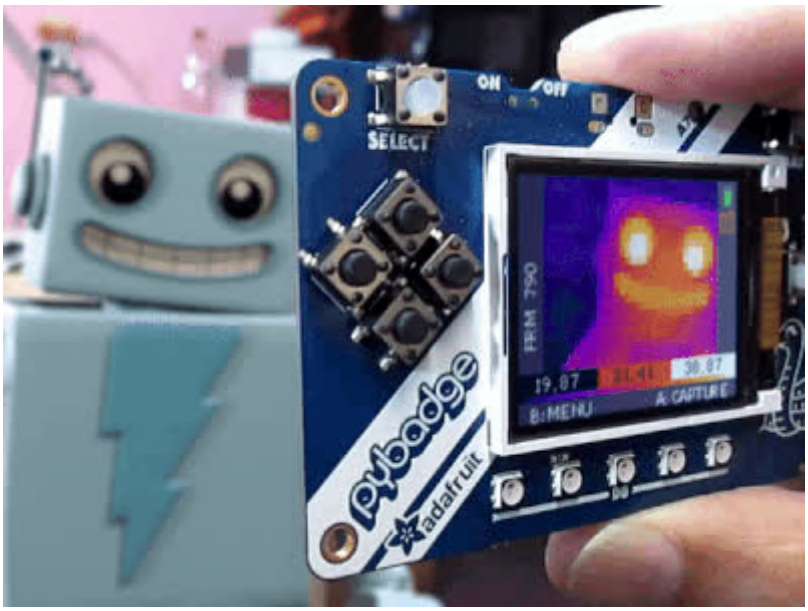
# MLX90640 Thermal Camera with Image Recording

Created by Eduardo Blume



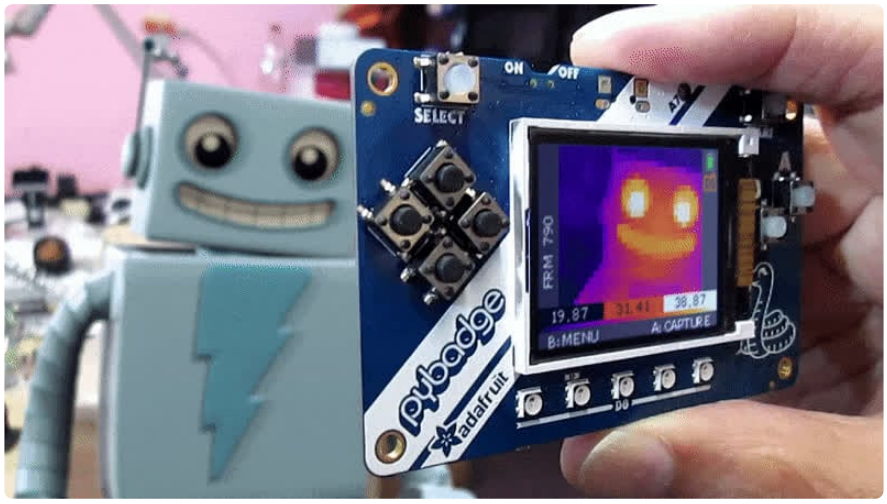

https://learn.adafruit.com/mlx90640-thermal-image-recording

Last updated on 2024-06-03 02:59:07 PM EDT

# Table of Contents

# Overview

Aren't thermal imaging cameras amazing? They can reveal the heat in circuits, engines, plumbing, home insulation, firefighting, medicine, forensics, environmental science, search and rescue, energy monitoring, chemical reactions, detecting wildlife, detecting trespassers. What could you do with one? Find the cold spots in your house? Fine-tune the cooling system in your overclocked game machine? Investigate whether Rover was sitting in your recliner? Too bad the price puts these cameras out of reach... or it used to, until now.




The Adafruit MLX90640 IR Thermal Camera Breakout (http://adafru.it/4407) makes low resolution thermal imaging affordable. It can detect a 32 by 24 pixel grid of temperatures within a 55 degree field of view, or a 110 degree field of view (http://adafru.it/4469). With a few extra goodies from Adafruit, you can turn it into your own mini thermal recording camera.

Your Feather-based thermal camera will be able to capture and store the images it senses as modified BMP files on flash media. These can be downloaded to a full-sized computer where the temperature data can be visualized and set in motion using open source tools written in the Processing (https://adafru.it/ldl) language.

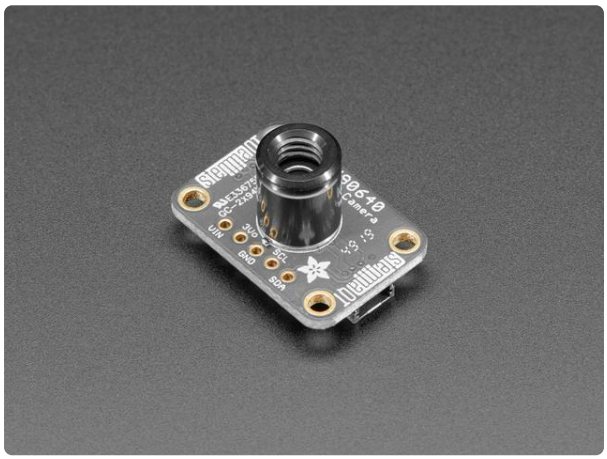
And don't miss the 3D printing guide, PyBadge Thermal Camera Case (https://adafru.it/Lla), for a project that will enhance your camera's protection, ergonomics, and style.

**Learn Guide: PyBadge Thermal Camera Case**

https://adafru.it/Lla

## Parts

The project is based on the MLX90640 from Melexis (https://adafru.it/JD2), a low cost sensor that uses thermopile (https://adafru.it/JD3) technology.

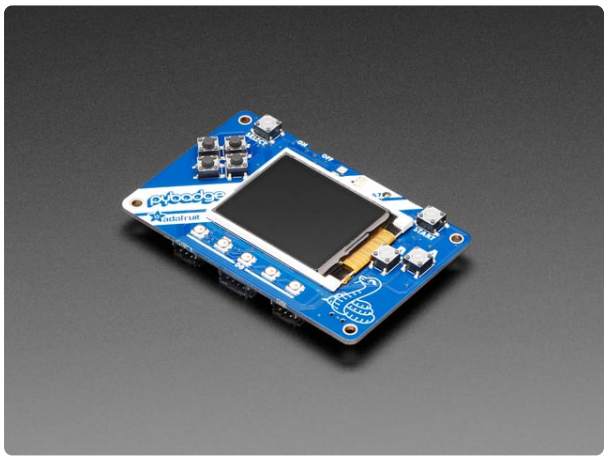


Adafruit MLX90640 IR Thermal Camera Breakout

You can now add affordable heat-vision to your project and with an Adafruit MLX90640 Thermal Camera Breakout. This sensor contains a 24x32 array of IR thermal sensors. When connected...

https://www.adafruit.com/product/4407

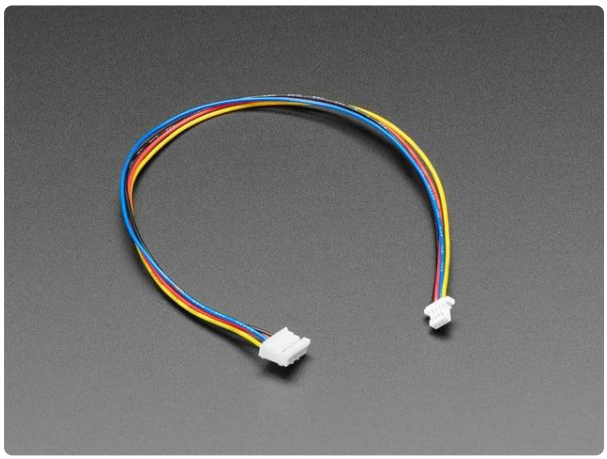
You'll need a board with Adafruit Arcada (https://adafru.it/EUk) support + a TFT LCD screen and STEMMA connectors.



Adafruit PyBadge for MakeCode Arcade, CircuitPython, or Arduino

What's the size of a credit card and can run CircuitPython, MakeCode Arcade or Arduino? That's right, its the Adafruit PyBadge! We wanted to see how much we...

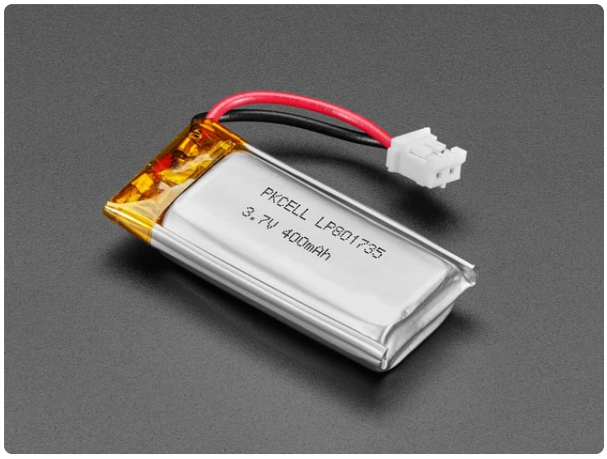
https://www.adafruit.com/product/4200



4-pin JST PH to JST SH Cable - STEMMA to QT / Qwiic

Are you a maker in the midst of a STEMMA dilemma? This 200mm long 4-wire...

https://www.adafruit.com/product/4424



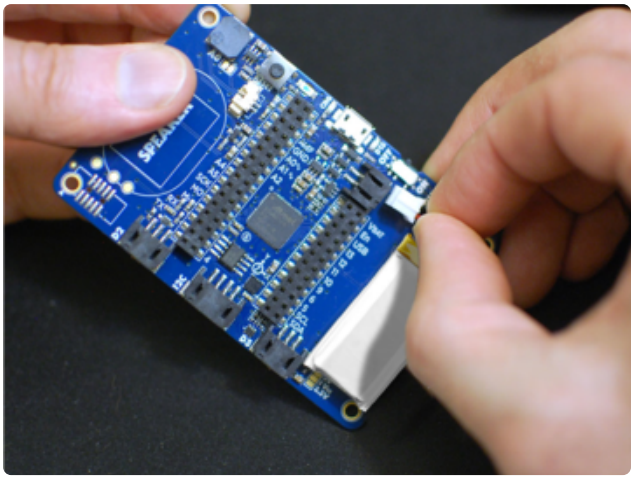

[Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh]

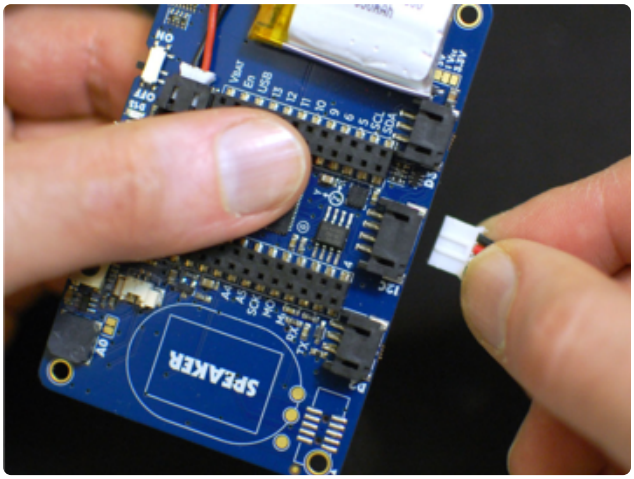
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/3898

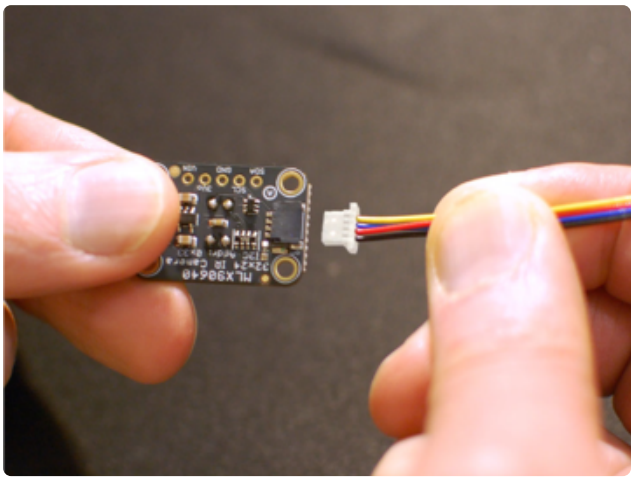
[USB cable - USB A to Micro-B]

This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

# Adafruit Assemble!





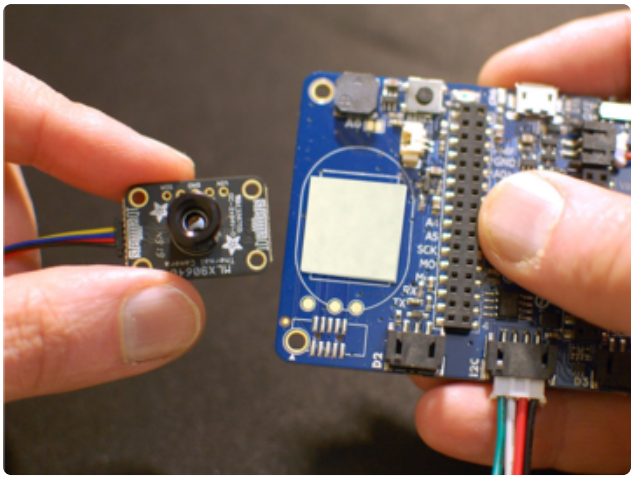



Assembling the project is pretty straightforward. Use the mini JST connectors to plug the Li-Po battery into its matching socket.

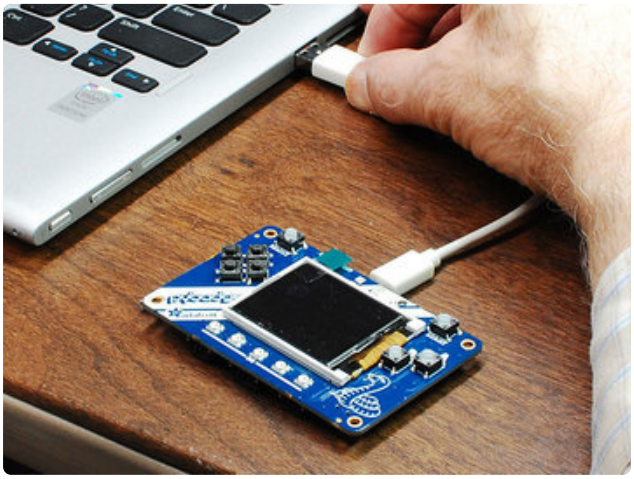
The PyBadge and PyGamer have a clear space on the circuit board where the battery can be attached with a bit of double-stick foam tape. Since Feather boards have built-in Li-Po chargers, the battery won't need to be removed.

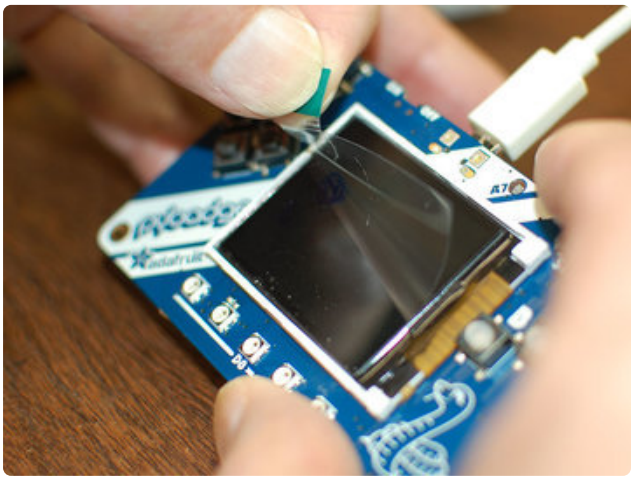
Use the short STEMMA wire to connect the MLX90640 board to the microcontroller's 4-pin connector.

You might want another piece of foam tape to attach the camera board to another empty space on the microcontroller board, or onto the outside of an enclosure. Pay attention to which way the sensor is oriented.



(**The circuit board's silkscreen is printed upside-down, so be careful.  Some sketches don't account for the MLX90640's bottom-to-top pixel mapping and appear flipped vertically.**)  If you're unsure which way is correct, complete the remaining steps and run the ThermalImager sketch to double check the orientation before taping down the MLX90640 board.





With the USB cable, you are ready to charge the battery and upload code to the microcontroller.

Lastly, peel away the screen protector film from the display, if you wish.

# Support Libraries

Are you new to using Arduino? No worries, you can get started using our series of guides on
Getting Started with Arduino (https://adafru.it/Clw).

You can learn more about Arduino libraries by reading this guide from Adafruit (https://adafru.it/dit).

Follow your board's guide on installing Arduino IDE, and support for the board you have, like the PyBadge (https://adafru.it/Fmm), for example. Then install the Adafruit Arcada libraries (https://adafru.it/EUk) (there's a lot of them!).

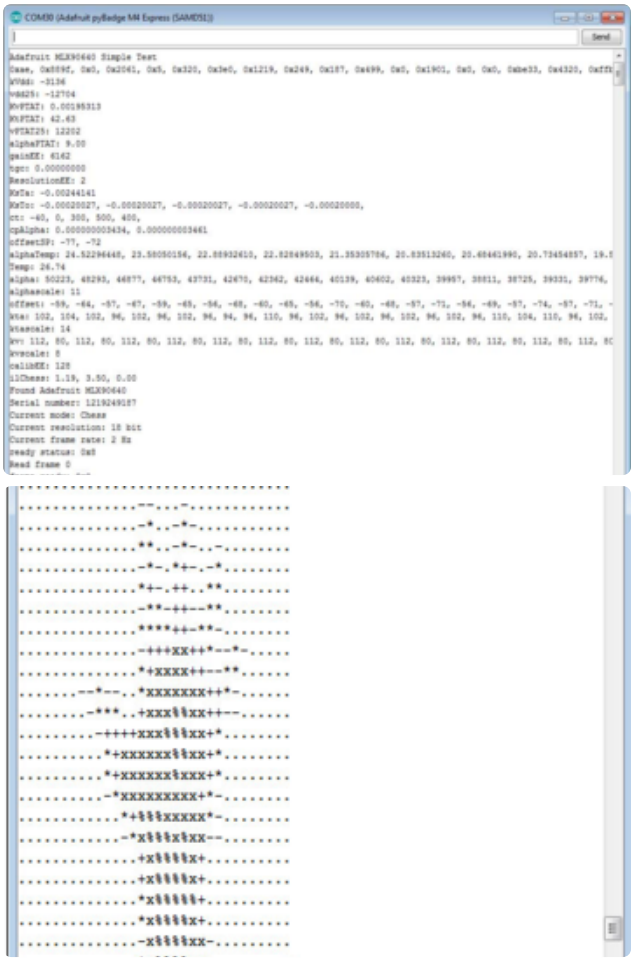
Once done, install the Adafruit_MLX90640 (https://adafru.it/IDj) library.

Now, even as I write, I've learned that the Adafruit_EPD (https://adafru.it/BRK) and Adafruit_BusIO (https://adafru.it/GxD) libraries are required as well, which prompts me to make this caution.

Adafruit's libraries are like any other ecosystem, changing and adapting. New languages and development tools can create new dependencies, and yesterday's requirements for a project might be different from today's. It's happened to me, so I recommend keeping the Adafruit Support Forums as a permanent bookmark.

## Test the Circuit

Included with the library are two example sketches which can confirm that the sensor is functioning. MLX90640_simpletest (https://adafru.it/JD4) is like a Hello World for your circuit. Compile it in your Arduino IDE, upload it, run it, and it should begin sending serial text that you can view in the serial console.



After sending the status of some of the sensor's hardware settings, the sketch goes on to serial print ASCII art (https://adafru.it/JD5) images of the temperature pattern sensed by the MLX90640.

