How to calibrate touch-screens

By Carlos E. Vidales
Software Engineering Manager
Tekelec

Resistive-type touch-screens are normally used in cost-conscious designs. Their construction is simple, their operation is well understood, and the hardware and software required to support them is readily available from multiple manufacturers.

Despite the advantages of resistive-type touch-screens, devices equipped with them almost always require that a calibration algorithm be the first task to run when the final product comes out of the box. Calibration is necessary because it is difficult to perfectly align a touch-screen's coordinates to the display (LCD or otherwise) behind it. If a button or other "live" feature on the display is to be properly activated, the coordinates of the area touched on the screen must be sufficiently close to the coordinates of the feature on the display. Otherwise, the software may not correctly act upon the soft button presses.

This article proposes a calibration algorithm for resistive-type touch-screens that is both efficient and effective. This algorithm was developed after identifying the sources of touch-screen errors and deriving the optimum method for transforming the coordinates provided by the touch-screen to match the coordinates of the display. The calibration method requires that three targets or test points-no more, no less-be displayed and touched in sequence to determine the screen's individual calibration factors. These calibration factors are then used to translate screen coordinates into true display coordinates.

Touch-screen technology

The cross section of a resistive-type touch-screen is shown in **Figure 1**. The construction is simple. Two sheets of glass are brought together to form a sandwich, the interior glass surfaces having been coated with a thin layer of conductive material. Small glass beads maintain a nominal separation between the conductive surfaces. When a finger or stylus presses against the surface of the glass, the material bends just enough to contact the lower sheet. In this construction the spacing between beads determines the sensitivity of the screen. The closer the beads are, the higher the pressure that must be exerted before the top glass sheet will bend enough to make contact.

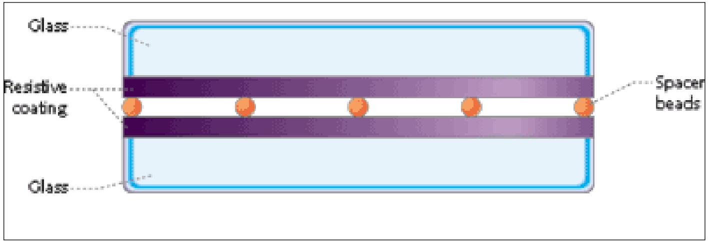

Figure 1: Cross section of a resistive-type touch-screen

An equivalent resistive circuit is shown in **Figure 2**: a touch-screen controller (digitiser or A/D) applies the V sources to the ends of one of the conductive layers, while the other conductive layer-on the opposite sheet of glass-plays the role of the potentiometer wiper. The Vtest value read by the digitiser depends on where the glass is touched and the conductive surfaces come into contact. The controller then translates the voltage reading into a binary quantity representing, for example, the X-coordinate of the point where the screen was touched. The voltage potential is then applied to the second surface's end-points and the first surface plays the role of the wiper, yielding a value that represents the Y-coordinate.

Controllers may collect 200 or more samples per second. The sampling rate usually depends on background noise and controller quality. A smart controller may also incorporate useful features like the ability to interrupt the CPU when a touch is detected, as well as the ability to sample continuously at a set rate as long as the screen is being touched. The device idles when the screen is not being touched.

Sources of error

Several sources of error affect the X and Y coordinates produced by the touch-screen controller. The most important sources of error are electrical noise, mechanical misalignments, and scaling factors. User idiosyncrasies may also play a role if, for example, the finger or instrument used to activate the screen does not maintain continuous contact or pressure against it. Any of these errors can produce unusable data, and all need to be compensated for if the touch-screen data is to be useful.

Electrical noise triggered by thermal or electromagnetic effects and system design weaknesses are omnipresent in all kinds of electrical systems. In the case of a touch-screen, the A/D conversion is particularly susceptible to electrical noise because of the high input impedance of the A/D front-end circuitry. In addition to carefully laying out the printed circuit board containing the touch-screen controller, noise problems are normally addressed by adding low-pass filtering to the A/D inputs. Software also comes to the rescue, usually discarding one or two of the least significant bits of the A/D conversion and implementing algorithms to remove from the sampled stream those data points that fall outside permissible ranges. The same software algorithms may remove errors introduced by the user. Such errors normally manifest themselves as sampled points that fall well outside permissible limits as the applied pressure varies.

Mechanical misalignments and scaling factors are the errors addressed by the calibration algorithm proposed in this article. Consider the images in **Figure 3**. The circle represents an image shown by the LCD under the touch-screen; the ellipse represents an exaggerated set of coordinates that the touch-screen could produce if the user were to trace the round image shown by the LCD. The reconstructed image appears rotated, translated, and scaled by a different factor in each direction. The challenge for the calibration algorithm is to translate the set of coordinates reported by the touch-screen into a set of coordinates that accurately represent the image on the display.

