system operation starts with a calibration process to map the projected image area in the whiteboard as seen by the video camera (cf. geometrical transformation), back into the original computer coordinate space (cf. monitor resolution). This task is essential to enable a real-time and robust process of Pointer Location detection using the computer coordinates. After the calibration phase, the LoCoBoard system starts interpreting the camera output identifying the mouse interaction through the displacement of the IR Pointer over the projected image. We also report in real-time the coordinates of the detected IR dots (one or two PL) using the TUIO protocol [3] and enabling the support for existing multitouch applications based on this protocol.

3.2. Implementing LoCoBoard. The LoCoBoard's block diagram, presented in Figure 3, reveals the core of the application. The image acquisition and preprocessing phase is responsible to grab and filter the images captured by the camera. Using OpenCV, we improve image quality and ease the IR Blob recognition. In the Pointer Location (PL) detection phase, we apply our developed algorithms to detect and track IR Blobs. Finally, the last component reports the

interaction coordinates using TUIO protocol and updates the cursor position.

3.2.1. Image Acquisition and Preprocessing. The image acquisition block maps the computer resolution on the projection image captured through the camera using the OpenCV framework transformation matrix. We apply filters, in each image, to remove eventual interferences and noises induced by other light sources. There are often other IR emitters in the environment, which may interfere with the IR pointing device (e.g., lamps, sunlight, etc.). Hence, to improve system accuracy, we use a physical (e.g., photo film) and logical IR band-pass filter (e.g., proactive background subtraction model). We use a foreground and background extraction model to identify and remove unwanted IR interferences from the image. The background filtering algorithm dynamically adapts its model to the variations on the environment light conditions.

3.2.2. Pointer Location Detection Algorithms. The image received from the previous block is then handled by the PL detection algorithms. An image is represented by a matrix where each pixel is coded by three values corresponding to the RGB color model. This image is then converted to gray scale in order to reduce the pixel state space. This transformation is useful in real time systems, where the color is not relevant since image processing can be more efficient and decrease the amount of time to process PL detection algorithm. PL coordinates are then searched on a gray scale image. For each PL found by the system, we return its pixel position using Cartesian coordinates. The Cartesian coordinates are reported setting the upper left corner pixel as (0, 0) and the lower right corner pixel as (width and height of the image). The system handles different image sizes depending on the camera resolution. The better the camera resolution the higher the system accuracy. However, large camera resolutions and high frame-rates substantially increase the CPU load. Image resolution and frame rate can be adjusted to allow better support on legacy computers.

FIGURE 3: Logical architecture of LoCoBoard.

FIGURE 4: Algorithm A1 process. The first step: analyzing and collecting Pointers Location over the threshold (a), the second step: identifying the centroid (b), and terminology used in the algorithms representation (c).

In the following two subsections, we describe the PL detection algorithms. Figure 4(c) shows the common visual symbology used in the algorithms descriptions.

Single Pointer Location (Algorithms A1, A2, A3, and A4). (See Algorithms 1, 2, 3, and 5).

(1) A1: this algorithm performs a scan on the image pixel by pixel as illustrated in Figure 4 and Algorithm 1, to find

a pixel with a luminance value exceeding a threshold set by the program during the calibration phase. This algorithm collects the coordinates of all points that have a value above the threshold and returns the average of the values in a row and column axis.

(2) A2: the traditional linear search algorithm (cf. A1) is modified using a step value S to speed up the search phase and returns when a pixel value greater than the threshold is found.

```

function ALGORITHM (image_height, image_width, image, threshold) return coordinates of the PL centroid
inputs:      image_height, height of the image
               image_width, width of the image
               image, captured grayscale image
               threshold, lower limit of brightness to identify pixels belonging to PL blob

size  $\leftarrow$  0
colValue  $\leftarrow$  0
rowValue  $\leftarrow$  0
row  $\leftarrow$  0
  while row < image_height do
    column  $\leftarrow$  0
    while column < image_width do
      if image[row][column]  $\geq$  threshold then
        colValue  $\leftarrow$  colValue + column
        rowValue  $\leftarrow$  rowValue + row
        size  $\leftarrow$  size + 1
      column  $\leftarrow$  column + 1
    row  $\leftarrow$  row + 1
colValue  $\leftarrow$  colValue/size
rowValue  $\leftarrow$  rowValue/size
return (rowValue, colValue)

