In *Proc. International Symposium on the CREST Digital Archiving Project*
pages 83–106, May 23–24, 2003, Tokyo, Japan.

Vision-based Interaction with Fingers and Papers

Zhengyou Zhang

Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
e-mail: zhang@microsoft.com
http://research.microsoft.com/users/zhang/

Abstract

This paper presents two vision-based interface systems. The first, VISUAL SCREEN, uses 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 runtime, our system locates the tip pointer (fingertip in our current implementation) in the image and converts the image position to the cursor position on the screen. The second system, VISUAL PANEL, extends the previous system for mobile applications. It employs an arbitrary quadrangle-shaped panel (e.g., an ordinary piece of paper) and a tip pointer as an intuitive, wireless and mobile input device. The system can accurately and reliably track the panel and the tip pointer. The panel tracking continuously determines the projective mapping between the panel at the current position and the display, which in turn maps the tip position to the corresponding position on the display. The system can fulfill many tasks such as controlling a remote large display, and simulating a physical keyboard. Users can naturally use their fingers or other tip pointers to issue commands and type texts. Furthermore, by tracking the 3D position and orientation of the visual panel, the system can also provide 3D information, serving as a virtual joystick, to control 3D virtual objects.

Keywords: Vision-based user interface, VISUAL SCREEN, VISUAL PANEL, new input device, new control device, virtual touch screen, virtual mouse, virtual keyboard, virtual joystick, plane projectivity.

1 Introduction

In many intelligent environments, instead of using conventional mice, keyboards and joysticks, people are looking for an intuitive, immersive and cost-efficient interaction device. We describe two vision-based interface systems. The first, VISUAL SCREEN, uses an inexpensive technique to transform an ordinary screen into a touch screen using an ordinary camera, and a user can use his/her finger to interact with the computer. The second, called VISUAL PANEL, employs an arbitrary quadrangle-shaped panel (e.g., an ordinary piece of paper) and a tip pointer (e.g., fingertip) as an intuitive input device.

There are several works [22, 13, 3] reported to use human finger as pointing device. Although the configurations of these systems differ in that [22] is using a wearable computer and [13, 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.

We can find many applications where this type of vision-based interfaces is desired. For an instance, in a smart room, the user wants to control a remote and large display or play a game, but he/she is in a sofa instead of in front of a computer, and therefore the mouse and keyboard or joystick may not be accessible. Then, what could he/she do? He/she may pick up an arbitrary paper at hand and move his fingers or pens on the paper to drive a cursor or to type some text, or move the paper to control the game. Certainly, such an interaction is made possible by having a camera look at the user and analyzing the movement of the paper and the user.

For another example, several people are discussing in a meeting room using a large display. They may need to draw some pictures to show their ideas. However, it is unrealistic to facilitate every user a mouse and a keyboard. What could they do? Again, they may pick up any paper and use their fingers to draw their ideas which will be shown in the large display. By this means, a more immersive discussion can be achieved.

Even more, in a large lecture room, the lecturer sometimes needs to write down something on a small whiteboard. However, the audience far from him or remote audience may not be able to see clearly what he writes. Due to the constraints of the bandwidth, it would not be feasible to broadcast the video of the writing. In this situation, a vision-based system is needed to analysis what the lecturer writes and retrieve it in remote displays.

In such scenarios as smart rooms, immersive discussions and tele-conferencing, conventional mice and keyboard turn out to be not suitable, which motivates the development of a vision-based gesture interface. There have been many implemented application systems ([2, 8, 10, 30, 28, 27, 1, 5, 11, 17, 18, 25, 29, 12, 21, 26, 7, 15, 20, 24] to cite a few).

Touch screens are very convenient because one can directly point to where it is interesting. No conventional mice are needed. The first prototype we will present is called Visual Screen. It uses an inexpensive technique to transform an ordinary screen into a touch screen using an ordinary camera, and the user can interact with the computer directing using his/her fingers.

The second prototype system, called Visual Panel, takes advantage of an arbitrary quadrangle-shaped planar object as a panel such that a user can use any tip pointer such as his fingertip to interact with the computer. By observing the tip pointer on the panel, we achieve accurate and robust interaction with the computer. The 3D pose of the panel can also be used to control the display of 3D objects.

The remainder of the paper is organized as follows. In Sect. 2, we first present the plane perspectivity, which describes the relationship between a plane in space and its image. This is the mathematical foundation of both systems because the computer screen/display and the panel are assumed to be planar, although non-planarity, as will also be shown, can be corrected. In Sect. 3, we provide an overview of the Visual Screen system, together with camera-screen calibration and other technical details. In Sect. 4, we first give an overview of the Visual Panel system and its main components, and then provide the details of the techniques used for detecting and tracking the panel and the tip pointer and for determining the pose of the Visual Panel. Four applications built on top of the system are finally presented. Section 5 concludes the paper.

2 Plane Perspectivity: Mapping Between A Rectangle in Space And Its Image

Since we use an arbitrarily rectangle-shaped panel to control the cursor position on the remote display, we have to know the mapping between a point on the panel and a point on the display. Furthermore, what is available is an image sequence of the panel which may undergo arbitrary motion (as long as the image of the panel does not degenerate into a line or a point), so we also need to know the mapping between a point in the image plane and a point on the panel. We assume the camera performs a perspective projection (pinhole model) [6]. As the display, the panel, and the image plane are all planes, both above relationships can be described by a *plane perspectivity* [19], as to be explained below.

Given a point $\mathbf{p} = [x, y]^T$ on a plane Π, we use $\tilde{\mathbf{p}} = [x, y, 1]^T$ to denote its homogeneous coordinates. Then, the plane perspectivity between planes Π and Π' is described by a 3×3 matrix $\mathbf{H}$ such that

$$\lambda \tilde{\mathbf{p}}' = \mathbf{H}\tilde{\mathbf{p}} \tag{1}$$

where λ is an arbitrary non-zero scalar. This implies that the homography matrix is only defined up to a scale factor, and therefore has 8 degrees of freedom. If four couples of corresponding points (no three of them are collinear) are given, the homography matrix can be determined (see, e.g., [31]).

It is not difficult to see that the composition of two plane perspectivities is still a plane perspectivity. Thus, the mapping between the image of the panel and the remote display can be described by a homography matrix. This is very important because what we really need is to use the detected tip position in the image to control the cursor position on the remote display. (If we instead estimate the above-mentioned two homographies, additional calibration is necessary, making the system setup unnecessarily more complicated.)

The composed homography can be easily determined once the four corners of the panel are located in the image. As we know the dimension of the display, we compute the homography by mapping each corner of the panel to a corner of the display. Thus, the position of the four corners of the panel is the only thing we need to perform the mapping. As the panel can be detected and tracked easily and robustly, as to be described in the next section, the camera can also move dynamically to achieve a higher degree of mobility.

3 The *Visual Screen* System

3.1 Overview

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.