The mathematics

To derive a general solution to the problem, it is convenient to describe each point as a mathematical quantity. We see from **Figure 4** that it is possible to describe each point on the display as a vector PD and its corresponding equivalent, as reported by the touch-screen, as a vector P.

It is also reasonable to assume that P and PD are related by a quantity that transforms the

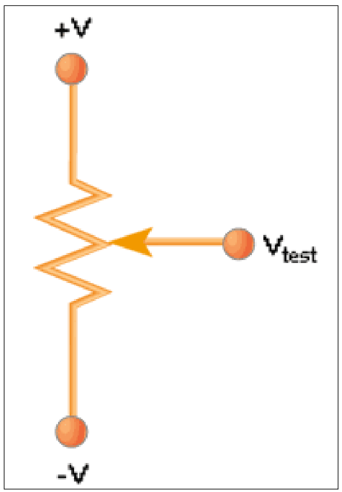

Figure 2: Equivalent circuit

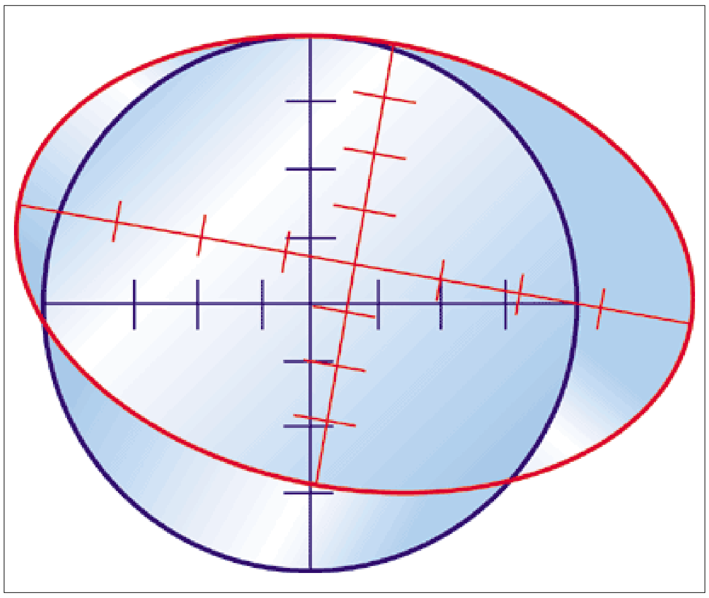
Figure 3: A circle on the display and its equivalent touch-screen coordinates

display coordinates into the apparent touch-screen coordinates, like this:

PD = MP (1)

where M is the appropriate transformation matrix and the object of this exercise. If we can assign values to the elements of the transformation matrix M, then we can solve for all PD points given the P points reported by the touch-screen.

Let us proceed under the assumption that each display point can be obtained from a corresponding touch-screen point, but that the touch-screen point has suffered rotation, scaling, and translation.

If we express the points as (X,Y) pairs based on the vector's length and angle, both the display and touch-screen points in question can be represented by the following equations:

$$P_D = \left[X_D, Y_D\right] = \left[R_D \cos\theta_D, R_D \sin\theta_D\right]$$

$$P = \left[X, Y\right] = \left[R\cos\theta, R\sin\theta\right]$$

If the reported touch-screen points have experienced rotation because of touch-screen misalignment, and qr = qD - q is the angular misalignment of the reported point, an intermediate point is expressed by:

$$P' = \left[R\cos\left(\theta+\theta_r\right), R\left(\theta+\theta_r\right)\right]$$

Each of the X and Y dimensions is also scaled by its own factor, which we will call KX and KY. Accounting for scaling produces the following equation, which comes closer to describing a display point in terms of screen point coordinates:

$$P'' = \left[K_X R\cos\left(\theta+\theta_r\right), K_Y R\sin\left(\theta+\theta_r\right)\right]$$

Finally, let us add translation elements XT and YT to yield the equation of a display point in terms of the data reported by the touch-screen:

$$P_D = \left[K_X R\cos\left(\theta+\theta_r\right)+X_T, K_Y R\sin\left(\theta+\theta_r\right)+Y_T\right]$$