```

ALGORITHM 1: Pseudo code Algorithm A1.

```

function ALGORITHM (image_height, image_width, image, threshold, step) return coordinates of the PL centroid
inputs:      image_height, height of the image
               image_width, width of the image
               image, captured grayscale image
               threshold, lower limit of brightness to identify pixels belonging to PL blob
               step, interval between consecutive processed pixels per row during search

size  $\leftarrow$  0
colValue  $\leftarrow$  0
rowValue  $\leftarrow$  0
index  $\leftarrow$  0
  while index < image_height * image_width do
    row  $\leftarrow$  index/image_width
    column  $\leftarrow$  index % image_width
    if image[row][column]  $\geq$  threshold then
      colValue  $\leftarrow$  colValue + column
      rowValue  $\leftarrow$  rowValue + row
      size  $\leftarrow$  size + 1
    index  $\leftarrow$  index + step
colValue  $\leftarrow$  colValue/size
rowValue  $\leftarrow$  rowValue/size
return (rowValue, colValue)

```

ALGORITHM 2: Pseudo code Algorithm A2.

The best results are obtained when S is lower than half the size of the PL. With S greater than the PL size (overfitting), we may never find the PL on the image.

Thus, between two pixels readings, the algorithm ignores $S-1$ pixels (see Figure 5 and Algorithm 2). The process to calculate the coordinates of PL is the same as in the previous algorithm (cf. A1). This algorithm provides accurate results on the centroid PL estimation, when S is small. However, the larger the S value the faster will be the image processing.

(3) A3: this algorithm focuses on reducing processing costs per image when a PL is present, but it is not in the right lower corner of the image. We use a linear search as described for Algorithm A2 (see Figure 6 and Algorithm 3). Unlike the previous algorithm (cf. A2), A3 stops the linear search as soon as it finds the first suitable pixel. After identifying this pixel in the image, another algorithm is applied, in this area, to determine the center coordinates of the PL (see Figure 6(b)). This algorithm searches in four directions

```

function ALGORITHM (image_height, image_width, image, threshold, step) return coordinates of PL centroid, or
failure
inputs:      image_height, height of the image
               image_width, width of the image
               image, captured grayscale image
               threshold, lower limit of brightness to identify pixels belonging to PL blob
               step, interval between consecutive processed pixels per row during search

size  $\leftarrow$  0
index  $\leftarrow$  0
while index < image_height * image_width do
    row  $\leftarrow$  index / image_width
    column  $\leftarrow$  index % image_width
    if image [row] [column]  $\geq$  threshold then
        (centroid.row, centroid.col)  $\leftarrow$  FINDCENTER(image_height, image_width, image, threshold, row, column)
        return (centroid.row, centroid.col)
    index  $\leftarrow$  index + step
return failure/* didn't found any center */

```

ALGORITHM 3: Pseudo code Algorithm A3.

FIGURE 5: Algorithm A2 process. The first step: analyzing and collecting Points Location over the threshold (a) and the second step: identifying the centroid (b).

starting from the over the threshold pixel previous found. We search horizontally (west and east) to detect the maximum and minimum values on the row-axis and vertically (north and south) to detect the maximum and minimum values on the column-axis. These two sets of values ({min row-axis, max row-axis}; {min column-axis, max column-axis}) correspond to the prediction center for the PL (see Figure 6(b)).

(4) A4: A4 Algorithm uses a different approach since its focus is on tracking PL over consecutive images. Hence, it should present better results, that is, less processing cost and time elapsed to detect PL, when a previous detected PL moves on consecutive images (e.g., dragging the pointing device). The algorithm uses a vector to store information about the PL displacement, that is, the difference of PL positions between two consecutive images. Consider three consecutive images: F_1 , F_2 , and F_3 . All respectively contain a PL in movement/displacement, which has the respective coordinates, $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, and $P_3 = (x_3, y_3)$.

