MVA2000 IAPR Workshop on Machine Vision Applications, Nov. 28–30, 2000, The University of Tokyo, Japan

6—3

Visual Screen: Transforming an Ordinary Screen into a Touch Screen

Zhengyou Zhang and Ying Shan
Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA

Abstract

Touch screens are very convenient because one can directly point to where it is interesting. This paper presents an inexpensive technique to transform an ordinary screen into a touch screen using an ordinary camera. The setup is easy: position a camera so it can see the whole screen. The system calibration involves the detection of the screen region in the image, which determines the projective mapping between the image plane and the screen. In order to compensate the non-flatness of the screen, an optional step can be applied by displaying a set of dots on the screen. In runtime, our system locates the indicator (finger tip in our current implementation) in the image and converts the image position to the cursor position on the screen. Finger gesture or key input executes appropriate actions. The system has been demonstrated to be quite accurate, and its applications include kiosk and kids entertainment.

1 Overview

There are several works [1, 2, 3] reported to use human finger as pointing device. Although the configurations of these systems differ in that [1] is using a wearable computer and [2, 3] are using camera-projector pairs, they are all assuming a plane to plane image-screen mapping. Since a large number of computer screens are not flat, this assumption does not apply in our case. We have proposed in this paper an image-screen mapping algorithm to correct the non-flatness of the computer screen. We have developed a segmentation method specially tailored for computer screen background. We have also developed a robust method to find the tip point location. The overall performance of the system is fast, accurate, and reliable.

The system setup essentially involves only positioning a camera so as to view the screen of a computer monitor. Ideally, the camera views the screen from a point along a line normal to the center of the screen. However, as this will likely interfere with the user who typically sits in front of the computer monitor, the camera can be shifted away from the normal line to get it out of the way of the user. The camera should not be moved too far away from the normal line, however, or errors will be introduced in the process. It has been observed that the camera can be positioned up to about 30 degrees off of the aforementioned normal line in any direction and still provide error-free performance.

Figure 1 is the diagram of the Visual Screen (VS in short) system. Four dash boxes represent four major parts of the system. From left to right, these boxes will be referred as Calibration block, Model Extraction I block, Main block, and Model Extraction II block, respectively. The Main block is the kernel of the system. Its functionality is to locate the tip point of the indicator and map its image coordinates to the screen coordinates. The task of tip point location contains two processes, i.e., to segment the indicator from the background, and to find the tip point of the indicator. The segmentation requires color models for both the background and the indicator. The Model Extraction blocks I, and II in Figure 1 are used to extract the background model and the foreground model, respectively. The Calibration block is used to establish the mapping between the image coordinates and the screen coordinates. This mapping is then used in the

Main block to find the corresponding screen coordinates for the tip point once its image coordinates are estimated.

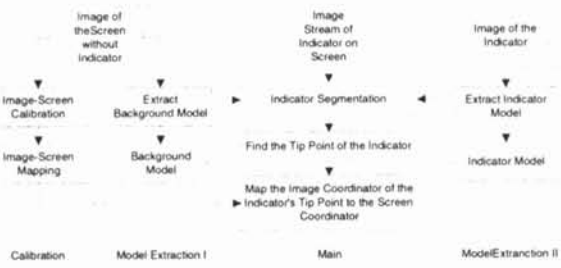

Figure 1: Diagram of the Visual Screen system. From left to right, four major functional parts of the system include calibration, extraction of a background model, extraction of a foreground model and a main processing block.

2 Accurate image-screen calibration

If the screen is flat, the plane perspectivity from the screen plane and its 2D projection on the image plane is described by a homography, a 3×3 matrix defined up to a scale factor. This matrix can be easily determined from 4 pairs of image-screen point correspondences. The correspondences are not difficult to obtain because we know the screen coordinates of four screen corners, and their corresponding image points can either be detected automatically or specified by the user.

