Self Calibration for the DigitalDesk

Pierre D. Wellner
wellner@europarc.xerox.com

Introduction to the problem

The DigitalDesk has one or more cameras that point down at the desk, and users can select areas in a camera's field of view to perform operations such as "copy and paste" or character recognition on selected areas of paper documents. This report describes how the system supports the ability to indicate a specific point or region precisely in the frame-grabber's coordinate system through a pointing device and projected display.

Relative pointing devices such as a mouse or trackball allow the user indirectly to position a cursor and make selections on the screen. The absolute position of the pointing device does not matter because the user dynamically calibrates it with the display by superimposing the cursor on desired screen points. With absolute pointing devices, such as a transparent digitizing tablet or touch screen, the user points directly to the screen, so the pointing device must be calibrated to map the pen or finger position to screen coordinates. The DigitalDesk uses absolute pointing devices because it aims to be as much like an ordinary desk as possible. When we use paper documents on an ordinary desk, we do not need to look at a video monitor to see where we are pointing: we simply point directly at the paper.

The DigitalDesk requires a two step calibration process. First, absolute positions of the digitizing tablet must be mapped to positions on the display (e.g. to provide feedback when the user points at something). Second, positions on the display must be mapped to corresponding positions in the frame grabber (e.g. to support grabbing of selected areas on the desk). If finger tracking is used for pointing instead of a tablet, however, then only the second step is necessary.

There are two ways to calibrate an absolute pointing device to the display: let us call them dynamic and static. A dynamic approach is only possible with certain device-display combinations; it continuously adjusts the pointer to the display, much as a person does when positioning a cursor on screen with a mouse. Although this approach has promise, it was not tried with the DigitalDesk and is only briefly discussed at the end of this report. Instead, devices on the DigitalDesk are calibrated using a static method which has two essential steps:

1) Obtain a number of known *tie points* that map from device space to display space.
2) Calculate a *warping* that generalizes the mapping from the tie points to all points.

The following sections describe first how the tablet is calibrated to the display, and then how the display is calibrated to the video cameras.

Calibrating the tablet to the display

The system obtains tie points for calibrating the tablet and display by prompting the user with a cross-hair for each point at which the user points with the stylus. The displayed location of the crosshair is paired with the data returned by the tablet. Because this is a manual process, we would like the user to enter the minimum number of tie points necessary for accurate calibration. This number of points depends on the type of warping that needs to be calculated.

Two point warping

When both the pointing device and the display use rectangular coordinate systems, it is often enough to obtain just two sets of tie points: (x_1, y_1), (x_1', y_1') and (x_2, y_2), (x_2', y_2') as illustrated in Figure 1. From these, the system can map any device coordinate (x, y) to the cor-

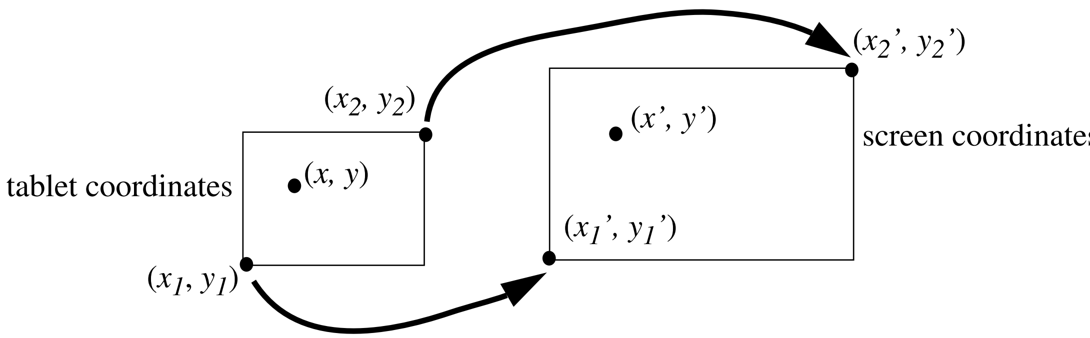


Figure 1: two point warping.

responding screen coordinate (x', y') by using linear interpolation with the following simple formulae:

$$x' = \frac{(x - x_1)(x'_2 - x'_1)}{x_2 - x_1} + x'_1$$

$$y' = \frac{(y - y_1)(y'_2 - y'_1)}{y_2 - y_1} + y'_1$$

In practice with the DigitalDesk, this technique is only an approximation at best. Unfortunately, the projected display is not a perfect rectangle. There are optical distortions such as "keystoning" (see Figure 2), and it is difficult to rotate the tablet and display so they

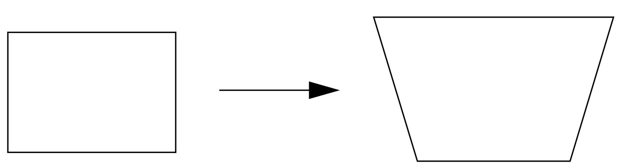


Figure 2: "keystoning" due to projecting a rectangular image at an angle.

are perfectly aligned. With this two point method, the tie points are mapped correctly, but as the pointer moves farther from a tie point, the cursor displayed at the warped screen coordinate moves farther away from the pen. The error depends on how large the optical and rotational distortions are, but it was not possible to maintain acceptable results with the current equipment, so more than two tie points are necessary. Calculating a warping from more than two points is a bit more complex and is described in the following sections.