We can now make a key, pragmatic assumption in order to represent **Equation 5** in a form that will enable us to solve for its unknown quantities. While the touch-screen may be rotated with respect to the display, the rotation angle qr should be sufficiently small that we can assume sinqr ≈qr and cosqr ≈1.0. This assumption yields another useful set of approximations:

$$\cos\left(\theta+\theta_r\right) \approx \left(\cos\theta - \theta_r \sin\theta\right)$$

$$\sin\left(\theta+\theta_r\right) \approx \left(\sin\theta + \theta_r \cos\theta\right)$$

and Equation 5 can then be re-expressed by the following combination of terms:

$$P_D = \left[K_X R\cos\theta - \theta_r K_X R\sin\theta +.\right.$$
$$\left. K_Y R\sin\theta + \theta_r K_Y R\cos\theta + Y_T\right]$$

The advantage of **Equation 7** is that we have expressed actual display coordinates in terms of touch-screen coordinates. Restating the above equation we get:

$$P_D = \left[K_X X - \theta_r K_X Y + X_T, \theta_r K_Y X + K_Y Y + Y_T\right]$$

Except for X and Y, all the terms on the right side of the above equation are constant, given our previous assumption that in the practical case the effects of scaling, rotation, and translation could be accounted for with constant quantities.

Expressing the XD and YD coordinates in more convenient terms, we arrive at the equations that translate touch-screen points to display points:

$$X_D = AX + BY + C$$

$$Y_D = DX + EY + F$$

The previous equations seem almost intuitive. Remember, though, that they are only valid when the angular misalignment between the display and the touch-screen is small.

Calibration matrix

Common calibration algorithms use data from two to as many as five sample points to collect calibration information. The exercise above assumes that display point coordinates can be calculated from touch-screen coordinates, and that by making simple assumptions we can obtain calibration data by using three sample points-no more, no less. Three sample points are needed because **Equations 9a** and **9b** each contain three unknowns. With three sample points we can obtain sufficient information to set up and solve the simultaneous equations.

Sample points should also be selected based on practical considerations. They must yield non-redundant simultaneous equations, they must not be too close to the edge of the touch-screen (where non-linearities are more likely to occur) and they must be widely spaced to minimise scaling errors. The set of points shown in **Figure 5** labelled P0, P1, and P2 meet the suggested requirements. Those points are roughly 10% off the edge of the screen, as far apart as the display geometry allows, and yield the following non-redundant equations:

$$X_{D0} = AX_0 + BY_0 + C$$

$$X_{D1} = AX_1 + BY_1 + C$$

$$X_{D2} = AX_2 + BY_2 + C$$

$$Y_{D0} = DX_0 + EY_0 + F$$

$$Y_{D1} = DX_1 + EY_1 + F$$

$$Y_{D2} = DX_2 + EY_2 + F$$

We need to solve for A, B, C, D, E, and F. Once these parameters are known, obtaining display coordinates is as simple as plugging the raw touch-screen data back into Equation 9.

The solution for the unknowns in the previous set of simultaneous equations is well known and will not be derived here; we will skip the intermediate steps and present the final solution. The unknowns are resolved as follows, using the value K as the common denominator for all solutions:

$$K=(X_0-X_2)(Y_1-Y_2)-(X_1-X_2)(Y_0-Y_2)$$

$$A=\left((X_{D0}-X_{D2})(Y_1-Y_2)-(X_{D1}-X_{D2})(Y_0-Y_2)\right)/K$$

$$B=\left((X_0-X_2)(X_{D1}-X_{D2})-(X_{D0}-X_{D2})(X_1-X_2)\right)/K$$

$$C=\left(Y_0(X_2X_{D1}-X_1X_{D2})+Y_1(X_0X_{D2}-X_2X_{D0})+Y_2(X_1X_{D0}-X_0X_{D1})\right)/K$$

$$D=\left((Y_{D0}-Y_{D2})(Y_1-Y_2)-(Y_{D1}-Y_{D2})(Y_0-Y_2)\right)/K$$

$$E=\left((X_0-X_2)(Y_{D1}-Y_{D2})-(Y_{D0}-Y_{D2})(X_1-X_2)\right)/K$$

$$F=\left(Y_0(X_2Y_{D1}-X_1Y_{D2})+Y_1(X_0Y_{D2}-X_2Y_{D0})+Y_2(X_1Y_{D0}-X_0Y_{D1})\right)/K$$

Software implementation

The hardest part of this problem (solving for the set of simultaneous **Equations 10** and **11**) is now behind us; the actual implementation in software is a straightforward exercise. The program sample provided at ftp://ftp.embedded.com/pub/2002/06vidales includes three files: calibrate.c, calibrate.h, and sample.c. The hard work is done in the first. It contains two functions called **setCalibrationMatrix()** and **getDisplayPoint()**. The former implements Equations 13 through 18; the latter implements Equation 9.

The header file contains miscellaneous declarations, and the other source file implements a console application that exercises the calibration functions, attempting to show how you would use these functions within a device.

