

## Data Graphing with Arduinos and a Small TFT Display

David R. Brooks, © May, 2017

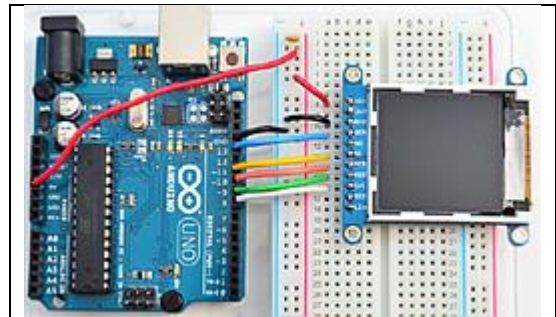
Thin-film-transistor liquid-crystal displays (TFTs) are high-resolution color displays. They are expensive to use for large displays, such as you would use to watch videos and movies, for example, but there are small versions that can be interfaced with microcontrollers.

[www.adafruit.com](http://www.adafruit.com) offers several small TFTs. For this project I chose the smallest one, a 1.44" square, 128 x 128 pixel display (ID 2088, \$15). These displays require two software libraries. One is for communications between an Arduino and the display and the other provides an extensive library of graphics routines that can be used to draw on the display – text, lines, and other shapes in a range of colors.

Here's the breadboard layout for using this device with an Arduino UNO (see <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-1-44-color-tft-with-micro-sd-socket.pdf>). The display comes with a clear peel-off protective cover, with a colored tab that is related to the size of the display; a green tab identifies this device as a 1.44" square display.

Here are the recommended pin connections for an Arduino UNO:

TFT	→ Arduino UNO
Vin	→ 5V
GND	→ GND
SCK	→ digital 13
SI	→ digital 11
TCS	→ digital 10
RST	→ digital 9
D/C	→ digital 8



Some of these pin assignments can be changed – a fact that I make use of later.

The artistic possibilities for these very cute pixel-addressable displays are intriguing, but my more mundane goal was to create data-based graph displays.

The coding for such projects can be tedious, but it is not really difficult because of the available graphing functions. The primary computational tasks are to keep track of the data being collected and to convert physical units for whatever measurement you are making to the available pixel space. For “continuous” monitoring, you could think of storing a series of measurements in one or more arrays and updating the measurements at some specified interval. For example, you might want to display several values on the bar graph – the current value and the most recent previous values.

For the first project, I created a bar graph using some made-up values. To start, I created a graphing space with some room for labeling the x- and y-axes. In common with many pixel-based displays, the graphing space starts with x values starting at the left edge of the display and moving to the right – 0 to 128 pixels for this display. The y values start at the top edge of the display and count down toward the bottom – 0 to 128 pixels. This is “upside down” from how you would usually think of addressing an x-y graphing space.

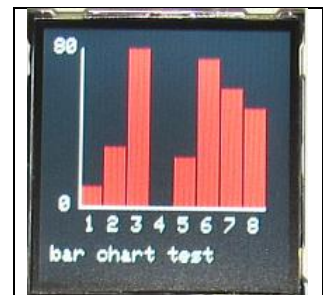
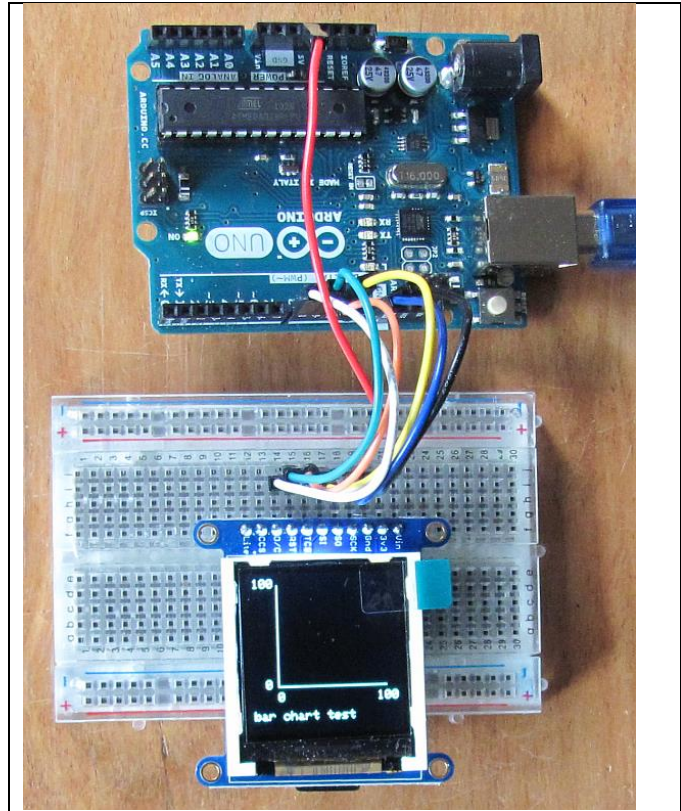
I set the background to black. For the x-axis, I chose a length of 90 pixels, starting 25 pixels from the left edge. For the y-axis I chose a length of 80 starting 90 pixels down from the top edge.

There is a default font included with the software library. The mono-spaced characters are 6 pixels wide by 8 pixels high – the characters themselves are 5 x 7 pixels, with a one-pixel empty row or column below or on the right edge of the character. Using  $(x_0, y_0) = (25, 90)$  as starting coordinates, I printed all text using offsets from those coordinates.  $(x, y)$  coordinates for positioning a text string or single character are at the upper lefthand corner of the string or character. The default text color is white.