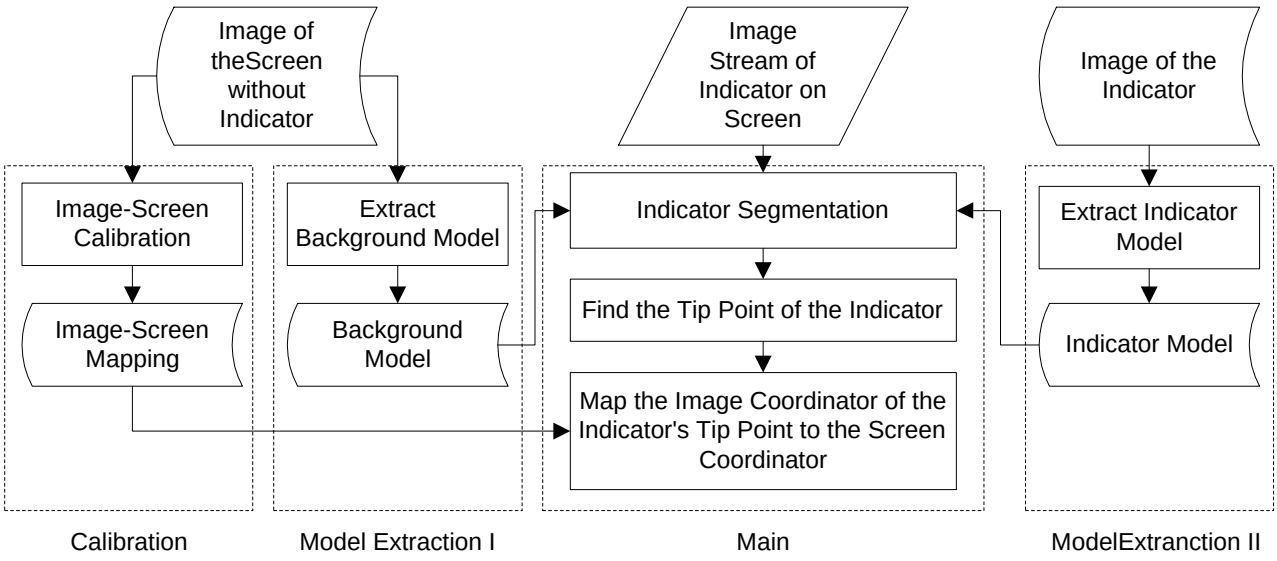


Figure 1: Diagram of the Visual Screen system. From left to right, four major functional parts of the system include calibration, extraction of a background model, extraction of a foreground model and a main processing block.

3.2 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.

3.3 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.

3.4 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.

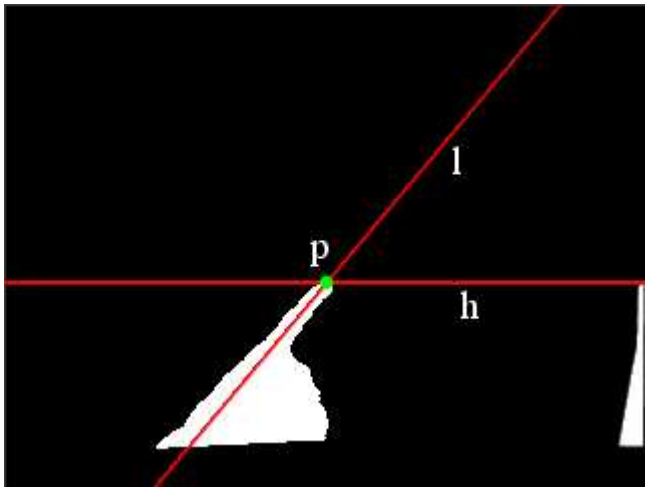


Figure 2: Tip point location. Line l is the center line of the indicator, and point p is the tip point. Both of them can been reliably found with reasonable accuracy. Line h indicates the scan line where the foreground pixels first appear in the direction of the y-axis. The horizontal histogram is shown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

3.5 Experiment results

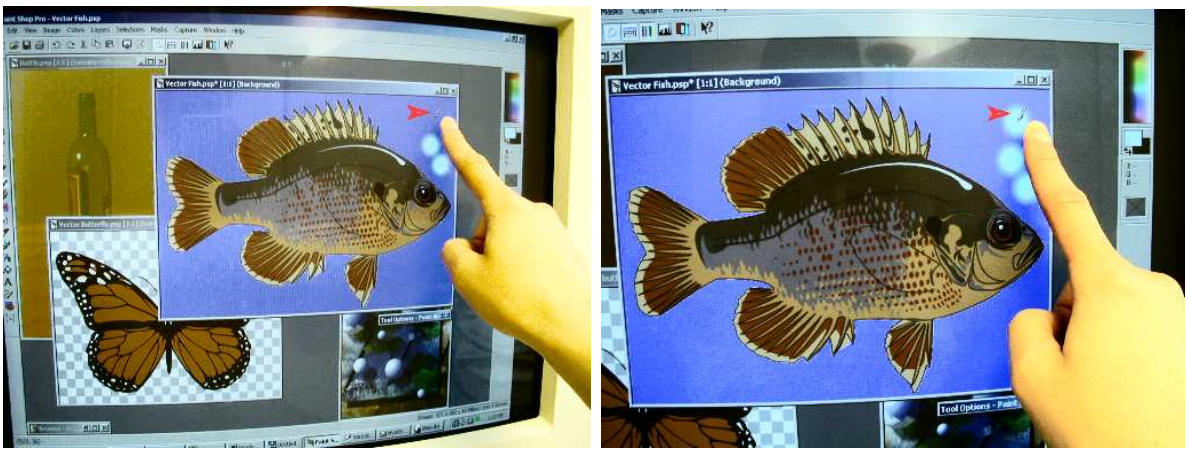


Figure 3: Photo editing with VS system. In the left image, the paint brush has been moved by finger tip to the place where the user wants to put the bubble. The right image shows the bubble. The paint brush is highlighted by an arrow in both images

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.

4 The *Visual Panel* System

4.1 Overview

Figure 6 shows the basic idea of visual tracking in the Visual Panel system. Figure 6a is one frame of the video input, and Figure 6b shows the tracking result of the panel and the fingertip. The quadrangle can be mapped to a remote display, and the mapping is a homography. As to be explained later, the homography describes a plane perspectivity because the panel and the display both are planar. The position of the fingertip is then mapped accordingly, and can be used, for example, to control the cursor on the display, thus serving as a virtual mouse.

The system consists of panel tracker, tip pointer tracker, homography calculation and update, and action detector and event generator. It can simulate both mouse and keyboard. The whole system is shown in Figure 7.

Video sequences are analyzed by a panel tracker and a tip pointer tracker. As already described in the last section, the panel tracker can accurately track an arbitrary quadrangle-shaped plane object by outputting the positions of the four corners. Since their positions are calculated in sub-pixels, we can accurately compute the homography, which describes the