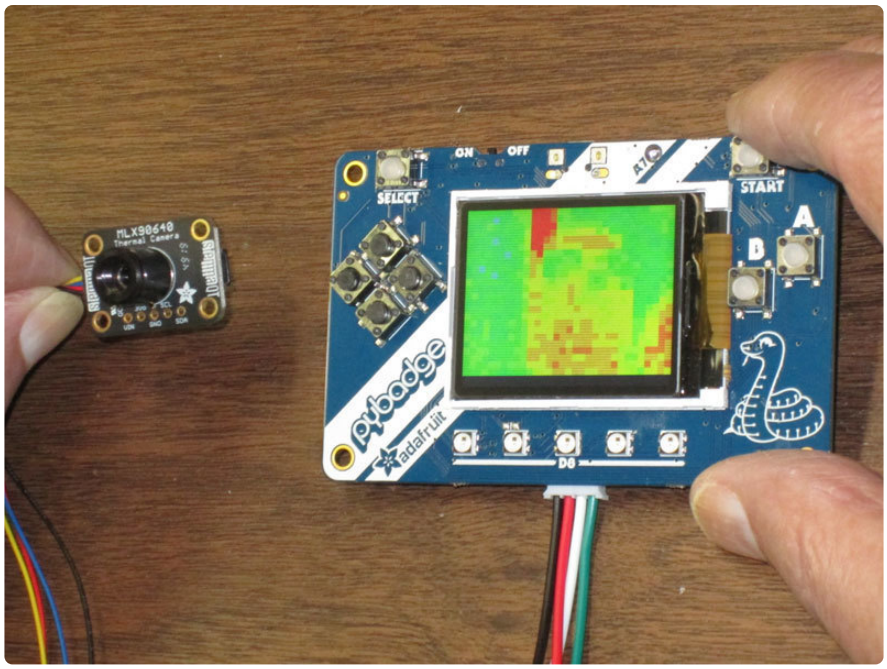
Look for this pair of lines near the head of the sketch.

```
//#define PRINT_TEMPERATURES
```

```
#define PRINT_ASCIIART
```

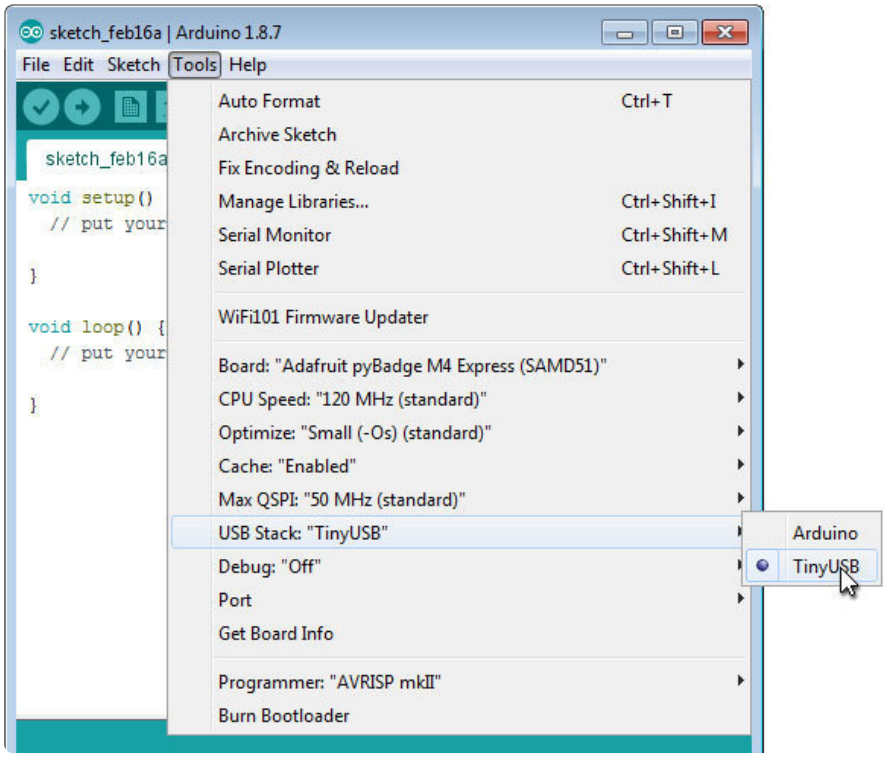
By changing which one of the two is uncommented, the ASCII art output will be replaced by arrays of numeric values.

The other example sketch, MLX90640_arcadacam (https://adafru.it/JD6), presents the array of temperatures as a color image on the TFT screen.  Have a look around at your fascinating thermal world!




## Enabling TinyUSB

One other important detail needs attention before proceeding.  If, like me, your images will be captured to built-in flash memory, like that on a PyBadge, look to the Arduino window menu bar. Choose this setting, Tools > USB Stack > TinyUSB.  It's an essential step in giving your computer access to QSPI flash storage as a virtual USB external drive. It's a handy way to read the images you'll capture.



The other settings in the list? I leave them at their standard values. You may want to change them for higher performance, whatever you prefer.

Congratulations!  Your hardware is ready for the next step, the full camera sketch.