The next step is to decide how many bars to draw. If you want each bar to have its own label (that wouldn't have to be true, of course), the limited pixel size of this device means that the number of bars is limited essentially by the number of pixels required to print an x-axis label. If the x-axis label string is

1 2 3 4 5 6 7 8

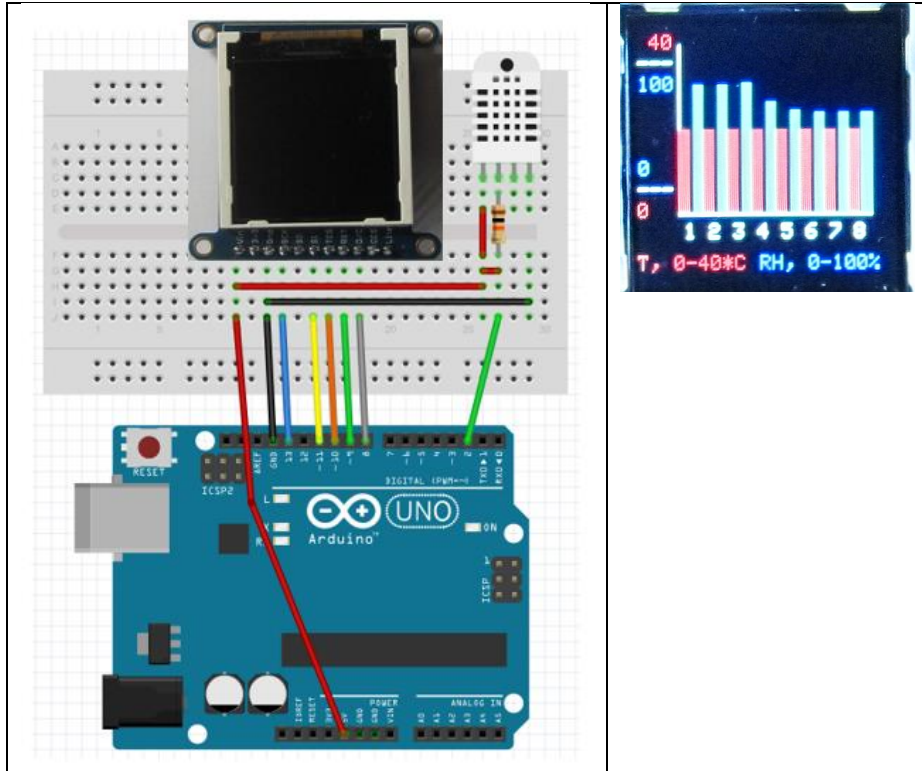
Where  $\square$  marks a blank space, 12 pixels for each label produces an x-axis of  $8 \times 12 = 96$  pixels. I reset the x-axis length to 96 pixels, starting 20 pixels from the left edge, and offset the start of the label string by 3 pixels to center the label under each bar. I left one blank pixel on each side of the bar, so the filled rectangle was 10 pixels wide rather than 12. Here is the code to generate this image. The `loop()` function is empty because this is just a “static” test for designing the display, rather than responding to data changing while a script is running.



## DataGraphTest\$

```
1 /*****
2   DataGraphTest.ino, D. Brooks, May 2017
3   *****/
4 #include <Adafruit_GFX.h>    // Core graphics library
5 #include <Adafruit_ST7735.h> // Hardware-specific library
6 #define TFT_CS    10
7 #define TFT_RST    9
8 #define TFT_DC    8
9 Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS,  TFT_DC, TFT_RST);
10 const int barW=12,nBars=8,x_axis=96,y_axis=80,x0=20,y0=90;
11 void setup(void) {
12   Serial.begin(9600);
13   tft.initR(INITR_144GREENTAB); // Initialize a 1.44" ST7735S chip (green tab).
14   //Serial.println("Screen initialized");
15   // (This 1.44" display is 128x128 pixels.)
16   //Serial.print("H= ");Serial.print(tft.height());
17   //Serial.print("W= ");Serial.println(tft.width());
18   tft.fillScreen(ST7735_BLACK);
19   // Set background and text.
20   tft.setCursor(x0-20,y0+20); tft.setTextColor(ST7735_WHITE);
21   tft.print(" bar chart test");
22   tft.drawFastHLine(x0,y0,x_axis,ST7735_WHITE);
23   tft.drawFastVLine(x0,y0-y_axis,y_axis,ST7735_WHITE);
24   // Print axis labels.
25   tft.setCursor(x0-10,y0-5); tft.print("0"); // offset 5 pix per character
26   tft.setCursor(x0-20,y0-y_axis-5); tft.print(" 80");
27   tft.setCursor(x0+3,y0+5); tft.print("1 2 3 4 5 6 7 8");
28   // Display data.
29   int staticData[]={10,30,80,0,25,75,60,50};
30   for (int i=0; i<nBars; i++) {
31     tft.fillRect(x0+i*barW+1,y0-staticData[i],barW-1,staticData[i],ST7735_RED);
32   }
33 }
34 void loop() {}
```

The next step was to replace the test data with data generated in real time by some kind of sensor. For this project, I split each 12-pixel bar into two 6-pixel bars, one for temperature and one for relative humidity, using data from a DHT22 temperature/relative humidity sensor updated every 5 seconds. The rightmost bar is always the most recent reading, with seven previous readings to the left. Considering the pixel limitations of this device, a surprising amount of information can be displayed! The display actually looks sharper, with more vivid colors, than the image here.



