```
// TouchScreen_Calibr_native for MCUFRIEND UNO Display Shields
// adapted by David Prentice
// for Adafruit's <TouchScreen.h> Resistive Touch Screen Library
// from Henning Karlsen's original UTouch_Calibration program.
// Many Thanks.

#define PORTRAIT  0
#define LANDSCAPE 1

#define TOUCH_ORIENTATION  PORTRAIT
#define TITLE "TouchScreen.h GFX Calibration"

#include <Adafruit_GFX.h>
#include <MCUFRIEND_kbv.h>
MCUFRIEND_kbv tft;

// MCUFRIEND UNO shield shares pins with the TFT.
#if defined(ESP32)
int XP = 27, YP = 4, XM = 15, YM = 14;  //most common configuration
#else
int XP = 6, YP = A1, XM = A2, YM = 7;  //most common configuration
//int XP = 7, YP = A2, XM = A1, YM = 6;  //most common configuration
#endif
//#include <TouchScreen.h>         //Adafruit Library
//TouchScreen ts(XP, YP, XM, YM, 300);   //re-initialised after diagnose
//TSPoint tp;                           //global point
#include "TouchScreen_kbv.h"         //my hacked version
TouchScreen_kbv ts(XP, YP, XM, YM, 300);   //re-initialised after diagnose
TSPoint_kbv tp;                           //global point

#define WHITE 0xFFFF
#define RED   0xF800
#define GRAY  0x8410
#define BLACK 0x0000

void readResistiveTouch(void)
{
    tp = ts.getPoint();
    pinMode(YP, OUTPUT);      //restore shared pins
    pinMode(XM, OUTPUT);
    digitalWrite(YP, HIGH);  //because TFT control pins
    digitalWrite(XM, HIGH);
    //    Serial.println("tp.x=" + String(tp.x) + ", tp.y=" + String(tp.y) + ", tp.z =" + String(tp.z));
}

bool ISPRESSED(void)
{
    // .kbv this was too sensitive !!
    // now touch has to be stable for 50ms
    int count = 0;
    bool state, oldstate;
    while (count < 10) {
```

```
        readResistiveTouch();
        state = tp.z > 200;     //ADJUST THIS VALUE TO SUIT YOUR SCREEN e.g. 20 ... 250
        if (state == oldstate) count++;
        else count = 0;
        oldstate = state;
        delay(5);
    }
    return oldstate;
}

uint32_t cx, cy, cz;
uint32_t rx[8], ry[8];
int32_t clx, crx, cty, cby;
float px, py;
int dispx, dispy, text_y_center, swapxy;
uint32_t calx, caly, cals;

char *Aval(int pin)
{
    static char buf[2][10], cnt;
    cnt = !cnt;
#if defined(ESP32)
    sprintf(buf[cnt], "%d", pin);
#else
    sprintf(buf[cnt], "A%d", pin - A0);
#endif
    return buf[cnt];
}

void showpins(int A, int D, int value, const char *msg)
{
    char buf[40];
    sprintf(buf, "%s (%s, D%d) = %d", msg, Aval(A), D, value);
    Serial.println(buf);
}

boolean diagnose_pins()
{
    int i, j, value, Apins[2], Dpins[2], Values[2], found = 0;
    Serial.println(F("Making all control and bus pins INPUT_PULLUP"));
    Serial.println(F("Typical 30k Analog pullup with corresponding pin"));
    Serial.println(F("would read low when digital is written LOW"));
    Serial.println(F("e.g. reads ~25 for 300R X direction"));
    Serial.println(F("e.g. reads ~30 for 500R Y direction"));
    Serial.println(F(""));
    for (i = A0; i < A5; i++) pinMode(i, INPUT_PULLUP);
    for (i = 2; i < 10; i++) pinMode(i, INPUT_PULLUP);
    for (i = A0; i < A4; i++) {
        pinMode(i, INPUT_PULLUP);
        for (j = 5; j < 10; j++) {
            pinMode(j, OUTPUT);