---

# Uploading the Sketch

Okay, take a deep breath.  Here's the latest version of the full sketch to copy and upload.  See you at the other end!

```
// SPDX-FileCopyrightText: 2020 Anne Barela for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*
  ThermalImager_009b - Collect thermal image values from a MLX90640 sensor array,
                       display them as color-mapped pixels on a TFT screen,
  include data capture to flash media, and a user configuration menu.
  Written by Eduardo using code from these sources.

  Arcada and MLX90640 libraries from adafruit.com

  Ver.  1 - Read temps, auto-range extremes, display gray squares on TFT
  Ver.  2 - Add Ironbow color palette, low+center+high markers
  Ver.  3 - Add crude BMP image write to SD
  Ver.  4 - Attach interrupts to improve button response
  Ver.  5 - Store BMPs to SD in an orderly manner, in folders
  Ver.  6 - Port to Teensy 3.2, where the libraries used are suited
  Ver.  7 - Port to Adafruit PyBadge using Arcada library.  Use simulated data
while awaiting hardware release
  Ver.  8 - Convert menu to scrolling style and add settings for emissivity and
frame rate, more if feasible.
  Ver.  9 - Bring in the current Adafruit library and read a real sensor.
*/

#include <Adafruit_MLX90640.h>
#include "Adafruit_Arcada.h"
Adafruit_MLX90640 mlx;
Adafruit_Arcada arcada;

#if !defined(USE_TINYUSB)
  #warning "Compile with TinyUSB selected!"
#endif

File myFile;

float mlx90640To[768];   // Here we receive the float vals acquired from MLX90640

#define DE_BOUNCE 200
  // Wait this many msec between button clicks
#define MENU_LEN 12
  // Number of total available menu choices
#define MENU_ROWS 9
  // Number of menu lines that can fit on screen
#define MENU_VPOS 6
#define GRAY_33 0x528A
#define BOTTOM_DIR "MLX90640"
#define DIR_FORMAT "/dir%05d"
#define BMP_FORMAT "/frm%05d.bmp"
#define CFG_FLNAME "/config.ini"
#define MAX_SERIAL 999
```

```
// BMP File Header, little end first, Photoshop ver.
const PROGMEM uint8_t BmpPSPHead[14] = {
 0x42, 0x4D,             // "BM" in hex
 0x38, 0x09, 0x00, 0x00, // File size, 2360
 0x00, 0x00,             // reserved for app data 1
 0x00, 0x00,             // reserved for app data 2
 0x36, 0x00, 0x00, 0x00  // Offset of first pixel, 54
};

// BMP 24-bit DIB Header, little end first, Photoshop ver.
const PROGMEM uint8_t DIBHeadPSP1[40] = {
 0x28, 0x00, 0x00, 0x00,  // Header size, 40
 0x20, 0x00, 0x00, 0x00,  // pixel width, 32
 0x18, 0x00, 0x00, 0x00,  // pixel height, 24
 0x01, 0x00,              // color planes, 1
 0x18, 0x00,              // bits per pixel, 24
 0x00, 0x00, 0x00, 0x00,  // Compression method, 0==none
 0x00, 0x00, 0x00, 0x00,  // Raw bitmap data size, dummy 0
 0x12, 0x0B, 0x00, 0x00,  // Pixels per meter H, 2834
 0x12, 0x0B, 0x00, 0x00,  // Pixels per meter V, 2834
 0x00, 0x00, 0x00, 0x00,  // Colors in palette, 0==default 2^n
 0x00, 0x00, 0x00, 0x00   // Number of important colors, 0
};

// BMP file data, 2 byte padding
const PROGMEM uint8_t PSPpad[2] = {0x00, 0x00};

//Byte arrays of bitmapped icons, 16 x 12 px:
const PROGMEM uint8_t battIcon[] = {
0x0f, 0x00, 0x3f, 0xc0, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40,
0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x3f, 0xc0};

const PROGMEM uint8_t camIcon[] = {
0x01, 0xe0, 0x61, 0x20, 0xff, 0xf0, 0x80, 0x10, 0x86, 0x10, 0x89, 0x10,
0x90, 0x90, 0x90, 0x90, 0x89, 0x10, 0x86, 0x10, 0x80, 0x10, 0xff, 0xf0};

const PROGMEM uint8_t SDicon[] = {
0x0f, 0xe0, 0x1f, 0xe0, 0x3c, 0x60, 0x78, 0x60, 0x70, 0x60, 0x60, 0x60,
0x60, 0x60, 0x60, 0x60, 0x6f, 0x60, 0x60, 0x60, 0x7f, 0xe0, 0x7f, 0xe0};

const PROGMEM uint8_t snowIcon[] = {
0x15, 0x00, 0x4E, 0x40, 0xC4, 0x60, 0x75, 0xC0, 0x9F, 0x20, 0x0E, 0x00,
0x0E, 0x00, 0x9F, 0x20, 0x75, 0xC0, 0xC4, 0x60, 0x4E, 0x40, 0x15, 0x00};

uint8_t pixelArray[2304];   // BMP image body, 32 pixels * 24 rows * 3 bytes

// Some global values that several functions will use, including
// 5 floats to append to the BMP pixel data:
// coldest pixel, coldest color, center temp, hottest color, hottest pixel
float sneakFloats[5] = {3.1415926, 0.0, -11.7, 98.6, -12.34};      // Test values
that get overwritten
uint16_t highAddr = 0, lowAddr = 0;                                // Append the
pixel addresses, too

uint16_t backColor, lowPixel, highPixel, buttonRfunc = 1,
         emissivity = 95, frameRate = 4,
         thermRange = 0, paletteNum = 1, colorPal[256],            // Array for
color palettes
         nextDirIndex = 0, nextBMPindex = 0, nextBMPsequence = 1;  // These keep
count of SD files and dirs, 0==error
uint32_t deBounce = 0, buttonBits = 0;
boolean mirrorFlag = false, celsiusFlag = false, markersOn = true,
        screenDim = false, smoothing = false, showLastCap = false,
        save1frame = false, recordingInProg = false, buttonActive = false;
float battAverage = 0.0, colorLow = 0.0, colorHigh = 100.0;        // Values for
managing color range
volatile boolean clickFlagMenu = false, clickFlagSelect = false;   // Volatiles for
timer callback handling
```

```
void setup()
{
  if (!arcada.arcadaBegin()) {    // Start TFT and fill with black
    // Serial.print("Failed to begin");
    while (1);
  }
  arcada.filesysBeginMSD();       // Set up SD or QSPI flash as an external USB
drive

  arcada.displayBegin();              // Activate TFT screen
  arcada.display->setRotation(1);     // wide orientation
  arcada.display->setTextWrap(false);
  arcada.setBacklight(255);           // Turn on backlight
  battAverage = arcada.readBatterySensor();

  Serial.begin(115200);
//  while(!Serial); // Wait for user to open terminal
  Serial.println("MLX90640 IR Array Example");

  if(arcada.filesysBegin()){              // Initialize flash storage, begin
setting up indices for saving BMPs
    if(!arcada.exists(BOTTOM_DIR)) {      // Is base "MLX90640" directory absent?
      if(arcada.mkdir(BOTTOM_DIR))        // Can it be added?
        nextDirIndex = nextBMPindex = 1;  // Success, prepare to store numbered
files & dirs
    } else {      // "MLX90640" directory exists, can we add files | directories?
      // Get the number of the next unused serial directory path
      nextDirIndex = availableFileNumber(1, BOTTOM_DIR + String(DIR_FORMAT));
      // and the next unused serial BMP name
      nextBMPindex = availableFileNumber(1, BOTTOM_DIR + String(BMP_FORMAT));
    }
  }  // By now each global index variable is either 0 (no nums available), or the
next unclaimed serial num

  if(!mlx.begin(MLX90640_I2CADDR_DEFAULT, &Wire)) {
    Serial.println("MLX90640 not found!");
    arcada.haltBox("MLX90640 not found!");
    while(1)
      delay(10);  // Halt here
  }
  Serial.println("Found Adafruit MLX90640");

  Serial.print("Serial number: ");
  Serial.print(mlx.serialNumber[0], HEX);
  Serial.print(mlx.serialNumber[1], HEX);
  Serial.println(mlx.serialNumber[2], HEX);

  //mlx.setMode(MLX90640_INTERLEAVED);
  mlx.setMode(MLX90640_CHESS);
  mlx.setResolution(MLX90640_ADC_18BIT);

  switch(frameRate) {
    case 0: mlx.setRefreshRate(MLX90640_0_5_HZ); break; // 6 frame rates, 0.5 to 16
FPS in powers of 2
    case 1: mlx.setRefreshRate(MLX90640_1_HZ); break;
    case 2: mlx.setRefreshRate(MLX90640_2_HZ); break;
    case 3: mlx.setRefreshRate(MLX90640_4_HZ); break;
    case 4: mlx.setRefreshRate(MLX90640_8_HZ); break;
    default: mlx.setRefreshRate(MLX90640_16_HZ); break;
  }
  Wire.setClock(1000000); // max 1 MHz

  for(int counter01 = 0; counter01 < 2304; ++counter01)
    pixelArray[counter01] = counter01 / 9;  // Initialize BMP pixel buffer with a
gradient

  loadPalette(paletteNum);             // Load false color palette
  backColor = GRAY_33;                 // 33% gray for BG
  setBackdrop(backColor, buttonRfunc); // Current BG, current button labels
```

```
  arcada.timerCallback(50, buttonCatcher);  // Assign a 50Hz callback function to
catch button presses
}

void loop()
{
  static uint32_t frameCounter = 0;
  float scaledPix, highPix, lowPix;
  uint16_t markColor;

// Show the battery level indicator, 3.7V to 3.3V represented by a 7 segment bar
  battAverage = battAverage * 0.95 + arcada.readBatterySensor() * 0.05; //
*Gradually* track battery level
  highPix = (int)constrain((battAverage - 3.3) * 15.0, 0.0, 6.0) + 1;   // Scale it
to a 7-segment bar
  markColor = highPix > 2 ? 0x07E0 : 0xFFE0;                            // Is the
battery level bar green or yellow?
  markColor = highPix > 1 ? markColor : 0xF800;                         // ...or
even red?
  arcada.display->fillRect(146, 2, 12, 12, backColor);                  // Erase
old battery icon
  arcada.display->drawBitmap(146, 2, battIcon, 16, 12, 0xC618);         // Redraw
gray battery icon
  arcada.display->fillRect(150, 12 - highPix, 4, highPix, markColor);   // Add the
level bar

// Fetch 768 fresh temperature values from the MLX90640
  arcada.display->drawBitmap(146, 18, camIcon, 16, 12, 0xF400); // Show orange
camera icon during I2C acquisition
  if(mlx.getFrame(mlx90640To) != 0) {
    Serial.println("Failed");
    return;
  }
  arcada.display->fillRect(146, 18, 12, 12, backColor);         // Acquisition
done, erase camera icon

// First pass: Find hottest and coldest pixels
  highAddr = lowAddr = 0;
  highPix  = lowPix  = mlx90640To[highAddr];

  for (int x = 1 ; x < 768 ; x++) { // Compare every pixel
    if(mlx90640To[x] > highPix) {   // Hotter pixel found?
      highPix = mlx90640To[x];      // Record its values
      highAddr = x;
    }
    if(mlx90640To[x] < lowPix) {    // Colder pixel found?
      lowPix = mlx90640To[x];       // Likewise
      lowAddr = x;
    }
  }
  if(thermRange == 0) {    // Are the colors set to auto-range?
    colorLow = lowPix;     // Then high and low color values get updated
    colorHigh = highPix;
  }
  sneakFloats[0] = lowPix;     // Retain these five temperature values
  sneakFloats[1] = colorLow;   // to append to the BMP file, if any
  sneakFloats[2] = mlx90640To[400];
  sneakFloats[3] = colorHigh;
  sneakFloats[4] = highPix;

// Second pass: Scale the float values down to 8-bit and plot colormapped pixels
  if(mirrorFlag) {                 // Mirrored display (selfie mode)?
    for(int y = 0; y < 24; ++y) {  // Rows count from bottom up
      for(int x = 0 ; x < 32 ; x++) {
        scaledPix = constrain((mlx90640To[32 * y + x] - colorLow) / (colorHigh -
colorLow) * 255.9, 0.0, 255.0);
        pixelArray[3 * (32 * y + x)] =
(uint8_t)scaledPix;                          // Store as a byte in BMP buffer
```

```
        arcada.display->fillRect(140 - x * 4, 92 - y * 4, 4, 4,
colorPal[(uint16_t)scaledPix]);  // Filled rectangles, bottom up
      }
    }
  } else {  // Not mirrored
    for(int y = 0; y < 24; ++y) {
      for(int x = 0 ; x < 32 ; x++) {
        scaledPix = constrain((mlx90640To[32 * y + x] - colorLow) / (colorHigh -
colorLow) * 255.9, 0.0, 255.0);
        pixelArray[3 * (32 * y + x)] = (uint8_t)scaledPix;
        arcada.display->fillRect(16 + x * 4, 92 - y * 4, 4, 4,
colorPal[(uint16_t)scaledPix]);
      }
    }
  }

// Post pass: Screen print the lowest, center, and highest temperatures
  arcada.display->fillRect(  0, 96, 53, 12, colorPal[0]);                  //
Contrasting mini BGs for cold temp
  arcada.display->fillRect(107, 96, 53, 12, colorPal[255]);                // and
for hot temperature texts
  scaledPix = constrain((mlx90640To[400] - colorLow) / (colorHigh - colorLow) *
255.9, 0.0, 255.0);
  arcada.display->fillRect(53, 96, 54, 12, colorPal[(uint16_t)scaledPix]); // Color
coded mini BG for center temp

  arcada.display->setTextSize(1);
  arcada.display->setCursor(10, 99);
  arcada.display->setTextColor(0xFFFF ^ colorPal[0]);   // Contrasting text color
for coldest value
  arcada.display->print(celsiusFlag ? lowPix : lowPix * 1.8 + 32.0);  // Print
Celsius or Fahrenheit

  arcada.display->setCursor(120, 99);
  arcada.display->setTextColor(0xFFFF ^ colorPal[255]); // Contrast text for
hottest value
  arcada.display->print(celsiusFlag ? highPix : highPix * 1.8 + 32.0);  // Print
Celsius or Fahrenheit

  arcada.display->setCursor(65, 99);
  if((mlx90640To[400] < (colorLow + colorHigh) * 0.5) == (paletteNum < 3))
    arcada.display->setTextColor(0xFFFF);               // A contrasting text color
for center temp
  else
    arcada.display->setTextColor(0x0000);
  arcada.display->print(celsiusFlag ? mlx90640To[400] : mlx90640To[400] * 1.8 +
32.0);  // Pixel 12 * 32 + 16

  markColor = 0x0600;    // Deep green color to draw onscreen cross markers
  if(markersOn) {        // Show markers?
    if(mirrorFlag) {     // ...over a mirrored display?
      arcada.display->drawFastHLine(156 - (( lowAddr % 32) * 4 + 16), 93 - 4 * (
lowAddr / 32), 4, markColor); // Color crosses mark cold pixel,
      arcada.display->drawFastVLine(159 - (( lowAddr % 32) * 4 + 17), 92 - 4 * (
lowAddr / 32), 4, markColor);
      arcada.display->drawFastHLine(156 - ((highAddr % 32) * 4 + 16), 93 - 4 *
(highAddr / 32), 4, markColor); // hot pixel,
      arcada.display->drawFastVLine(159 - ((highAddr % 32) * 4 + 17), 92 - 4 *
(highAddr / 32), 4, markColor);
      arcada.display->drawFastHLine(76, 45, 4,
markColor);                                                 // and center pixel
      arcada.display->drawFastVLine(78, 44, 4, markColor);
    } else {             // Not mirrored
      arcada.display->drawFastHLine(( lowAddr % 32) * 4 + 16, 93 - 4 * ( lowAddr /
32), 4, markColor); // Color crosses mark cold pixel,
      arcada.display->drawFastVLine(( lowAddr % 32) * 4 + 17, 92 - 4 * ( lowAddr /
32), 4, markColor);
      arcada.display->drawFastHLine((highAddr % 32) * 4 + 16, 93 - 4 * (highAddr /
32), 4, markColor); // hot pixel,
```

```
      arcada.display->drawFastVLine((highAddr % 32) * 4 + 17, 92 - 4 * (highAddr /
32), 4, markColor);
      arcada.display->drawFastHLine(80, 45, 4,
markColor);                                          // and center pixel
      arcada.display->drawFastVLine(81, 44, 4, markColor);
    }
  }

// Print the frame count on the left sidebar
  arcada.display->setRotation(0);    // Vertical printing
  arcada.display->setCursor(48, 4);
  arcada.display->setTextColor(0xFFFF, backColor); // White text, current BG
  arcada.display->print("FRM ");
  arcada.display->print(++frameCounter);
  arcada.display->setRotation(1);    // Back to horizontal

// Handle any button presses
  if(!buttonActive && clickFlagMenu) {         // Was B:MENU button pressed?
    buttonActive = true;                       // Set button flag
    deBounce = millis() + DE_BOUNCE;           // and start debounce timer
    menuLoop(backColor);                       // Execute menu routine until
finished
    clickFlagSelect = recordingInProg = false; // Clear unneeded flags
    nextBMPsequence = 1;
    setBackdrop(backColor, buttonRfunc);       // Repaint current BG & button labels
  }

  if(!buttonActive && clickFlagSelect) { // Was the A button pressed?
    buttonActive = true;                 // Set button flag
    deBounce = millis() + DE_BOUNCE;     // and start debounce timer

    if(buttonRfunc == 0) {                                          // Freeze
requested?
      arcada.display->drawBitmap(146, 48, snowIcon, 16, 12, 0xC61F); // Freeze icon
on
      while(buttonBits & ARCADA_BUTTONMASK_A)                       // Naive
freeze: loop until button released
        delay(10);                                                  // Short pause
      deBounce = millis() + DE_BOUNCE;                              // Restart
debounce timer
      arcada.display->fillRect(146, 48, 12, 12, backColor);         // Freeze icon
off
    } else if(buttonRfunc == 1) {                         // Capture requested?
      if((nextBMPindex = availableFileNumber(nextBMPindex, BOTTOM_DIR +
String(BMP_FORMAT))) != 0) { // Serialized BMP filename available?
        save1frame = true;                                // Set the flag to save a
BMP
        arcada.display->fillRect(0, 96, 160, 12, 0x0600); // Display a green strip
        arcada.display->setTextColor(0xFFFF);             // with white capture
message text
        arcada.display->setCursor(16, 99);
        arcada.display->print("Saving frame ");
        arcada.display->print(nextBMPindex);
      }
    } else {                            // Begin or halt recording a sequence of
BMP files
      if(!recordingInProg) {            // "A:START RECORDING" was pressed
        if((nextDirIndex = availableFileNumber(nextDirIndex, BOTTOM_DIR +
String(DIR_FORMAT))) != 0) { // Serialized directory name available?
          // Make the directory
          if(newDirectory()) {          // Success in making a new sequence
directory?
            recordingInProg = true;     // Set the flag for saving BMP files
            nextBMPsequence = 1;        // ...numbered starting with 00001
            setBackdrop(backColor, 3);  // Show "A:STOP RECORDING" label
          } else                        // Couldn't make the new directory, so
            nextDirIndex = 0;           // disable further sequences
        }
      } else {                          // "A:STOP RECORDING" was pressed
```

```
        recordingInProg = false;
        setBackdrop(backColor, 2);      // Clear "A:STOP RECORDING" label
      }
    }
  }

// Saving any BMP images to flash media happens here
  if(save1frame || recordingInProg) {      // Write a BMP file to SD?
    arcada.display->drawBitmap(146, 32, SDicon, 16, 12, 0x07E0); // Flash storage
activity icon on

    prepForSave();                         // Save to flash.  Use global values for
parameters
    nextBMPsequence += recordingInProg ? 1 : 0;  // If recording a series,
increment frame count
    save1frame = false;                    // If one frame saved, clear the flag
afterwards

    arcada.display->fillRect(146, 32, 12, 12, backColor);        // Flash storage
activity icon off
  }

  if(showLastCap) {                       // Redisplay the last BMP saved?
    buttonActive = true;                  // Set button flag
    deBounce = millis() + DE_BOUNCE;      // and start debounce timer
    recallLastBMP(backColor);             // Redisplay last bitmap from buffer until
finished
    setBackdrop(backColor, buttonRfunc); // Repaint current BG & button labels
    showLastCap = false;
  }

// Here we protect against button bounces while the function loops
  if(buttonActive && millis() > deBounce && (buttonBits
     & (ARCADA_BUTTONMASK_B | ARCADA_BUTTONMASK_A)) == 0)  // Has de-bounce wait
expired & all buttons released?
    buttonActive = false;                 // Clear flag to allow another button press

  clickFlagMenu = clickFlagSelect = false; // End of the loop, clear all interrupt
flags
}

// Compute and fill an array with 256 16-bit color values
void loadPalette(uint16_t palNumber) {
  uint16_t x, y;
  float fleX, fleK;

  switch(palNumber) {
    case 1:  // Compute ironbow palette
      for(x = 0; x < 256; ++x) {
        fleX = (float)x / 255.0;

        // fleK = 65535.9 * (1.02 - (fleX - 0.72) * (fleX - 0.72) * 1.96);
        // fleK = (fleK > 65535.0) || (fleX > 0.75) ? 65535.0 : fleK;  // Truncate
red curve
        fleK = 63487.0 * (1.02 - (fleX - 0.72) * (fleX - 0.72) * 1.96);
        fleK = (fleK > 63487.0) || (fleX > 0.75) ? 63487.0 : fleK;  // Truncate red
curve
        colorPal[x] = (uint16_t)fleK & 0xF800;                     // Top 5 bits
define red

        // fleK = fleX * fleX * 2047.9;
        fleK = fleX * fleX * 2015.0;
        colorPal[x] += (uint16_t)fleK & 0x07E0;  // Middle 6 bits define green

        // fleK = 31.9 * (14.0 * (fleX * fleX * fleX) - 20.0 * (fleX * fleX) + 7.0
* fleX);
        fleK = 30.9 * (14.0 * (fleX * fleX * fleX) - 20.0 * (fleX * fleX) + 7.0 *
fleX);
        fleK = fleK < 0.0 ? 0.0 : fleK;          // Truncate blue curve
```

```
        colorPal[x] += (uint16_t)fleK & 0x001F;  // Bottom 5 bits define blue
      }
      break;
    case 2:  // Compute quadratic "firebow" palette
      for(x = 0; x < 256; ++x) {
        fleX = (float)x / 255.0;

        // fleK = 65535.9 * (1.00 - (fleX - 1.0) * (fleX - 1.0));
        fleK = 63487.0 * (1.00 - (fleX - 1.0) * (fleX - 1.0));
        colorPal[x] = (uint16_t)fleK & 0xF800;                      // Top 5 bits
define red

        // fleK = fleX < 0.25 ? 0.0 : (fleX - 0.25) * 1.3333 * 2047.9;
        fleK = fleX < 0.25 ? 0.0 : (fleX - 0.25) * 1.3333 * 2015.0;
        colorPal[x] += (uint16_t)fleK & 0x07E0;  // Middle 6 bits define green

        // fleK = fleX < 0.5 ? 0.0 : (fleX - 0.5) * (fleX - 0.5) * 127.9;
        fleK = fleX < 0.5 ? 0.0 : (fleX - 0.5) * (fleX - 0.5) * 123.0;
        colorPal[x] += (uint16_t)fleK & 0x001F;  // Bottom 5 bits define blue
      }
      break;
    case 3:  // Compute "alarm" palette
      for(x = 0; x < 256; ++x) {
        fleX = (float)x / 255.0;

        fleK = 65535.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : 1.0);
        colorPal[x] = (uint16_t)fleK & 0xF800;                      // Top 5 bits
define red

        fleK = 2047.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : (fleX - 0.875) *
8.0);
        colorPal[x] += (uint16_t)fleK & 0x07E0;  // Middle 6 bits define green

        fleK = 31.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : 0.0);
        colorPal[x] += (uint16_t)fleK & 0x001F;  // Bottom 5 bits define blue
      }
      break;
    case 4:  // Compute negative gray palette, black hot
      for(x = 0; x < 256; ++x)
        colorPal[255 - x] = (((uint16_t)x << 8) & 0xF800) + (((uint16_t)x << 3) &
0x07E0) + (((uint16_t)x >> 3) & 0x001F);
      break;
    default:  // Compute gray palette, white hot
      for(x = 0; x < 256; ++x)
        colorPal[x] = (((uint16_t)x << 8) & 0xF800) + (((uint16_t)x << 3) & 0x07E0)
+ (((uint16_t)x >> 3) & 0x001F);
      break;
  }
}

void setColorRange(int presetIndex) { // Set coldest/hottest values in color range
  switch(presetIndex) {
    case 1:  // Standard range, from FLIR document: 50F to 90F
      colorLow = 10.0;
      colorHigh = 32.22;
      break;
    case 2:  // Cool/warm range, for detecting mammals outdoors
      colorLow = 5.0;
      colorHigh = 32.0;
      break;
    case 3:  // Warm/warmer range, for detecting mammals indoors
      colorLow = 20.0;
      colorHigh = 32.0;
      break;
    case 4:  // Hot spots, is anything hotter than it ought to be?
      colorLow = 20.0;
      colorHigh = 50.0;
      break;
    case 5:  // Fire & ice, extreme temperatures only!
```

```
      colorLow = -10.0;
      colorHigh = 200.0;
      break;
    default:  // Default is autorange, so these values will change with every frame
      colorLow = 0.0;
      colorHigh = 100.0;
      break;
  }
}

// Draw the stationary screen elements behind the live camera window
void setBackdrop(uint16_t bgColor, uint16_t buttonFunc) {
  arcada.display->fillScreen(bgColor);

  for(int x = 0; x < 160; ++x)   // Paint current palette across bottom
    arcada.display->drawFastVLine(x, 110, 6, colorPal[map(x, 0, 159, 0, 255)]);

  arcada.display->setCursor(16, 120);
  arcada.display->setTextColor(0xFFFF, bgColor);    // White text, current BG for
button labels
  switch(buttonFunc) {
    case 0:
      arcada.display->print("B:MENU        A:FREEZE");
      break;
    case 1:
      arcada.display->print("B:MENU       ");
      if(nextBMPindex == 0)                         // No room to store a BMP in
flash media?
        arcada.display->setTextColor(GRAY_33 >> 1); // Grayed button label
      arcada.display->print("A:CAPTURE");
      break;
    case 2:
      arcada.display->print("B:MENU    ");
      if(nextDirIndex == 0)                         // Has flash storage no room
for a new directory?
        arcada.display->setTextColor(GRAY_33 >> 1); // Grayed button label
      arcada.display->print("A:START RECORD");
      break;
    case 3:
      arcada.display->print("B:MENU    ");
      arcada.display->setTextColor(0xFFFF, 0xF800);  // White text, red BG
recording indicator
      arcada.display->print("A:STOP RECORD");
      break;
    case 4:
      arcada.display->print("               A:EXIT"); // Use for bitmap redisplay
only
      break;
  }
}

void prepForSave() {
  for(int x = 0; x < 768; ++x)
    pixelArray[3 * x + 2] = pixelArray[3 * x + 1] = pixelArray[3 * x];  // Copy
each blue byte into R & G for 256 grays in 24 bits

  if(!writeBMP()) {                                     // Did BMP write to flash
fail?
    arcada.display->fillRect(0, 96, 160, 12, 0xF800);    // Red error signal
    arcada.display->setTextColor(0xFFFF);                // with white text
    arcada.display->setCursor(20, 99);
    arcada.display->print("Storage error!");
  }
}

boolean newDirectory() { // Create a subdirectory, converting the name between char
arrays and string objects
  char fileArray[64];
  String fullPath;
```

```
  sprintf(fileArray, DIR_FORMAT, nextDirIndex); // Generate subdirectory name
  fullPath = BOTTOM_DIR + String(fileArray);    // Make a filepath out of it, then
  return arcada.mkdir(fullPath.c_str());        // try to make a real subdirectory
from it
}

// Here we write the actual bytes of a BMP file (plus extras) to flash media
boolean writeBMP() {
  uint16_t counter1, shiftedFloats[14]; // A buffer for the appended floats and
uint16_t's
  uint32_t timeStamp;
  float shiftAssist;
  char fileArray[64];
  String fullPath;

// First, figure out a name and path for our new BMP
  fullPath = BOTTOM_DIR;                              // Build a filepath starting
with the base subdirectory
  if(buttonRfunc == 2) {                              // BMP sequence recording in
progress?
    sprintf(fileArray, DIR_FORMAT, nextDirIndex);     // Generate subdirectory name
    fullPath += String(fileArray);                    // Add it to the path
    sprintf(fileArray, BMP_FORMAT, nextBMPsequence);  // Generate a sequential
filename
    fullPath += String(fileArray);                    // Complete the filepath
string
  } else {                                            // Not a sequence, solitary
BMP file
    sprintf(fileArray, BMP_FORMAT, nextBMPindex);     // Generate a serial filename
    fullPath += String(fileArray);                    // Complete the filepath
string
  }

  myFile = arcada.open(fullPath.c_str(), FILE_WRITE); // Only one file can be open
at a time

  if(myFile) {                      // If the file opened okay, write to it:
    myFile.write(BmpPSPHead, 14);   // BMP header 1
    myFile.write(DIBHeadPSP1, 40);  // BMP header 2
    myFile.write(pixelArray, 2304); // Array of 768 BGR byte triples
    myFile.write(PSPpad, 2);        // Pad with 2 zeros 'cause Photoshop does it.

    // My BMP hack - append 5 fixed-point temperature values as 40 extra bytes
    for(counter1 = 0; counter1 < 5; ++counter1) {    // Shift 5 floats
      shiftAssist = sneakFloats[counter1] + 1000.0;  // Offset MLX90640 temps to
positive
      shiftedFloats[counter1 * 2] = (uint16_t)shiftAssist;
      shiftAssist = (shiftAssist - (float)shiftedFloats[counter1 * 2]) *
49152.0; // Scale up fraction
      shiftedFloats[counter1 * 2 + 1] = (uint16_t)shiftAssist;
    }

    shiftedFloats[10] = lowAddr;   // Two more appended numbers, the 2 extreme
pixel addresses
    shiftedFloats[11] = highAddr;

    timeStamp = millis();         // Recycle this variable to append a time stamp
    lowAddr = timeStamp & 0xFFFF;
    highAddr = timeStamp >> 16;
    shiftedFloats[12] = lowAddr;
    shiftedFloats[13] = highAddr;

    myFile.write(shiftedFloats, 28);  // Write appended uint16_t's

    myFile.close();
    return true;
  } else {          // The file didn't open, return error
    return false;
```

```
  }
}

void recallLastBMP(uint16_t bgColor) {  // Display 8-bit values left in buffer from
the last BMP save
  int counter1, counter2;
  boolean exitFlag = false;

  setBackdrop(bgColor, 4);  // Clear screen, just a color palette & "A:EXIT" in the
BG

  for(int counter1 = 0; counter1 < 24; ++counter1) {  // Redraw using leftover red
byte values, not yet overwritten
    for(int counter2 = 0 ; counter2 < 32 ; ++counter2) {
      arcada.display->fillRect(16 + counter2 * 4, 92 - counter1 * 4, 4, 4,
                  colorPal[(uint16_t)pixelArray[3 * (32 * counter1 + counter2) +
2]]);
    }
  }

  while(!exitFlag) {  // Loop here until exit button
    if(!buttonActive && (buttonBits & ARCADA_BUTTONMASK_A)) { // "A:EXIT" button
freshly pressed?
      exitFlag = true;
      buttonActive = true;
      deBounce = millis() + DE_BOUNCE;
    }

    if(buttonActive && millis() > deBounce
       && (buttonBits & (ARCADA_BUTTONMASK_A | ARCADA_BUTTONMASK_B)) == 0)  // Has
de-bounce wait expired & all buttons released?
      buttonActive = false;               // Clear flag to allow another button
press
  }
}

uint16_t availableFileNumber(uint16_t startNumber, String formatBase) { // Find
unclaimed serial number for file series
  uint16_t counter1;
  char nameArray[80];

  for(counter1 = startNumber; counter1 % MAX_SERIAL != 0; ++counter1) { // Start
counting
    sprintf(nameArray, formatBase.c_str(), counter1);                   // Generate
a serialized filename
    if(!arcada.exists(nameArray))                                       // If it
doesn't already exist
      return counter1;                                                  // return
the number as available
  }
  return 0; // Loop finished, no free number found, return fail
}

boolean menuLoop(uint16_t bgColor) {  // Lay out a menu screen, interact to change
values
  int counter1 = 0, scrollPosition = 0;
  boolean exitFlag = false, settingsChanged = false;
  uint32_t menuButtons;

  arcada.display->fillScreen(bgColor);
  arcada.display->fillRect(0, 12 * (counter1 + scrollPosition) + MENU_VPOS - 2, 160,
12, 0x0000);  // Black stripe cursor on menu

  arcada.display->setTextColor(0xFFFF);             // White text
  arcada.display->setCursor(16, 120);               // at screen bottom
  arcada.display->print("B:ADVANCE     A:CHANGE");  // for button labels

  for(counter1 = 0; counter1 < MENU_ROWS; ++counter1) {  // Display menu texts
    menuLines(counter1, scrollPosition);
```

```
  }
  counter1 = 0;

  while(!exitFlag) { // Loop until exit is activated
    if(!buttonActive && (buttonBits & ARCADA_BUTTONMASK_B)) {  // Fresh press of
B:ADVANCE button?
      buttonActive = true;                                     // Set button flag
      deBounce = millis() + DE_BOUNCE;                         // and start
debounce timer.

      arcada.display->fillRect(0, 12 * (counter1 - scrollPosition) + MENU_VPOS - 2,
160, 12, bgColor); // Erase cursor & text
      menuLines(counter1, scrollPosition);                     // Refresh menu text
line
      counter1 = (counter1 + 1) % MENU_LEN;                    // Advance menu
counter

      if(counter1 == 0) {                                      // Have we cycled
around to the menu top?
        scrollPosition = 0;
        for(int counter2 = 0; counter2 < MENU_ROWS; ++counter2) {  // Redisplay all
menu texts
          arcada.display->fillRect(0, 12 * counter2 + MENU_VPOS - 2, 160, 12,
bgColor); // Erase old text
          menuLines(counter2 + scrollPosition, scrollPosition);    // Redraw each
text line
        }
      } else if((counter1 + 1 < MENU_LEN) && (counter1 - scrollPosition == MENU_ROWS
- 1)) { // Should we scroll down 1 menu line?
        ++scrollPosition;
        for(int counter2 = 0; counter2 < MENU_ROWS; ++counter2) {  // Redisplay all
menu texts
          arcada.display->fillRect(0, 12 * counter2 + MENU_VPOS - 2, 160, 12,
bgColor); // Erase old text
          menuLines(counter2 + scrollPosition, scrollPosition);    // Redraw each
text line
        }
      }

      arcada.display->fillRect(0, 12 * (counter1 - scrollPosition) + MENU_VPOS - 2,
160, 12, 0x0000);  // New black cursor
      menuLines(counter1, scrollPosition);                     // Refresh text line
      deBounce = millis() + DE_BOUNCE;                         // Restart debounce
timer, just for safety
    }

    if(!buttonActive && (buttonBits & ARCADA_BUTTONMASK_A)) {  // Fresh press of
A:CHANGE button?
      buttonActive = true;                                     // Set button flag
      deBounce = millis() + DE_BOUNCE;                         // and start
debounce timer.

      switch(counter1) {       // Change whichever setting is currently hilighted
        case 0:
          showLastCap = true;  // Set flag to display the last frame captured to SD
          exitFlag = true;     // and exit
          break;
        case 1:
          celsiusFlag = !celsiusFlag; // Toggle Celsius/Fahrenheit
          break;
        case 2:
          buttonRfunc = (buttonRfunc + 1) % 3; // Step through button functions
          break;
        case 3:
          loadPalette(paletteNum = (paletteNum + 1) % 5); // Step through various
color palettes
          break;
        case 4:
          thermRange = (thermRange + 1) % 6; // Step through various temp range
```

```
presets
          break;
        case 5:
          markersOn = !markersOn; // Toggle hot/cold marker visibility
          break;
        case 6:
          mirrorFlag = !mirrorFlag; // Toggle mirrored display
          break;
        case 7:
          switch(frameRate = (frameRate + 1) % 6) {             // 6 frame rates,
0.5 to 16 in powers of 2
            case 0: mlx.setRefreshRate(MLX90640_0_5_HZ); break;
            case 1: mlx.setRefreshRate(MLX90640_1_HZ); break;
            case 2: mlx.setRefreshRate(MLX90640_2_HZ); break;
            case 3: mlx.setRefreshRate(MLX90640_4_HZ); break;
            case 4: mlx.setRefreshRate(MLX90640_8_HZ); break;
            default: mlx.setRefreshRate(MLX90640_16_HZ); break;
          }
          break;
        case 8:
          emissivity = (emissivity + 90) % 100; // Step from 95% to 5% by -10%
          break;
        case 9:
          smoothing = !smoothing; // Toggle pixel smoothing
          break;
        case 10:
          arcada.setBacklight((screenDim = !screenDim) ? 64 : 255); // Change
backlight LED
          break;
        default:
          exitFlag = true;
          break;
      }
      if((counter1 > 0) && (counter1 < MENU_LEN - 1))    // Was any setting just
changed?
        settingsChanged = true;

      arcada.display->fillRect(0, 12 * (counter1 - scrollPosition) + MENU_VPOS - 2,
160, 12, 0x0000); // Erase hilit menu line
      menuLines(counter1, scrollPosition);   // Retype hilit menu line
    }

    if(buttonActive && millis() > deBounce
       && (buttonBits & (ARCADA_BUTTONMASK_A | ARCADA_BUTTONMASK_B)) == 0)  // Has
de-bounce wait expired & all buttons released?
      buttonActive = false;               // Clear flag to allow another button
press
  }
  return(settingsChanged);
}

void menuLines(int lineNumber, int scrollPos) {  // Screen print a single line in
the settings menu

  arcada.display->setTextColor(0xFFFF);               // White text
  arcada.display->setCursor(10, 12 * (lineNumber - scrollPos) + MENU_VPOS); // Menu
lines 12 pixels apart

  if(lineNumber - scrollPos == 0 && scrollPos > 0) {  // Are any menu lines
scrolled off screen top?
    arcada.display->print("           ^");            // Print a small up arrow
indicator
  } else if(lineNumber - scrollPos == 8 && lineNumber + 1 < MENU_LEN) { // How
about off the bottom?
    arcada.display->print("           v");            // Print a small down arrow
indicator... yeah, it's a v
  } else {

    switch(lineNumber) {
```

```
      case 0:
        arcada.display->print("  Display last capture");
        break;
      case 1:
        arcada.display->print("     Scale - ");
        arcada.display->print(celsiusFlag ? "CELSIUS" : "FAHRENHEIT");
        break;
      case 2:
        arcada.display->print(" Rt button - ");
        switch(buttonRfunc) {
          case 1:
            arcada.display->print("CAPTURE"); break;
          case 2:
            arcada.display->print("RECORD"); break;
          default:
            arcada.display->print("FREEZE"); break;
        }
        break;
      case 3:
        arcada.display->print("   Palette - ");
        for(int xPos = 0; xPos < 72; ++xPos)   // Display the current heat spectrum
colors
          arcada.display->drawFastVLine(xPos + 87, (lineNumber - scrollPos) * 12 +
MENU_VPOS,
                                        8, colorPal[map(xPos, 0, 71, 0, 255)]);
        switch(paletteNum) {
          case 1:
            arcada.display->print("IRONBOW");
            break;
          case 2:
            arcada.display->print("FIREBOW");
            break;
          case 3:
            arcada.display->setTextColor(0x0000);    // Black text for reverse
contrast
            arcada.display->print("ALARM");
            break;
          case 4:
            arcada.display->setTextColor(0x0000);    // Black text
            arcada.display->print("BLACK HOT");
            break;
          default:
            arcada.display->print("WHITE HOT");
            break;
        }
        break;
      case 4:
        arcada.display->print("Temp range - ");
        setColorRange(thermRange);
        switch(thermRange) {
          case 1:
            arcada.display->print("STANDARD"); break;
          case 2:
            arcada.display->print("COOL/WARM"); break;
          case 3:
            arcada.display->print("WARM/WARMER"); break;
          case 4:
            arcada.display->print("HOT SPOTS"); break;
          case 5:
            arcada.display->print("FIRE & ICE"); break;
          default:
            arcada.display->print("AUTO-RANGE"); break;
        }
        break;
      case 5:
        arcada.display->print("   Markers - ");
        arcada.display->print(markersOn ? "ON" : "OFF");
        break;
      case 6:
```

```
        arcada.display->print("     Image - ");
        arcada.display->print(mirrorFlag ? "MIRRORED" : "FORWARD");
        break;
      case 7:
        arcada.display->print("Frame rate - ");
        arcada.display->print((float)(1 << frameRate) * 0.5);
        arcada.display->print(" FPS");
        break;
      case 8:
        arcada.display->setTextColor(GRAY_33 << 1); // Grayed menu item
        arcada.display->print("Emissivity - ");
        arcada.display->print(emissivity);
        arcada.display->print("%");
        break;
      case 9:
        arcada.display->setTextColor(GRAY_33 << 1); // Grayed menu item
        arcada.display->print(" Smoothing - ");
        arcada.display->print(smoothing ? "ON" : "OFF");
        break;
      case 10:
        arcada.display->print(" Backlight - ");
        arcada.display->print(screenDim ? "DIM" : "FULL");
        break;
      case 11:
        arcada.display->print("       Exit menu");
    }
  }
}

// This is the function that substitutes for GPIO external interrupts
// It will check for A and B button presses at 50Hz
void buttonCatcher(void) {
  buttonBits = arcada.readButtons();
  clickFlagMenu |= (buttonBits & ARCADA_BUTTONMASK_B) != 0;
  clickFlagSelect |= (buttonBits & ARCADA_BUTTONMASK_A) != 0;
}
```