Here's the code that produced this display:

```

/* DataGraphTest.ino, D. Brooks, May 2017
   Uses a DHT22 sensor to create T/RH bar graph on adafruit.com
   ID 2088 tft, displaying 8 most recent values, with newest
   value on the right.
   T 0-40 *C, RH 0-100%
*/
#include <Adafruit_GFX.h> // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library
#include <DHT.h>
#define TFT_CS 10
#define TFT_RST 9
#define TFT_DC 8
#define DHTpin 2
#define DHTTYPE DHT22
DHT dht(DHTpin,DHTTYPE);
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);
const int barW=12,nBars=8,x_axis=96,y_axis=80,x0=20,y0=90,dt=5000;
const int T_min=0, T_max=40; // temperature limits, *C, see L41
float T,temperature[]={0,0,0,0,0,0,0,0};
float RH,humidity[]={0,0,0,0,0,0,0,0};
int i,knt,fillFlag;
int y,m,d,hr,mi,sec;
void setup(void) {
  Serial.begin(9600); dht.begin();
  tft.initR(INITR_144GREENTAB); // Initialize a 1.44" ST7735S chip (green tab).
  //Serial.println("Screen initialized");
  // (This 1.44" display is 128x128 pixels.)
  //Serial.print("H= ");Serial.print(tft.height());
  //Serial.print("W= ");Serial.println(tft.width());
  tft.fillScreen(ST7735_BLACK);
  // Set background and text.

```

```

tft.setCursor(x0-20,y0+20);
tft.setTextColor(ST7735_RED); tft.print("T, 0-40°C ");
tft.setCursor(x0+40,y0+20);
tft.setTextColor(ST7735_BLUE); tft.print("RH, 0-100%");
tft.setTextColor(ST7735_WHITE);

tft.drawFastHLine(x0,y0,x_axis,ST7735_WHITE);
tft.drawFastVLine(x0,y0-y_axis,y_axis,ST7735_WHITE);
// Print axis labels.
tft.setTextColor(ST7735_BLUE);
tft.setCursor(x0-20,y0-5); tft.print("0 "); // offset 5 pix per character
tft.setTextColor(ST7735_WHITE);
tft.setCursor(x0-20,y0-15); tft.print("---");
tft.setTextColor(ST7735_RED);
tft.setCursor(x0-20,y0-25); tft.print("0 ");
tft.setTextColor(ST7735_RED);
tft.setCursor(x0-20,y0-y_axis-5); tft.print(" 40");
tft.setTextColor(ST7735_WHITE);
tft.setCursor(x0-20,y0-y_axis+5); tft.print("---");
tft.setTextColor(ST7735_BLUE);
tft.setCursor(x0-20,y0-y_axis+15); tft.print("100");
tft.setTextColor(ST7735_WHITE);
tft.setCursor(x0+3,y0+5); tft.print("1 2 3 4 5 6 7 8");
// Display starting data (humidity=0).
for (int i=0; i<nBars; i++) {
    tft.fillRect(x0+i*barW+1,y0-0,barW/2-1,0,ST7735_RED);
    tft.fillRect(x0+i*barW+barW/2,y0-0,barW/2,0,ST7735_RED);
}
// Get first pair of readings to start populating array.
knt=0; fillFlag=0;
temperature[knt]=dht.readTemperature();
humidity[knt]=dht.readHumidity();

RH=scaleRH(humidity[i]); T=scaleT(temperature[knt]);
tft.fillRect(x0+i*barW+2,y0-T,barW/2,T,ST7735_RED);
tft.fillRect(x0+i*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
displayArray(0,0,0,0,0,0);
delay(dt);
}
void loop() {
// Fill all array elements for the first time.
if (fillFlag==0) {
    knt++;
    temperature[knt]=dht.readTemperature();
    humidity[knt]=dht.readHumidity();
    displayArray(0,0,0,0,0,0);
    RH=scaleRH(humidity[knt]); T=scaleT(temperature[knt]);
    tft.fillRect(x0+knt*barW+2,y0-T,barW/2,T,ST7735_RED);
    tft.fillRect(x0+knt*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
    if (knt == (nBars-1)) fillFlag=1;
}
else {
    for (i=0; i<(nBars-1); i++) {
        temperature[i]=temperature[i+1];
        humidity[i]=humidity[i+1];
        RH=scaleRH(humidity[i]);
        T=scaleT(temperature[i]);
        // next 2 lines "erase" previous bars
        tft.fillRect(x0+i*barW+2,y0-T_max,barW/2,T_max,ST7735_BLACK);
        tft.fillRect(x0+i*barW+barW+1,y0-100,barW/2,100,ST7735_BLACK);
        tft.fillRect(x0+i*barW+2,y0-T,barW/2,T,ST7735_RED);
        tft.fillRect(x0+i*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
    }
}
}