In most cases, however, screens are not flat. This problem has been addressed as follows. First, we display on the screen a grid of circles (referred as calibration points hereafter), whose centers are known in the screen plane. A circle is usually projected in the image plane as an ellipse. We can easily compute the centroid of an ellipse. As the extent of an ellipse is small in our case, the centroid of an ellipse can be well considered as the projection of the center of the corresponding circle. Second, we compute a homography from the image-screen correspondence of the calibration points. Since the screen is actually not flat, the homography thus computed is just an approximation. Third, we map the calibration points on the image back to the screen (called the estimated calibration points), and compare the estimated calibration points with the original ones. The difference between the original and the estimated points then defines a residual vector on each grid point. This grid of residual vectors are then used to compensate the mapping errors caused by the non-flatness of the screen. Bilinear interpolation is used to compute the residual vectors of screen points not on the grid. Figure 4 demonstrates the calibration result.

3 Reliable color segmentation

It sounds difficult to separate the indicator from the background screen because its contents change frequently. However, it has been observed in our experiments that the images of screen pixels have some degrees of invariance in the color space. That is, they are dominated by a kind of blue color. This observation forms the base of our segmentation algorithm described as follows. We firstly compute a color model for the screen without the indicator. A number of pictures with rich color are displayed on the screen in order to make the model as general as possible. To compute this background model all of the pixels in the image are histogrammed-namely, for each pixel its color intensity is placed in the proper bin of a preferred possible 256 intensity levels. This is preferably done for each of the red, green and blue (RGB) channels thus generating three separate histograms. Alternately, one histogram could be generated using some joint space representation of the channels. Once the histogram has been computed, a Gaussian distribution for each histogram is calculated to provide the mean pixel intensity of the background and the variance.

Once the modeling of the background of the screen has been completed, the model for the indicator or pointer is computed in order to separate the indicator from the background. This is done by asking the user to select a polygonal bounding area displayed on the screen for the indicator of choice. Only the pixels inside this polygonal area are used to compute the color

model for the indicator. The computation is done in the same way the background model was produced. Usually the color model for the indicator will be dominated by a different color in color space than the background. Once a color model for the indicator has been determined, this model will not have to be recalculated unless a pointer with a significantly different color is employed.

Once both the screen background and indicator models are determined, a standard Bayes classifier (or the like) is used to segment the indicator from the screen background. If the extracted models of the foreground and background are split into separate RGB channels, the Bayes classifier determines the probability a given pixel color is a background pixel for each channel and these probabilities are multiplied together. The classifier also determines the probability a given pixel is a foreground pixel for each channel and multiplies the probabilities together. Next, the background pixel probability product is divided by the foreground pixel probability product. If this quotient is greater than one then the pixel is determined to be a background pixel, otherwise it is determined to be a foreground or indicator pixel. Figure 5 demonstrates the segment result of a hand, of which any finger could be an indicator.

4 Robust finger tip locating

It is not trivial to define the tip point of an indicator. What is really desired is the consistency, or the invariance of the definition. In our current implementation, the tip point is defined as the intersection of the indicator's center line and its boundary along the direction that the indicator is pointing towards. In our prototype system, we have simplified the definition by allowing only the upwards pointing direction. We have developed an algorithm to robustly find the center line of the indicator, as well as its intersection with the upper boundary of the indicator. The boundary of the indicator can be found easily from the segmentation result mentioned above.

The algorithm can be elaborated as the follows. A cumulative total of the number of pixels that belong to the foreground are calculated on a scan line by scan line basis starting at the top of the image containing the indicator. The resultant histogram will be referred as *horizontal histogram*. The horizontal histogram is next analyzed to determine the scan line where the foreground pixels first appear and increase in cumulative total thereafter (i.e., representing a step). The identified scan line roughly corresponds to where the indicator tip location may be found. Next, a number of lines above and below the identified line (e.g., 15 lines) are selected and each is scanned to find the start and end of the foreground pixels in the horizontal direction. In addition, the center point of each series of foreground pixels along each of the scan lines is determined and a line is robustly fit through these points. The pixel corresponding to the indicator tip location is then determined by scanning all pixels within the previously identified indicator window (e.g., 15 lines) to find the boundary pixels. The pixel corresponding with the tip of the indicator is the boundary pixel where the previously determined centerline intersects the boundary of the indicator. Figure 2 demonstrates the result of tip point location.