The code in sample.c assumes that your device implements a routine to collect calibration data before attempting to use these functions. An outline of the proposed calibration process is shown in **Listing 1**. After taking those steps, the calibration procedure is complete and your applications can begin receiving accurate position information from the touch-screen system. The function **getDisplay-Point()** is intended to be called within the touch-screen controller's interrupt routine, after the routine has filtered and debounced the data coming out of the digitiser. In a typical implementation, calling **getDisplayPoint()** would be the last step needed before a point of touch-screen data could be placed in the user-input queue.

Listing 1 Steps in the calibration process

1. Call setCalibrationMatrix() with a perfect set of values (see sample.c) to set the touch-screen driver to provide raw (non-translated) data. This way you do not need to build special functions to access the data. Instead you simply get coordinates from the same mechanism as your program.
2. Draw the first target at coordinates (XD0,YD0).
3. Collect the data returned by the touch-screen driver and save it as (X0,Y0).
4. Draw the second target at coordinates (XD1,YD1).
5. Collect the data returned by the touch-screen driver and save it as (X1,Y1).
6. Draw the third target at coordinates (XD2,YD2).
7. Collect the data returned by the touch-screen driver and save it as (X2,Y2).
8. Call setCalibrationMatrix() with the reference display data and resulting touch-screen data as arguments.

Special considerations

The sample functions have been implemented to ensure that accurate results are produced every time. Thus, intermediate values have not been scaled.

Integer 32-bit math is required because the raw numbers provided by most digitisers are 10-bit quantities, and **Equations 15** and **18** may yield 31-bit signed values. The equations, as implemented, will work with touch-screen digitiser resolutions up to 1,024 pixels, which may be suitable for display resolutions up to about 512 pixels (height or width).

If you want to use these formulas for larger touch-screen resolutions, it will be necessary to either use 64-bit integer values or judiciously scale input values or intermediate results to avoid register overflow when calculating the calibration factors.

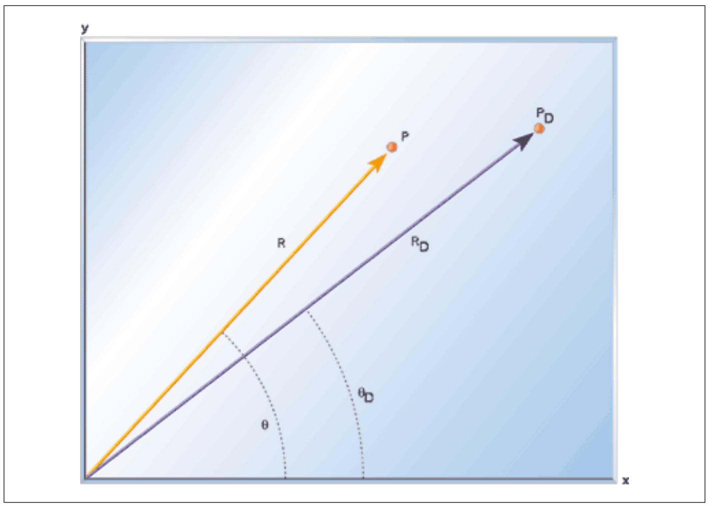

Figure 4: Vector representations of point

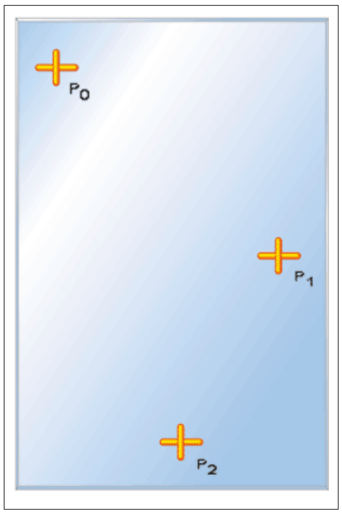

Figure 5: Sample calibration points

The function **getDisplay-Point()** does depend on the calibration matrix containing a set of valid quantities to provide reliable results. The structure of the sample provided uses automatic variables to highlight the relationship between the display coordinates, screen coordinates, and the calibration matrix, but in a practical implementation the calibration matrix may best be a global structure that is properly initialised at boot time.

The code available for download at ftp://ftp.embedded.com/pub/2002/06vidales is offered as a free reference implementation that can be adapted to a variety of applications. No attempt was made to optimise the code.

One final touch

The touch-screen calibration algorithm described here is simple, flexible, and capable of correcting for common mechanical errors. These errors-translation, scaling, and rotation-result from mismatches between the display and the touch-screen located above it. They can be compensated for with mechanical means, but the proposed software solution reduces demanding electro-mechanical manufacturing and quality problems.

Email   Send inquiry