```

```

    temperature[nBars-1]=dht.readTemperature();
    humidity[nBars-1]=dht.readHumidity();
    T=scaleT(temperature[nBars-1]); RH=scaleRH(humidity[nBars-1]);
    // next 2 lines "erase" previous bars
    tft.fillRect(x0+(nBars-1)*barW+2,y0-T_max,barW/2,T_max,ST7735_BLACK);
    tft.fillRect(x0+(nBars-1)*barW+barW/2+1,y0-100,barW/2,100,ST7735_BLACK);
    tft.fillRect(x0+(nBars-1)*barW+2,y0-T,barW/2,T,ST7735_RED);
    tft.fillRect(x0+(nBars-1)*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
    displayArray(0,0,0,0,0,0);
}
delay(dt);
}
void displayArray(int y,int m,int d,int hr,int mi,int sec) {
  for (i=0; i<nBars; i++) {
    //Serial.print(y); Serial.print('/');
    //Serial.print(m); Serial.print('/');
    //Serial.print(d); Serial.print(' ');
    //Serial.print(hr); Serial.print(':');
    //Serial.print(mi); Serial.print(':');
    //Serial.print(sec); Serial.print(' ');
    Serial.print("i = ");Serial.print(i);
    Serial.print(" RH = ");Serial.print(humidity[i],1);
    Serial.print(" T = ");Serial.println(temperature[i],1);
  }
  Serial.println("-----");
}
int scaleRH(float RH) {
  return round(y_axis*RH/100.);
}
int scaleT(float T) {
  return round(y_axis*(T-T_min)/(T_max-T_min));
}

```

## Logging data simultaneously with graphing data

Considering how the previous project might be used to monitor temperature and relative humidity in a greenhouse, for example, it would make sense to record data over time, ideally with a date/time stamp. The data logging shield from adafruit.com (ID 1141) provides that capability. My only concern was whether the TFT display was compatible with the data logging shield. As it turns out, only one pin connection needs to be changed. Because on the data logging shield, pin 10 is “hard wired” for communicating with the SD card interface, the TFT’s TCS pin needs to be moved; I moved it to pin 7:

TFT	→ Arduino UNO
Vin	→ 5V
GND	→ GND
SCK	→ digital 13
SI	→ digital 11
TCS	→ digital 7
RST	→ digital 9
D/C	→ digital 8

As shown in the breadboard layout, moving the orange wire from pin 10 to pin 7 is the only change from the previous breadboard layout. Of course, the code for accessing the TFT display needs to be changed accordingly. As before, the DHT22 sensor is read from pin 2.

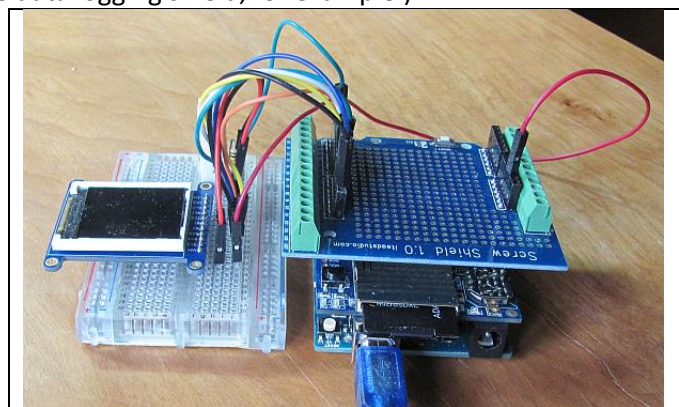
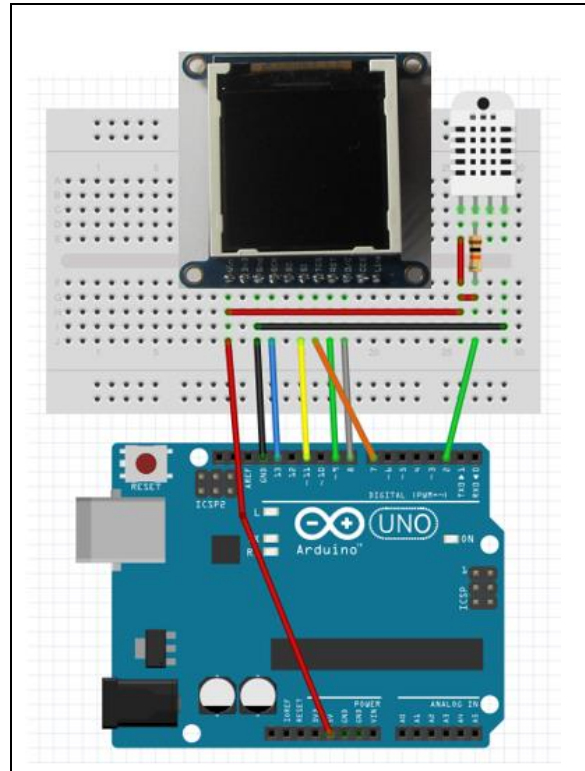
The layout shown here is incomplete because it doesn’t show the adafruit data logging shield that sits on top of the Arduino board. For this project I also mounted a screw terminal shield (RB-lte-127 from robotshop.com) so I would have access to the Arduino and data logging shield for other purposes without having to remove all the pin connections. (It is sometimes necessary to reset the clock on the data logging shield, for example.)

The pin connections shown are fed up from the Arduino board headers through the headers on the data logging and screw terminal shields as shown in this image. The SD card is visible in its holder just above the USB cable.