Four point warping

If we obtain four pairs of tie points, as illustrated in Figure 1, then we can calculate a warping

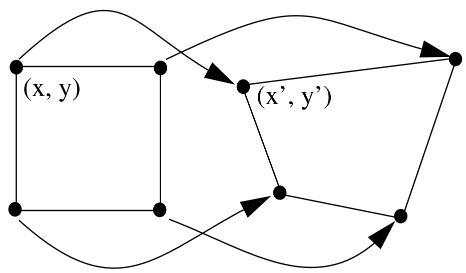


Figure 3: four point warping.

by using the following equations:

$$x' = c_1 x + c_2 y + c_3 xy + c_4$$

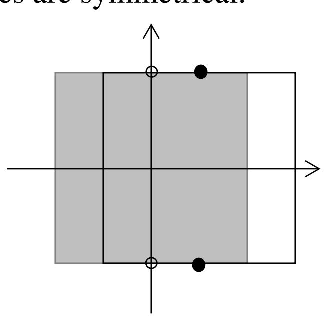
$$y' = c_5 x + c_6 y + c_7 xy + c_8$$

For each of these two equations, we can find the values of c_n by solving the set of four simultaneous equations obtained from plugging in values of the four tie points. These sets of simultaneous linear equations can be quickly and exactly solved (no need for statistical methods) by using Gauss-Jordan Elimination or Gaussian Elimination [Press88]. The following section shows how this solution not only compensates for translation and scaling of coordinate systems (as the two point warping does) but it also compensates for keystoning and rotation.

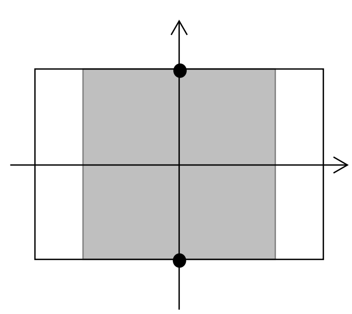
Why these particular equations? We need to have at least as many simultaneous equations as unknowns, so if we have only four points, then we must not have more than four unknown terms for x' or y'. Let us look at the six possible terms in a second order polynomial:

$$x' = c_0 + c_1 x + c_2 y + c_3 xy + c_4 xy + c_5 xy$$

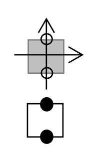
The diagrams below illustrate the effect of increasing each term on a simple square centered about the origin. These only look at terms affecting the *x* coordinates, but effects on *y* coordinates are symmetrical.



c_0 uniform translation

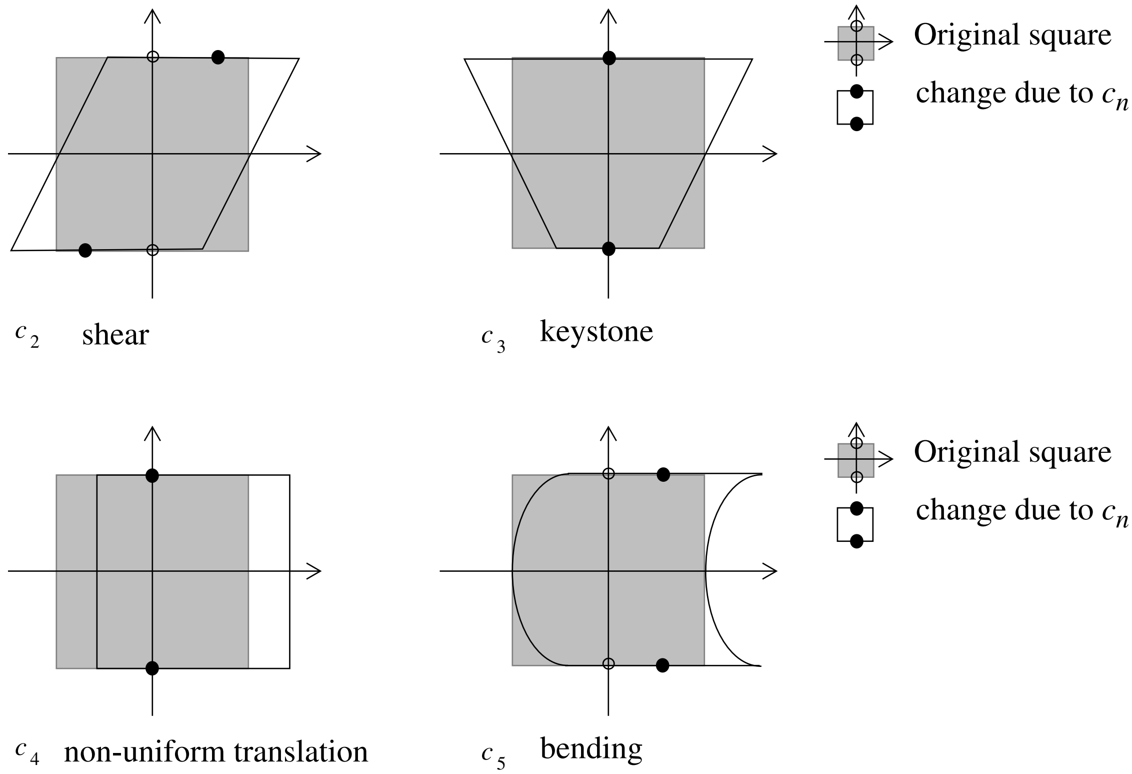


c_1 scaling



Original square

change due to c_n



Of these terms c_4 and c_5 are the least important., so $c_{0\text{-}3}$ were chosen as the terms to use in the warping equations. A shear in the *x* direction combined with a shear in the *y* direction rotates the coordinate space.