The value of the PL displacement vector, between F_1 and F_2 , Δs is calculated as follows:

$$\Delta s = (\Delta x, \Delta y) = (x_2 - x_1, y_2 - y_1). \quad (1)$$

The algorithm uses Δs to predict the future position of P_3 and then execute from this prediction a spiral search to find the real coordinates of PL in F_3 (see Figure 7(a)). This approach assumes that the displacement is similar between consecutive images unless the IR pen stops abruptly. The estimation of P_3 coordinates in F_3 is calculated through the formula:

$$\hat{P}_3 = P_2 + \Delta s = (x_2 + \Delta x, y_2 + \Delta y). \quad (2)$$

The algorithm uses the estimation of P_3 as a starting point to initiate the spiral search. On average, this algorithm provides better results when compared with searching PL from the origin. A traditional spiral search can be time consuming due to nonlinear memory accesses. Hence, we use a lookup

FIGURE 6: The Algorithm A3 Process (a) and how to find the centroid of PL (b).

FIGURE 7: The representation of Algorithm A4 process—prediction (a) centroid process (b).

table with the distance related to row-axis and column-axis, which by addition would apply the movement of a spiral as can be seen in Figure 7(a). Spiral search algorithm can be inefficient, when compared to linear search methods, when a PL suddenly moves to an unpredicted position or disappears from the image. The spiral search repeatedly performs jumps between different memory areas while the linear search processes consecutive storage positions. Therefore, when the algorithm reaches the maximum spiral size N and fails, a linear search strategy is applied. After the search phase, the coordinates for the PL centroid are obtained as previously described (see Figure 7(b)).

Multiple Pointers Location (Algorithm A5). The multipoint algorithm uses an ordinary linear search and recursively collects and labels the foreground pixels information for all existing PLs in the image (see in Figure 8(a)). For each of those groups, the centroid coordinates are determined as illustrated in Figure 8(b). Although this algorithm can detect

a large number of PLs, we only track up to two simultaneous PLs in the LoCoBoard system.

3.3. LoCoBoard Output. After PL detection algorithm, we need to report the centroid coordinates using the TUIO protocol to enable user interaction or directly control the mouse/cursor on the computer. For TUIO report, the kind of action triggered depends on the client TUIO application; optionally, we could also include in the TUIO reports the intensity value and the size of the detected PL per image. For the computer, LoCoBoard [1] system handles these actions and emulates on the same equipment: click (single or double) and move/drag. We interpret that the user wants to do a double click when the IR emitter (cf. Pointer Location) remains active on the same region in consecutive images during one second (time is setup as a property and could be overwrite). In other cases, we will consider that the user emulates a single click. The drag movement is a particular

FIGURE 8: The representation of Algorithm A5 process (a), find groups and their centroid (b).

FIGURE 9: LoCoBoard project interaction with other application.

case of a single click, where we detect a click followed by a movement of the PL in consecutive images, source and destination correspond, respectively, to the coordinate first appearance and the disappearance for this PL movement.

We also need to execute a calibration between the projection and the native computer resolution. This will allow the determination of the correspondence between the PL detected coordinate on the projection and the computer's mouse.

3.3.1. Interactive Whiteboard. We tested the interaction accuracy performing the following functions operations: (i) opening files in finder/explorer, (ii) navigating to a URL using the Firefox WEB browser and a virtual keyboard to insert the characters, (iii) drawing in Paint and (iv) interacting with Google Earth. We select applications that require an intensive use of the mouse and that are better suited to use with an interactive whiteboard (e.g., Google Earth)—see Figure 9.


```

function FINDCENTER (image_height, image_width, image, threshold, row, column) return identifies the centroid
coordinates of a PL region
inputs:   image_height, height of the image
            image_width, width of the image
            image, captured grayscale image
            threshold, lower limit of brightness to identify pixels belonging to PL blob
            row, row coordinate of known pixel belonging to PL blob
            column, column coordinate of known pixel belonging to PL blob

    minCol  $\leftarrow$  column
    maxCol  $\leftarrow$  column
    While maxCol < image_width AND image[row][maxCol]  $\geq$  threshold do
        maxCol  $\leftarrow$  maxCol + 1
    while minCol  $\geq$  0 AND image[row][minCol]  $\geq$  threshold do
        minCol  $\leftarrow$  minCol - 1
    distanceCol  $\leftarrow$  maxCol - minCol
    centerCol  $\leftarrow$  minCol + distanceCol/2
    minRow  $\leftarrow$  row
    maxRow  $\leftarrow$  row
    while maxRow < image_height AND image[maxRow][centerCol]  $\geq$  threshold do
        maxRow  $\leftarrow$  maxRow + 1
    while minRow  $\geq$  0 AND image[minRow][centerCol]  $\geq$  threshold do
        minRow  $\leftarrow$  minRow - 1
    distanceRow  $\leftarrow$  maxRow - minRow
    centerRow  $\leftarrow$  minRow + distanceRow/2
    return (centerRow, centerCol)