The code for this project is shown below. The bar display is updated every 5 seconds, plus the few milliseconds required to execute the code. Data are saved to the SD card every 2 minutes – this value can be changed as desired. The code reads the real time clock with every cycle through the `loop()` function. The minute value is checked using `if... tests`:

```
if (minute) mod (logging interval) equals 0...
```

```
if data for this minute haven’t been logged before...
```



```

/* T_RH_GraphLog.ino, D. Brooks, May 2017
  Uses a DHT22 sensor to create T/RH bar graph on adafruit.com
  ID 2088 tft, displaying 8 most recent values, with newest
  value on the right.
  Uses adafruit.com ID 1141 datalogging shield for date/time
  and to save T/RH data to SD card. See line 41 for logging interval.
  Limits on temperature and relative humidity for graph:
  T 0-40 *C, RH 0-100% (see lines 39, 40, 75, 77, 83-93)
  TFT --> Arduino UNO
  Vin --> 5V
  GND --> GND
  SCK --> 13
  SI --> 11
  TCS --> 7
  RST --> 9
  D/C --> 8
  SD card pin --> 10 (from datalogger shield)
  DHT22 pin --> 2
*/
#include <Wire.h>
#include <SD.h>
#include <RTCLib.h>
#include <SPI.h>
#include <DHT.h>
#include <Adafruit_GFX.h> // Core graphics library
#include <Adafruit_ST7735.h> // TFT hardware library
#include <DHT.h>
#define DHTPIN 2
#define DHTTYPE DHT22
#define TFT_CS 7 // not 10, to avoid conflict with SD card
#define TFT_RST 9
#define TFT_DC 8
#define DHTpin 2
#define SDpin 10
DHT dht(DHTpin,DHTTYPE);
RTC_DS1307 rtc;
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);
const int barW=12,nBars=8,x_axis=96,y_axis=80,x0=20,y0=90,dt=5000;
const int T_min=0, T_max=40; // temperature limits, *C
const int RH_min=0, RH_max=100; // humidity limits, %
const int dm=2; // time interval for logging
float T,temperature[]={0,0,0,0,0,0,0,0,0};
float RH,humidity[]={0,0,0,0,0,0,0,0,0};
float dayFrac;
int i,knt,fillFlag,minit,last_written=99;
int y,m,d,hr,mi,sec;
File logfile;
char filename[]="LOGGER00.CSV";
void setup(void) {
  Serial.begin(9600); dht.begin(); Wire.begin(); rtc.begin();
  if (!rtc.isrunning()) {
    Serial.println(F("RTC not running.)); delay(50); exit;
  }
  if (!SD.begin(SDpin)) {Serial.println(F("Card failed.));
  delay(50);exit(0);}
  Serial.println(F("card initialized.));
  for (uint8_t i = 0; i < 100; i++) {
    filename[6] = i/10 + '0';
    filename[7] = i%10 + '0';
    // Create a new file LOGGERxx.CSV, xx= 01 to 99, only if it doesn't already exist.
    if (! SD.exists(filename)) {
      logfile = SD.open(filename, FILE_WRITE);
      if (!logfile) {Serial.println(F("Couldn't create file.)); delay(50); exit(0);}
    }
  }
}

```



```

        break; // Leave the loop once a file is created.
    }
}
logfile=SD.open(filename,FILE_WRITE);
Serial.print(F("Logging to: ")); Serial.println(filename);
logfile.println(F("yr,mon,day,hr,min,sec,day_frac,T(*C),RH(%)"));
logfile.flush();
tft.initR(INITR_144GREENTAB); // Initialize a 1.44" ST7735S chip (green tab).
tft.fillScreen(ST7735_BLACK);
// Set background and text.
tft.setCursor(x0-20,y0+20);
tft.setTextColor(ST7735_RED); tft.print("T, 0-40*C ");
tft.setCursor(x0+40,y0+20);
tft.setTextColor(ST7735_BLUE); tft.print("RH, 0-100%");
tft.setTextColor(ST7735_WHITE);
tft.drawFastHLine(x0,y0,x_axis,ST7735_WHITE);
tft.drawFastVLine(x0,y0-y_axis,y_axis,ST7735_WHITE);
// Print axis labels.
tft.setTextColor(ST7735_BLUE);
tft.setCursor(x0-20,y0-5); tft.print("0 "); // offset 5 pix per character
tft.setTextColor(ST7735_WHITE);
tft.setCursor(x0-20,y0-15); tft.print("---");
tft.setTextColor(ST7735_RED);
tft.setCursor(x0-20,y0-25); tft.print("0 ");
tft.setTextColor(ST7735_RED);
tft.setCursor(x0-20,y0-y_axis-5); tft.print(" 40");
tft.setTextColor(ST7735_WHITE);
tft.setCursor(x0-20,y0-y_axis+5); tft.print("---");
tft.setTextColor(ST7735_BLUE);
tft.setCursor(x0-20,y0-y_axis+15); tft.print("100");
tft.setTextColor(ST7735_WHITE);
tft.setCursor(x0+3,y0+5); tft.print("1 2 3 4 5 6 7 8");
// Display starting data (humidity=0).
for (int i=0; i<nBars; i++) {
    tft.fillRect(x0+i*barW+1,y0-0,barW/2-1,0,ST7735_RED);
    tft.fillRect(x0+i*barW+barW/2,y0-0,barW/2,0,ST7735_RED);
}
// Get first pair of readings to start populating array.
knt=0; fillFlag=0;
temperature[knt]=dht.readTemperature();
humidity[knt]=dht.readHumidity();
RH=scaleRH(humidity[i]); T=scaleT(temperature[knt]);
tft.fillRect(x0+i*barW+2,y0-T,barW/2,T,ST7735_RED);
tft.fillRect(x0+i*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
displayArray();
DateTime now=rtc.now();
dayFrac=now.day()+now.hour()/24.+now.minute()/1440.+now.second()/86400.;
logfile.print(now.year()); logfile.print(',');
logfile.print(now.month()); logfile.print(',');
logfile.print(now.day()); logfile.print(',');
logfile.print(now.hour()); logfile.print(',');
logfile.print(now.minute()); logfile.print(',');
logfile.print(now.second()); logfile.print(',');
logfile.print(dayFrac,5); logfile.print(',');
logfile.print(T,1); logfile.print(',');
logfile.print(RH,1); logfile.println();
logfile.flush(); delay(dt);
}
void loop() {
    DateTime now=rtc.now();
    // Fill all array elements for the first time.
    if (fillFlag==0) {
        knt++; temperature[knt]=dht.readTemperature();