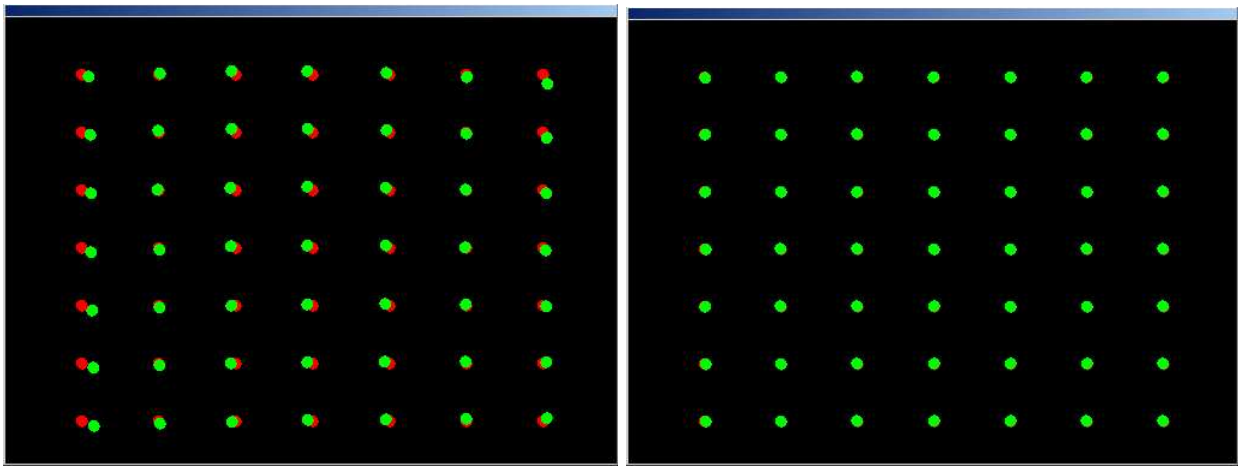


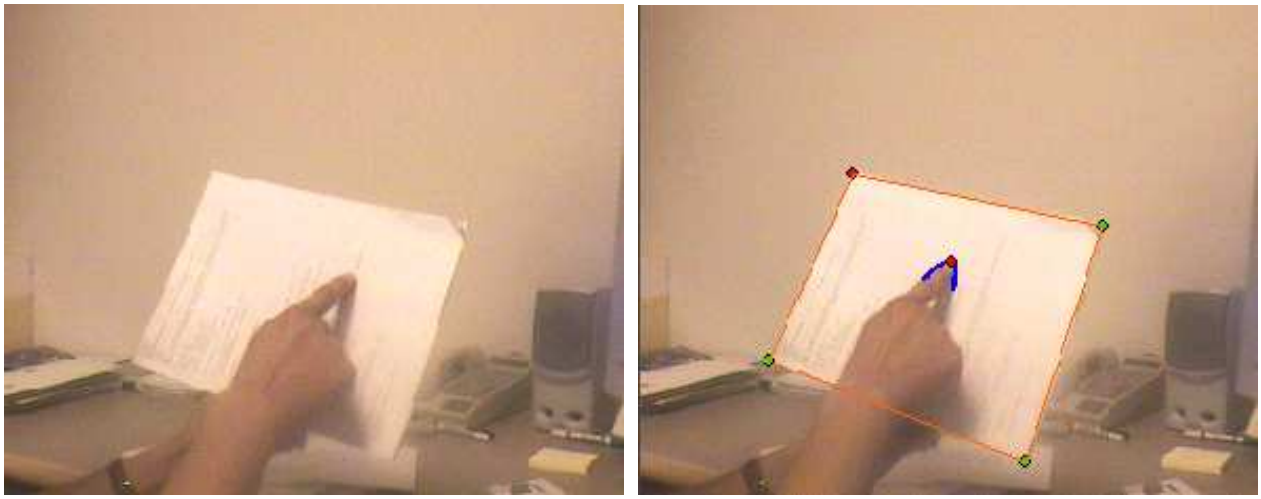
Figure 4: Calibration result. The left image shows the original calibration points (in red) and the estimated points (in green) with homography only. The right image shows the result with residual vector interpolation. The error of image-screen mapping is largely reduced in the right image

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

mapping between the panel and a remote display. Through the homography, any point on the panel is mapped to the corresponding position on the remote display.

The panel detector, also described in the last section, can automatically detect the panel when it just enters the camera's field of view, or recover the panel if its tracking is lost due to abrupt motion.

In the Visual Panel system, users can use their fingertip as a mouse to simulate a cursor for the remote display. This requires an accurate and stable tracking of the fingertip, because a small error in the tip position will be magnified in the remote large screen. To see why, let us assume the resolution of the input video is 320×240 and that of the remote display is 1024×768. Since the panel usually occupies about half of the image, it is obvious that a tracking error of 1 pixel will incur an error of about 6 pixels on the display, making the mapped cursor position very shaky. This problem is solved in our system by representing a tip pointer



(a) (b)

Figure 6: The tracking in the VISUAL PANEL system. (a) an input image (b) tracking result: a tracked panel and a tracked fingertip.

as a conic and fitting a parametric conic to image observations. Therefore, the tip position is also calculated in sub-pixels.

The fingertip detector automatically detects the fingertip when it is posed on the panel. Both fingertip tracking and detection will be described later.

The current system simulates the clicking/pressing gestures by holding the tip pointer on the same position for a while. The event generator reads input from the action detector, and issues various mouse and keyboard events. More details will be given later.

We describe below several system issues:

- **Camera Setting**: The setting is quite flexible. It can be anywhere as long as the panel is not significantly occluded. In our current implementation, a fixed camera is mounted on the ceiling. The user can rotate, translate and tilt the panel to reach a comfortable pose for use. Under circumstances where more mobility is necessary, we can use a pan-tilt-zoom camera or mount a camera on top of his head by wearing a hat, on his glasses, or on his shoulders, such that the user can be anywhere to interact with the computer. This would be quite useful, for example, for a speaker who gives a presentation while walking around.
- **Panel Design**: The panel can be anything as long as it is quadrangle-shaped. For example, we can use a piece of white paper or a cardboard, which is widely available in offices and at homes. Because we rely on a homography mapping, we should not bend the quadrangle-shaped object during operation.
- **Tip Pointers**: The system allows arbitrary tip pointers, such as fingertips and pens, as long as their color is distinguishable from the panel's color. In our usability studies, many users prefer pens to fingertips in some applications like finger painting, because pens are more intuitive for them, although they have fun to use fingertips.
- **Clicking**: The current VISUAL PANEL system simulates clicking and pressing by holding the tip pointer in the same position for a short period of time. We are exploring the possibility of using some natural gestures.

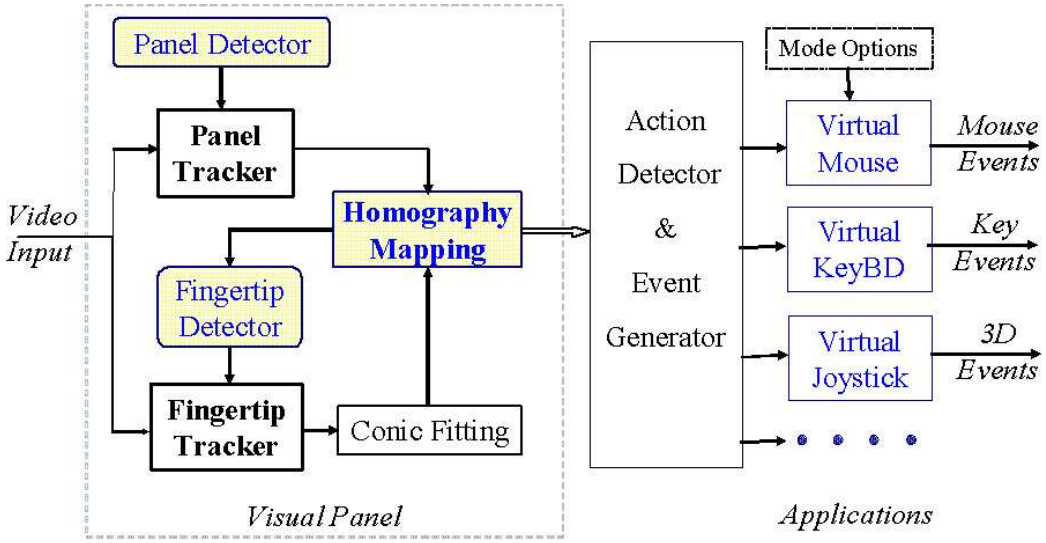


Figure 7: The system of VISUAL PANEL, which consists of panel tracker, pointer tracker, action detector and event generator.

Building on top of these techniques, our system is capable of performing two types of input: virtual mouse and virtual keyboard, as will be shown in the application section. As a virtual mouse, the position of the tip pointer is mapped onto the remote display to simulate a cursor. We can also use a paper with a keyboard pattern printed on it as a virtual keyboard, with which a user can point the keys on the paper to input text.

Furthermore, our system can track the 3D position and orientation of the visual panel, and the 3D information can be used to control 3D virtual objects. Therefore, our system can also serve as a virtual joystick.

A video filming a live demonstration of an older version of this system is also submitted. We have not yet had time to record a demonstration of the newer version including the 3D control functionality (virtual joystick), but we plan to do it for the PUI. The whole system runs at close to 29 frames per second.