Another advantage of this four point warping is that it can be used to recursively break up a plane of coordinates to use as many points as necessary to make the warping more accurate over a larger area with more complex distortions. If four points are not enough, then the plane can be broken up into more tiles, each of which has its own four point warping. Although it was the intention to do this for the DigitalDesk, four points worked well enough that more turned out not to be necessary. See the appendix for the interface to the Warping.T object.

Calibrating the frame grabber to the display

The projector displays feedback directly on the page, so users can see exactly what is selected, but how does the system know what pixels in the frame grabber correspond to the area displayed by the projector? A precise warping is necessary to map pixels in the projected display to pixels in the frame grabber.

Unfortunately, the same problems that make calibrating the tablet difficult (e.g. keystoning and rotation) also make it difficult to calibrate the camera. In addition, there are specific factors due to the camera: it may view the display at an angle, it may frequently need to be zoomed or pointed at a new position, and it is sensitive to vibrations caused by air conditioners or slamming doors.

Whereas obtaining tie points for calibrating the tablet is relatively straight forward, obtaining them for calibrating the camera is a little more complex. One obvious way to do this is to adjust the size and shape of a projected rectangle on the desk while looking at its image on a video monitor, and this rectangle can be aligned with the edges of the monitor. This provides only two tie points, and if there is any keystoning, rotation, or under-scanning by the video monitor the result is poor, even after performing this tedious manual procedure. To obtain multiple tie points, users could control a cursor on the digitized video image seen through the camera; then they could point at projected cross hairs in much the same way as users calibrate the tablet. A better approach, however, is for the system to locate these crosshairs without any assistance from the user: for the system to *self* calibrate. The camera is connected to an image processing system, so it precisely locates marks projected by the display, then for each mark the system knows the screen coordinates and its corresponding frame grabber coordinates. These coordinates are the tie points needed to calculate a warping.

Precisely locating a projected cross

The basic steps in locating a projected mark through the video camera are as follows:

1) Prompt the user to place a projected window on the desk to give the system an approximation of where the camera is pointing.
2) Inside each of the four corners and in the center of this window:
    a) project a thick "plus mark."
    b) pause to allow the display system (X windows and the LCD panel) to catch up.
    c) grab the image through the camera and threshold it.
    d) perform mathematical morphology operations on the image to locate the mark.
    e) pinpoint the center of the mark.
3) Use the four corner point pairs to calculate a warping.
4) Use the warping to predict where the center point should appear.
5) Test if the center point is close to its prediction. If it's not then try again, else use the calculated warping.

Now let us go through each of these steps in a bit more detail:

Step 1: This step is necessary to ensure the system can project a mark that can be seen by the camera. The camera may be zoomed into a small part of the display, or it may be zoomed out to cover the entire display. The image processing system expects the apparent size of the marks (in frame grabber pixels) to be within a certain range, so if the camera is zoomed out, then the marks must be projected larger than if the camera is zoomed in. The size of the window the user sweeps out indicates to the system how large to make the projected marks. This manual step is not strictly necessary, because it is possible for the system to discover this approximate rectangle on its own, but this facility has not yet been implemented. A way to do it would be to start by grabbing two frames: one with the entire display white, and one with the entire display black. If there is no difference between the frames, then the camera cannot see the display and an error message would be generated. If there is a difference, then a smaller part of the display would be changed. The goal of this algorithm would be to find the largest rectangular area of the display on which a change is noticed by the camera, and where enlarging this area makes no additional difference.

Step 2: This step finds 5 tie point pairs (display coordinates $\leftrightarrow$ grabber coordinates). The four corner points are projected 1/4 of the distance in from each corner, and the 5th test point is placed in the center.

Step 2a: this mark is designed to be easily found using mophological techniques, even in the presence of substantial noise. Another approach that was tried was to XOR the mark into the display. The idea here was that no matter what was currently being displayed, the camera would be able to subtract the before and after images and detect a mark, and the hope was that the system could periodically (or continuously) self-calibrate even while it was being used.

This worked well on a background of mostly solid black and white areas, but because of the sometimes large difference in resolution between the camera and display, XORing the mark into a complex or patterned background often did not show up (on a fine gray background, even the human eye can hardly detect an XORed mark). Another problem with the mark was that it did not work well when projected onto paper with black marks on it. There is very little difference between projected black and projected white when projected onto black ink. So to make things simple and reliable, the current system requires the user to clear the area below the camera (e.g. by putting down a blank sheet of paper), and it only projects black marks onto a white background.