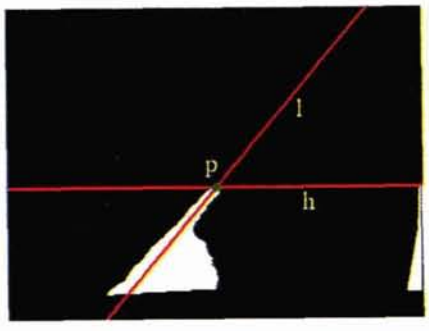

Figure 2: Tip point location. Line l is the center line of the indicator, and point p is the tip point. Both of them can been reliably found with reasonable accuracy. Line h indicates the scan line where the foreground pixels first appear in the direction of the y-axis. The horizontal histogram is shown on the right side of the image

The stability of the finger tip location is further

improved by filtering the location over time with a Kalman filter.

5 Experiment results

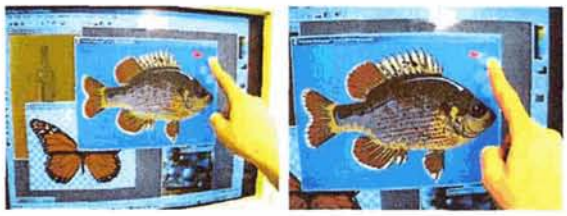

Figure 3: Photo editing with VS system. In the left image, the paint brush has been moved by finger tip to the place where the user wants to put the bubble. The right image shows the bubble. The paint brush is highlighted by an arrow in both images

The VS system is implemented on a Pentium II 450MHz machine with an average frame rate of 19fps. The system is designed to work in conjunction with the native mouse. The native mouse is used to control the cursor as usual when no indicator is detected within the screen area. When an indicator is detected, it takes control of the cursor over the mouse. Figure 3 shows our system working on a photo editing environment (Paint Shop Pro) where the finger tip is used to control the paint brush. Despite the complicated background, which is frequently encountered in graphical systems, the VS system is able to control the brush robustly and consistently. Actions like left-button click is now simulated by key strokes. They will be triggered by a gesture recognition subsystem in the next version of the VS system.

6 Conclusion

We have introduced a system and method for turning a regular computer monitor screen into a touch screen using an ordinary camera. It includes an image-screen mapping procedure to correct for the non-flatness of the computer screen. It also includes a segmentation method to distinguish the foreground from the background of a computer screen. Additionally, this system and method includes a robust technique of finding the tip point location of the indicator (such as the finger tip). The screen coordinates of the tip points are then used to control the position of the system indicator.

References

[1] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. W. Picard, and A. Pentland. Augmented reality through wearable computing. *Presence, Special Issue on Augmented Reality*, 6(4), 1997. Also as MIT Media Lab Technical Report TR-397.

[2] C. Maggioni and B. Kammerer. Gesturecomputer - history, design and applications. In Ed. R. Cipolla and A. Pentland, editors, *Computer Vision for Human-Machine Interaction*. Cambridge University Press, 1998.

[3] J. Coutaz, Crowley, J. L., and F. B rard. Things that see: Machine perception for human computer interaction. *Communications of the ACM*, 43(3):54–64, 2000.

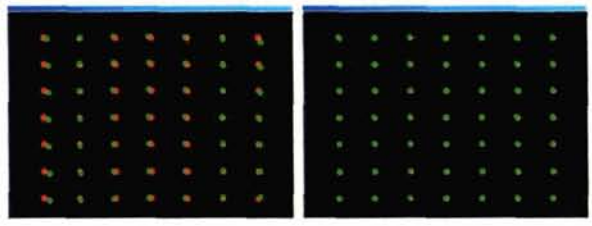

Figure 4: Calibration result. The left image shows the original calibration points (in red) and the estimated points (in green) with homography only. The right image shows the result with residual vector interpolation. The error of image-screen mapping is largely reduced in the right image

Figure 5: Segmentation result. Left image shows a picture of a hand on the screen. Indicator could be any finger of the hand. Right image shows the segmented hand. Note that the segmentation result is not affected by the complicated screen background