            digitalWrite(j, LOW);
```

```
            value = analogRead(i);              // ignore first reading
            value = analogRead(i);
            if (value < 100 && value > 0) {
                showpins(i, j, value, "Testing :");
                if (found < 2) {
                    Apins[found] = i;
                    Dpins[found] = j;
                    Values[found] = value;
                }
                found++;
            }
            pinMode(j, INPUT_PULLUP);
        }
        pinMode(i, INPUT_PULLUP);
    }
    if (found == 2) {
        Serial.println(F("Diagnosing as:-"));
        int idx = Values[0] < Values[1];
        for (i = 0; i < 2; i++) {
            showpins(Apins[i], Dpins[i], Values[i],
                    (Values[i] < Values[!i]) ? "XM,XP: " : "YP,YM: ");
        }
        XM = Apins[!idx]; XP = Dpins[!idx]; YP = Apins[idx]; YM = Dpins[idx];
//        ts = TouchScreen(XP, YP, XM, YM, 300);    //re-initialise with pins
        ts = TouchScreen_kbv(XP, YP, XM, YM, 300);    //re-initialise with pins
        return true;                            //success
    }
    Serial.println(F("BROKEN TOUCHSCREEN"));
    return false;
}

void setup()
{
    Serial.begin(9600);
    Serial.println(TITLE);
    bool ret = true;
#if defined(__arm__) || defined(ESP32)
    Serial.println(F("Not possible to diagnose Touch pins on ARM or ESP32"));
#else
    ret = diagnose_pins();
#endif
    uint16_t ID = tft.readID();
    Serial.print("ID = 0x");
    Serial.println(ID, HEX);
    tft.begin(ID);
    tft.setRotation(TOUCH_ORIENTATION);
    dispx = tft.width();
    dispy = tft.height();
    text_y_center = (dispy / 2) - 6;
    if (ret == false) {
        centerprint("BROKEN TOUCHSCREEN", text_y_center);
        while (true);    //just tread water
```

```
    }
}

void loop()
{
    startup();

    tft.fillScreen(BLACK);
    drawCrossHair(dispx - 11, 10, GRAY);
    drawCrossHair(dispx / 2, 10, GRAY);
    drawCrossHair(10, 10, GRAY);
    drawCrossHair(dispx - 11, dispy / 2, GRAY);
    drawCrossHair(10, dispy / 2, GRAY);
    drawCrossHair(dispx - 11, dispy - 11, GRAY);
    drawCrossHair(dispx / 2, dispy - 11, GRAY);
    drawCrossHair(10, dispy - 11, GRAY);
    centerprint("**********", text_y_center - 12);
    centerprint("**********", text_y_center + 12);

    calibrate(10, 10, 0, F(" LEFT, TOP, Pressure"));
    calibrate(10, dispy / 2, 1, F(" LEFT, MIDH, Pressure"));
    calibrate(10, dispy - 11, 2, F(" LEFT, BOT, Pressure"));
    calibrate(dispx / 2, 10, 3, F(" MIDW, TOP, Pressure"));
    calibrate(dispx / 2, dispy - 11, 4, F(" MIDW, BOT, Pressure"));
    calibrate(dispx - 11, 10, 5, F(" RT, TOP, Pressure"));
    calibrate(dispx - 11, dispy / 2, 6, F(" RT, MIDH, Pressure"));
    calibrate(dispx - 11, dispy - 11, 7, F(" RT, BOT, Pressure"));

    cals = (long(dispx - 1) << 12) + (dispy - 1);
    if (TOUCH_ORIENTATION == PORTRAIT) swapxy = rx[2] - rx[0];
    else swapxy = ry[2] - ry[0];
    swapxy = (swapxy < -500 || swapxy > 500);
    if ((TOUCH_ORIENTATION == PORTRAIT) ^ (swapxy != 0)) {
        clx = (rx[0] + rx[1] + rx[2]) / 3;
        crx = (rx[5] + rx[6] + rx[7]) / 3;
        cty = (ry[0] + ry[3] + ry[5]) / 3;