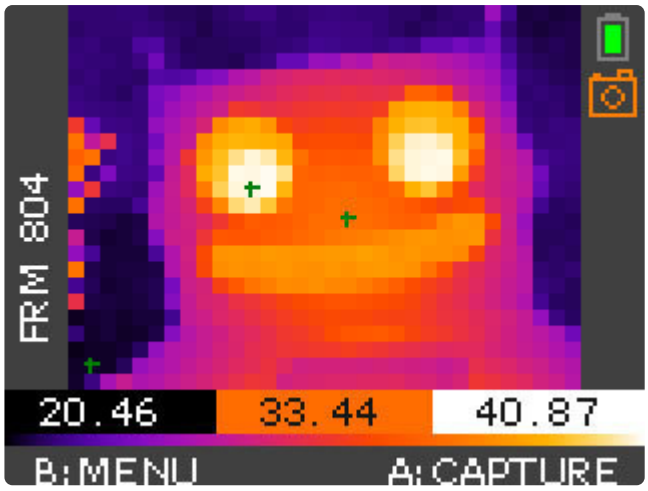
Did you make it to the finish?  High five!

Next, we'll look at the sketch in operation, and peek under the hood to see what the code does.

---

# Pixels on a Screen!

With the sketch successfully uploaded and running, what appears on the screen?  If the sensor isn't connected properly, a red message will appear saying the MLX90640 is not detected, and everything halts.  But if the hardware is working, look for a screen full of features like this.




The picture in the middle is the nifty part, radiant heat converted into a color image. At 32x24 pixels, only coarse details can be discerned, but they can reveal a lot. The other features on the screen provide more information.

In the left border is a frame counter, which counts the number of image frames displayed since the last restart of the sketch. It was a performance diagnostic that's still part of the program.

In the right border are small status icons that indicate what the sketch is doing from moment to moment.

- A battery level indicator shows when battery power is running low. It doesn't indicate charging status, however.
- An orange camera icon appears when temperature data is transferring from the sensor to the processor.
- A green SD card icon appears as image data is written to flash storage.
- A snowflake icon indicates when an image freeze is active.

Three numbers below the image display three temperatures copied straight from the sensor data. The coldest temperature detected in the image is shown on the left, the hottest on the right, and the temperature of the center pixel is in the middle. The locations of these temperatures can be indicated in the image by three small green crosses.

A small strip showing the color palette from cold to hot lies below the numbers, and below this are the function labels for buttons B and A.

## Capturing Images

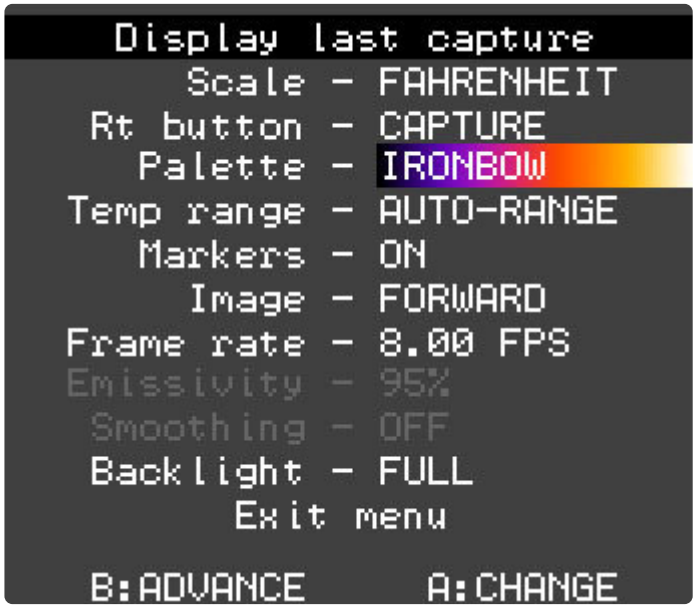
Buttons B and A are to the right of the screen, and can perform more than one function.  It's button A that can capture a thermal image and save it into flash storage as a tiny gray BMP file with embedded temperature data, like this one.

Depending on its user setting, the button can capture single images, or start and stop recording BMP image sequences, or simply freeze the display without capturing anything.

The sketch tries to test for flash availability, and will gray out the button label if it detects flash storage as unavailable, but it's not a sophisticated system.  Do pay attention to storage space.  Each BMP consumes over 2K, and the sketch can't recall or delete them.  That can happen only when your controller is USB connected to a full-size computer.

## The Settings Menu

From the camera view screen, button B brings up the settings menu.  Give it a press and see what appears.

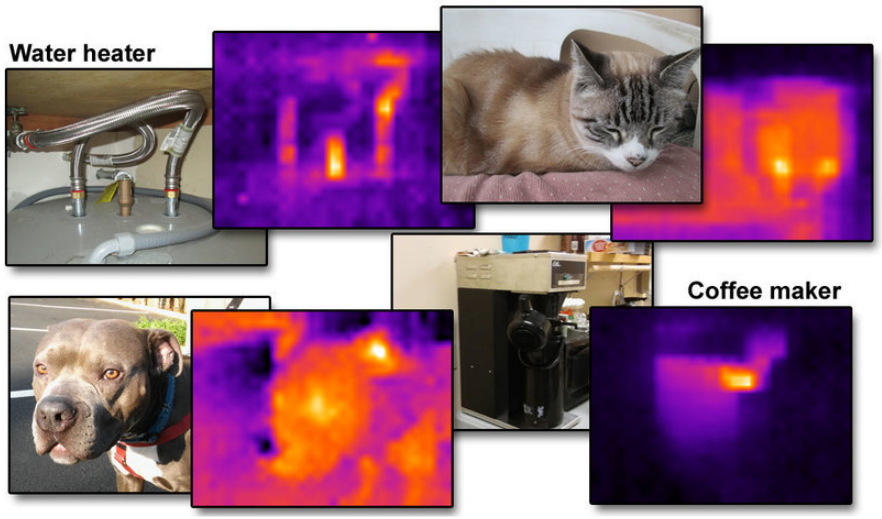



The menu screen looks sort of like this image, but not quite.  That's because the full menu doesn't fit on the small screen.  No problem, since the menu automatically scrolls as you click button B to advance the highlight bar down the list.  Any setting

you wish to change can be done by advancing the bar to the item and clicking button A. These are the settings you can change.

- Display last capture - If any thermal image data has been captured to flash storage, the last image stored can be momentarily re-shown on screen. If no image has been captured since the latest restart, a generic gradient gets displayed.
- Scale - On screen temperatures can be shown in Fahrenheit or Celsius.
- Rt button - The right button triggers the capture of images from the camera screen to flash storage. You can set it to capture single images, to record image sequences, or to freeze the live image without storing it.
- Palette - Controls which of several false-color palettes will be used to paint the live thermal image. The palettes are described further in the Putting it in Motion (https://adafru.it/K0C) section.
- Temp range - The colors in your image, what temperatures do they really represent? You can choose. **Auto-range** changes with each frame, adjusting the colors to span the coldest to hottest pixels detected. **Standard** fixes the color range from 50F to 90F, a range the high-priced thermal cameras use. **Cool/warm** is good for imaging your warm-blooded friends in the cool outdoors. **Warm/warmer** is similar, but is better for the warmer indoors. **Hot spots** can help sense if something's getting dangerously warm. **Fire & ice** barely senses everyday temperatures, extremes only.
- Markers - The small crosses in the image can be made invisible.
- Image - **Forward** is normal. **Mirrored** flips the image if you want a backwards-facing selfie camera.
- Frame rate - Control the exposure time per frame. Actual frame rates vary due to computing overhead.
- Emissivity and Smoothing - These settings are under development. Check back often.
- Backlight - The LCD screen's backlight can be dimmed for nighttime viewing, or simply to extend battery life.
- Exit menu - We now return you to your regularly scheduled thermal camera.

That's it. Go ahead and experiment with different settings. Get a feel for what your thermal imaging camera can reveal. Capture some images and record some sequences. The data you obtain will give you something to analyze in upcoming chapters, but mainly it's for the fun of discovery.




The next few pages walk through the camera sketch in more detail.

---

# Code Walkthrough: Setting Up

The sketch is on the large side. I tried to sprinkle it richly with comments, but here I'll give a few more details as we break it down into chunks. (Is that the technical term?)

At the beginning are the libraries, constants, global variables and arrays that will be used throughout the sketch. They dictate how BMP files are named, the bytes that go into their file headers, and bitmaps for the onscreen icons. Variables and Booleans are used to keep track of what the sketch needs to do at each step. Some of them can be changed to different startup values if you wish, like `frameRate`.

```
#include &lt;Adafruit_MLX90640.h&gt;
#include "Adafruit_Arcada.h"
Adafruit_MLX90640 mlx;
Adafruit_Arcada arcada;

#if !defined(USE_TINYUSB)
  #warning "Compile with TinyUSB selected!"
#endif

File myFile;

float mlx90640To[768];   // Here we receive the float vals acquired from MLX90640

#define DE_BOUNCE 200
  // Wait this many msec between button clicks
#define MENU_LEN 12
  // Number of total available menu choices
#define MENU_ROWS 9
  // Number of menu lines that can fit on screen
#define MENU_VPOS 6
#define GRAY_33 0x528A
#define BOTTOM_DIR "MLX90640"
#define DIR_FORMAT "/dir%05d"
#define BMP_FORMAT "/frm%05d.bmp"
#define CFG_FLNAME "/config.ini"
#define MAX_SERIAL 999
```

```
// BMP File Header, little end first, Photoshop ver.
const PROGMEM uint8_t BmpPSPHead[14] = {
 0x42, 0x4D,             // "BM" in hex
 0x38, 0x09, 0x00, 0x00, // File size, 2360
 0x00, 0x00,             // reserved for app data 1
 0x00, 0x00,             // reserved for app data 2
 0x36, 0x00, 0x00, 0x00  // Offset of first pixel, 54
};

// BMP 24-bit DIB Header, little end first, Photoshop ver.
const PROGMEM uint8_t DIBHeadPSP1[40] = {
 0x28, 0x00, 0x00, 0x00,  // Header size, 40
 0x20, 0x00, 0x00, 0x00,  // pixel width, 32
 0x18, 0x00, 0x00, 0x00,  // pixel height, 24
 0x01, 0x00,              // color planes, 1
 0x18, 0x00,              // bits per pixel, 24
 0x00, 0x00, 0x00, 0x00,  // Compression method, 0==none
 0x00, 0x00, 0x00, 0x00,  // Raw bitmap data size, dummy 0
 0x12, 0x0B, 0x00, 0x00,  // Pixels per meter H, 2834
 0x12, 0x0B, 0x00, 0x00,  // Pixels per meter V, 2834
 0x00, 0x00, 0x00, 0x00,  // Colors in palette, 0==default 2^n
 0x00, 0x00, 0x00, 0x00   // Number of important colors, 0
};

// BMP file data, 2 byte padding
const PROGMEM uint8_t PSPpad[2] = {0x00, 0x00};

//Byte arrays of bitmapped icons, 16 x 12 px:
const PROGMEM uint8_t battIcon[] = {
0x0f, 0x00, 0x3f, 0xc0, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40,
0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x20, 0x40, 0x3f, 0xc0};

const PROGMEM uint8_t camIcon[] = {
0x01, 0xe0, 0x61, 0x20, 0xff, 0xf0, 0x80, 0x10, 0x86, 0x10, 0x89, 0x10,
0x90, 0x90, 0x90, 0x90, 0x89, 0x10, 0x86, 0x10, 0x80, 0x10, 0xff, 0xf0};

const PROGMEM uint8_t SDicon[] = {
0x0f, 0xe0, 0x1f, 0xe0, 0x3c, 0x60, 0x78, 0x60, 0x70, 0x60, 0x60, 0x60,
0x60, 0x60, 0x60, 0x60, 0x6f, 0x60, 0x60, 0x60, 0x7f, 0xe0, 0x7f, 0xe0};

const PROGMEM uint8_t snowIcon[] = {
0x15, 0x00, 0x4E, 0x40, 0xC4, 0x60, 0x75, 0xC0, 0x9F, 0x20, 0x0E, 0x00,
0x0E, 0x00, 0x9F, 0x20, 0x75, 0xC0, 0xC4, 0x60, 0x4E, 0x40, 0x15, 0x00};

uint8_t pixelArray[2304];   // BMP image body, 32 pixels * 24 rows * 3 bytes

// Some global values that several functions will use, including
// 5 floats to append to the BMP pixel data:
// coldest pixel, coldest color, center temp, hottest color, hottest pixel
float sneakFloats[5] = {3.1415926, 0.0, -11.7, 98.6, -12.34};      // Test values
that get overwritten
uint16_t highAddr = 0, lowAddr = 0;                                // Append the
pixel addresses, too

uint16_t backColor, lowPixel, highPixel, buttonRfunc = 1,
         emissivity = 95, frameRate = 4,
         thermRange = 0, paletteNum = 1, colorPal[256],            // Array for
color palettes
         nextDirIndex = 0, nextBMPindex = 0, nextBMPsequence = 1;  // These keep
count of SD files and dirs, 0==error
uint32_t deBounce = 0, buttonBits = 0;
boolean mirrorFlag = false, celsiusFlag = false, markersOn = true,
        screenDim = false, smoothing = false, showLastCap = false,
        save1frame = false, recordingInProg = false, buttonActive = false;
float battAverage = 0.0, colorLow = 0.0, colorHigh = 100.0;        // Values for
managing color range
volatile boolean clickFlagMenu = false, clickFlagSelect = false;   // Volatiles for
timer callback handling
```

The `setup()` function executes first and just once per startup. Its job is to initialize the system, starting with the Arcada code and the onboard hardware it will control, like the screen and buttons, etc.

Then a set of tests determines whether flash storage is ready to receive fresh BMP files.

```
void setup()
{
  if (!arcada.arcadaBegin()) {    // Start TFT and fill with black
    // Serial.print("Failed to begin");
    while (1);
  }
  arcada.filesysBeginMSD();       // Set up SD or QSPI flash as an external USB
drive

  arcada.displayBegin();              // Activate TFT screen
  arcada.display-&gt;setRotation(1);     // wide orientation
  arcada.display-&gt;setTextWrap(false);
  arcada.setBacklight(255);           // Turn on backlight
  battAverage = arcada.readBatterySensor();

  Serial.begin(115200);
//  while(!Serial); // Wait for user to open terminal
  Serial.println("MLX90640 IR Array Example");

  if(arcada.filesysBegin()){              // Initialize flash storage, begin
setting up indices for saving BMPs
    if(!arcada.exists(BOTTOM_DIR)) {      // Is base "MLX90640" directory absent?
      if(arcada.mkdir(BOTTOM_DIR))        // Can it be added?
        nextDirIndex = nextBMPindex = 1;  // Success, prepare to store numbered
files &amp;amp; dirs
    } else {      // "MLX90640" directory exists, can we add files | directories?
      // Get the number of the next unused serial directory path
      nextDirIndex = availableFileNumber(1, BOTTOM_DIR + String(DIR_FORMAT));
      // and the next unused serial BMP name
      nextBMPindex = availableFileNumber(1, BOTTOM_DIR + String(BMP_FORMAT));
    }
  }  // By now each global index variable is either 0 (no nums available), or the
next unclaimed serial num
```

Next, the MLX90640 itself must be found, initialized, and configured for use.

If this is successful, the screen is cleared and a basic backdrop is displayed. A function called `timerCallback()` is invoked. It will cause another function to be executed at a set rate. In our case, the function is named `buttonCatcher()`. It's located at the end of the sketch, and it will execute 50 times a second. Its job will be to rapidly test and remember whether buttons B or A are being pressed. (It's a way of forcing the sketch to regularly check for button clicks, even when it's tied up doing a slow process.)

```
if(!mlx.begin(MLX90640_I2CADDR_DEFAULT, &amp;amp;Wire)) {
    Serial.println("MLX90640 not found!");
    arcada.haltBox("MLX90640 not found!");
    while(1)
      delay(10);  // Halt here
  }
```

```
  Serial.println("Found Adafruit MLX90640");

  Serial.print("Serial number: ");
  Serial.print(mlx.serialNumber[0], HEX);
  Serial.print(mlx.serialNumber[1], HEX);
  Serial.println(mlx.serialNumber[2], HEX);

  //mlx.setMode(MLX90640_INTERLEAVED);
  mlx.setMode(MLX90640_CHESS);
  mlx.setResolution(MLX90640_ADC_18BIT);

  switch(frameRate) {
    case 0: mlx.setRefreshRate(MLX90640_0_5_HZ); break; // 6 frame rates, 0.5 to 16
FPS in powers of 2
    case 1: mlx.setRefreshRate(MLX90640_1_HZ); break;
    case 2: mlx.setRefreshRate(MLX90640_2_HZ); break;
    case 3: mlx.setRefreshRate(MLX90640_4_HZ); break;
    case 4: mlx.setRefreshRate(MLX90640_8_HZ); break;
    default: mlx.setRefreshRate(MLX90640_16_HZ); break;
  }
  Wire.setClock(1000000); // max 1 MHz

  for(int counter01 = 0; counter01 &lt; 2304; ++counter01)
    pixelArray[counter01] = counter01 / 9;  // Initialize BMP pixel buffer with a
gradient

  loadPalette(paletteNum);             // Load false color palette
  backColor = GRAY_33;                 // 33% gray for BG
  setBackdrop(backColor, buttonRfunc); // Current BG, current button labels

  arcada.timerCallback(50, buttonCatcher);  // Assign a 50Hz callback function to
catch button presses
}
```

The sketch by now should be ready for the next part, sensing temperatures, displaying pixels, and clicking buttons.  Cool stuff ahead!