Step 2b: If the program does not pause long enough after displaying the mark and before grabbing the image, then the mark does not appear in the frame grabber, or it may appear only very faintly, because the LCDs have not had enough time to finish changing state.

Step 2c: Simply grab the entire field of view of the camera as a gray-scale image and threshold it using the quick adaptive thresholding technique described in [Wellner93].

Step 2d: Mathematical morphology and how it is used to locate calibration marks is described in the next section.

Step 2e: The end result of the morphological analysis is a bitmap with all bits clear except for the central part of the plus mark. This step finds the north-west and south-east corners of this small rectangle. If the corners are farther apart than expected, then something has gone wrong; *e.g.* more than one mark was in the camera's field of view. Otherwise, the midpoint between these two corners is used as the position for the mark.

Step 4: The four pairs of tie points are used to calculate a warping as described above, by solving the simultaneous linear equations. Then it is used to calculate where a mark displayed in the center of the four displayed tie points should appear in frame grabber coordinates.

Step 5: The predicted location is compared to the measured location. This is done by calculating the sum of the squares of the differences between the x coordinates and the y coordinates. The current system uses 4 as the upper bounds for this. If something goes wrong during the calibration process (*e.g.* the user moves a piece of paper under the camera), and it was not detected before, then the problem is usually discovered at this point because this distance is too great. Even if a little noise, vibration, or changing lighting conditions caused inaccurate readings, then the system will go around and locate the five points again. In normal circumstances, however, the system usually finds the sum of the squares of the errors at this point to be 1 or 2, *i.e.* neither x or y coordinate is more than one pixel off.

The original intention was for this algorithm to continue recursing down into each of the four quadrants and to continue refining the warping, but a single set of four points turns out to be more than precise enough to allow the user to select areas on the desk and rely on the displayed feedback to precisely indicate what will appear in the grabbed image. The entire process takes about n seconds, of which displaying and finding marks take about n seconds each. This could be made to go much faster by displaying and locating all five points in a single step.

Mathematical Morphology

A proper description of mathematical morphology (sometimes called image morphology) is outside the scope of this report (for this see [Haralick87]), but this section briefly introduces the concepts and walks through how they are applied to the problem of locating the calibration mark.

Mathematical morphology is an approach to image processing that is based on set theory and shape. Two basic operations are *dilation* and *erosion.* In the following discussion, A and B are sets of points that represent binary images; for each black pixel in the image, the corresponding point is an element of the set (See Figure 3 for example).

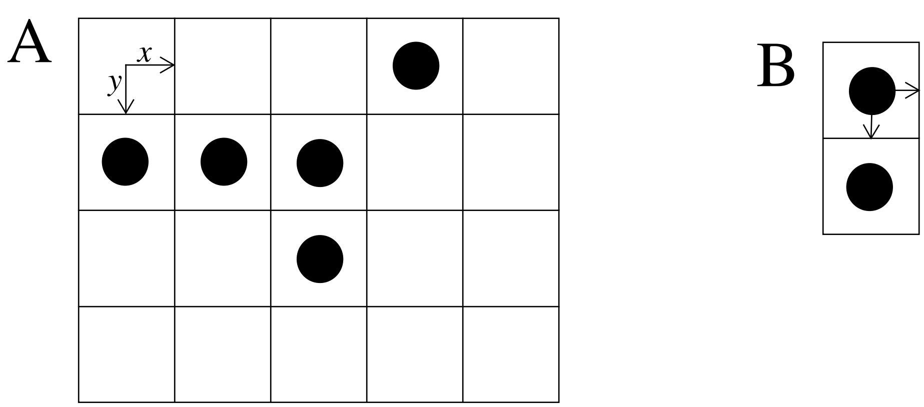


Figure 4: A = {(0,1), (1,1), (2, 1), (2, 2), (3, 0)}   B = {(0,0), (0, 1)}

Dilation

The dilation of A by B is denoted by A ⊕ B and is defined below, treating the points *p*, *a*, and *b* as vectors for the purpose of addition.

$A \oplus B = \{p \mid p = a + b \text{ for some } a \in A \text{ and } b \in B\}$.

See Figure 4 for an example.

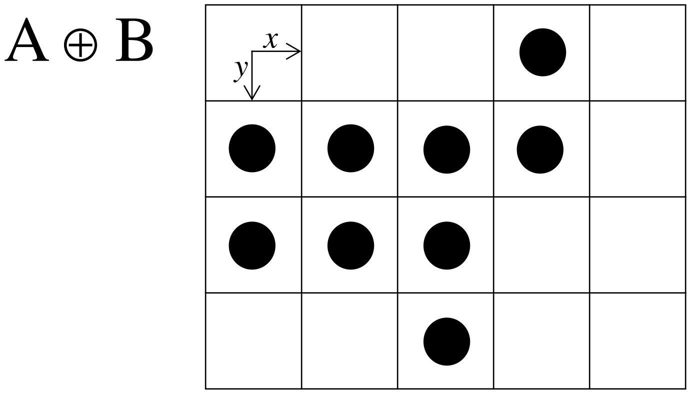


Figure 5: $A \oplus B = \{(0,1), (1,1), (2, 1), (2, 2), (3, 0)$
$(0,2), (1,2), (2, 2), (2, 3), (3, 1)\}$