Let us now look at each component in more details.

4.2 Visual Panel Detection and Tracking

We use an arbitrary rectangular object such as a piece of paper as our visual panel. Its projection in the image is a quadrangle.

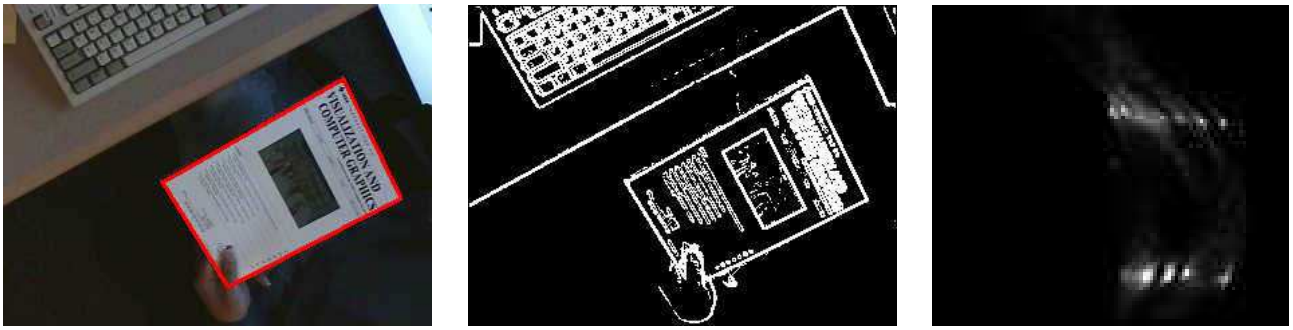
4.2.1 Quadrangle Representation

The image of the panel can be represented by a quadrangle:

$$\mathcal{Q} = \{l_1, l_2, l_3, l_4\} \tag{2}$$

where l_i is a side line. It can also be represented by the four corners $\mathcal{Q} = \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4\}$ with $l_k = \mathbf{q}_{k-1}\mathbf{q}_k$ (we assume $\mathbf{q}_0 = \mathbf{q}_4$).

Each side of the quadrangle in the image is expected to be a set of edge points due to the difference between the panel and the background. We model the appearance of each side as a



(a) (b) (c)

Figure 8: Automatic quadrangle detection. (a) Original image with detected quadrangle overlaid as green lines; (b) Edges image obtained with Sobel detector; (c) Hough space.

random vector $\mathbf{x} = \{G, I\}$, where G is the average gradient and I is the average intensity. The distribution of $\mathcal{X}$ is assumed to be a Gaussian, i.e., $\mathbf{x} \sim N(\mu_x, \Sigma_x)$. More richer modeling of the appearance is under investigation.

4.2.2 Automatic Detection

We have developed a simple technique based on Hough transform [9] to automatically detect a quadrangle in an image. Take the image shown in Fig. 8a as an example. A Sobel edge operator is first applied, and the resulting edges are shown in Fig. 8b. We then build a 2D Hough space for lines. A line is represented by (ρ, θ), and a point (u, v) on the line satisfies $\cos(\theta)u + \sin(\theta)v - \rho = 0$. An edge point with orientation is mapped into the (ρ, θ) space. In our implementation, θ is divided into 90 intervals from $-90°$ to $90°$, and ρ is divided into 100 intervals from range from $-d$ to d, where d is the half of the image diagonal. The Hough space for the edges in Fig. 8b is shown in Fig. 8c.

We then examine the strong peaks in the Hough space whether four of them form a reasonable quadrangle. By "reasonable", we mean:

- the neighboring sides should differ at least by $20°$ in orientation;
- the opposite sides are close to be parallel (the orientation difference is less than $20°$);
- the opposite sides are not close to each other (at least 40 pixels of difference in ρ); and
- there are indeed a large number of edges on the quadrangle.

The last test is necessary because a point in the Hough space corresponds to an infinite line, and a quadrangle formed by 4 lines may not correspond to any physical quadrangle in an image. The quadrangle detected in Fig. 8a is shown with red lines on the image. Our current implementation of quadrangle detection achieves 22 frames per second for image resolution 320×240 on a PC III 1G Hz.

4.2.3 Tracking Through Dynamic Programming

At time frame t, the location of the quadrangle is at $\mathcal{Q}(t) = \{\mathbf{q}_1(t), \mathbf{q}_2(t), \mathbf{q}_3(t), \mathbf{q}_4(t)\}$, and the appearance of the quadrangle is $\mathbf{x}(t)$. The tracking can be formulated as a MAP (maximum a posteriori) problem:

$$\mathcal{Q}^*(t+1) = \arg\max_{\mathcal{Q}} p(\mathcal{Q}(t+1)|\mathcal{Q}(t), \mathbf{x}(t), \mathbf{x}(t+1))$$

Because the panel motion between successive image frames is limited, we assume at time $t+1$ these four corner points will be in a range D_i around $p_i(t)$, respectively. The above problem can then be approximated by

$$\mathcal{Q}^*(t+1) = \arg\max_{\mathcal{Q}} p(\mathcal{Q}(t+1), \mathbf{x}(t+1) | \mathcal{Q}(t), \mathbf{x}(t) : \{D_1, D_2, D_3, D_4\})$$

Here, ":" means that $\{D_1, D_2, D_3, D_4\}$ are parameters for the probability. Obviously, this is a formidable searching problem. To illustrate this (see Figure 9), we assume the size of each search area of D_i is N. The complexity of the exhausted search for this problem is $O(4^N)$. However, since the four sides of the quadrangle are sequentially connected, this problem can be solved by the *dynamic programming* technique [23].

$$\begin{aligned}
&\mathcal{Q}^*(t+1) \\
&\quad = \arg\max_Q \textstyle\sum_{i=1}^4 p(\mathcal{Q}(t+1), \mathbf{x}_i(t+1) | \mathbf{x}_i(t), \mathcal{Q}_i(t) \\
&\qquad\qquad : D_i(\mathbf{q}_i(t), \mathbf{q}^*_{i-1}(t))) \\
&\quad = \arg\max_{\{\mathbf{q}_i\}} \textstyle\sum_{i=1}^4 p(\mathbf{x}_i(t+1) | \mathbf{x}_i(t), \mathbf{q}_i(t), \mathbf{q}^*_{i-1}(t))
\end{aligned}$$

That is, we try to estimate each side of the quadrangle sequentially by maximizing a suitable criterion (see below).

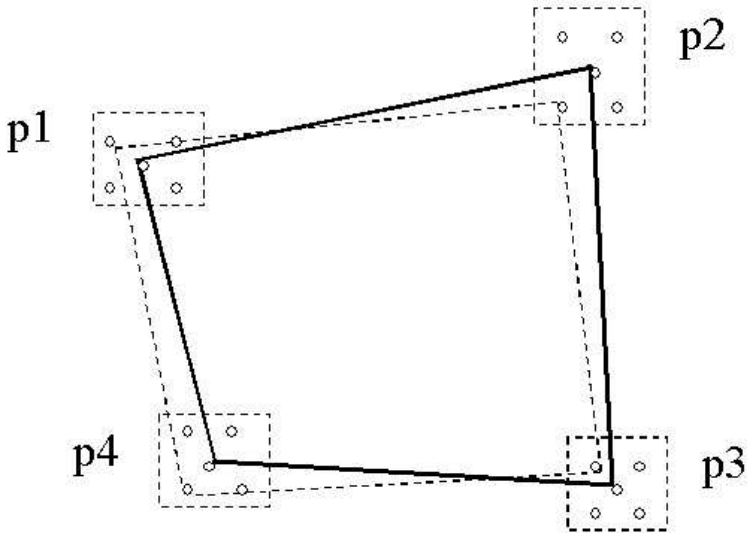


Figure 9: Tracking a quadrangle by dynamic programming technique

In our implementation, in order to reduce further the computational complexity, we do not search for the corners directly, where each corner should be examined in a 2D region. Instead, we search side lines, where the search region of each corner can be approximated by a line segment. Once side lines are determined, corner points are computed from the intersection of these lines.