---

# The Main Loop

This is where the display action happens.

For every cycle, after getting some variables ready, what does the loop do first?  It checks the battery level!  The Arcada library allows the processor to calculate the battery's voltage, and the sketch converts that into an unscientific onscreen indicator.  It's still pretty good at showing when power's getting low.

After that is the main event.  An array of 768 floating point numbers gets filled with temperatures from the MLX90640 sensor.  Everything else that happens depends on this.

```
void loop()
{
  static uint32_t frameCounter = 0;
  float scaledPix, highPix, lowPix;
  uint16_t markColor;

// Show the battery level indicator, 3.7V to 3.3V represented by a 7 segment bar
```

```
  battAverage = battAverage * 0.95 + arcada.readBatterySensor() * 0.05; //
*Gradually* track battery level
  highPix = (int)constrain((battAverage - 3.3) * 15.0, 0.0, 6.0) + 1;   // Scale it
to a 7-segment bar
  markColor = highPix &gt; 2 ? 0x07E0 : 0xFFE0;                            // Is
the battery level bar green or yellow?
  markColor = highPix &gt; 1 ? markColor : 0xF800;                         // ...or
even red?
  arcada.display-&gt;fillRect(146, 2, 12, 12, backColor);                  // Erase
old battery icon
  arcada.display-&gt;drawBitmap(146, 2, battIcon, 16, 12, 0xC618);         //
Redraw gray battery icon
  arcada.display-&gt;fillRect(150, 12 - highPix, 4, highPix, markColor);   // Add
the level bar

// Fetch 768 fresh temperature values from the MLX90640
  arcada.display-&gt;drawBitmap(146, 18, camIcon, 16, 12, 0xF400); // Show orange
camera icon during I2C acquisition
  if(mlx.getFrame(mlx90640To) != 0) {
    Serial.println("Failed");
    return;
  }
  arcada.display-&gt;fillRect(146, 18, 12, 12, backColor);         // Acquisition
done, erase camera icon
```

Now all the numbers in the array get inspected.  Before they are turned into pixels, the coldest and the hottest values must be located.  The center temperature gets saved, too.  These are some of the numbers that get appended to the pixels in a BMP file, whenever a capture is performed.

```
// First pass: Find hottest and coldest pixels
  highAddr = lowAddr = 0;
  highPix  = lowPix  = mlx90640To[highAddr];

  for (int x = 1 ; x &lt; 768 ; x++) { // Compare every pixel
    if(mlx90640To[x] &gt; highPix) {   // Hotter pixel found?
      highPix = mlx90640To[x];      // Record its values
      highAddr = x;
    }
    if(mlx90640To[x] &lt; lowPix) {    // Colder pixel found?
      lowPix = mlx90640To[x];       // Likewise
      lowAddr = x;
    }
  }
  if(thermRange == 0) {    // Are the colors set to auto-range?
    colorLow = lowPix;     // Then high and low color values get updated
    colorHigh = highPix;
  }
  sneakFloats[0] = lowPix;     // Retain these five temperature values
  sneakFloats[1] = colorLow;   // to append to the BMP file, if any
  sneakFloats[2] = mlx90640To[400];
  sneakFloats[3] = colorHigh;
  sneakFloats[4] = highPix;
```

Then the array values are inspected again.  This time each one is converted to a color, a pixel-shaped rectangle is painted on the TFT screen, and a thermal picture finally appears.  At the same time, the scaled 8-bit values get stored in another array, one that can contain pixels in a form suitable for writing to a BMP file, if needed.

```
// Second pass: Scale the float values down to 8-bit and plot colormapped pixels
  if(mirrorFlag) {                // Mirrored display (selfie mode)?
    for(int y = 0; y &lt; 24; ++y) {  // Rows count from bottom up
      for(int x = 0 ; x &lt; 32 ; x++) {
        scaledPix = constrain((mlx90640To[32 * y + x] - colorLow) / (colorHigh -
colorLow) * 255.9, 0.0, 255.0);
        pixelArray[3 * (32 * y + x)] =
(uint8_t)scaledPix;                           // Store as a byte in BMP buffer
        arcada.display-&gt;fillRect(140 - x * 4, 92 - y * 4, 4, 4,
colorPal[(uint16_t)scaledPix]);  // Filled rectangles, bottom up
      }
    }
  } else {  // Not mirrored
    for(int y = 0; y &lt; 24; ++y) {
      for(int x = 0 ; x &lt; 32 ; x++) {
        scaledPix = constrain((mlx90640To[32 * y + x] - colorLow) / (colorHigh -
colorLow) * 255.9, 0.0, 255.0);
        pixelArray[3 * (32 * y + x)] = (uint8_t)scaledPix;
        arcada.display-&gt;fillRect(16 + x * 4, 92 - y * 4, 4, 4,
colorPal[(uint16_t)scaledPix]);
      }
    }
  }
```

After this, all the other onscreen stuff that shows up around the image gets added. That means printing the three important temperatures, the frame counter, and the three small crosses on the important pixels. All of these are shown in carefully selected contrasting colors. Pretty fancy, eh?

```
// Post pass: Screen print the lowest, center, and highest temperatures
  arcada.display-&gt;fillRect(  0, 96, 53, 12, colorPal[0]);                  //
Contrasting mini BGs for cold temp
  arcada.display-&gt;fillRect(107, 96, 53, 12, colorPal[255]);                //
and for hot temperature texts
  scaledPix = constrain((mlx90640To[400] - colorLow) / (colorHigh - colorLow) *
255.9, 0.0, 255.0);
  arcada.display-&gt;fillRect(53, 96, 54, 12, colorPal[(uint16_t)scaledPix]); //
Color coded mini BG for center temp

  arcada.display-&gt;setTextSize(1);
  arcada.display-&gt;setCursor(10, 99);
  arcada.display-&gt;setTextColor(0xFFFF ^ colorPal[0]);   // Contrasting text
color for coldest value
  arcada.display-&gt;print(celsiusFlag ? lowPix : lowPix * 1.8 + 32.0);  // Print
Celsius or Fahrenheit

  arcada.display-&gt;setCursor(120, 99);
  arcada.display-&gt;setTextColor(0xFFFF ^ colorPal[255]); // Contrast text for
hottest value
  arcada.display-&gt;print(celsiusFlag ? highPix : highPix * 1.8 + 32.0);  // Print
Celsius or Fahrenheit

  arcada.display-&gt;setCursor(65, 99);
  if((mlx90640To[400] &lt; (colorLow + colorHigh) * 0.5) == (paletteNum &lt; 3))
    arcada.display-&gt;setTextColor(0xFFFF);              // A contrasting text
color for center temp
  else
    arcada.display-&gt;setTextColor(0x0000);
  arcada.display-&gt;print(celsiusFlag ? mlx90640To[400] : mlx90640To[400] * 1.8 +
32.0);  // Pixel 12 * 32 + 16

  markColor = 0x0600;    // Deep green color to draw onscreen cross markers
  if(markersOn) {        // Show markers?
    if(mirrorFlag) {     // ...over a mirrored display?
```

```
      arcada.display-&gt;drawFastHLine(156 - (( lowAddr % 32) * 4 + 16), 93 - 4 * (
lowAddr / 32), 4, markColor); // Color crosses mark cold pixel,
      arcada.display-&gt;drawFastVLine(159 - (( lowAddr % 32) * 4 + 17), 92 - 4 * (
lowAddr / 32), 4, markColor);
      arcada.display-&gt;drawFastHLine(156 - ((highAddr % 32) * 4 + 16), 93 - 4 *
(highAddr / 32), 4, markColor); // hot pixel,
      arcada.display-&gt;drawFastVLine(159 - ((highAddr % 32) * 4 + 17), 92 - 4 *
(highAddr / 32), 4, markColor);
      arcada.display-&gt;drawFastHLine(76, 45, 4,
markColor);                                                  // and center pixel
      arcada.display-&gt;drawFastVLine(78, 44, 4, markColor);
    } else {             // Not mirrored
      arcada.display-&gt;drawFastHLine(( lowAddr % 32) * 4 + 16, 93 - 4 * (
lowAddr / 32), 4, markColor); // Color crosses mark cold pixel,
      arcada.display-&gt;drawFastVLine(( lowAddr % 32) * 4 + 17, 92 - 4 * (
lowAddr / 32), 4, markColor);
      arcada.display-&gt;drawFastHLine((highAddr % 32) * 4 + 16, 93 - 4 *
(highAddr / 32), 4, markColor); // hot pixel,
      arcada.display-&gt;drawFastVLine((highAddr % 32) * 4 + 17, 92 - 4 *
(highAddr / 32), 4, markColor);
      arcada.display-&gt;drawFastHLine(80, 45, 4,
markColor);                                                   // and center pixel
      arcada.display-&gt;drawFastVLine(81, 44, 4, markColor);
    }
  }

// Print the frame count on the left sidebar
  arcada.display-&gt;setRotation(0);    // Vertical printing
  arcada.display-&gt;setCursor(48, 4);
  arcada.display-&gt;setTextColor(0xFFFF, backColor); // White text, current BG
  arcada.display-&gt;print("FRM ");
  arcada.display-&gt;print(++frameCounter);
  arcada.display-&gt;setRotation(1);    // Back to horizontal
```

## The Interactive Stuff

What happens when a button gets pressed? The response starts waayyy at the end of the sketch, where the `buttonCatcher()` function is being called 50 times a second. If it detects either button A or B clicking, a corresponding Boolean is set. When the loop reaches this point, the sketch deals with those Boolean flags. It will store timer values for suppressing button bounces, set more Booleans to block double-presses, and even operate the camera's features.

First, the B button's Boolean is tested, and the onscreen settings menu is called up if clicked.

Then the A button is checked. It will either send the sketch into a small loop that freezes everything until all buttons are released, or it will set even more Booleans, ones that will allow other functions to write BMP images to flash.

If neither button gets clicked, none of these events happen and the loop continues showing thermal pixels.

```
// Handle any button presses
  if(!buttonActive &amp;&amp; clickFlagMenu) {         // Was B:MENU button pressed?
```

```
    buttonActive = true;                      // Set button flag
    deBounce = millis() + DE_BOUNCE;          // and start debounce timer
    menuLoop(backColor);                      // Execute menu routine until
finished
    clickFlagSelect = recordingInProg = false; // Clear unneeded flags
    nextBMPsequence = 1;
    setBackdrop(backColor, buttonRfunc);      // Repaint current BG &amp; button
labels
  }

  if(!buttonActive &amp;&amp; clickFlagSelect) { // Was the A button pressed?
    buttonActive = true;                 // Set button flag
    deBounce = millis() + DE_BOUNCE;     // and start debounce timer

    if(buttonRfunc == 0) {                                          // Freeze
requested?
      arcada.display-&gt;drawBitmap(146, 48, snowIcon, 16, 12, 0xC61F); // Freeze
icon on
      while(buttonBits &amp; ARCADA_BUTTONMASK_A)                        // Naive
freeze: loop until button released
        delay(10);                                                  // Short pause
      deBounce = millis() + DE_BOUNCE;                              // Restart
debounce timer
      arcada.display-&gt;fillRect(146, 48, 12, 12, backColor);          // Freeze
icon off
    } else if(buttonRfunc == 1) {                         // Capture requested?
      if((nextBMPindex = availableFileNumber(nextBMPindex, BOTTOM_DIR +
String(BMP_FORMAT))) != 0) { // Serialized BMP filename available?
        save1frame = true;                                // Set the flag to save a
BMP
        arcada.display-&gt;fillRect(0, 96, 160, 12, 0x0600); // Display a green
strip
        arcada.display-&gt;setTextColor(0xFFFF);             // with white capture
message text
        arcada.display-&gt;setCursor(16, 99);
        arcada.display-&gt;print("Saving frame ");
        arcada.display-&gt;print(nextBMPindex);
      }
    } else {                            // Begin or halt recording a sequence of
BMP files
      if(!recordingInProg) {            // "A:START RECORDING" was pressed
        if((nextDirIndex = availableFileNumber(nextDirIndex, BOTTOM_DIR +
String(DIR_FORMAT))) != 0) { // Serialized directory name available?
          // Make the directory
          if(newDirectory()) {          // Success in making a new sequence
directory?
            recordingInProg = true;     // Set the flag for saving BMP files
            nextBMPsequence = 1;        // ...numbered starting with 00001
            setBackdrop(backColor, 3);  // Show "A:STOP RECORDING" label
          } else                        // Couldn't make the new directory, so
            nextDirIndex = 0;           // disable further sequences
        }
      } else {                          // "A:STOP RECORDING" was pressed
        recordingInProg = false;
        setBackdrop(backColor, 2);      // Clear "A:STOP RECORDING" label
      }
    }
  }
```

Okay.  Now that the buttons have been checked, one of them might have been clicked for an action to happen, so here's where the responses start.

The first chunk happens when a capture is triggered, either singly or in a series. Whenever `prepForSave()` is called, a BMP image complete with added temperature values is written to flash storage, if available.

If Display last capture was selected from the menu, then the `recallLastBMP()` function gets called, and whatever 8-bit pixel data is still lingering in the output array gets shown in color on the screen once again.

Finally, all the timer values and Booleans used for handling buttons are checked and updated, and if all conditions are met, the values are cleared to permit new button clicks. (We'll see this code block again in some of the support functions.)

```
// Saving any BMP images to flash media happens here
  if(save1frame || recordingInProg) {      // Write a BMP file to SD?
    arcada.display-&gt;drawBitmap(146, 32, SDicon, 16, 12, 0x07E0); // Flash
storage activity icon on

    prepForSave();                         // Save to flash.  Use global values for
parameters
    nextBMPsequence += recordingInProg ? 1 : 0;  // If recording a series,
increment frame count
    save1frame = false;                    // If one frame saved, clear the flag
afterwards

    arcada.display-&gt;fillRect(146, 32, 12, 12, backColor);        // Flash
storage activity icon off
  }

  if(showLastCap) {                       // Redisplay the last BMP saved?
    buttonActive = true;                  // Set button flag
    deBounce = millis() + DE_BOUNCE;      // and start debounce timer
    recallLastBMP(backColor);             // Redisplay last bitmap from buffer until
finished
    setBackdrop(backColor, buttonRfunc); // Repaint current BG &amp; button labels
    showLastCap = false;
  }

// Here we protect against button bounces while the function loops
  if(buttonActive &amp;&amp; millis() &gt; deBounce &amp;&amp; (buttonBits
     &amp; (ARCADA_BUTTONMASK_B | ARCADA_BUTTONMASK_A)) == 0)  // Has de-bounce
wait expired &amp; all buttons released?
    buttonActive = false;                 // Clear flag to allow another button press

  clickFlagMenu = clickFlagSelect = false; // End of the loop, clear all interrupt
flags
}
```

The main loop repeats from here, but that's not even half the sketch. The heavy lifting happens in the supporting functions that follow.

---

# Supporting Functions

Next come the additional functions for actions that happen often, or that might be useful in other sketches.

This one is `loadPalette()`, and it fills a 16-bit array with 256 color values computed using various formulas. It's the most math-y part of the whole sketch, but it's not nasty math. The arrangement of the 16 bits in a color value is described more fully in the Adafruit_GFX tutorial (https://adafru.it/JD8) (which describes many of the Arcada GFX functions as well).

The sharp-eyed reader might notice that some of the lines appear in pairs, one commented out, the other one not. Why? It's due to an odd characteristic of the TFT screens on my PyBadges. It turns out that the colors appear in gradual, even steps as their value increases from black to full color...except for the last step. The brightest color has more contrast than other values, and makes harsh contours appear where there should be smooth gradients on my screens, so the math stops short of using the maximum values of RGB, and the smoothness is preserved. If you wish to use the full range of color on your screen, just reverse which line is commented out for each pair. Your hardware may vary.

```
// Compute and fill an array with 256 16-bit color values
void loadPalette(uint16_t palNumber) {
  uint16_t x, y;
  float fleX, fleK;

  switch(palNumber) {
    case 1:  // Compute ironbow palette
      for(x = 0; x &lt; 256; ++x) {
        fleX = (float)x / 255.0;

        // fleK = 65535.9 * (1.02 - (fleX - 0.72) * (fleX - 0.72) * 1.96);
        // fleK = (fleK &gt; 65535.0) || (fleX &gt; 0.75) ? 65535.0 : fleK;  //
Truncate red curve
        fleK = 63487.0 * (1.02 - (fleX - 0.72) * (fleX - 0.72) * 1.96);
        fleK = (fleK &gt; 63487.0) || (fleX &gt; 0.75) ? 63487.0 : fleK;  //
Truncate red curve
        colorPal[x] = (uint16_t)fleK &amp;amp; 0xF800;                      // Top 5
bits define red

        // fleK = fleX * fleX * 2047.9;
        fleK = fleX * fleX * 2015.0;
        colorPal[x] += (uint16_t)fleK &amp;amp; 0x07E0;  // Middle 6 bits define green

        // fleK = 31.9 * (14.0 * (fleX * fleX * fleX) - 20.0 * (fleX * fleX) + 7.0
* fleX);
        fleK = 30.9 * (14.0 * (fleX * fleX * fleX) - 20.0 * (fleX * fleX) + 7.0 *
fleX);
        fleK = fleK &lt; 0.0 ? 0.0 : fleK;          // Truncate blue curve
        colorPal[x] += (uint16_t)fleK &amp;amp; 0x001F;  // Bottom 5 bits define blue
      }
      break;
    case 2:  // Compute quadratic "firebow" palette
      for(x = 0; x &lt; 256; ++x) {
        fleX = (float)x / 255.0;

        // fleK = 65535.9 * (1.00 - (fleX - 1.0) * (fleX - 1.0));
        fleK = 63487.0 * (1.00 - (fleX - 1.0) * (fleX - 1.0));
        colorPal[x] = (uint16_t)fleK &amp;amp; 0xF800;                      // Top 5
bits define red

        // fleK = fleX &lt; 0.25 ? 0.0 : (fleX - 0.25) * 1.3333 * 2047.9;
        fleK = fleX &lt; 0.25 ? 0.0 : (fleX - 0.25) * 1.3333 * 2015.0;
```

```
        colorPal[x] += (uint16_t)fleK &amp; 0x07E0;  // Middle 6 bits define green

        // fleK = fleX &lt; 0.5 ? 0.0 : (fleX - 0.5) * (fleX - 0.5) * 127.9;
        fleK = fleX &lt; 0.5 ? 0.0 : (fleX - 0.5) * (fleX - 0.5) * 123.0;
        colorPal[x] += (uint16_t)fleK &amp; 0x001F;  // Bottom 5 bits define blue
      }
      break;
    case 3:  // Compute "alarm" palette
      for(x = 0; x &lt; 256; ++x) {
        fleX = (float)x / 255.0;

        fleK = 65535.9 * (fleX &lt; 0.875 ? 1.00 - (fleX * 1.1428) : 1.0);
        colorPal[x] = (uint16_t)fleK &amp; 0xF800;                      // Top 5
bits define red

        fleK = 2047.9 * (fleX &lt; 0.875 ? 1.00 - (fleX * 1.1428) : (fleX - 0.875) *
8.0);
        colorPal[x] += (uint16_t)fleK &amp; 0x07E0;  // Middle 6 bits define green

        fleK = 31.9 * (fleX &lt; 0.875 ? 1.00 - (fleX * 1.1428) : 0.0);
        colorPal[x] += (uint16_t)fleK &amp; 0x001F;  // Bottom 5 bits define blue
      }
      break;
    case 4:  // Compute negative gray palette, black hot
      for(x = 0; x &lt; 256; ++x)
        colorPal[255 - x] = (((uint16_t)x &lt;&lt; 8) &amp; 0xF800) + (((uint16_t)x
&lt;&lt; 3) &amp; 0x07E0) + (((uint16_t)x &gt;&gt; 3) &amp; 0x001F);
      break;
    default:  // Compute gray palette, white hot
      for(x = 0; x &lt; 256; ++x)
        colorPal[x] = (((uint16_t)x &lt;&lt; 8) &amp; 0xF800) + (((uint16_t)x
&lt;&lt; 3) &amp; 0x07E0) + (((uint16_t)x &gt;&gt; 3) &amp; 0x001F);
      break;
  }
}
```

Here are two more functions. `setColorRange()` just receives an integer and changes two floating point values. They define the low and high temperatures for several preset color ranges.