```

```

humidity[knt]=dht.readHumidity();
displayArray();
RH=scaleRH(humidity[knt]); T=scaleT(temperature[knt]);
tft.fillRect(x0+knt*barW+2,y0-T,barW/2,T,ST7735_RED);
tft.fillRect(x0+knt*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
if (knt == (nBars-1)) fillFlag=1;
}
else {
for (i=0; i<(nBars-1); i++) {
temperature[i]=temperature[i+1]; humidity[i]=humidity[i+1];
RH=scaleRH(humidity[i]); T=scaleT(temperature[i]);
// next 2 lines "erase" previous bars
tft.fillRect(x0+i*barW+2,y0-T_max,barW/2,T_max,ST7735_BLACK);
tft.fillRect(x0+i*barW+barW+1,y0-100,barW/2,100,ST7735_BLACK);
tft.fillRect(x0+i*barW+2,y0-T,barW/2,T,ST7735_RED);
tft.fillRect(x0+i*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
}
temperature[nBars-1]=dht.readTemperature();
humidity[nBars-1]=dht.readHumidity();
RH=scaleRH(humidity[nBars-1]); T=scaleT(temperature[nBars-1]);
// Next 2 lines "erase" previous bars.
tft.fillRect(x0+(nBars-1)*barW+2,y0-T_max,barW/2,T_max,ST7735_BLACK);
tft.fillRect(x0+(nBars-1)*barW+barW/2+1,y0-100,barW/2,100,ST7735_BLACK);
// Now print new bars.
tft.fillRect(x0+(nBars-1)*barW+2,y0-T,barW/2,T,ST7735_RED);
tft.fillRect(x0+(nBars-1)*barW+barW/2+1,y0-RH,barW/2,RH,ST7735_BLUE);
displayArray();
}
now=rtc.now(); minit=now.minute();
// These two if... tests write data at the specified (dm) interval and make
// sure that data for any selected minute isn't recorded more than once.
if (minit%dm == 0) {
if (minit != last_written) {
Serial.println(F("Write to SD."));
dayFrac=now.day()+now.hour()/24.+now.minute()/1440.+now.second()/86400.;
logfile.print(now.year()); logfile.print(',');
logfile.print(now.month()); logfile.print(',');
logfile.print(now.day()); logfile.print(',');
logfile.print(now.hour()); logfile.print(',');
logfile.print(now.minute()); logfile.print(',');
logfile.print(now.second()); logfile.print(',');
logfile.print(dayFrac,5); logfile.print(',');
logfile.print(T,1); logfile.print(',');
logfile.print(RH,1); logfile.println();
logfile.flush(); last_written=minit;
}
}
delay(dt);
}
void displayArray() {
for (i=0; i<nBars; i++) {
Serial.print("i = ");Serial.print(i);
Serial.print(" RH = ");Serial.print(humidity[i],1);
Serial.print(" T = ");Serial.println(temperature[i],1);
}
Serial.println("-----");
}
int scaleRH(float RH) {
return round(y_axis*(RH-RH_min)/(RH_max-RH_min));
}
int scaleT(float T) {
return round(y_axis*(T-T_min)/(T_max-T_min));
}
}