As mentioned earlier, the appearance of each side line of the quadrangle is modeled by $\mathbf{x}$ that contains both the gradient information and the color information. Maximizing the probability in (3) implies to finding a pair of line segments between t and $t+1$ such that their appearances are closest. This can be done by minimizing the relative entropy between their distributions [4].

Assume Gaussian distribution of X and Y, then the relative entropy:

$$\begin{aligned} D(X||Y) &= E[\lg \frac{p_x(u)}{p_y(u)}] = \int p(u) \lg \frac{p_x(u)}{p_y(u)} du \\ &= \frac{d}{2} \lg \frac{|\Sigma_y|}{|\Sigma_x|} - \frac{1}{2} + \frac{1}{2} E[(x-\mu_y)' \Sigma_y^{-1} (x-\mu_y)] \\ &= \frac{d}{2} \lg \frac{|\Sigma_y|}{|\Sigma_x|} - \frac{1}{2} + \frac{|\Sigma_y|}{2|\Sigma_x|} + \frac{1}{2} (\mu_x - \mu_y)' \Sigma_y^{-1} (\mu_x - \mu_y) \end{aligned}$$

Thus, we have a symmetric distance metric:

$$\begin{aligned} D(X,Y) &= 2(D(X||Y) + D(Y||X)) \\ &= \frac{|\Sigma_y|}{|\Sigma_x|} + \frac{|\Sigma_x|}{|\Sigma_y|} + (\mu_x - \mu_y)'(\Sigma_x^{-1} + \Sigma_y^{-1})(\mu_x - \mu_y) - 2 \end{aligned}$$

By this means, we can find the best-matched line at time $t+1$ by:

$$l_i^*(t+1) = \arg \min_{\{\mathbf{q}_i, \mathbf{q}_{i-1}\}} D(\mathbf{x}(t), \mathbf{x}(t+1) : \{\mathbf{q}_i, \mathbf{q}_{i-1}\}) \tag{3}$$

Note that, because our panel tracking is based on locating the side lines, it is very robust to occlusion. It works well even when a significant fraction of a side line, including the corners, is occluded by, for example, hands, or moves out of the camera's field of view. Obviously, the tracking fails when a whole side line is occluded or invisible, and in this case the quadrangle detection algorithm described in the last subsection is activated.

Our current implementation of quadrangle tracking achieves 29.5 frames per second.

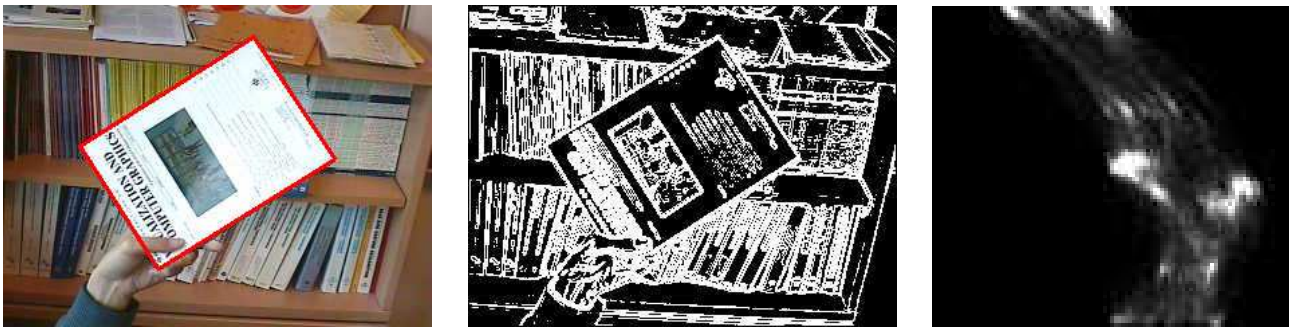
4.2.4 An Example

Figures 10 and 11 show another tracking sequence with different background. Figure 10 shows the automatic detection result, while Figure 11 shows a few sample results of the tracking under various situations. Note that this sequence is quite difficult since the background contains books of similar color and there are a large number of edges. Note also that we have not used any background subtraction or frame difference technique to reduce the background clutter. As can be observed, our technique tracks very well under perspective distortion, illumination change, partial disappearance, size change, and partial occlusion.

4.3 Fingertip Detection and Tracking

The tracking of a tip pointer is quite intuitive. Assume the position of the tip at time t is $\mathbf{p}(t)$. Kalman filtering technique is employed to predict the tip position $\bar{\mathbf{p}}(t+1)$ at time $t=1$. In a small window, say 30×30, we identify as many as possible edge pixels that probably belong to the edge of the tip by thresholding the gradient and taking advantage of color of previous edge of tracked tip. After that, we fit a conic to those pixels and identify the extreme point of the conic to be the tip position $\mathbf{p}(t+1)$ for time $t+1$. In this way, we achieve subpixel precision for tip location. The combination of quadrangle tracking and tip tracking runs at close to 29 frames per second.

Since a tip pointer is on/off the panel frequently, the system should have the capability of detecting the tip pointer automatically when it appears on the panel. We have developed a technique through dynamic background subtraction, which is illustrated in Fig. 12.



(a) (b) (c)

Figure 10: Another example of automatic quadrangle detection. (a) Original image with detected quadrangle overlaid as red lines; (b) Edges image obtained with Sobel detector; (c) Hough space.

(a) (b) (c)

(d) (e) (f)

Figure 11: Sample results of a tracking sequence under various situations.

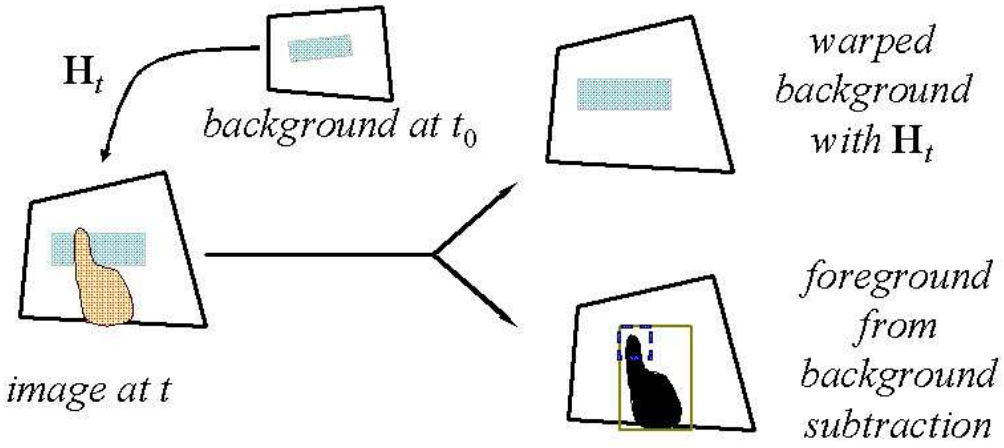


Figure 12: Detecting tip pointer. The foreground, i.e. hand, can be segmented out from the background, since the current position of the panel is tracked and a background template is maintained.