`setBackdrop()` clears the screen and draws the parts of the display that don't change from frame to frame, like the color palette and the button labels.

```
void setColorRange(int presetIndex) { // Set coldest/hottest values in color range
  switch(presetIndex) {
    case 1:  // Standard range, from FLIR document: 50F to 90F
      colorLow = 10.0;
      colorHigh = 32.22;
      break;
    case 2:  // Cool/warm range, for detecting mammals outdoors
      colorLow = 5.0;
      colorHigh = 32.0;
      break;
    case 3:  // Warm/warmer range, for detecting mammals indoors
      colorLow = 20.0;
      colorHigh = 32.0;
      break;
    case 4:  // Hot spots, is anything hotter than it ought to be?
      colorLow = 20.0;
      colorHigh = 50.0;
      break;
    case 5:  // Fire &amp; ice, extreme temperatures only!
      colorLow = -10.0;
```

```
      colorHigh = 200.0;
      break;
    default:  // Default is autorange, so these values will change with every frame
      colorLow = 0.0;
      colorHigh = 100.0;
      break;
  }
}

// Draw the stationary screen elements behind the live camera window
void setBackdrop(uint16_t bgColor, uint16_t buttonFunc) {
  arcada.display-&gt;fillScreen(bgColor);

  for(int x = 0; x &lt; 160; ++x)   // Paint current palette across bottom
    arcada.display-&gt;drawFastVLine(x, 110, 6, colorPal[map(x, 0, 159, 0, 255)]);

  arcada.display-&gt;setCursor(16, 120);
  arcada.display-&gt;setTextColor(0xFFFF, bgColor);    // White text, current BG
for button labels
  switch(buttonFunc) {
    case 0:
      arcada.display-&gt;print("B:MENU        A:FREEZE");
      break;
    case 1:
      arcada.display-&gt;print("B:MENU       ");
      if(nextBMPindex == 0)                          // No room to store a BMP in
flash media?
        arcada.display-&gt;setTextColor(GRAY_33 &gt;&gt; 1); // Grayed button label
      arcada.display-&gt;print("A:CAPTURE");
      break;
    case 2:
      arcada.display-&gt;print("B:MENU    ");
      if(nextDirIndex == 0)                          // Has flash storage no room
for a new directory?
        arcada.display-&gt;setTextColor(GRAY_33 &gt;&gt; 1); // Grayed button label
      arcada.display-&gt;print("A:START RECORD");
      break;
    case 3:
      arcada.display-&gt;print("B:MENU    ");
      arcada.display-&gt;setTextColor(0xFFFF, 0xF800);  // White text, red BG
recording indicator
      arcada.display-&gt;print("A:STOP RECORD");
      break;
    case 4:
      arcada.display-&gt;print("               A:EXIT"); // Use for bitmap
redisplay only
      break;
  }
}
```

The next two functions should have been swapped in order, making the program's flow a little clearer. Sorry about that.

Anyway, the second one, `newDirectory()`, is used before recording a sequence. Each new BMP sequence gets a freshly minted flash directory all its own, and its name is built using a serial number. This function takes a serial number and converts it back and forth between an array of characters and a string object to construct the directory name. With that, it tries to create a subdirectory, and returns a true value if successful.

`prepForSave()` simply expands the 8-bit values in the output array to 24 grayscale bits and calls another function, `writeBMP()`, to complete the process of... well, writing a BMP file. If the write fails, an alert message pops up on the screen.

```
void prepForSave() {
  for(int x = 0; x &lt; 768; ++x)
    pixelArray[3 * x + 2] = pixelArray[3 * x + 1] = pixelArray[3 * x];  // Copy
each blue byte into R &amp; G for 256 grays in 24 bits

  if(!writeBMP()) {                                        // Did BMP write to flash
fail?
    arcada.display-&gt;fillRect(0, 96, 160, 12, 0xF800);    // Red error signal
    arcada.display-&gt;setTextColor(0xFFFF);                // with white text
    arcada.display-&gt;setCursor(20, 99);
    arcada.display-&gt;print("Storage error!");
  }
}

boolean newDirectory() { // Create a subdirectory, converting the name between char
arrays and string objects
  char fileArray[64];
  String fullPath;

  sprintf(fileArray, DIR_FORMAT, nextDirIndex); // Generate subdirectory name
  fullPath = BOTTOM_DIR + String(fileArray);    // Make a filepath out of it, then
  return arcada.mkdir(fullPath.c_str());        // try to make a real subdirectory
from it
}
```

## The Augmented BMP Files

This function is what makes captured images fun.  It starts by building a string containing a path for saving the image under an unused serial file name. Then the bytes of a normal BMP get written next, two headers, the array of pixel values, and two pad bytes. But keep reading. The extra data comes next.

```
// Here we write the actual bytes of a BMP file (plus extras) to flash media
boolean writeBMP() {
  uint16_t counter1, shiftedFloats[14]; // A buffer for the appended floats and
uint16_t's
  uint32_t timeStamp;
  float shiftAssist;
  char fileArray[64];
  String fullPath;

// First, figure out a name and path for our new BMP
  fullPath = BOTTOM_DIR;                                // Build a filepath starting
with the base subdirectory
  if(buttonRfunc == 2) {                                // BMP sequence recording in
progress?
    sprintf(fileArray, DIR_FORMAT, nextDirIndex);     // Generate subdirectory name
    fullPath += String(fileArray);                    // Add it to the path
    sprintf(fileArray, BMP_FORMAT, nextBMPsequence);  // Generate a sequential
filename
    fullPath += String(fileArray);                    // Complete the filepath
string
  } else {                                            // Not a sequence, solitary
BMP file
    sprintf(fileArray, BMP_FORMAT, nextBMPindex);     // Generate a serial filename
```

```
    fullPath += String(fileArray);                    // Complete the filepath
string
  }

  myFile = arcada.open(fullPath.c_str(), FILE_WRITE); // Only one file can be open
at a time

  if(myFile) {                        // If the file opened okay, write to it:
    myFile.write(BmpPSPHead, 14);   // BMP header 1
    myFile.write(DIBHeadPSP1, 40);  // BMP header 2
    myFile.write(pixelArray, 2304); // Array of 768 BGR byte triples
    myFile.write(PSPpad, 2);        // Pad with 2 zeros 'cause Photoshop does it.
```

Next, five extra floating point values are stored in a peculiar way. All temperatures are temporarily made positive by adding 1000. The integer portion becomes a 16-bit integer, and the fractional portion gets multiplied by 49152 to produce another 16-bit integer, and these are copied byte by byte into another small array... 20 bytes to hold 5 floating point temperatures. It's not elegant, but it allows other sketches to access these numbers and reconstruct nearly every pixel temperature in the image.

Three more useful numbers get broken into bytes and added, the positions of the coldest and hottest pixels, and the elapsed time in milliseconds.

```
// My BMP hack - append 5 fixed-point temperature values as 40 extra bytes
    for(counter1 = 0; counter1 &lt; 5; ++counter1) {    // Shift 5 floats
      shiftAssist = sneakFloats[counter1] + 1000.0;  // Offset MLX90640 temps to
positive
      shiftedFloats[counter1 * 2] = (uint16_t)shiftAssist;
      shiftAssist = (shiftAssist - (float)shiftedFloats[counter1 * 2]) *
49152.0; // Scale up fraction
      shiftedFloats[counter1 * 2 + 1] = (uint16_t)shiftAssist;
    }

    shiftedFloats[10] = lowAddr;   // Two more appended numbers, the 2 extreme
pixel addresses
    shiftedFloats[11] = highAddr;

    timeStamp = millis();         // Recycle this variable to append a time stamp
    lowAddr = timeStamp &amp; 0xFFFF;
    highAddr = timeStamp &gt;&gt; 16;
    shiftedFloats[12] = lowAddr;
    shiftedFloats[13] = highAddr;

    myFile.write(shiftedFloats, 28);  // Write appended uint16_t's

    myFile.close();
    return true;
  } else {          // The file didn't open, return error
    return false;
  }
}
```

`recallLastBMP()` is a bit of a cheap hack, but useful. The byte array that's used when saving BMPs gets overwritten with every new frame displayed. That is, parts of it get overwritten and parts of it don't. Some bytes still persist unchanged from the last image capture, and this function can clear the screen and redisplay those bytes

as pixels again.  Not as good as recalling an image from flash storage, but handy in its own way.

It requires a button click to exit, so the debounce instructions from the main loop appear at the end here as well.

```
void recallLastBMP(uint16_t bgColor) {  // Display 8-bit values left in buffer from
the last BMP save
  int counter1, counter2;
  boolean exitFlag = false;

  setBackdrop(bgColor, 4);  // Clear screen, just a color palette &amp; "A:EXIT" in
the BG

  for(int counter1 = 0; counter1 &lt; 24; ++counter1) {  // Redraw using leftover
red byte values, not yet overwritten
    for(int counter2 = 0 ; counter2 &lt; 32 ; ++counter2) {
      arcada.display-&gt;fillRect(16 + counter2 * 4, 92 - counter1 * 4, 4, 4,
                colorPal[(uint16_t)pixelArray[3 * (32 * counter1 + counter2) +
2]]);
    }
  }

  while(!exitFlag) {  // Loop here until exit button
    if(!buttonActive &amp;&amp; (buttonBits &amp; ARCADA_BUTTONMASK_A)) { //
"A:EXIT" button freshly pressed?
      exitFlag = true;
      buttonActive = true;
      deBounce = millis() + DE_BOUNCE;
    }

    if(buttonActive &amp;&amp; millis() &gt; deBounce
       &amp;&amp; (buttonBits &amp; (ARCADA_BUTTONMASK_A | ARCADA_BUTTONMASK_B)) ==
0)  // Has de-bounce wait expired &amp; all buttons released?
      buttonActive = false;               // Clear flag to allow another button
press
  }
}
```

The function `availableFileNumber()` is small, but I use it a lot when I need a name for saving a new file.  It takes a number and a character string and combines them into a numbered filename, then checks whether a file by that name already exists in storage.  If so, it tries the next number, and keeps counting until an unoccupied filename is found, or the value `MAX_SERIAL` is exceeded.  The number of the first free filename is returned, or a zero if none is found.

```
uint16_t availableFileNumber(uint16_t startNumber, String formatBase) { // Find
unclaimed serial number for file series
  uint16_t counter1;
  char nameArray[80];

  for(counter1 = startNumber; counter1 % MAX_SERIAL != 0; ++counter1) { // Start
counting
    sprintf(nameArray, formatBase.c_str(), counter1);                  // Generate
a serialized filename
    if(!arcada.exists(nameArray))                                      // If it
doesn't already exist
      return counter1;                                                 // return
the number as available
```

```
  }
  return 0; // Loop finished, no free number found, return fail
}
```

---

# What's on the Menu?

`menuLoop()` is where settings can be changed while the sketch runs.  The top half of this function deals with the menu on the screen as the user clicks the B button.  It keeps track of which menu lines should appear on the screen as the user scrolls down, which line is highlighted with the black stripe cursor, but not what each menu text should say.  That's done later by the `menuLines()` function.

```
boolean menuLoop(uint16_t bgColor) {  // Lay out a menu screen, interact to change
values
  int counter1 = 0, scrollPosition = 0;
  boolean exitFlag = false, settingsChanged = false;
  uint32_t menuButtons;

  arcada.display-&gt;fillScreen(bgColor);
  arcada.display-&gt;fillRect(0, 12 * (counter1 + scrollPosition) + MENU_VPOS - 2,
160, 12, 0x0000);  // Black stripe cursor on menu

  arcada.display-&gt;setTextColor(0xFFFF);             // White text
  arcada.display-&gt;setCursor(16, 120);               // at screen bottom
  arcada.display-&gt;print("B:ADVANCE     A:CHANGE");  // for button labels

  for(counter1 = 0; counter1 &lt; MENU_ROWS; ++counter1) {  // Display menu texts
    menuLines(counter1, scrollPosition);
  }
  counter1 = 0;

  while(!exitFlag) { // Loop until exit is activated
    if(!buttonActive &amp;&amp; (buttonBits &amp; ARCADA_BUTTONMASK_B)) {  // Fresh
press of B:ADVANCE button?
      buttonActive = true;                                    // Set button flag
      deBounce = millis() + DE_BOUNCE;                        // and start
debounce timer.

      arcada.display-&gt;fillRect(0, 12 * (counter1 - scrollPosition) + MENU_VPOS -
2, 160, 12, bgColor); // Erase cursor &amp; text
      menuLines(counter1, scrollPosition);                    // Refresh menu text
line
      counter1 = (counter1 + 1) % MENU_LEN;                   // Advance menu
counter

      if(counter1 == 0) {                                     // Have we cycled
around to the menu top?
        scrollPosition = 0;
        for(int counter2 = 0; counter2 &lt; MENU_ROWS; ++counter2) {  // Redisplay
all menu texts
          arcada.display-&gt;fillRect(0, 12 * counter2 + MENU_VPOS - 2, 160, 12,
bgColor); // Erase old text
          menuLines(counter2 + scrollPosition, scrollPosition);    // Redraw each
text line
        }
      } else if((counter1 + 1 &lt; MENU_LEN) &amp;&amp; (counter1 - scrollPosition
== MENU_ROWS - 1)) { // Should we scroll down 1 menu line?
        ++scrollPosition;
        for(int counter2 = 0; counter2 &lt; MENU_ROWS; ++counter2) {  // Redisplay
all menu texts
          arcada.display-&gt;fillRect(0, 12 * counter2 + MENU_VPOS - 2, 160, 12,
bgColor); // Erase old text
```

```
          menuLines(counter2 + scrollPosition, scrollPosition);    // Redraw each
text line
        }
      }

      arcada.display-&gt;fillRect(0, 12 * (counter1 - scrollPosition) + MENU_VPOS -
2, 160, 12, 0x0000);  // New black cursor
      menuLines(counter1, scrollPosition);                    // Refresh text line
      deBounce = millis() + DE_BOUNCE;                        // Restart debounce
timer, just for safety
    }
```

Here's the other half of `menuLoop()`. It controls what happens when button A clicks. A `switch()` statement jumps to the right code snippet, depending on where the menu cursor is standing. Some choices just toggle a Boolean flag, some step through a range of numbers, and some immediately affect the hardware, like the backlight LED. Once done, the screen menu gets updated, the button debounce values are refreshed, and the loop either exits or continues, depending on what got clicked.

```
if(!buttonActive &amp;&amp; (buttonBits &amp; ARCADA_BUTTONMASK_A)) {  // Fresh
press of A:CHANGE button?
      buttonActive = true;                                    // Set button flag
      deBounce = millis() + DE_BOUNCE;                        // and start
debounce timer.

      switch(counter1) {       // Change whichever setting is currently hilighted
        case 0:
          showLastCap = true;  // Set flag to display the last frame captured to SD
          exitFlag = true;     // and exit
          break;
        case 1:
          celsiusFlag = !celsiusFlag; // Toggle Celsius/Fahrenheit
          break;
        case 2:
          buttonRfunc = (buttonRfunc + 1) % 3; // Step through button functions
          break;
        case 3:
          loadPalette(paletteNum = (paletteNum + 1) % 5); // Step through various
color palettes
          break;
        case 4:
          thermRange = (thermRange + 1) % 6; // Step through various temp range
presets
          break;
        case 5:
          markersOn = !markersOn; // Toggle hot/cold marker visibility
          break;
        case 6:
          mirrorFlag = !mirrorFlag; // Toggle mirrored display
          break;
        case 7:
          switch(frameRate = (frameRate + 1) % 6) {             // 6 frame rates,
0.5 to 16 in powers of 2
            case 0: mlx.setRefreshRate(MLX90640_0_5_HZ); break;
            case 1: mlx.setRefreshRate(MLX90640_1_HZ); break;
            case 2: mlx.setRefreshRate(MLX90640_2_HZ); break;
            case 3: mlx.setRefreshRate(MLX90640_4_HZ); break;
            case 4: mlx.setRefreshRate(MLX90640_8_HZ); break;
            default: mlx.setRefreshRate(MLX90640_16_HZ); break;
          }
          break;
        case 8:
          emissivity = (emissivity + 90) % 100; // Step from 95% to 5% by -10%
          break;
```

```
        case 9:
          smoothing = !smoothing; // Toggle pixel smoothing
          break;
        case 10:
          arcada.setBacklight((screenDim = !screenDim) ? 64 : 255); // Change
backlight LED
          break;
        default:
          exitFlag = true;
          break;
      }
      if((counter1 &gt; 0) &amp;&amp; (counter1 &lt; MENU_LEN - 1))    // Was any
setting just changed?
        settingsChanged = true;

      arcada.display-&gt;fillRect(0, 12 * (counter1 - scrollPosition) + MENU_VPOS -
2, 160, 12, 0x0000); // Erase hilit menu line
      menuLines(counter1, scrollPosition);   // Retype hilit menu line
    }

    if(buttonActive &amp;&amp; millis() &gt; deBounce
       &amp;&amp; (buttonBits &amp; (ARCADA_BUTTONMASK_A | ARCADA_BUTTONMASK_B)) ==
0)  // Has de-bounce wait expired &amp; all buttons released?
      buttonActive = false;               // Clear flag to allow another button
press
  }
  return(settingsChanged);
}
```

When `menuLoop()` displays a menu on the screen, it lets the `menuLines()` function print each line, one at a time. This function receives two numbers, the number of the line item to print, and where to print it.  The name for each setting appears on the left, and its current value on the right. A couple of frills are there as well, like scrolling arrows and mini color palettes.

```
void menuLines(int lineNumber, int scrollPos) {  // Screen print a single line in
the settings menu

  arcada.display-&gt;setTextColor(0xFFFF);               // White text
  arcada.display-&gt;setCursor(10, 12 * (lineNumber - scrollPos) + MENU_VPOS); //
Menu lines 12 pixels apart

  if(lineNumber - scrollPos == 0 &amp;&amp; scrollPos &gt; 0) {  // Are any menu
lines scrolled off screen top?
    arcada.display-&gt;print("           ^");            // Print a small up arrow
indicator
  } else if(lineNumber - scrollPos == 8 &amp;&amp; lineNumber + 1 &lt; MENU_LEN)
{ // How about off the bottom?
    arcada.display-&gt;print("           v");            // Print a small down
arrow indicator... yeah, it's a v
  } else {

    switch(lineNumber) {
      case 0:
        arcada.display-&gt;print("  Display last capture");
        break;
      case 1:
        arcada.display-&gt;print("     Scale - ");
        arcada.display-&gt;print(celsiusFlag ? "CELSIUS" : "FAHRENHEIT");
        break;
      case 2:
        arcada.display-&gt;print(" Rt button - ");
        switch(buttonRfunc) {
          case 1:
```

```
            arcada.display-&gt;print("CAPTURE"); break;
          case 2:
            arcada.display-&gt;print("RECORD"); break;
          default:
            arcada.display-&gt;print("FREEZE"); break;
        }
        break;
      case 3:
        arcada.display-&gt;print("   Palette - ");
        for(int xPos = 0; xPos &lt; 72; ++xPos)   // Display the current heat
spectrum colors
          arcada.display-&gt;drawFastVLine(xPos + 87, (lineNumber - scrollPos) * 12
+ MENU_VPOS,
                                           8, colorPal[map(xPos, 0, 71, 0, 255)]);
        switch(paletteNum) {
          case 1:
            arcada.display-&gt;print("IRONBOW");
            break;
          case 2:
            arcada.display-&gt;print("FIREBOW");
            break;
          case 3:
            arcada.display-&gt;setTextColor(0x0000);    // Black text for reverse
contrast
            arcada.display-&gt;print("ALARM");
            break;
          case 4:
            arcada.display-&gt;setTextColor(0x0000);    // Black text
            arcada.display-&gt;print("BLACK HOT");
            break;
          default:
            arcada.display-&gt;print("WHITE HOT");
            break;
        }
        break;
      case 4:
        arcada.display-&gt;print("Temp range - ");
        setColorRange(thermRange);
        switch(thermRange) {
          case 1:
            arcada.display-&gt;print("STANDARD"); break;
          case 2:
            arcada.display-&gt;print("COOL/WARM"); break;
          case 3:
            arcada.display-&gt;print("WARM/WARMER"); break;
          case 4:
            arcada.display-&gt;print("HOT SPOTS"); break;
          case 5:
            arcada.display-&gt;print("FIRE &amp; ICE"); break;
          default:
            arcada.display-&gt;print("AUTO-RANGE"); break;
        }
        break;
      case 5:
        arcada.display-&gt;print("   Markers - ");
        arcada.display-&gt;print(markersOn ? "ON" : "OFF");
        break;
      case 6:
        arcada.display-&gt;print("     Image - ");
        arcada.display-&gt;print(mirrorFlag ? "MIRRORED" : "FORWARD");
        break;
      case 7:
        arcada.display-&gt;print("Frame rate - ");
        arcada.display-&gt;print((float)(1 &lt;&lt; frameRate) * 0.5);
        arcada.display-&gt;print(" FPS");
        break;
      case 8:
        arcada.display-&gt;setTextColor(GRAY_33 &lt;&lt; 1); // Grayed menu item
        arcada.display-&gt;print("Emissivity - ");
```

```
        arcada.display-&gt;print(emissivity);
        arcada.display-&gt;print("%");
        break;
      case 9:
        arcada.display-&gt;setTextColor(GRAY_33 &lt;&lt; 1); // Grayed menu item
        arcada.display-&gt;print(" Smoothing - ");
        arcada.display-&gt;print(smoothing ? "ON" : "OFF");
        break;
      case 10:
        arcada.display-&gt;print(" Backlight - ");
        arcada.display-&gt;print(screenDim ? "DIM" : "FULL");
        break;
      case 11:
        arcada.display-&gt;print("       Exit menu");
    }
  }
}
```

The last little function is the one I promised way earlier. Its only job is to set Boolean flags whenever it detects button A or B pressing. That's it, but it gets called by a timer 50 times a second no matter what else is in progress. That way almost no button clicks get missed, and interactivity is preserved. Nice function.

```
// This is the function that substitutes for GPIO external interrupts
// It will check for A and B button presses at 50Hz
void buttonCatcher(void) {
  buttonBits = arcada.readButtons();
  clickFlagMenu |= (buttonBits &amp; ARCADA_BUTTONMASK_B) != 0;
  clickFlagSelect |= (buttonBits &amp; ARCADA_BUTTONMASK_A) != 0;
}
```

You made it to the end! Well done! The only prize I can offer is a deeper insight into the workings of your thermal camera software. That, plus the Processing sketches that can turn your captured images into fascinating visual media. They're next.