```

## Drawing other kinds of graphs

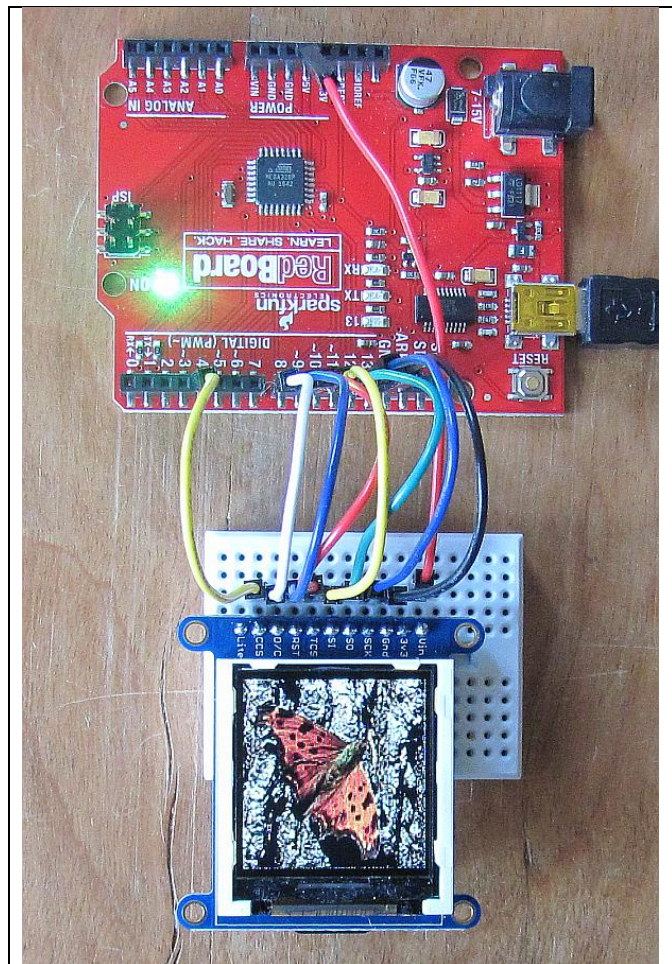
The techniques used for drawing bar graphs can also be applied to drawing line graphs, using the graphic library's line drawing command. You can graph several data sets at once, using different colors, and you can use the same approach to updating data so that the last (rightmost) value is always the most recent value from one or more sensors.

## Drawing bitmap images

As a bonus, the ID 2088 TFT display from adafruit.com includes a slot for a micro SD card that you can use to store images to be displayed on the screen. (It's on the bottom of the TFT display board and you can just see the edge of the microSD card extending a little below the lower edge of the display board in the image below.) The images must be in 24-color format (the usual format for such images), and they should be no larger than 127 x 127 pixels. You can convert any image to .bmp format and resize it by using a program such as the freeware IrfanView image processing program. Two more connections are required to access the SD card:

SD → digital 12  
CCS → digital 4

You can store as many images as you like on the SD card, and you can position multiple smaller images anywhere on the display. File names can be no more than 7 characters, plus the .bmp extension. Here's an example. This file, `butrfly.bmp`, is cropped and resized from my own butterfly JPG image and saved as a .bmp file using IrfanView. The RedBoard is an Arduino UNO equivalent that I often use.



Here is the code, verbatim from adafruit.com except for changing the file name:

```
/******  
  This is a library for the Adafruit 1.8" SPI display.  
  This library works with the Adafruit 1.8" TFT Breakout w/SD card  
  ----> http://www.adafruit.com/products/358  
  The 1.8" TFT shield  
  ----> https://www.adafruit.com/product/802  
  The 1.44" TFT breakout  
  ----> https://www.adafruit.com/product/2088  
  as well as Adafruit raw 1.8" TFT display  
  ----> http://www.adafruit.com/products/618  
  Check out the links above for our tutorials and wiring diagrams  
  These displays use SPI to communicate, 4 or 5 pins are required to  
  interface (RST is optional)  
  Adafruit invests time and resources providing this open source code,  
  please support Adafruit and open-source hardware by purchasing  
  products from Adafruit!  
  Written by Limor Fried/Ladyada for Adafruit Industries.  
  MIT license, all text above must be included in any redistribution  
  *****/  
#include <Adafruit_GFX.h>    // Core graphics library  
#include <Adafruit_ST7735.h> // Hardware-specific library  
#include <SPI.h>  
#include <SD.h>  
  
// TFT display and SD card will share the hardware SPI interface.  
// Hardware SPI pins are specific to the Arduino board type and  
// cannot be remapped to alternate pins.  For Arduino Uno,  
// Duemilanove, etc., pin 11 = MOSI, pin 12 = MISO, pin 13 = SCK.  
#define TFT_CS  10 // Chip select line for TFT display  
#define TFT_RST  9 // Reset line for TFT (or see below...)  
#define TFT_DC   8 // Data/command line for TFT  
  
#define SD_CS    4 // Chip select line for SD card  
  
//Use this reset pin for the shield!  
//#define TFT_RST 0 // you can also connect this to the Arduino reset!  
  
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_RST);  
  
void setup(void) {  
  Serial.begin(9600);  
  
  // Use this initializer if you're using a 1.8" TFT  
  //tft.initR(INITR_BLACKTAB);  
  
  // Use this initializer (uncomment) if you're using a 1.44" TFT  
  tft.initR(INITR_144GREENTAB);  
  
  Serial.print("Initializing SD card...");  
  if (!SD.begin(SD_CS)) {  
    Serial.println("failed!");  
    return;  
  }  
  Serial.println("OK!");  
}
```

```

    // change the name here!
    bmpDraw("butfly.bmp", 0, 0);
    // wait 5 seconds
    delay(5000);
}

void loop() {
// uncomment these lines to draw bitmaps in different locations/rotations!
/*
    tft.fillScreen(ST7735_BLACK); // Clear display
    for(uint8_t i=0; i<4; i++) // Draw 4 parrots
        bmpDraw("parrot.bmp", tft.width() / 4 * i, tft.height() / 4 * i);
    delay(1000);
    tft.setRotation(tft.getRotation() + 1); // Inc rotation 90 degrees
*/
}

// This function opens a Windows Bitmap (BMP) file and
// displays it at the given coordinates. It's sped up
// by reading many pixels worth of data at a time
// (rather than pixel by pixel). Increasing the buffer
// size takes more of the Arduino's precious RAM but
// makes loading a little faster. 20 pixels seems a
// good balance.

#define BUFFPIXEL 20

void bmpDraw(char *filename, uint8_t x, uint8_t y) {

    File    bmpFile;
    int     bmpWidth, bmpHeight; // W+H in pixels
    uint8_t bmpDepth;           // Bit depth (currently must be 24)
    uint32_t bmpImageoffset;    // Start of image data in file
    uint32_t rowSize;           // Not always = bmpWidth; may have padding
    uint8_t  sdbuffer[3*BUFFPIXEL]; // pixel buffer (R+G+B per pixel)
    uint8_t  buffidx = sizeof(sdbuffer); // Current position in sdbuffer
    boolean  goodBmp = false;    // Set to true on valid header parse
    boolean  flip    = true;     // BMP is stored bottom-to-top
    int      w, h, row, col;
    uint8_t  r, g, b;
    uint32_t pos = 0, startTime = millis();

    if((x >= tft.width()) || (y >= tft.height())) return;

    Serial.println();
    Serial.print("Loading image '");
    Serial.print(filename);
    Serial.println('\n');

    // Open requested file on SD card
    if ((bmpFile = SD.open(filename)) == NULL) {
        Serial.print("File not found");
        return;
    }

    // Parse BMP header
    if(read16(bmpFile) == 0x4D42) { // BMP signature
        Serial.print("File size: "); Serial.println(read32(bmpFile));
        (void)read32(bmpFile); // Read & ignore creator bytes
        bmpImageoffset = read32(bmpFile); // Start of image data
        Serial.print("Image Offset: "); Serial.println(bmpImageoffset, DEC);
        // Read DIB header