Assume at the beginning of the application, the panel $\mathcal{Q}(0)$ at time 0 is detected, and there is no tip pointer on the panel. We save the image as I_0. At time t, since the system tracks the panel position $\mathcal{Q}(t)$, the homography $\mathbf{H}(t)$ between $\mathcal{Q}(0)$ and $\mathcal{Q}(t)$ can be easily calculated. Through the homography $\mathbf{H}(t)$, the pixels $\mathbf{p}_t(0)$ in I_0 are mapped to the panel at time t as $\mathbf{p}_b(t)$ by:

$$\tilde{\mathbf{p}}_b(t) = \mathbf{H}(t)\tilde{\mathbf{p}}_t(0)$$

We thus have a warped image $I_0(\mathbf{p}_b(t))$. This virtual background image is what the panel should look like if there is no tip pointer. Subtracting $I_0(\mathbf{p}_b(t))$ from the current image gives us a difference image. The tip pointer is likely located in areas with large color difference. A mask image is computed for the foreground, and the most distant pixel from the mask centroid is considered to the tip. Figure 12 shows the basic idea of our approach.

4.4 Action Detection and Two Mouse Pressing Modes

Our system has two mouse button pressing modes: mode I (clicking mode) which simulates the left button down then up automatically and mode II (dragging mode) which simulates the left button down until released. In our current implementation, clicking/pressing is simulated by holding the tip pointer for a short period of time, say, 1 second, and a beep sound is generated as a feedback to indicate that an event is activated. This is illustrated in Fig. 13, where the horizontal axis indicates the time and the vertical axis for the top row indicates the tip location (i.e., trajectory).

A variable S with two states (UP and DN) is maintained to simulate the two natural states of a button. The variable S is initialized to be UP. In the clicking mode (mode I), when the system detects that the tip pointer has been at a fixed place for a predefined amount of time, the state variable S is set to DN. After 0.1 second, the state variable S will be automatically set to UP to simulate button release. Appropriate mouse events are then generated, and a clicking action is performed.

Obviously, in clicking mode (mode I), the ability of dragging is very limited, since the release is automatic. To simulate dragging, mode II uses another state variable D to memorize the

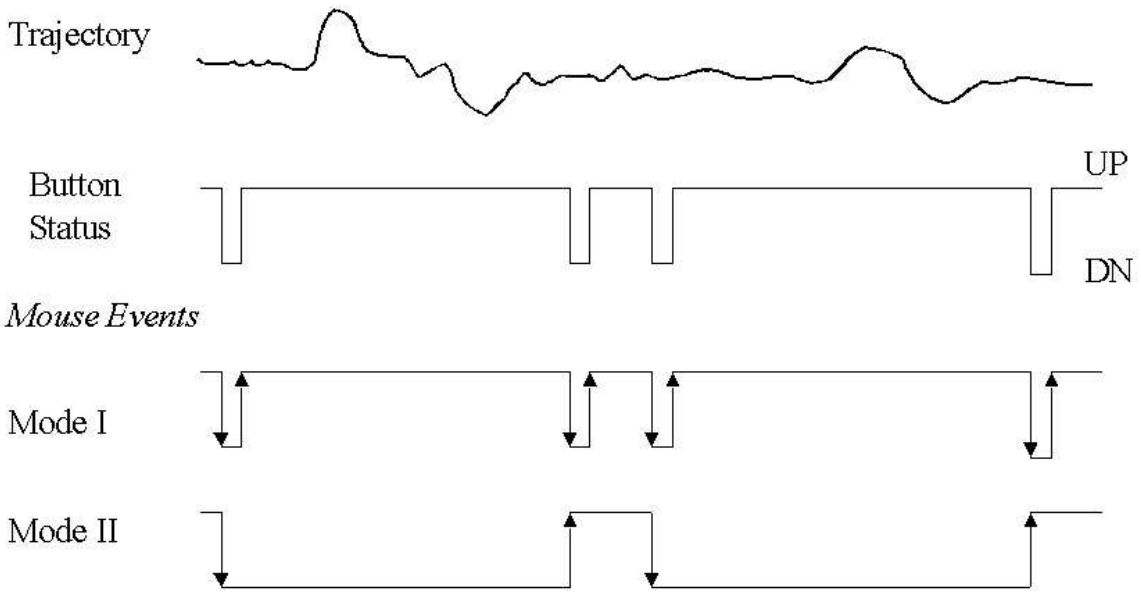


Figure 13: Simulating clicking (mode I) and dragging (mode II)

flip of clicking. When the system detects that the tip pointer has been at a fixed place for a predefined amount of time, variable D changes its state. When the D-state changes from UP to DN, a pressing event is triggered; when the D-state changes from DN to UP, a releasing event is triggered. Thus, we can pick up a window and drag it to a different place.

Note that the clicking event can also be triggered in the dragging mode if the pointer tip stays in the same location twice longer.

In our current implementation, an icon is provided in the menu bar. By clicking on that icon, the system switches between the dragging and clicking modes. Because of noise, there are some small jitters in the tip location during the wait time for activating an event. We consider that the tip is immobile if the jitters are within 2 pixels.

4.5 3D Pose Estimation

In this section, we describe how the 3D pose of the Visual Panel and the camera parameters are determined and how that information is used to control the visualization of a 3D object. The intrinsic parameters of a camera can be calibrated in advance with many different techniques such as the one described in [31]. In our current implementation, we use a simplified camera model, and calibrate the camera using the known size of the Visual Panel.

We assume that the principal point of the camera is at the center of the image, the aspect ratio of the pixels is known (1 in our case), and the pixels are squared (no skew), so the only unknown camera parameter is the focal length f. Let $\mathbf{R}$ and $\mathbf{t}$ be the rotation matrix (defined by 3 parameters) and the translation vector (also defined by 3 parameters) of the Visual Panel with respect to the camera coordinate system. We assume that the width w and height h of the Visual Panel are known, so the coordinates of the four corners, $\mathtt{Q}_i$ $(i = 1, \ldots, 4)$, can be defined in a coordinate system attached to the Visual Panel as $[0, 0, 0]^T$, $[w, 0, 0]^T$, $[w, h, 0]^T$ and $[0, h, 0]^T$. As described in Sect. 4.2, we detect/track the four corners in the image which are denoted by $\mathbf{q}_i$ $(i = 1, \ldots, 4)$. The relationship between $\mathtt{Q}_i$ and $\mathbf{q}_i$ is described by the

perspective projection model:

$$s \begin{bmatrix} \mathbf{q}_i \\ 1 \end{bmatrix} = \mathbf{A}[\mathbf{R}\ \mathbf{t}] \begin{bmatrix} \mathbf{Q}_i \\ 1 \end{bmatrix} \quad \text{with } \mathbf{A} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

where s is a non-zero scale factor and (u_0, v_0) are the coordinates of the principal point. Eliminating s yields two scalar equations. Since we have 4 points, we have in total 8 equations. Because we only have 7 parameters (focal length + the 6 pose parameters), a least-squares solution can be obtained for the focal length by minimizing the errors in the image space. In order to achieve higher accuracy, we track the VISUAL PANEL through 30 frames and estimate the focal length using all the images, again in least-squares.

Once the focal length is known, the pose $(\mathbf{R}, \mathbf{t})$ of the VISUAL PANEL at each time instant can be determined based on the same equation (4). We still have 8 equations, but only 6 unknowns (the pose parameters). A least-squares solution can be obtained by minimizing the errors in the image space. The computation time for pose estimation is negligible.

To control the visualization of a 3D object, we use the relative pose of the VISUAL PANEL at the current time instant with respect to the pose when it was detected.

4.6 Sample Applications

Based on the VISUAL PANEL system, several applications are made to demonstrate the capacity of the system. A video filming a live demonstration of an older version of the system (without automatic panel detection and the virtual joystick functionality), is available at URL:
`research.microsoft.com/~zhang/VisualPanel/video.avi`
The sound track is not edited, and a beep signals a Windows event is generated.

In this section, we explain four applications: control a calculator, draw a picture with a finger, input text without using any keyboard, and control a 3D object with the Visual Panel.