---

# The Processing Language

The camera alone can do plenty, but with the data it captures you can do even more. One way to explore your thermal images is to use sketches in a programming language from Processing.org (https://adafru.it/aPt) that runs on a full-blown computer. There is an Adafruit tutorial (https://adafru.it/JD9) on installing the Processing environment on a Raspberry Pi, and their website (https://adafru.it/jsC) has links for Mac and Windows users as well.

## The Inspector Sketch

Once you have the Processing package installed and set up, here is a sketch to download and try. Open a new sketch window, copy the following code and paste it there, and save it as ConvertBMPinspector01. **Be sure you're working in the**

**Processing environment and not in the Arduino environment**. It's an easy mistake to make.

```
// ConvertBMPinspector01 - Read and enlarge a modified 32x24 24-bit gray BMP file,
//                         display an upscaled 256x192 BMP image in 256 colors.
// Ver. 1 - Fetch filenames and display BMPs in sequence.
//          Add nav buttons and mouseover pixel temperatures
//          This sketch does no checking for file compatibility.
//          Only frm_____.bmp images from the thermal camera sketch will work.
//          Any other files in the data folder will fail.

import java.util.Date;

byte b[], colorPal[];     // Buffers for input file bytes and for colors

int i, fileCount = 0, BGcolor = 48, colorMap = 1,
    butnsX = 30, butnsY = 290,
    offsetX = 153, offsetY = 6,    // These value pairs control where the onscreen
features appear
    numbersX = 40, numbersY = 48,
    probeX = 190, probeY = 210;
boolean celsiusFlag = false;
float fixedPoint[];
String[] filenames;

void setup() {

  size(480, 360);      // Size must be the first statement
  background(BGcolor); // Clear the screen with a gray background

  colorPal = new byte[1024];   // Prepare a 1K color table
  loadColorTable(colorMap, 0); // Load color table, 1 == ironbow palette
  fixedPoint = new float[5];   // A buffer for appended fixed point values

  String path = sketchPath() + "/data";  // Read from the "/data" subdirectory

  filenames = listFileNames(path);
  fileCount = filenames.length;

  i = 0;
  if(fileCount < 1) {
    println("No files found.  Stopping.");
    noLoop();
  } else {
    loadBMPscreen(i);  // Read in the first frame for inspection
  }
}

void draw() {
  int sampleX, sampleY, pixelVal;
  float sampleTemp;

  sampleX = (mouseX - offsetX) >> 3;       // Map mouse position to BMP pixel space
  sampleY = 23 - ((mouseY - offsetY) >> 3);

  noStroke();
  smooth();
  fill(BGcolor + 16);
  rect(probeX, probeY, 180, 40);   // Clear the interactive window space

  if((sampleX >= 0) && (sampleX < 32) && (sampleY >= 0) && (sampleY < 24)) { //
Mouse within BMP image bounds?
    pixelVal = b[54 + (32 * sampleY + sampleX) * 3] & 0xff;                  //
Read the 8-bit pixel value

    fill(colorPal[4 * pixelVal + 2] & 0xFF, colorPal[4 * pixelVal + 1] & 0xFF,
colorPal[4 * pixelVal + 0] & 0xFF);
```

```
    rect(probeX, probeY, 180, 40);
    fill(BGcolor);
    rect(probeX + 10, probeY + 10, 160, 20);   // Draw a colorized frame for the
interactive temp readout

    sampleTemp = (float(pixelVal) + 1.0) / 257.0 * (fixedPoint[3] - fixedPoint[1]) +
fixedPoint[1];
    if(!celsiusFlag)
      sampleTemp = sampleTemp * 1.8 + 32.0;

    fill(255);       // Ready to display white interactive text
    textSize(11);
    text(sampleX, probeX + 154, probeY + 19);   // Display X Y position
    text(sampleY, probeX + 154, probeY + 29);
    textSize(15);
    text(sampleTemp, probeX + 60, probeY + 25); // Display temperature

    if(pixelVal ==   0 && fixedPoint[0] < fixedPoint[1]) // Pixel values clipped at
bottom limit?
      text("<", probeX + 40, probeY + 25);               // Show out-of-range
indicator
    if(pixelVal == 255 && fixedPoint[4] > fixedPoint[3]) // Clipped at top?
      text(">", probeX + 40, probeY + 25);               // Same
  }

  noSmooth();     // Clear any highlighted buttons
  stroke(0);
  noFill();
  for(sampleX = 0; sampleX < 8; ++sampleX)
    rect(butnsX + sampleX * 52, butnsY, 51, 24);

  sampleX = mouseX - butnsX;
  sampleY = mouseY - butnsY;
  if(sampleX >=0 && sampleX < 416 && sampleY >= 0 && sampleY < 24) { // Mouse over
buttons?
    sampleX = sampleX / 52;                       // Map mouse X to button X space
    stroke(BGcolor + 64);
    rect(butnsX + sampleX * 52, butnsY, 51, 24);  // Highlight border around a
button
  }
}

void keyPressed() {  // Load a different thermal BMP image based on keystroke
  switch(key) {
    case '.':   // Next image
      i = (i + 1) % fileCount;
      break;
    case ',':   // Prev Image
      i = (i + fileCount - 1) % fileCount;
      break;
    case '>':   // 16 images forward
      i = i + 16 < fileCount ? i + 16 : fileCount - 1;
      break;
    case '<':   // 16 images back
      i = i - 16 < 0 ? 0 : i - 16;
      break;
    case '/':   // Last image
      i = fileCount - 1;
      break;
    case 'm':   // First image
      i = 0;
      break;
  }
  loadBMPscreen(i);
}

void mousePressed() {
  int sampleX, sampleY;
```

```
  sampleX = mouseX - butnsX;
  sampleY = mouseY - butnsY;
  if(sampleX >=0 && sampleX < 416 && sampleY >= 0 && sampleY < 24) { // Is mouse
over button row?
    sampleX = sampleX / 52;                       // Map mouse X to button X space

    switch(sampleX) {
      case 1:   // First image
        i = 0;
        break;
      case 2:   // 16 images back
        i = i - 16 < 0 ? 0 : i - 16;
        break;
      case 3:   // Prev Image
        i = (i + fileCount - 1) % fileCount;
        break;
      case 4:   // Next image
        i = (i + 1) % fileCount;
        break;
      case 5:   // 16 images forward
        i = i + 16 < fileCount ? i + 16 : fileCount - 1;
        break;
      case 6:   // Last image
        i = fileCount - 1;
        break;
      case 7:   // Change color map
        loadColorTable(colorMap = (colorMap + 1) % 5, 0); // Load color table
        break;
      default:  // Toggle C/F
        celsiusFlag = !celsiusFlag;
        break;
    }
    loadBMPscreen(i);
  }
}

void loadBMPscreen(int fileIndex) {
  int x, y;

  b = loadBytes(filenames[fileIndex]);   // Open a file and read its 8-bit data
  background(BGcolor);                   // Clear screen
  enlarge8bitColor();                    // Place colored enlarged image on screen

  for(x = 0; x < 5; ++x) {  // Rebuild 5 float values from next 4*n bytes in the
file
    fixedPoint[x] = expandFloat(b[2360 + (x * 4) + 0], b[2360 + (x * 4) + 1],
                                b[2360 + (x * 4) + 2], b[2360 + (x * 4) + 3]);
  }
  y = ((b[2387] & 0xff) << 24) + ((b[2386] & 0xff) << 16)
    + ((b[2385] & 0xff) <<  8) +  (b[2384] & 0xff);       // Reassemble a
milliseconds time stamp

  textSize(10);        // Print text labels for the frame stats
  smooth();
  fill(255);
  text(filenames[fileIndex], numbersX + 5, numbersY + 40); // Show current filename

  if(celsiusFlag)
    text("Frame\n\n\nSeconds\n\nDegrees C", numbersX + 5, numbersY + 8);
  else
    text("Frame\n\n\nSeconds\n\nDegrees F", numbersX + 5, numbersY + 8);

  text("Approximate temperatures based on 8-bit pixel values", probeX - 42, probeY +
52); // Show approximation disclaimer

  textSize(15);
  text(fileIndex, numbersX + 5, numbersY + 25);     // Print frame number
  text(float(y) * 0.001, numbersX, numbersY + 74);  // Print time stamp in seconds
```

```
  if(celsiusFlag) {      // Show 3 temps in Celsius
    fill(255, 128, 64);
    text(fixedPoint[4], numbersX, numbersY + 108);
    fill(255, 200, 64);
    text(fixedPoint[2], numbersX, numbersY + 128);
    fill(128, 128, 255);
    text(fixedPoint[0], numbersX, numbersY + 148);

  } else {               // or show them in Farenheit
    fill(255, 128, 64);
    text(fixedPoint[4] * 1.8 + 32.0, numbersX, numbersY + 108);
    fill(255, 200, 64);
    text(fixedPoint[2] * 1.8 + 32.0, numbersX, numbersY + 128);
    fill(128, 128, 255);
    text(fixedPoint[0] * 1.8 + 32.0, numbersX, numbersY + 148);
  }

  noSmooth();
  stroke(0);
  fill(BGcolor + 24);
  for(x = 0; x < 8; ++x)     // Draw 8 button rectangles
    rect(butnsX + x * 52, butnsY, 51, 24);
  for(x = 0; x < 50; ++x) {  // Paint a mini colormap gradient within last button
    y = int(map(x, 0, 50, 0, 255));
    stroke(colorPal[4 * y + 2] & 0xFF, colorPal[4 * y + 1] & 0xFF, colorPal[4 * y +
0] & 0xFF);
    line(butnsX + 365 + x, butnsY + 1, butnsX + 365 + x, butnsY + 23);
  }
  smooth();  // Add text labels to buttons
  fill(255);
  textSize(15);
  text("|<      <<       <         >        >>      >|", butnsX + 70, butnsY + 17);
  if(celsiusFlag)
    text("C", butnsX + 20, butnsY + 18);
  else
    text("F", butnsX + 20, butnsY + 18);
}

void enlarge8bitColor() {  // Convert a small gray BMP array and plot an enlarged
colormapped version
  int x, y;

  noStroke();

  for(y = 0; y < 24; ++y) {   // Count all source pixels
    for(x = 0; x < 32; ++x) {
      int pixMid = b[54 + ((32 * y + x) +  0) * 3] & 0xFF;
      fill(colorPal[4 * pixMid + 2] & 0xFF, colorPal[4 * pixMid + 1] & 0xFF,
colorPal[4 * pixMid + 0] & 0xFF);  // Get color from table
      rect(offsetX + 8 * x, offsetY + 8 * (23 - y), 8, 8);  // Draw a square pixel,
bottom up
    }
  }
}

void loadColorTable(int choiceNum, int offset) {
  int i, x;

  switch(choiceNum) {
    case 1:     // Load 8-bit BMP color table with computed ironbow curves
      for(x = 0; x < 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (1.02 - (fleX - 0.72) * (fleX - 0.72) * 1.96);
        fleG = (fleG > 255.0) || (fleX > 0.75) ? 255.0 : fleG;  // Truncate curve
        i = (int)fleG;
        colorPal[offset + x * 4 + 2] = byte(i & 0xFF);    // Red vals

        fleG = fleX * fleX * 255.9;
```

```
        i = (int)fleG;
        colorPal[offset + x * 4 + 1] = byte(i & 0xFF);    // Grn vals

        fleG = 255.9 * (14.0 * (fleX * fleX * fleX) - 20.0 * (fleX * fleX) + 7.0 *
fleX);
        fleG = fleG < 0.0 ? 0.0 : fleG;  // Truncate curve
        i = (int)fleG;
        colorPal[offset + x * 4 + 0] = byte(i & 0xFF);    // Blu vals
      }
      break;
    case 2:  // Compute quadratic "firebow" palette
      for(x = 0; x < 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (1.00 - (fleX - 1.0) * (fleX - 1.0));
        i = (int)fleG;
        colorPal[offset + x * 4 + 2] = byte(i & 0xFF);    // Red vals

        fleG = fleX < 0.25 ? 0.0 : (fleX - 0.25) * 1.3333 * 255.9;
        i = (int)fleG;
        colorPal[offset + x * 4 + 1] = byte(i & 0xFF);    // Grn vals

        fleG = fleX < 0.5 ? 0.0 : (fleX - 0.5) * (fleX - 0.5) * 1023.9;
        i = (int)fleG;
        colorPal[offset + x * 4 + 0] = byte(i & 0xFF);    // Blu vals
      }
      break;
    case 3:  // Compute "alarm" palette
      for(x = 0; x < 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : 1.0);
        i = (int)fleG;
        colorPal[offset + x * 4 + 2] = byte(i & 0xFF);    // Red vals

        fleG = 255.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : (fleX - 0.875) *
8.0);
        i = (int)fleG;
        colorPal[offset + x * 4 + 1] = byte(i & 0xFF);    // Grn vals

        fleG = 255.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : 0.0);
        i = (int)fleG;
        colorPal[offset + x * 4 + 0] = byte(i & 0xFF);    // Blu vals
      }
      break;
    case 4:    // Grayscale, black hot
      for(x = 0; x < 256; ++x) {
        colorPal[offset + x * 4 + 2] = byte(255 - x & 0xFF);    // Red vals
        colorPal[offset + x * 4 + 1] = byte(255 - x & 0xFF);    // Grn vals
        colorPal[offset + x * 4 + 0] = byte(255 - x & 0xFF);    // Blu vals
      }
      break;
    default:    // Grayscale, white hot
      for(x = 0; x < 256; ++x) {
        colorPal[offset + x * 4 + 2] = byte(x & 0xFF);    // Red vals
        colorPal[offset + x * 4 + 1] = byte(x & 0xFF);    // Grn vals
        colorPal[offset + x * 4 + 0] = byte(x & 0xFF);    // Blu vals
      }
  }
}

// Rebuild a float from a fixed point decimal value encoded in 4 bytes
float expandFloat(byte m1, byte m2, byte e1, byte e2) {
  int fracPart;
  float floatPart;

  fracPart = ((e2 & 0xff) << 8) + (e1 & 0xff);   // Reassemble 16-bit value
  floatPart = (float)fracPart / 49152.0;         // Convert into fractional portion
of float
```

```
  fracPart = ((m2 & 0xff) << 8) + (m1 & 0xff);   // Reassemble 16-bit value
  return ((float)fracPart + floatPart) - 1000.0; // Complete reconstructing
original float
}

String[] listFileNames(String dir) {   // Return the filenames from a directory as
an array of Strings
  File file = new File(dir);

  if (file.isDirectory()) {
    String names[] = file.list();
    return names;
  } else    // It's not a directory
    return null;
}
```

Again, the programming interfaces for Processing and Arduino are very similar, and it's easy to mistake one for the other. In fact, both environments store sketches in separate Sketchbook folders, and sometimes the wrong folder will appear when saving a new sketch. Double check every time. You may have to manually navigate to the correct folder.

Once you've saved this sketch in the Sketchbook folder with a filename, Processing has placed it in a subfolder with the same name. It's good to know where to find this subfolder when importing thermal image datasets. The editor window can quickly open it for you by starting at the menu bar and selecting Sketch>Show sketch folder, where the new file should appear.

So far, so good. There are two more sketches to go.