```

ALGORITHM 4: Find center function.

3.3.2. *TUIO Applications.* These applications interact with the LoCoBoard using the TUIO protocol. The LoCoBoard system is able to report the PL locations using the same protocol. That the protocol has been used for three reasons: (i) establish a common communication protocol between interactive applications and sensing frameworks like CCV/Tbeta or Touchlib; (ii) separate the PL tracking framework development from any interactive application, and (iii) reuse a large number of already developed TUIO compatible, interactive applications. TUIO compliant applications are generally developed in flash in order to allow multiple PL interaction since current operating systems only handle one single pointing device interaction at a time. We use a demo application to manipulate some pictures and other applications, that is, Piano Player on Figure 9(c) or TUIO Pictures Gallery, use two PL allowing users to: (i) zoom, (ii) move/drag, and (iii) rotate pictures. The interpretation of the perceived coordinates depends on the TUIO compliant application (Algorithm 4).

4. Evaluation

To illustrate the PL detection approach, we first have applied it to a set of prerecorded videos which depicts a classic use case interaction scenario and we assess the performance of each PL detection algorithm. Later, we present a comparison between the LoCoBoard approach and other related systems, namely, CCV aka Tbeta. All the experiments have been conducted on the same machine: an Apple MacBook (Model

TABLE 1: Subset of videos with a description of their characteristics used for the evaluation.

Videos	Conditions	
	Simulation kind of activity	Environment condition
Video 1	Interaction clicks	Without noise
Video 2	Drag/movement	Without noise
Video 3	Mixture	Without noise
Video 4	Interaction clicks	With noise
Video 5	Drag/movement	With noise
Video 6	Mixture	With noise

4.1), 2.4 Ghz and 4 GB ram. Connectivity on the equipment has been limited to guarantee the same situation for different algorithms analysis.

4.1. *Benchmark Analysis.* We evaluate the different PL detection algorithms presented in Section 3.2.2 regarding its computational performance under different test conditions. Six videos were created with several interaction scenarios, and the algorithms (A1 to A5) were used for PL tracking on each of them (see Table 1). The videos include various types of interactions with the IR pointer: clicks, drags, and a mixture of both. Some videos had the presence of sources of interference in the background image (e.g., natural lamp lights, sun light, etc.). During the simulation, IR interaction clicks appear at random places in the image and drags happened through the image using random movements.

FIGURE 10: Duration of each processing run-time per algorithm per video.

TABLE 2: Error estimation of PL per algorithm.

Ground truth	Algorithms				
	A1	A2	A3	A4	A5
First PL	1	0	3	4.12	0
Second PL	1.41	1	3.16	3.16	1
Third PL	1.41	1.41	3.61	3.16	1
Fourth PL	1.41	1.41	3.61	3.16	1
Fifth PL	1.41	1.41	3.61	3.16	1

Light variations (noise) were created in the environment allowing the test of PL detection accuracy in the presence of light interference. Videos with noise are preprocessed with a background subtraction algorithm, using a proactive learning method, to remove background and identify any noise.

4.1.1. Performance. In order to measure the performance of each algorithm, we compare time profiles from each of them during execution. The figures represent that the amount of time each algorithm was executing during sampling. The parameters values used were: for A2 and A3, $S = 3$ (step value); and for the A4, $N = 5$ (maximum spiral size). We can see in Figure 10 that the most efficient algorithms were A3 and A4 (due to prediction mechanism in videos 2 and 5), followed by A2, A1, and A5, the latter being the least efficient (due to the exhaustive linear search on all the image and an estimation on possible multiple PL).