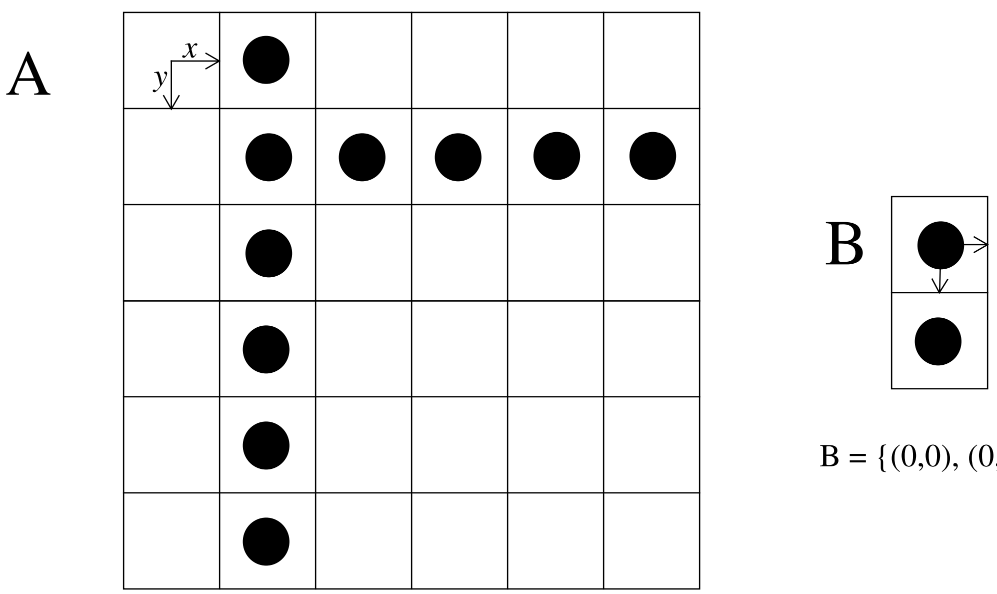
Although $A \oplus B = B \oplus A$, by convention they are handled quite differently. The first operand A represents the image bitmap, and B represents a smaller *structuring element* which is thought of as a single shape parameter for the dilation transformation.

Erosion

The erosion of A by B is denoted by $A \theta B$ and is defined by

$A \theta B = \{p \mid p + b \in A \text{ for every } b \in B\}$.

See Figure 5 for an example.



$B = \{(0,0), (0, 1)\}$

$A = \{(1,0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (3, 1), (4, 1), (5, 1)\}$

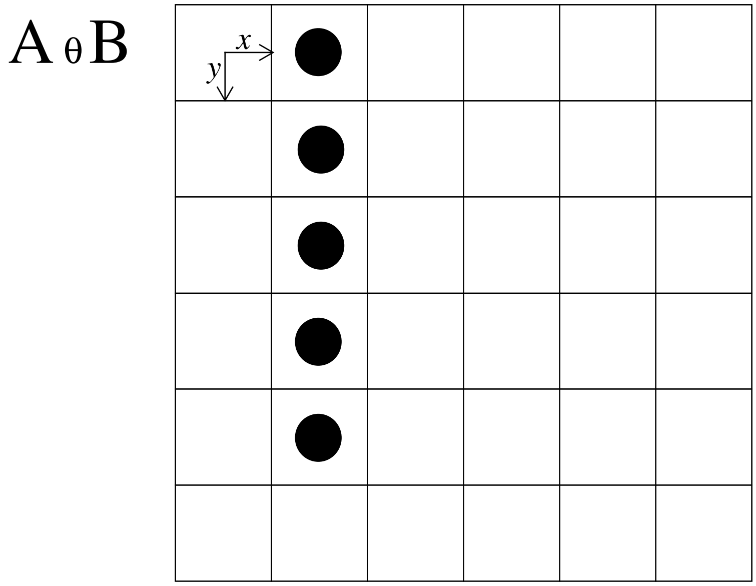


Figure 6: $A \theta B = \{(1,0), (1, 1), (1, 2), (1, 3), (1, 4)\}$

Opening and Closing

Dilation and erosion are often applied in pairs.

The *opening* of an image A by structuring element B is denoted by $A \circ B$ and is defined by

$A \circ B = (A \theta B) \oplus B$.

The *closing* of an image A by structuring element B is denoted by $A \bullet B$ and is defined by

$A \bullet B = (A \oplus B) \theta B$.

As proven in [Haralick87], these two operations are idempotent, *i.e.* applying them a second time does not make further changes. Their effect is to eliminate specific details of the image smaller than the structuring element without distorting larger features. For example, opening an image with a disk structuring element smooths the contours, breaks narrow isthmuses, and eliminates small black islands. Closing an image with a disk structuring element also smooths the contours, fuses narrow breaks, and eliminates small white holes.

Finding the calibration mark

This section describes the use of morphological operations to find the calibration mark in the presence of noise and other bits of black in the image. The basic strategy is to eliminate all pixels in the image other than long horizontal areas and long vertical areas, then to look at the intersection between them. The code calls mathematical morphology routines implemented by Dan Bloomberg [Bloomberg88].