        cby = (ry[2] + ry[4] + ry[7]) / 3;
    } else {
        clx = (ry[0] + ry[1] + ry[2]) / 3;
        crx = (ry[5] + ry[6] + ry[7]) / 3;
        cty = (rx[0] + rx[3] + rx[5]) / 3;
        cby = (rx[2] + rx[4] + rx[7]) / 3;
    }
    px = float(crx - clx) / (dispx - 20);
    py = float(cby - cty) / (dispy - 20);
    //  px = 0;
    clx -= px * 10;
    crx += px * 10;
    cty -= py * 10;
    cby += py * 10;

    calx = (long(clx) << 14) + long(crx);
```

```
    caly = (long(cty) << 14) + long(cby);
    if (swapxy)
        cals |= (1L << 31);

    report();          // report results
    while (true) {}    // tread water
}

void readCoordinates()
{
    int iter = 5000;
    int failcount = 0;
    int cnt = 0;
    uint32_t tx = 0;
    uint32_t ty = 0;
    boolean OK = false;

    while (OK == false)
    {
        centerprint("*  PRESS  *", text_y_center);
        while (ISPRESSED() == false) {}
        centerprint("*  HOLD!  *", text_y_center);
        cnt = 0;
        iter = 400;
        do
        {
            readResistiveTouch();
            if (tp.z > 20)
            {
                tx += tp.x;
                ty += tp.y;
                cnt++;
            }
            else
                failcount++;
        } while ((cnt < iter) && (failcount < 10000));
        if (cnt >= iter)
        {
            OK = true;
        }
        else
        {
            tx = 0;
            ty = 0;
            cnt = 0;
        }
        if (failcount >= 10000)
            fail();
    }

    cx = tx / iter;
    cy = ty / iter;
```

```
    cz = tp.z;
}

void calibrate(int x, int y, int i, String msg)
{
    drawCrossHair(x, y, WHITE);
    readCoordinates();
    centerprint("* RELEASE *", text_y_center);
    drawCrossHair(x, y, GRAY);
    rx[i] = cx;
    ry[i] = cy;
    Serial.print("\r\ncx="); Serial.print(cx);
    Serial.print(" cy="); Serial.print(cy);
    Serial.print(" cz="); Serial.print(cz);
    if (msg) Serial.print(msg);
    while (ISPRESSED() == true) {}
}

void report()
{
    uint16_t TS_LEFT, TS_RT, TS_TOP, TS_BOT, TS_WID, TS_HT, TS_SWAP;
    int16_t tmp;
    char buf[60];
    centertitle(TITLE);

    tft.println(F("To use the new calibration"));
    tft.println(F("settings you must map the values"));
    tft.println(F("from Point p = ts.getPoint() e.g. "));
    tft.println(F("x = map(p.x, LEFT, RT, 0, tft.width());"));
    tft.println(F("y = map(p.y, TOP, BOT, 0, tft.height());"));
    tft.println(F("swap p.x and p.y if diff ORIENTATION"));

    //.kbv show human values
    TS_LEFT = (calx >> 14) & 0x3FFF;
    TS_RT   = (calx >>  0) & 0x3FFF;
    TS_TOP  = (caly >> 14) & 0x3FFF;
    TS_BOT  = (caly >>  0) & 0x3FFF;
    TS_WID  = ((cals >> 12) & 0x0FFF) + 1;
    TS_HT   = ((cals >>  0) & 0x0FFF) + 1;
    TS_SWAP = (cals >> 31);
    if (TOUCH_ORIENTATION == LANDSCAPE) { //always show PORTRAIT first
        tmp = TS_LEFT, TS_LEFT = TS_BOT, TS_BOT = TS_RT, TS_RT = TS_TOP, TS_TOP =
tmp;
        tmp = TS_WID, TS_WID = TS_HT, TS_HT = tmp;
    }
    tft.setCursor(0, 120);
    Serial.println("");
    sprintf(buf, "MCUFRIEND_kbv ID=0x%04X  %d x %d",
            tft.readID(), TS_WID, TS_HT);
    tft.println(buf);
    Serial.println(buf);