4.1.2. Accuracy. The accuracy in the detection of PL is essential. We use five images (500×500 pixels), each with a unique PL, where we previously known the real coordinates (i.e., ground truth). We then used the algorithms to estimate the coordinates and compare the results with the real PL coordinates. This procedure was repeated with the five algorithms, in each of the images. The results presented in Table 2 correspond to the Euclidean distance between the estimated and the actual positions of the PL. Lower Euclidean distances correspond to better algorithm accuracy. In Table 2, we can see that A2 and A5 have the best result in this test.

We also analyzed the behavior of the PL detection algorithms using six different geometric forms for the PL (see Figure 11). To quantify the accuracy, we reused Euclidean distance as previously stated. Algorithm A4 reuses a detection

FIGURE 11: Different geometric representations used to depict PL.

FIGURE 12: Evaluation of the accuracy of each of the algorithms in terms of the geometric forms of PL.

TABLE 3: CPU consumption comparison between the LocoBoard and the CCV (Tbeta).

CPU %	Applications system for PL detection	
	CCV (Tbeta)	LoCoBoard
Minimum	64.1%	14.0%
Maximum	73.3%	52.8%
Mean	68.3%	32.1%

algorithms (A1, A2, or A3) if the search in the predicted area of interest/region was not successful and as such was not relevant to be tested here. From our results (see Figure 12), it is clear that the geometric shape of PL affects the precision of some algorithms. In particular, the A3 offers the worst accuracy while A1 and A5 show the highest precision levels. However, all algorithms performed worse when the PL shape approximates a lozenge. Algorithm A2 maintains an almost constant error rate regardless of the PL shape.

From this experiment, we realized that certain PL shapes can provide more accurate results than others. This may be relevant in the choice for the IR light source used in the pointing device. We found that the best results were achieved when the PL shape is close to a square or circle. Also, as we were expecting, we observed that algorithms using a total linear search in the image (A1 and A5) present better results.

4.1.3. Processing/Profiling. This experiment compares the CPU load of the LoCoBoard and CCV applications for the same tasks. This comparison was made by collecting twenty samples of the CPU load per application using a profiling tool. Results for this test are presented in Table 3 where we can see that LoCoBoard induces a lower CPU load than CCV. This

TABLE 4: Qualitative comparison LocoBoard with related systems.

App./criteria	Cost	Equipment	Compatibility	Support	TUIO	Usability
LoCoBoard [1]	Low	Camera, IR-Pointer, Pass-IR filter	Windows, Mac OS X, Linux	Single*	Yes	Simple
Johnny Lee Whiteboard [12]	Moderate	Wiimote, Bluetooth adapter, IR-Pointer	Windows	Single	No	Moderate
Uweschmidt [13]	Moderate	Wiimote, Bluetooth adapter, IR-Pointer	Windows, Mac OS X, Linux	Single	Yes	Moderate
CCV/Tbeta [16]	Low	Camera, IR-Pointer, Pass-IR filter	Windows, Mac OS X, Linux	Multi	Yes	Moderate
Touchlib [15]	Low	Camera, IR-Pointer, Pass-IR filter, IDE compiler	Win	Multi	Yes	Complex

* note: multipoint is supported but was not the main focus, since operating systems are not prepared off the shelf to handle multiple source of input.

```

function ALGORITHM (image_height, image_width, image, threshold, lookup_table, est_row, est_col) return centroid
coordinates of a PL, or failure
inputs:    image_height, height of the image
             image_width, width of the image
             image, captured grayscale image
             threshold, lower limit of brightness to identify pixels belonging to PL blob
             lookup_table, predefined spiral search path relative to estimation of PL centroid in current image
             est_row, estimated row location for PL centroid in current image
             est_col, estimated column location for PL centroid in current image

for index  $\leftarrow$  0 to SIZE(lookup_table) do
    column  $\leftarrow$  est_col + lookup_table[index].x
    row  $\leftarrow$  est_row + lookup_table[index].y
    if image[row][column]  $\geq$  threshold then
        (centroid.row, centroid.col)  $\leftarrow$  FINDCENTER(image_height, image_width, image, threshold, row, column)
        return (centroid.row, centroid.col)
return failure/*didn't found any center—will use another search algorithm for the rest of the image*/

```

ALGORITHM 5: Pseudo code Algorithm A4.

is perhaps due to the fine tuned PL detection algorithms we use while the CCV uses generic functions implemented in the OpenCV library.