```

```

Serial.print("Header size: "); Serial.println(read32 bmpFile));
bmpWidth = read32 bmpFile);
bmpHeight = read32 bmpFile);
if(read16 bmpFile) == 1) { // # planes -- must be '1'
  bmpDepth = read16 bmpFile); // bits per pixel
  Serial.print("Bit Depth: "); Serial.println bmpDepth);
  if((bmpDepth == 24) && (read32 bmpFile) == 0) { // 0 = uncompressed

    goodBmp = true; // Supported BMP format -- proceed!
    Serial.print("Image size: ");
    Serial.print bmpWidth);
    Serial.print('x');
    Serial.println bmpHeight);

    // BMP rows are padded (if needed) to 4-byte boundary
    rowSize = (bmpWidth * 3 + 3) & ~3;

    // If bmpHeight is negative, image is in top-down order.
    // This is not canon but has been observed in the wild.
    if bmpHeight < 0) {
      bmpHeight = -bmpHeight;
      flip = false;
    }

    // Crop area to be loaded
    w = bmpWidth;
    h = bmpHeight;
    if((x+w-1) >= tft.width()) w = tft.width() - x;
    if((y+h-1) >= tft.height()) h = tft.height() - y;

    // Set TFT address window to clipped image bounds
    tft.setAddrWindow(x, y, x+w-1, y+h-1);

    for (row=0; row<h; row++) { // For each scanline...

      // Seek to start of scan line. It might seem labor-
      // intensive to be doing this on every line, but this
      // method covers a lot of gritty details like cropping
      // and scanline padding. Also, the seek only takes
      // place if the file position actually needs to change
      // (avoids a lot of cluster math in SD library).
      if(flip) // Bitmap is stored bottom-to-top order (normal BMP)
        pos = bmpImageoffset + (bmpHeight - 1 - row) * rowSize;
      else // Bitmap is stored top-to-bottom
        pos = bmpImageoffset + row * rowSize;
      if bmpFile.position() != pos) { // Need seek?
        bmpFile.seek(pos);
        buffidx = sizeof(sdbuffer); // Force buffer reload
      }

      for (col=0; col<w; col++) { // For each pixel...
        // Time to read more pixel data?
        if (buffidx >= sizeof(sdbuffer)) { // Indeed
          bmpFile.read(sdbuffer, sizeof(sdbuffer));
          buffidx = 0; // Set index to beginning
        }

        // Convert pixel from BMP to TFT format, push to display
        b = sdbuffer[buffidx++];
        g = sdbuffer[buffidx++];
        r = sdbuffer[buffidx++];
        tft.pushColor(tft.Color565(r,g,b));
      } // end pixel
    }

```

```

        } // end scanline
        Serial.print("Loaded in ");
        Serial.print(millis() - startTime);
        Serial.println(" ms");
    } // end goodBmp
}

bmpFile.close();
if(!goodBmp) Serial.println("BMP format not recognized.");
}

// These read 16- and 32-bit types from the SD card file.
// BMP data is stored little-endian, Arduino is little-endian too.
// May need to reverse subscript order if porting elsewhere.

uint16_t read16(File f) {
    uint16_t result;
    ((uint8_t *)&result)[0] = f.read(); // LSB
    ((uint8_t *)&result)[1] = f.read(); // MSB
    return result;
}

uint32_t read32(File f) {
    uint32_t result;
    ((uint8_t *)&result)[0] = f.read(); // LSB
    ((uint8_t *)&result)[1] = f.read();
    ((uint8_t *)&result)[2] = f.read();
    ((uint8_t *)&result)[3] = f.read(); // MSB
    return result;
}

```