    sprintf(buf, "\nconst int XP=%d,XM=A%d,YP=A%d,YM=%d; //%dx%d ID=0x%04X",
```

```
        XP, XM - A0, YP - A0, YM, TS_WID, TS_HT, tft.readID());
    Serial.println(buf);
    sprintf(buf, "const int TS_LEFT=%d,TS_RT=%d,TS_TOP=%d,TS_BOT=%d;",
        TS_LEFT, TS_RT, TS_TOP, TS_BOT);
    Serial.println(buf);
    sprintf(buf, "PORTRAIT CALIBRATION     %d x %d", TS_WID, TS_HT);
    tft.println("");
    tft.println(buf);
    Serial.println(buf);
    sprintf(buf, "x = map(p.x, LEFT=%d, RT=%d, 0, %d)", TS_LEFT, TS_RT, TS_WID);
    tft.println(buf);
    Serial.println(buf);
    sprintf(buf, "y = map(p.y, TOP=%d, BOT=%d, 0, %d)", TS_TOP, TS_BOT, TS_HT);
    tft.println(buf);
    Serial.println(buf);
    sprintf(buf, "Touch Pin Wiring XP=%d XM=%s YP=%s YM=%d",
        XP, Aval(XM), Aval(YP), YM);
    tft.println("");
    tft.println(buf);
    Serial.println(buf);

    tmp = TS_LEFT, TS_LEFT = TS_TOP, TS_TOP = TS_RT, TS_RT = TS_BOT, TS_BOT = tmp;
    tmp = TS_WID, TS_WID = TS_HT, TS_HT = tmp;

    sprintf(buf, "LANDSCAPE CALIBRATION    %d x %d", TS_WID, TS_HT);
    tft.println("");
    tft.println(buf);
    Serial.println(buf);
    sprintf(buf, "x = map(p.y, LEFT=%d, RT=%d, 0, %d)", TS_LEFT, TS_RT, TS_WID);
    tft.println(buf);
    Serial.println(buf);
    sprintf(buf, "y = map(p.x, TOP=%d, BOT=%d, 0, %d)", TS_TOP, TS_BOT, TS_HT);
    tft.println(buf);
    Serial.println(buf);
}

void drawCrossHair(int x, int y, uint16_t color)
{
    tft.drawRect(x - 10, y - 10, 20, 20, color);
    tft.drawLine(x - 5, y, x + 5, y, color);
    tft.drawLine(x, y - 5, x, y + 5, color);
}

void centerprint(const char *s, int y)
{
    int len = strlen(s) * 6;
    tft.setTextColor(WHITE, RED);
    tft.setCursor((dispx - len) / 2, y);
    tft.print(s);
}

void centertitle(const char *s)
```

```
{
   tft.fillScreen(BLACK);
   tft.fillRect(0, 0, dispx, 14, RED);
   tft.fillRect(0, 14, dispx, 1, WHITE);
   centerprint(s, 1);
   tft.setCursor(0, 30);
   tft.setTextColor(WHITE, BLACK);
}

void startup()
{
   centertitle(TITLE);

   tft.println(F("#define NUMSAMPLES 3 in Library\n"));
   tft.println(F("Use a stylus or something"));
   tft.println(F("similar to touch as close"));
   tft.println(F("to the center of the"));
   tft.println(F("highlighted crosshair as"));
   tft.println(F("possible. Keep as still as"));
   tft.println(F("possible and keep holding"));
   tft.println(F("until the highlight is"));
   tft.println(F("removed. Repeat for all"));
   tft.println(F("crosshairs in sequence.\n"));
   tft.println(F("Report can be pasted from Serial\n"));
   tft.println(F("Touch screen to continue"));

   while (ISPRESSED() == false) {}
   while (ISPRESSED() == true) {}
   //    waitForTouch();
}

void fail()
{
   centertitle("Touch Calibration FAILED");

   tft.println(F("Unable to read the position"));
   tft.println(F("of the press. This is a"));
   tft.println(F("hardware issue and can"));
   tft.println(F("not be corrected in"));
   tft.println(F("software."));
   tft.println(F("check XP, XM pins with a multimeter"));
   tft.println(F("check YP, YM pins with a multimeter"));
   tft.println(F("should be about 300 ohms"));

   while (true) {};
}
```