Figure 6 illustrates the type of image (after thresholding) in which the system must locate the thick “plus” mark. This example is the picture of a final test mark projected in the center of the

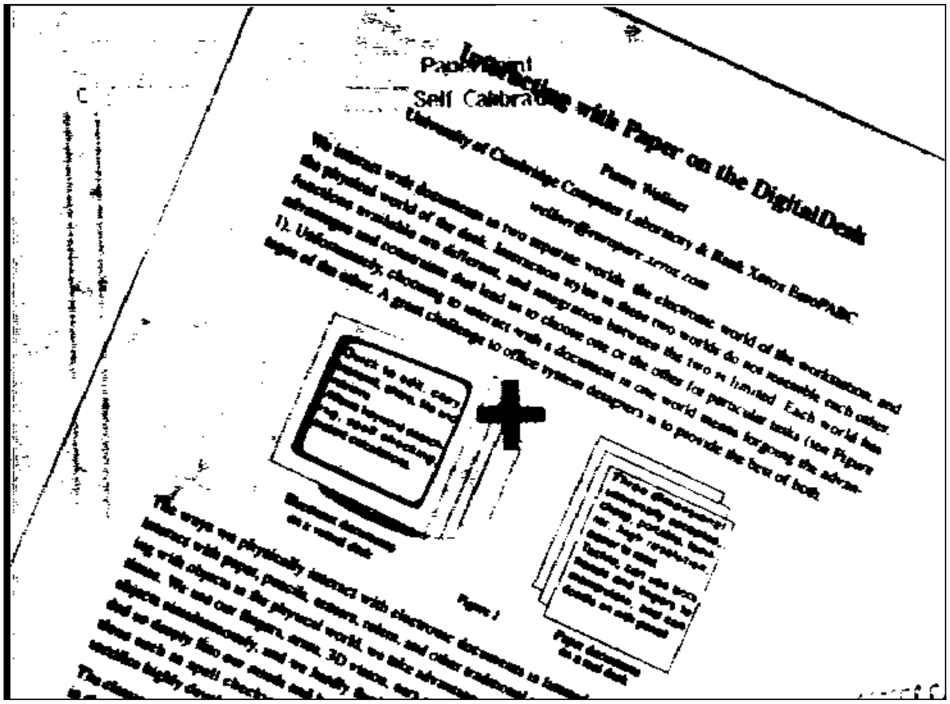


Figure 7: thresholded image of the final test point

calibration window to check if it appears in the location predicted. In this example, a paper with text and graphics was left on the desk under the camera to make the location of the cross a bit more challenging — normally it is best for the desk to be completely clear of extraneous marks. Also, the projected self calibration window is a bit too small for the field of view of the camera, so some projected text appears near the top of the image. In addition, the camera and display are slightly rotated relative to each other, as can be seen by the angle of the (fuzzy) projected window borders on the left side. All these features must be filtered out by the morphological operations in order to locate the cross in the center.

The first step is to erode the thresholded image by a square structuring element 5 pixels by 5 pixels with its origin in the center. This leaves a mark only in black areas at least this big, but removing 2 pixels off the top, bottom, left and right sides these areas. The size of the projected plus mark, based on the size of the self calibration window, is designed to be thick enough to comfortably survive an erosion by this structuring element. In an image less noisy than this one, most other marks would disappear completely. The result is shown in Figure 7.