4.2. Comparison with Other Systems. This section presents a qualitative evaluation of LoCoBoard [1] prototype with related systems, using the following criteria.

- (1) Cost: this is defined through a three levels scale: Low (contemplating systems that cost less than thirty euro), Average (for costs less than fifty euro), and High (corresponding to all others). The cost does not include computer and projector values.
- (2) Equipment: this serves to list the material required to mount an interactive whiteboard, again, without considering the computer and projector.
- (3) Compatibility: this property relates to the supported platforms (e.g., Mac, Linux, or Windows).
- (4) Multiple PL support: states if the system is able or unable to support multiple pointers location.
- (5) TUIO [20]: defines if the application provides support for the TUIO protocol.

- (6) Ease of setup: here we access the complexity to deploy the system. We use a three level scale: Simple (reproduce the deployment is less than four steps), Moderate (less than six steps) and Complex (all the others).

The qualitative evaluation results are summarized in Table 4 [1]. Besides the fact that LoCoBoard is a low-cost and multiplatform system, one can observe that an important additional attribute is the fact that it represents a turnkey system for someone searching a ready to use interactive whiteboard solution to implement.

5. Conclusions and Future Work

The work described in this paper consisted in the designing of a simple, economic, easy to setup, user friendly and effective solution for an interactive whiteboard. LoCoBoard is an open-source project shared with the community and offers an alternative to the already existing platforms of PL detection (cf. Touchlib and CCV).

```

function ALGORITHM (image_height, image_width, image, threshold) return centroids' coordinates of each
identified PL
inputs:    image_height, height of the image
            image_width, width of the image
            image, captured grayscale image
            threshold, lower limit of brightness to identify pixels belonging to PL blob
/* recursive function to compute the connected components of a binary image */
LB  $\leftarrow$  (-1)*threshold(image, threshold)
label  $\leftarrow$  0
for L  $\leftarrow$  0 to image_height
    for P  $\leftarrow$  0 to image_width
        if LB[L,P] = -1 then
            label  $\leftarrow$  label + 1
            SEARCH(LB, label, L, P)
            /* FINDCENTROID function returns centroid of just labeled region */
            (centroids.row[label], centroids.col[label])  $\leftarrow$  FINDCENTROID(LB, L, P, label)
return (centroids.row, centroids.col)

function SEARCH(LB, label, L, P)
    LB[L,P]  $\leftarrow$  label
    Nset  $\leftarrow$  NEIGHBORS(L,P)
    for each [L', P'] in Nset
        if LB [L', P'] = -1 then
            SEARCH(LB, label, L', P')

```

ALGORITHM 6: Pseudo code Algorithm A5 adapted from [21, page 57].

We have conducted several experiments to assess the functionality and efficiency of the LoCoBoard system and to benchmark it with some popular alternative open-source options. Using different experiment scenarios (cf. six videos, see Section 4), we observed that background noise (e.g., extreme environment illumination conditions) could disturb the functioning and affect the efficiency of the presented algorithms. Also, based on these experiments we could conclude that Algorithm A4 attained better results, using less time than all the others for processing the data set used.

As a future work directions, we pretend to continue the improvement of the platform and also the test and deployment of applications, to interact with the LoCoBoard, to be used in the classroom. We pretend to test and evaluate the entire setting in, larger scale, real environments. We also initiate working on multiple PL algorithms (cf. A5), since a full multitouch aware system is becoming increasingly prominent. Additionally, we plan to develop a complete test scenario to identify the most promising algorithmic approaches for the several software modules of the LoCoBoard system (e.g., increasing the number of videos used as a sample).

Finally, we intent to improve the adopted background subtraction model to filter any noise from the captured images. This is fundamental since all estimation algorithms are dramatically affected by the noise presence in the natural environment.