4.6.1 Controlling a Calculator

This application demonstrates the accuracy and stability of the VISUAL PANEL system. The Calculator, with around 30 buttons, takes a very small part area of the display. In this demo, a user can freely use his fingertip to click any buttons or menus of the Calculator. A snapshot is shown in Fig. 14 where the cursor is positioned on the button "×" (multiplication). The tracking error is less than 1 pixel, and the motion of the cursor is very smooth. Indeed, our system can be used as a virtual mouse to control any Windows application.

4.6.2 Finger Painting

This application demonstrates different mouse button pressing modes. A user can now use his finger to select tools and draw anything with `Paint`, a standard Windows application. Our usability study shows that users learn quickly how to draw a picture and control the remote display with their finger using our VISUAL PANEL system. Figure 15 shows a snapshot of the display while a user was finishing painting "hello world". The left window displays the panel and the hand viewed from the camera.

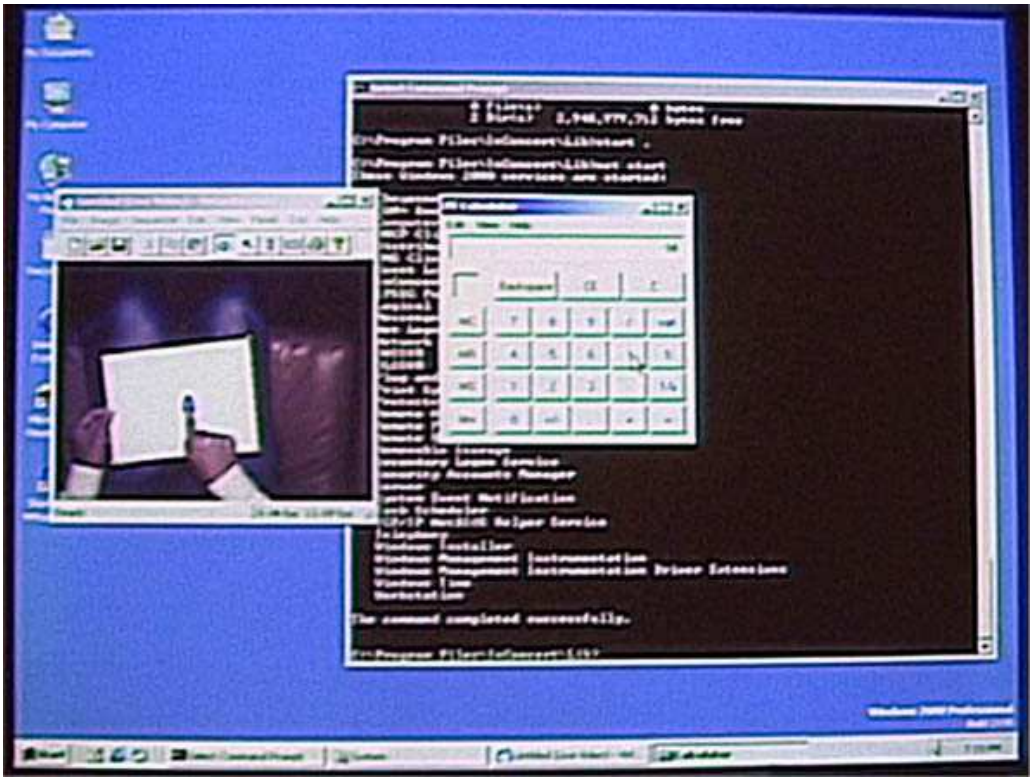


Figure 14: Controlling a calculator.

Figure 15: Finger painting.

4.6.3 Virtual Keyboard

This application demonstrates that the physical keyboard can be replaced by a printed virtual keyboard in our Visual Panel system. We print a keyboard pattern on the panel, which is shown in Figure 16. When the user points to any of the keys on the panel, a key-down message is sent to the operating system, such that the current active application will receive such key. For example, we can use Windows `Notepad` to receive text inputted by the user. Figure 17 shows a snapshot of the display while a user was inputting "hello world. THANK YOU" with the virtual keyboard. (Note that the `lock` key was pressed in the middle.)

In our current implementation, users can only use one of their fingers. The typing speed is slow because of the one-second wait time. We are not claiming that we can get rid of the physical keyboard. However, our system could be a very promising alternative when the physical keyboard is not available under some circumstances. The typing speed could be increased by using hand gestures to activate the events rather than waiting for a short period of time. Alternatively, instead of using the traditional keyboard, we could use special keyboards similar to Cirrin [14] and Quickwriting [16] which can input multiple letters without explicit keyboard events.

4.6.4 Virtual Joystick

As described in Sect. 4.5, we can determine the 3D orientation and position of the Visual Panel at each time instant, and we can therefore use this type of information to control the display of a 3D object. That is, the Visual Panel can serve as a virtual joystick. Figure 18 shows a few screen dumps of controlling a tiger model using the Visual Panel. In each picture, the left side displays the live video with the tracked Visual Panel indicated by red lines; the right side displays a 3D tiger model. As we can see, when the Visual Panel moves closer to the camera, the tiger is zoomed; when the Visual Panel rotates, so does the tiger; when the Visual Panel translates, so does the tiger.

One obvious limitation of our system is that we cannot rotate an object all around continuously because the Visual Panel may degenerate to a line in the image, making the pose estimation failed. There are several ways to overcome this difficulty: detect explicitly this case; perform temporal filtering. In our current implementation, we just re-initialize the Visual Panel, and pursue the 3D control from where the tracking was lost previously.

5 Conclusion and Future Work

In this paper, we have described two vision-based human-computer interaction systems.

The first system, called Visual Screen, turns 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.

The second system, called Visual Panel, is capable of performing accurate control of remote display and simulating mouse, keyboard and joystick. The Visual Panel system employs an

Figure 16: Virtual keyboard

Figure 17: Virtual keyboard in action

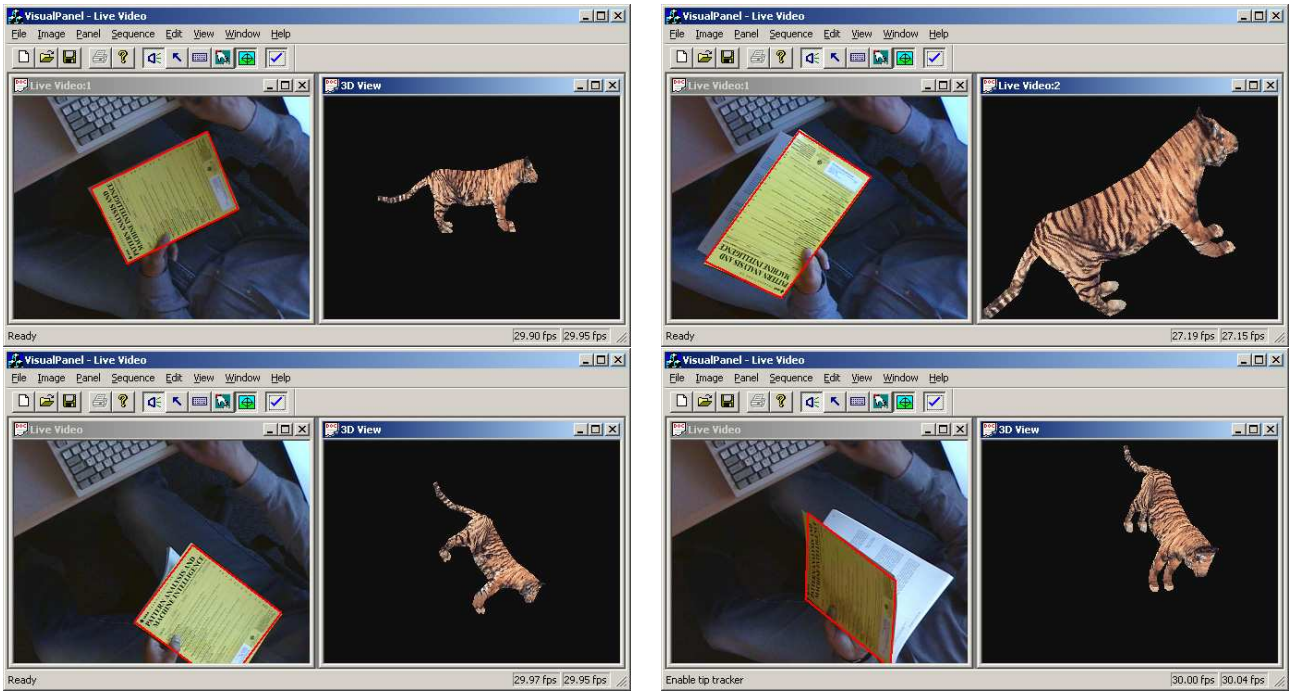


Figure 18: Virtual joystick: Control the visualization of a 3D object by moving the Visual Panel