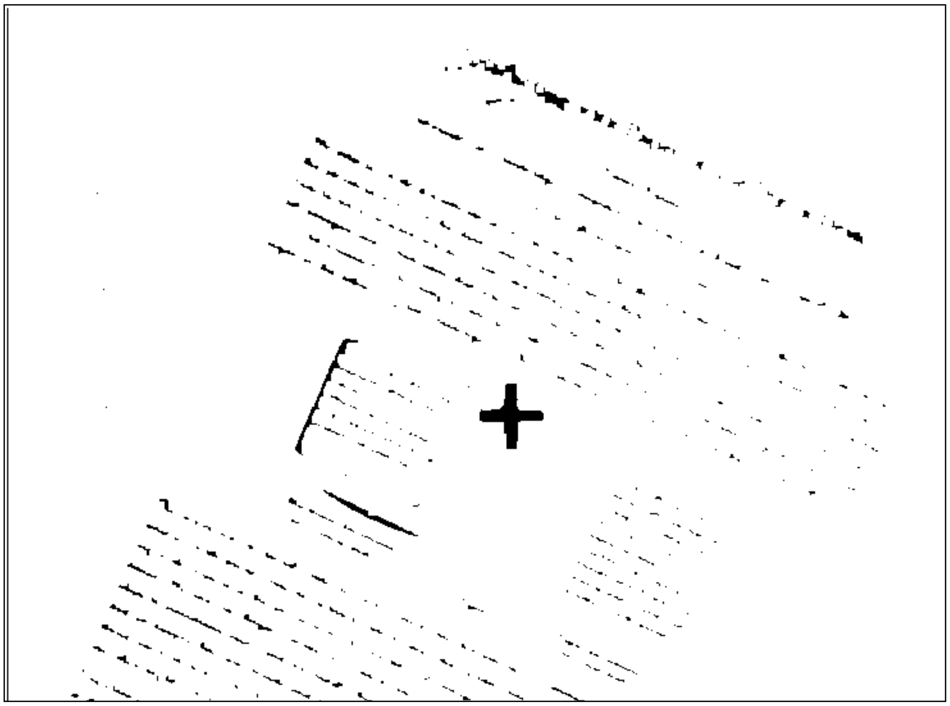


Figure 8: result after eroding with a 5x5 stucturing element

The next step is to open the image with a structuring element 32 pixels tall, 1 pixel wide, and with its origin in the center. This has the effect of removing all marks except those composed of vertical lines 32 or more pixels tall. The vertical stroke of the plus mark and the over-scanned left edge are the only features which survive this operation (see Figure 8a). As long as the camera and display are not rotated more than about 30 degrees relative to each other, then this stroke should survive. The following step is to open the eroded image with a structuring element 32 pixels wide and 1 pixel tall, preserving only marks that are 32 or more pixels wide (see Figure 8b).

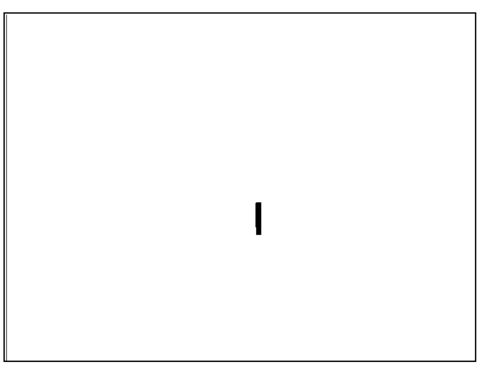


Figure 9a

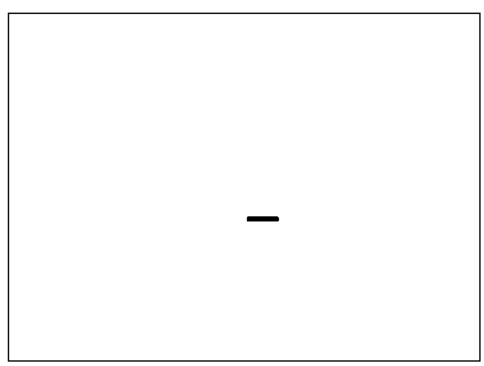


Figure 8b

Finally, these two images are ANDed together to produce the result in Figure 9, where the only pixels remaining are from the center of the plus mark. The system uses the midpoint of this mark to determine the frame grabber coordinates of the displayed cross.

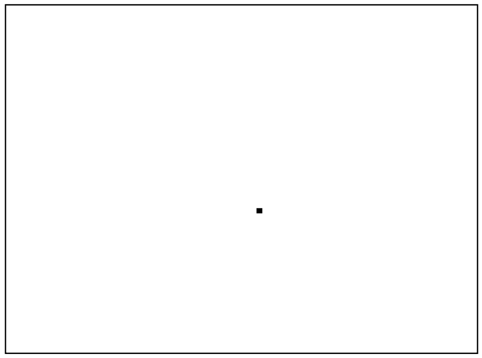


Figure 10

Dynamic Calibration

The DigitalDesk currently uses static calibration technique, with a warping that is initially calculated from a set of tie points and then used, unchanged, until the system is calibrated again. Although this works well with the current equipment and set of applications, the disadvantage of this method is that users must stop everything whenever the system needs to be recalibrated. A future implementation of the DigitalDesk (such as the DigitalDeskLamp) may allow users to adjust the camera and the display frequently depending on the resolution required or on the part of the desk in use. If users must frequently pause to recalibrate, this can make the system much less usable. It would be much better if the system could continuously calibrate itself without needing to stop working.

In principle, this should be possible with a fast enough image processing system. The key is for the system continuously to recognize and locate what it displays on the desk as part of the user interface. This can be especially useful when displaying dynamic feedback to the user such as a cursor or rubber-band rectangle.

Finger following might also be implemented with dynamic calibration instead of the current static method; the basic process would be simple. When the system locates the user’s finger in the image processing system it would guess, based on current information, what display coordinate this position corresponds to, and the system would display a cursor at that point. It then would quickly grab another frame, locate this cursor and see how far it appears from the fingertip. Based on this error, it moves the cursor closer to the finger and checks again. To determine where to move the cursor, the system can use a crude warping (perhaps a two point warping) that is dynamically updated based on recent local cursor positions. The process is essentially the same one a person would use with a mouse to follow someone’s finger pointing at a workstation screen. If this loop can be executed at video rates, the cursor might appear to follow the finger very smoothly and precisely. For this, not only a fast frame grabber and image processing system are necessary, but also an LCD panel that can change the state of its pixels at video rates. This method could work even while the camera and display are moving.