Acknowledgment

C. Soares thanks the FCT—Foundation for Science and Technology, Portugal, for Ph.D. Grant SFRH/BD/64210/2009.

References

- [1] C. Soares, *LoCoBoard: Quadro Interactivo de Baixo Custo recorrendo a Algoritmos de Visão por Computador* [M.S. thesis], Universidade Fernando Pessoa, Porto, Portugal, 2009.
- [2] Y. Benezeth, P. M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger, "Comparative study of background subtraction algorithms," *Journal of Electronic Imaging*, vol. 19, no. 3, 2010.
- [3] T. Bovermann, R. Bencina, E. Costanza, and M. Kaltenbrunner, "TUIO: a protocol for table-top tangible user interfaces," in *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation (GW '05)*, 2005.
- [4] R. Chang, F. Wang, and P. You, "A survey on the development of multi-touch technology," in *Proceedings of the 2010 Asia-Pacific Conference on Wearable Computing Systems (APWCS '10)*, pp. 363–366, IEEE Computer Society, Washington, DC, USA, April 2010.
- [5] D. Glover and D. Miller, "Running with technology: the pedagogic impact of the large-scale introduction of interactive whiteboards in one secondary school," *Journal of Information Technology For Teacher Education*, vol. 10, no. 3, pp. 257–278, 2001.
- [6] J. Walny, B. Lee, P. Johns, N. H. Riche, and S. Carpendale, "Understanding pen and touch interaction for data exploration on interactive whiteboards," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2779–2788, 2012.
- [7] N. Group, *Multi-Touch Technologies*, NUI Group, 1st edition, 2009.
- [8] J. Teichert, M. Herrlich, B. Walther-Franks et al., "Advancing large interactive surfaces for use in the real world," *Advances in Human-Computer Interaction*, vol. 2010, Article ID 657937, 10 pages, 2010.
- [9] P. I. S. Lei and A. K. Y. Wong, "The multiple-touch user interface revolution," *IT Professional*, vol. 11, no. 1, pp. 42–49, 2009.

- [10] A. D. Wilson, “Playanywhere: a compact interactive tabletop projection-vision system,” in *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST '05)*, pp. 83–92, ACM, New York, NY, USA, 2005.
- [11] A. Agarwal, S. Izadi, M. Chandraker, and A. Blake, “High precision multi-touch sensing on surfaces using overhead cameras,” in *Proceedings of the 2nd Annual IEEE International Workshop on Horizontal Interactive Human-Computer Systems (Tabletop '07)*, pp. 197–200, IEEE Computer Society, Washington, DC, USA, October 2007.
- [12] J. C. Lee, “Hacking the nintendo wii remote,” *IEEE Pervasive Computing*, vol. 7, no. 3, pp. 39–45, 2008.
- [13] U. Schmidt, Wiimote whiteboard, 2008, <http://www.uweschmidt.org/wiimote-whiteboard>.
- [14] T. Vajk, P. Coulton, W. Bamford, and R. Edwards, “Using a mobile phone as a “wii-like” controller for playing games on a large public display,” *International Journal of Computer Games Technology*, vol. 2008, Article ID 539078, 6 pages, 2008.
- [15] N. Group, Touchlib: A multi-touch development kit, 2008 <http://nuigroup.com/touchlib/>.
- [16] N. Group, Community core vision—ccv, <http://ccv.nuigroup.com/>, 2010.
- [17] M. Kaltenbrunner and R. Bencina, “Reactivation: a computer-vision framework for table-based tangible interaction,” in *Proceedings of the 1st international conference on Tangible and embedded interaction (TEI '07)*, pp. 69–74, ACM, New York, NY, USA, February 2007.
- [18] M. Wright, A. Freed, and A. Momeni, “Opensound control: state of the art 2003,” in *in Proceedings of the 2003 conference on New interfaces for musical expression (NIME '03)*, pp. 153–160, National University of Singapore, Singapore, 2003.
- [19] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, 1st edition, 2008.
- [20] R. Bencina, “oscpack—a simple c++ OSC packet manipulation library,” 2006, <http://www.rossbencina.com/code/oscpack>.
- [21] R. Haralick and L. Shapiro, *Computer and Robot Vision*, vol. 1, Addison Wesley, 1992.