arbitrary quadrangle-shaped plane object as a panel, which can be considered as a display or a keyboard. It can robustly and accurately track the panel and the tip pointer. A user can use their fingers or other tip pointers to simulate a cursor pointer and issue clicking/pressing instructions. After the action is detected, various events can be generated and sent to the computer operating system. Furthermore, by tracking the 3D position and orientation of the visual panel, the system can also provide 3D information, serving as a virtual joystick, to control 3D virtual objects. Four applications have been described: control a calculator, paint with fingers, input text with a virtual keyboard, and control the display of 3D objects. They have clearly shown the high robustness, accuracy and flexibility that the Visual Panel can provide. Many other applications are possible. For example, the 3D pose information of the Visual Panel will allow real-time insertion of a texture-mapped virtual object into the scene to make it appear as if it is a part of the scene and move with the Visual Panel. The Visual Panel system leaves a lot of room for extensions and improvements in various aspects, especially in action recognition. In our current implementation, action is triggered when the tip pointer stays immobile for a short duration (say 1 second). We are investigating more natural ways to do that, for example, by combining hand gesture recognition.

Acknowledgment

The author thanks Ying Shan, Ying Wu and Steven Shafer for their contributions. The work described in this paper was first presented in [32, 33]

References

[1] S. Ahmad. A usable real-time 3D hand tracker. In *Proc. IEEE Asilomar Conf.*, 1994.

[2] Greg Berry. Small-wall: A multimodal human computer intelligent interaction test bed with applications. Master's thesis, Dept. of ECE, University of Illinois at Urbana-Champaign, 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.

[4] T. M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[5] J. Crowley, F. Berard, and J. Coutaz. Finger tracking as an input device for augmented reality. In *Proc. Int'l Workshop on Automatic Face and Gesture Recognition*, pages 195–200, Zurich, 1995.

[6] Olivier Faugeras. *Three-Dimensional Computer Vision: a Geometric Viewpoint*. MIT Press, 1993.

[7] D. Hall, C. Le Gal, J. Martin, O. Chomat, T. Kapuscinski, and J. Crowley. Magicboard: A contribution to an intelligent office environment. In *Proceedings of the International Symposium on Intelligent Robotic Systems*, University of Coimbra, Portugal, 1999.

[8] K. Imagawa, S. Lu, and S. Igi. Color-Based hands tracking system for sign language recognition. In *Proc. of Int'l Conf. on Face and Gesture Recognition*, pages 462–467, 1998.

[9] R. Jain, R. Kasturi, and B.G. Schunck. *Machine Vision*. McGraw-Hill, New York, 1995.

[10] K. Jo, Y. Kuno, and Y. Shirai. Manipulative hand gestures recognition using task knowledge for human computer interaction. In *Proc. of IEEE Int'l Conf. on Automatic Face and Gesture Recognition*, Japan, 1998.

[11] Nebojsa Jojic, Barry Brumitt, Brian Meyers, and Steve Harris. Detecting and estimating pointing gestures in dense disparity maps. In *Proc. IEEE Int'l Conf. on Face and Gesture Recognition*, pages 468–475, Greboble, France, 2000.

[12] S. Ju, M. Black, S. Minneman, and D. Kimber. Analysis of gesture and action in technical talks for video indexing. In *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, 1997.

[13] 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.

[14] J. Mankoff and G. Abowd. Cirrin: A world-level unistroke keyboard for pen input. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages 213–214, San Francisco, CA, 1998.

[15] T. Moran, E. Saund, W. van Melle, A. Gujar, K. Fishkin, and B. Harrison. Design and technology for collaborage: Collaborative collages of information on physical walls. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, Asheville, NC, 1999.

[16] K. Perlin. Quikwriting: Continuous stylus-based text entry. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages 215–216, San Francisco, CA, 1998.

[17] F. Quek. Unencumbered gesture interaction. *IEEE Multimedia*, 1997.

[18] J. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. In *Proc. of IEEE Int'l Conf. Computer Vision*, pages 612–617, 1995.

[19] J.G. Semple and L. Roth. *Introduction to Algebraic Geometry*. Oxford: Clarendon Press, 1949. Reprinted 1987.

[20] T. Starner, J. Auxier, D. Ashbrook, and M. Gandy. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring. In *Proceedings of the Fourth International Symposium on Wearable Computers (ISWC'00)*, pages 87–94, Atlanta, Georgia, October 2000.

[21] T. Starner and et.al. A wearable computer based american sign language recognizer. In *Proc. IEEE Int'l Symposium on Wearable Computing*, Oct. 1997.

[22] 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.

[23] G. Strang. *Introduction To Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[24] N. Takao, J. Shi, S. Baker, I. Matthews, and B. Nabble. Tele-Graffiti: A pen and paper-based remote sketching system. In *Proceedings of the 8th International Conference on Computer Vision*, volume II, page 750, Vancouver, Canada, July 2001. IEEE Computer Society Press.

[25] J. Triesch and C. von de Malsburg. Robust classification of hand postures against complex background. In *Proc. Int'l Conf. On Automatic Face and Gesture Recognition*, 1996.

[26] Christian Vogler and Dimitris Metaxas. ASL recognition based on a coupling between HMMs and 3D motion analysis. In *Proc. of IEEE Int'l Conf. on Computer Vision*, pages 363–369, Mumbai, India, Jan. 1998.

[27] P. Wellner. Interacting with paper on the DigitalDesk. *Communications of the ACM*, 36(7):86–96, July 1993.

[28] C. Wren, A. Azarbayejani, T. Darrel, and A. Pentland. Pfinder: Real-time tracking of the human body. In *Photonics East, SPIE Proceedings*, volume 2615, Bellingham, WA, 1995.

[29] Ying Wu and Thomas S. Huang. Human hand modeling, analysis and animation in the context of HCI. In *Proc. IEEE Int'l Conf. on Image Processing (ICIP'99)*, Kobe, Japan, Oct. 1999.

[30] M. Zeller and et al. A visual computing environment for very large scale biomolecular modeling. In *Proc. IEEE Int'l Conf. on Application-Specific Systems, Architectures and Processors*, pages 3–12, Zurich, 1997.

[31] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[32] Z. Zhang and Y. Shan. Visual Screen: Transforming an ordinary screen into a touch screen. In *Proc. IAPR Workshop on Machine Vision Applications*, pages 215–218. Tokyo, Japan, November 2000.

[33] Zhengyou Zhang, Ying Wu, Ying Shan, and Steve Shafer. Visual Panel: Virtual mouse, keyboard and 3D controller with an ordinary piece of paper. In *Proceedings of ACM Workshop on Perceptive User Interfaces (PUI)*, Orlando, Florida, November 2001.