Acknowledgements

Thanks to Mik Lamming and Peter Robinson for pointing out the possibility of self-calibration, to Marcel Wellner for help with the mathematics of four point warping, to William Newman for synthesizing perspective warping data to test the validity of four point warping, to Dan Bloomberg for help with understanding and using his morphology code, to Steve Freeman for help making the Modula-3 Warping objects remotely acessible by RPC, and to Mike Flynn for interesting discussions on dynamic calibration (and lots of other topics too).

References

[Bloomberg90] Bloomberg, D., and Maragos, P. “Image Algebra and Morphological Image Processing.” In *SPIE Conference Proc.*, San Diego, CA, July 1990.

[Elrod92] Elrod, Scott, et al. “Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations, and Remote Collaboration” in proceedings of CHI 92 (Monterey, California, May 3 - May 7, 1992) ACM, New York 1992. pp. 599-607.

[Press88] Press, William H. et al. “Numerical Recipes: The art of Scientific Computing” Cambridge University Press, 1988.

[Haralick87] Haralick, Robert M., Sternberg, Stanley R., and Zhuang, Xinhua “Image Analysis Using Mathematical Morphology” IEEE Transactions on Pattern Analysis and Machine Intelligence, Bol PAMI-9, No. 4, July 1987.

[Wellner93] Wellner, Pierre D. “Adaptive thresholding for the DigitalDesk” EuroPARC Technical report EPC-93-110.

Appendix: Modula-3 interface for Warping

```
INTERFACE Warping;
(*
 *
 * A Warping provides a mapping between two coordinate spaces: one
 * “unwarped” and the other “warped.” The mapping is calculated based on
 * a set of tiepoints where each tiepoint is a PointPair that maps a
 * single unwarped point to a single warped point. The init() methods
 * determine how many tiepoints the Warping is based on, and return the
 * appropriate subtype of Warping.T These tiepoints are then set by
 * the various set methods. The implementation of the warping
 * (or subtype returned by init) depends on the number of points. Only
 * 2 and 4 point warpings are currently supported, although this interface
 * is meant to support warpings based on arbitrary numbers of points.
 *
 * The Types Points and PointPairs should be the same as those in
 * GrabberParms.i3 generated from the .x file. Also see the file warping.x
 *
 * T is a MUTEX and all methods lock self to prevent thread conflicts.
 *
 * ReadPointPairs() uses the Sx package to parse a lisp-like nested
 * parenthetical expression that specifies a list of PointPairs, e.g.
 *
 * ( ((0 0) (100 100)) ((10 10) (200 200)) ((1 2) (3 4)) ((5 6) (7 8)) )
 *
 * To initialize a Warping.T from a file, you could do something like this:
 *
 * TRY
 *    pointPairs := Warping.ReadPointPairs(warpingFile);
 *    w4 := NEW(Warping.T).init(NUMBER(pointPairs^));
 *    EVAL w4.setPairs(pointPairs^); (* if it’s not full then will fail
 *    later, or you can check and raise an error here *)
 * EXCEPT
 * | Warping.Failed (arg) => Utils.PutText(arg.info);
 * | Warping.BadFormat => Utils.PutText(
 *    Fmt.F(“bad Warping format file\”%s\”\n”, warpingFilename));
 * | Rd.Failure => Utils.PutText(Fmt.F(“can’t read file”);
 * END;
 *
 *)

IMPORT Point, Rect, Rd, Wr, ExceptionArg;

EXCEPTION
 BadFormat(Failure);
 Failed(Failure);

TYPE
 Failure = ExceptionArg.T BRANDED OBJECT END;
 FullFailure = Failure BRANDED OBJECT size: CARDINAL; END;
 NotReady = Failure BRANDED OBJECT minimum: CARDINAL; END;
```

```
TYPE
 PointPair = RECORD
      p : Point.T;
      wp: Point.T; (* warped point *)
 END;

 PointPairs = REF ARRAY OF PointPair;

 T <: Public;

 Public = MUTEX OBJECT
 METHODS
      init (READONLY numberOfTiepoints: CARDINAL): T RAISES {Failed};

      set (READONLY p, wp: Point.T): BOOLEAN RAISES {Failed};
      setPairs (READONLY tiepoints: ARRAY OF PointPair): BOOLEAN
      RAISES {Failed};
      (* set and setPairs reset existing PointPairs if either the point or
      warped point equal a corresponding tiepoint. This means that it is
      impossible to have the same point tied to two different warped
      points or vice versa.
      These procedures return TRUE when the Warping is ready, FALSE other-
      wise. *)

      getPairs (): PointPairs;
      (* returns tiepoints as a value suitable for passing to setPairs *)

      warp (READONLY p: Point.T): Point.T RAISES {Failed};
      unwarp (READONLY pw: Point.T): Point.T RAISES {Failed};
      warpRect (READONLY rect: Rect.T): Rect.T RAISES {Failed};
      unwarpRect (READONLY rect: Rect.T): Rect.T RAISES {Failed};
 END;

PROCEDURE ReadPointPairs (sx: Rd.T): PointPairs
 RAISES{Failed, BadFormat, Rd.Failure};

PROCEDURE WritePointPairs (sx: Wr.T; pps: PointPairs)
 RAISES{Wr.Failure, Failed};


VAR
 Identity: T; (* initialized at start up and should never be changed.
       Treat this like a constant. *)

END Warping.
```