---

# More Processing Sketches

Here's the second of our three sketches in Processing. Open another new window and paste this code into it, then save it as ConvertBMPto8bit.

```
// ConvertBMPto8bit - Read and enlarge a modified 32x24 24-bit gray BMP file,
//                    write an upscaled 256x192 BMP image with a 256 color table.
// Ver. 2 - Fetch filenames and convert all suitable BMPs we find.
//          Builds sequences suitable for online animated GIF converters

import java.util.Date;

// BMP File Header, little end first
int BmpPSPHead[] = {
 0x42, 0x4D,             // "BM" in hex
 0x36, 0xC4, 0x00, 0x00, // File size, 50230
 0x00, 0x00,             // reserved for app data 1
 0x00, 0x00,             // reserved for app data 2
 0x36, 0x04, 0x00, 0x00  // Offset of pixel 0, 1078
};

// BMP 8-bit DIB Header, little end first
int DIBHeadPSP1[] = {
 0x28, 0x00, 0x00, 0x00,  // Header size, 40
```

```
 0x00, 0x01, 0x00, 0x00,  // pixel width, 256
 0xC0, 0x00, 0x00, 0x00,  // pixel height, 192
 0x01, 0x00,              // color planes, 1
 0x08, 0x00,              // bits per pixel, 8
 0x00, 0x00, 0x00, 0x00,  // Compression method, 0==none
 0x00, 0x00, 0x00, 0x00,  // Raw bitmap data size, dummy 0
 0x12, 0x0B, 0x00, 0x00,  // Pixels per meter H, 2834
 0x12, 0x0B, 0x00, 0x00,  // Pixels per meter V, 2834
 0x00, 0x00, 0x00, 0x00,  // Colors in palette, 0==default 2^n
 0x00, 0x00, 0x00, 0x00   // Number of important colors, 0
};

byte outBytes[], b[];     // Buffer for the input file bytes

PImage img;  // Declare variable of type PImage
int fileCount = 0, imageIndex = 0;
String[] filenames;

// "paletteChoice" selects a false color palette:
// 0 == Grayscale, white hot
// 1 == Ironbow
// 2 == Firebow
// 3 == Hot alarm
// 4 == Grayscale, black hot
int paletteChoice = 1;

void setup() {
  int i, j, x, y;
  String nameHead, nameTail;

  size(256, 192);  // Size must be the first statement
//  noStroke();
  frameRate(5);
  background(0);   // Clear the screen with a black background

  outBytes = new byte[50230]; // 54 header + 1K colors + 12K pixels

  String path = sketchPath() + "/data"; // Read from the "/data" subdirectory

  println("Listing filenames: ");
  filenames = listFileNames(path);
  println(filenames);
  fileCount = filenames.length;
  println(fileCount + " entries");

  if(fileCount < 1) {
    println("No images found.  Stopping.");
  } else {    // Filenames exist in the directory
    for(i = 0; i < fileCount; ++i) {   // Test each name
      nameHead = filenames[i].substring(0, 3);
      nameTail = filenames[i].substring(8);
      j = int(filenames[i].substring(3, 8));

      if(nameHead.equals("frm") && nameTail.equals(".bmp") && j != 0)   // Source
"frm_____.bmp" found?
        enlarge8bit(i);    // Process and write an enlarged 8-bit version
    }
  }
  noLoop();
}

void draw() {
  int countX, countY;

  noSmooth();

  for(countY = 0; countY < 192; ++countY) {
    for(countX = 0; countX < 256; ++countX) {
      stroke(0xFF & outBytes[1078 + (countY * 256 + countX)]); // Color from BMP
```

```
buffer
      point(countX, 191 - countY);                              // Draw a pixel,
bottom up
    }
  }
}

void enlarge8bit(int fileNumber) {  // Read a small gray "frm" BMP image and write
an enlarged colormapped "out" BMP
  int i, x, y;

  b = loadBytes(filenames[fileNumber]);   // Open a file and read its 8-bit data

  for(i = 0; i < 14; ++i)
    outBytes[i] = byte(BmpPSPHead[i] & 0xFF);        // Copy BMP header 1 into
output buffer
  for(i = 0; i < 40; ++i)
    outBytes[i + 14] = byte(DIBHeadPSP1[i] & 0xFF);  // Copy header 2

  loadColorTable(paletteChoice, 54);  // Load color table, 54 byte BMP header offset

  for(y = 0; y < 23; ++y) {        // Bilinear interpolation, count the source
pixels less one
    for(x = 0; x < 31; ++x) {
      for(int yLirp = 0; yLirp < 9; ++yLirp) {
        int corner0 = b[54 + ((32 * y + x) + 32) * 3] & 0xFF;
        int corner1 = b[54 + ((32 * y + x) +  0) * 3] & 0xFF;
        int pixLeft  = (corner0 * yLirp + corner1 * (8 - yLirp)) >> 3;  // Lirp 1
endpoint from 2 L pixels,

        int corner2 = b[54 + ((32 * y + x) + 33) * 3] & 0xFF;
        int corner3 = b[54 + ((32 * y + x) +  1) * 3] & 0xFF;
        int pixRight = (corner2 * yLirp + corner3 * (8 - yLirp)) >> 3;  // and the
other from 2 R pixels

        for(int xLirp = 0; xLirp < 9; ++xLirp) {
          int pixMid = (pixRight * xLirp + pixLeft * (8 - xLirp)) >> 3; // Lirp
between lirped endpoints, bilinear interp
          outBytes[1078 + y * 2048 + x * 8 + yLirp * 256 + xLirp + 771] =
byte(pixMid & 0xFF);
        }
      }
    }
  }
  for(y = 0; y < 192; ++y) {   // Pad out the empty side pixels
    for(x = 0; x < 4; ++x) {
      outBytes[1078 + (3 - x) + 256 * y] = outBytes[1082 + 256 * y];
      outBytes[1330 +      x  + 256 * y] = outBytes[1329 + 256 * y];
    }
  }
  for(x = 0; x < 256; ++x) {   // Pad out the empty above/below pixels
    for(y = 0; y < 4; ++y) {
      outBytes[ 1078 + 256 * (3 - y) + x] = outBytes[ 2102 + x];
      outBytes[49206 + 256 *      y  + x] = outBytes[48950 + x];
    }
  }

  saveBytes("data/out" + filenames[fileNumber].substring(3), outBytes);   // Save a
recolored 8-bit BMP as "out_____.bmp"
}

void loadColorTable(int choiceNum, int offset) {
  int i, x;

  switch(choiceNum) {
    case 1:     // Load 8-bit BMP color table with computed ironbow curves
      for(x = 0; x < 256; ++x) {
        float fleX = (float)x / 255.0;
```

```
        float fleG = 255.9 * (1.02 - (fleX - 0.72) * (fleX - 0.72) * 1.96);
        fleG = (fleG > 255.0) || (fleX > 0.75) ? 255.0 : fleG;  // Truncate curve
        i = (int)fleG;
        outBytes[offset + x * 4 + 2] = byte(i & 0xFF);    // Red vals

        fleG = fleX * fleX * 255.9;
        i = (int)fleG;
        outBytes[offset + x * 4 + 1] = byte(i & 0xFF);    // Grn vals

        fleG = 255.9 * (14.0 * (fleX * fleX * fleX) - 20.0 * (fleX * fleX) + 7.0 *
fleX);
        fleG = fleG < 0.0 ? 0.0 : fleG;  // Truncate curve
        i = (int)fleG;
        outBytes[offset + x * 4 + 0] = byte(i & 0xFF);    // Blu vals
      }
      break;
    case 2:  // Compute quadratic "firebow" palette
      for(x = 0; x < 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (1.00 - (fleX - 1.0) * (fleX - 1.0));
        i = (int)fleG;
        outBytes[offset + x * 4 + 2] = byte(i & 0xFF);    // Red vals

        fleG = fleX < 0.25 ? 0.0 : (fleX - 0.25) * 1.3333 * 255.9;
        i = (int)fleG;
        outBytes[offset + x * 4 + 1] = byte(i & 0xFF);    // Grn vals

        fleG = fleX < 0.5 ? 0.0 : (fleX - 0.5) * (fleX - 0.5) * 1023.9;
        i = (int)fleG;
        outBytes[offset + x * 4 + 0] = byte(i & 0xFF);    // Blu vals
      }
      break;
    case 3:  // Compute "alarm" palette
      for(x = 0; x < 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : 1.0);
        i = (int)fleG;
        outBytes[offset + x * 4 + 2] = byte(i & 0xFF);    // Red vals

        fleG = 255.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : (fleX - 0.875) *
8.0);
        i = (int)fleG;
        outBytes[offset + x * 4 + 1] = byte(i & 0xFF);    // Grn vals

        fleG = 255.9 * (fleX < 0.875 ? 1.00 - (fleX * 1.1428) : 0.0);
        i = (int)fleG;
        outBytes[offset + x * 4 + 0] = byte(i & 0xFF);    // Blu vals
      }
      break;
    case 4:    // Grayscale, black hot
      for(x = 0; x < 256; ++x) {
        outBytes[offset + x * 4 + 2] = byte(255 - x & 0xFF);    // Red vals
        outBytes[offset + x * 4 + 1] = byte(255 - x & 0xFF);    // Grn vals
        outBytes[offset + x * 4 + 0] = byte(255 - x & 0xFF);    // Blu vals
      }
      break;
    default:    // Grayscale, white hot
      for(x = 0; x < 256; ++x) {
        outBytes[offset + x * 4 + 2] = byte(x & 0xFF);    // Red vals
        outBytes[offset + x * 4 + 1] = byte(x & 0xFF);    // Grn vals
        outBytes[offset + x * 4 + 0] = byte(x & 0xFF);    // Blu vals
      }
  }
}

String[] listFileNames(String dir) {   // Return the filenames from a directory as
an array of Strings
```

```
  File file = new File(dir);

  if (file.isDirectory()) {
    String names[] = file.list();
    return names;
  } else    // It's not a directory
    return null;
}
```

## Sketch 3: ConvertBMPtoSeq01

Just one more sketch to go!  Do the same as before, saving this code as another new sketch named ConvertBMPtoSeq01.

```
// ConvertBMPtoSeq01 - Read and enlarge a modified 32x24 24-bit gray BMP file,
//                     saving 256x192 BMP images in 256 colors for converting to
MOV.
// Ver. 1 - Fetch filenames and scan all suitable BMPs we find for their time/temp
data,
//          to set the scale for graphing these numbers through the MOV.

import java.util.Date;

byte colorPal[], b[];     // Buffers for a color palette, and reading bytes from
files

PImage img;
int i, fileCount = 0, frameTotal = 0, earlyFrame = 0, lastFrame = 0,
    hotLowFrame, hotHighFrame, coldLowFrame, coldHighFrame, targLowFrame,
targHighFrame,
    framX1, framX2, coldY1, coldY2, targY1, targY2, hotY1, hotY2,
    offsetX = 153, offsetY = 6, numbersX = 40, numbersY = 30, graphX = 8, graphY =
342,
    histoX = 410, histoY = 342, histoH = 140, histoW = 64, BGcolor = 48;
float hottestLow, hottestHigh, coldestLow, coldestHigh, targetLow, targetHigh;
String[] filenames;



// Change the following values to customize the output images.
// "paletteChoice" selects a false color palette:
// 0 == Grayscale, white hot
// 1 == Ironbow
// 2 == Firebow
// 3 == Hot alarm
// 4 == Grayscale, black hot
int paletteChoice = 1;
boolean markersVisible = true, celsiusFlag = false, lirpSmoothing = true;

void setup() {
  int x, y;
  float fixedPoint[];
  String nameHead, nameTail;

  size(480, 360);      // Size must be the first statement
  background(BGcolor); // Clear the screen with a gray background
  noSmooth();

  colorPal = new byte[1024];        // Reserve a 1K color table
  loadColorTable(paletteChoice, 0); // Load color table
  fixedPoint = new float[5];        // Buffer for added fixed point values

  String path = sketchPath() + "/data";  // Read from the "/data" subdirectory
```

```
//  println("Listing filenames: ");
  filenames = listFileNames(path);
//  println(filenames);
  fileCount = filenames.length;
//  println(fileCount + " entries");

  if(fileCount &lt; 1) {
    println("No images found.  Stopping.");
  } else {    // Filenames exist in the directory, convert what we can

// First pass: Read the embedded times/temps and find maxes/mins for graphing
    print("Counting through files: ");
    for(i = 0; i &lt; fileCount; ++i) {   // Test each filename for conformity
      if((i &amp; 0x3F) == 0)
        print(i + ", ");
      nameHead = filenames[i].substring(0, 3);
      nameTail = filenames[i].substring(8);

      if(nameHead.equals("frm") &amp;&amp; nameTail.equals(".bmp") &amp;&amp;
int(filenames[i].substring(3, 8)) != 0) { // Source "frm_____.bmp" found?
        b = loadBytes(filenames[i]);   // Open a file and read its 8-bit data

        for(x = 0; x &lt; 5; ++x) {  // Rebuild float values from next 4*n bytes in
the file
          fixedPoint[x] = expandFloat(b[2360 + (x * 4) + 0], b[2360 + (x * 4) + 1],
                                      b[2360 + (x * 4) + 2], b[2360 + (x * 4) +
3]); // 2360 == headers + pixels + 2
        }
        y = ((b[2387] &amp; 0xff) &lt;&lt; 24) + ((b[2386] &amp; 0xff) &lt;&lt; 16)
          + ((b[2385] &amp; 0xff) &lt;&lt;  8) +  (b[2384] &amp; 0xff);   //
Reassemble a uint32_t millis() stamp

        if(++frameTotal == 1) { // First frame found so far?
          coldestLow = coldestHigh = fixedPoint[0];
          targetLow  = targetHigh  = fixedPoint[2];  // Initialize all values
          hottestLow = hottestHigh = fixedPoint[4];
          hotLowFrame = hotHighFrame = coldLowFrame = coldHighFrame = targLowFrame =
targHighFrame = earlyFrame = lastFrame = y;
        } else {   // Compare everything, update where necessary

          if(y &lt; earlyFrame)
            earlyFrame = y;       // These will set the left and right bounds
          else if(y &gt; lastFrame)  // of the temperature over time graphs
            lastFrame = y;

          if(fixedPoint[0] &lt; coldestLow) {       // These will define the high
and low bounds
            coldestLow = fixedPoint[0];
            coldLowFrame = y;
          } else if(fixedPoint[0] &gt; coldestHigh) {
            coldestHigh = fixedPoint[0];
            coldHighFrame = y;
          }

          if(fixedPoint[2] &lt; targetLow) {
            targetLow = fixedPoint[2];
            targLowFrame = y;
          } else if(fixedPoint[2] &gt; targetHigh) {
            targetHigh = fixedPoint[2];
            targHighFrame = y;
          }

          if(fixedPoint[4] &lt; hottestLow) {
            hottestLow = fixedPoint[4];
            hotLowFrame = y;
          } else if(fixedPoint[4] &gt; hottestHigh) {
            hottestHigh = fixedPoint[4];
            hotHighFrame = y;
          }
```

```
        }
      }
    }
    println(i + ", done.\n");

// The high and low points of three datasets are found, display them
    println("Frame times " + earlyFrame + " to " + lastFrame + " totaling " +
(lastFrame - earlyFrame));
    println("Cold values " + coldestLow + " at " + coldLowFrame + " to " +
coldestHigh + " at " + coldHighFrame);
    println("Targ values " +  targetLow + " at " + targLowFrame + " to " +
targetHigh + " at " + targHighFrame);
    println("Hot values  " + hottestLow + " at " +  hotLowFrame + " to " +
hottestHigh + " at " +  hotHighFrame);

    stroke(BGcolor + 48);
    for(y = 0; y &lt;= 140; y += 35)
      line(graphX, graphY - y, graphX + 400, graphY - y);  // Draw a generic grid
for the time graph
    for(x = 0; x &lt;= 400; x += 40)
      line(graphX + x, graphY - 140, graphX + x, graphY);

    noStroke();     // Text labels for the top &amp; bottom temp values of the graph
    textSize(10);
    fill(255);
    if(celsiusFlag) {
      text(hottestHigh, graphX + 402, graphY - 142);
      text(coldestLow,  graphX + 402, graphY +  12);
    } else {
      text(hottestHigh * 1.8 + 32.0, graphX + 402, graphY - 142);
      text(coldestLow * 1.8 + 32.0,  graphX + 402, graphY +  12);
    }

    fill(BGcolor + 128);           // Predraw 6 little high/low markers in the
graph space
    rect(graphX + 400 * (coldLowFrame  - earlyFrame) / (lastFrame - earlyFrame) - 1,
         graphY - int((coldestLow -  coldestLow) / (coldestLow - hottestHigh) *
140.0) - 1, 3, 3);
    rect(graphX + 400 * (coldHighFrame - earlyFrame) / (lastFrame - earlyFrame) - 1,
         graphY - int((coldestLow - coldestHigh) / (coldestLow - hottestHigh) *
140.0) - 1, 3, 3);

    rect(graphX + 400 * (targLowFrame  - earlyFrame) / (lastFrame - earlyFrame) - 1,
         graphY - int((coldestLow -   targetLow) / (coldestLow - hottestHigh) *
140.0) - 1, 3, 3);
    rect(graphX + 400 * (targHighFrame - earlyFrame) / (lastFrame - earlyFrame) - 1,
         graphY - int((coldestLow -  targetHigh) / (coldestLow - hottestHigh) *
140.0) - 1, 3, 3);

    rect(graphX + 400 * (hotLowFrame   - earlyFrame) / (lastFrame - earlyFrame) - 1,
         graphY - int((coldestLow -  hottestLow) / (coldestLow - hottestHigh) *
140.0) - 1, 3, 3);
    rect(graphX + 400 * (hotHighFrame  - earlyFrame) / (lastFrame - earlyFrame) - 1,
         graphY - int((coldestLow - hottestHigh) / (coldestLow - hottestHigh) *
140.0) - 1, 3, 3);
  }
  i = 0;
}

// Second pass: Read each frame again, plot color mapped enlarged image,
temperature values and graph, save each frame
void draw() {
  int x, y, histogram[];
  float tempY, fixedPoint[];
  String nameHead, nameTail;

  noSmooth();
  fixedPoint = new float[5];   // Buffer for appended fixed point values
  histogram  = new int[256];   // Buffer for color histogram
```

```
  for(x = 0; x &lt; 256; ++x)
    histogram[x] = 0;          // Initialize histogram

  if(i &lt; fileCount) {   // Test each filename for conformity
    nameHead = filenames[i].substring(0, 3);
    nameTail = filenames[i].substring(8);

    if(nameHead.equals("frm") &amp;&amp; nameTail.equals(".bmp") &amp;&amp;
int(filenames[i].substring(3, 8)) != 0) { // Source "frm_____.bmp" found?
      b = loadBytes(filenames[i]);   // Open a file and read its 8-bit data
//      println(i + " " + filenames[i]);
      enlarge8bitColor();            // Place colored enlarged image on screen

      for(x = 0; x &lt; 5; ++x) {  // Rebuild float values from next 4*n bytes in
the file
        fixedPoint[x] = expandFloat(b[2360 + (x * 4) + 0], b[2360 + (x * 4) + 1],
                                    b[2360 + (x * 4) + 2], b[2360 + (x * 4) + 3]);
      }
      y = ((b[2387] &amp; 0xff) &lt;&lt; 24) + ((b[2386] &amp; 0xff) &lt;&lt; 16)
        + ((b[2385] &amp; 0xff) &lt;&lt;  8) +  (b[2384] &amp; 0xff);       //
Reassemble a milliseconds time stamp

      smooth();
      framX2 = graphX + 400 * (y - earlyFrame) / (lastFrame - earlyFrame);
      coldY2 = graphY - int((coldestLow - fixedPoint[0]) / (coldestLow -
hottestHigh) * 140.0); // Map data values into graph space
      targY2 = graphY - int((coldestLow - fixedPoint[2]) / (coldestLow -
hottestHigh) * 140.0);
      hotY2  = graphY - int((coldestLow - fixedPoint[4]) / (coldestLow -
hottestHigh) * 140.0);

      if(i == 0) {
        framX1 = framX2;  // Set starting points for 3 graphs
        coldY1 = coldY2;
        targY1 = targY2;
        hotY1 = hotY2;
      }

      stroke(128, 128, 255);
      line(framX1, coldY1, framX2, coldY2);  // Graph cold data point
      stroke(255, 200, 64);
      line(framX1, targY1, framX2, targY2);  // Graph center data point
      stroke(255, 128, 64);
      line(framX1,  hotY1, framX2,  hotY2);  // Graph hot data point

      framX1 = framX2;  // Remember endpoints of graphed lines
      coldY1 = coldY2;
      targY1 = targY2;
      hotY1  = hotY2;

      noStroke();          // Print key values onscreen for current frame
      fill(BGcolor);
      rect(numbersX, numbersY, 82, 152);  // Erase number region

      fill(BGcolor + 32);  // A color to highlight any extreme values
      if(y == hotLowFrame || y == hotHighFrame)
        rect(numbersX, numbersY + 95, 80, 16);
      if(y == targLowFrame || y == targHighFrame)
        rect(numbersX, numbersY + 115, 80, 16);
      if(y == coldLowFrame || y == coldHighFrame)
        rect(numbersX, numbersY + 135, 80, 16);

      textSize(10);
      fill(255);
      text(filenames[i], numbersX + 5, numbersY + 40); // Show current filename

      if(celsiusFlag)
        text("Frame\n\n\nElapsed sec\n\nDegrees C", numbersX + 5, numbersY + 8);
      else
```

```
        text("Frame\n\n\nElapsed sec\n\nDegrees F", numbersX + 5, numbersY + 8);

      textSize(15);
      text(i, numbersX + 5, numbersY + 25);                          // Print frame
number
      text(float(y - earlyFrame) * 0.001, numbersX, numbersY + 74);  // Print
elapsed time

      if(celsiusFlag) {      // Print temps in Celsius
        fill(255, 128, 64);
        text(fixedPoint[4], numbersX, numbersY + 108);
        fill(255, 200, 64);
        text(fixedPoint[2], numbersX, numbersY + 128);
        fill(128, 128, 255);
        text(fixedPoint[0], numbersX, numbersY + 148);
      } else {               // or print them in Farenheit
        fill(255, 128, 64);
        text(fixedPoint[4] * 1.8 + 32.0, numbersX, numbersY + 108);
        fill(255, 200, 64);
        text(fixedPoint[2] * 1.8 + 32.0, numbersX, numbersY + 128);
        fill(128, 128, 255);
        text(fixedPoint[0] * 1.8 + 32.0, numbersX, numbersY + 148);
      }

      for(x = 0; x &lt; 768; ++x)
        ++histogram[b[54 + 3 * x] &amp; 0xFF];  // Count all colors
      framX2 = histogram[0];
      for(x = 1; x &lt; 256; ++x) {            // Find most numerous color
        if(histogram[x] &gt; framX2) {
          framX2 = histogram[x];
          targY2 = x;
        }
      }

      fill(BGcolor);
      rect(histoX, histoY - 140, histoW, histoH + 1);  // Erase histogram region

      for(y = 0; y &lt; 256; ++y) {
        if(histogram[y] &gt; 0) {
          tempY = float(y) * (fixedPoint[3] - fixedPoint[1]) / 255.0 +
fixedPoint[1];    // Convert a 8-bit value to a temperature
          tempY = float(histoH) * (coldestLow - tempY) / (coldestLow -
hottestHigh);     // Position it on the graph Y axis
          stroke(colorPal[4 * y + 2] &amp; 0xFF, colorPal[4 * y + 1] &amp; 0xFF,
colorPal[4 * y + 0] &amp; 0xFF);  // Color map the stroke
          line(histoX, histoY - int(tempY), histoX + (histoW - 1) * histogram[y] /
framX2, histoY - int(tempY)); // Draw a line proportional to the pixel count
        }

        noStroke();
        noSmooth();
        textSize(10);
        if(targY2 &lt; 0x80) // Histogram peak in the dark side?
          fill(255);      // Set contrasting test to white
        else
          fill(0);

        tempY = float(targY2) * (fixedPoint[3] - fixedPoint[1]) / 255.0 +
fixedPoint[1];    // Convert a 8-bit value to a temperature
        if(celsiusFlag)    // Print the Y-positioned float value in C?
          text(tempY, histoX, histoY + 3 - int(float(histoH) * (coldestLow -
tempY) / (coldestLow - hottestHigh)));
        else
          text(tempY * 1.8 + 32.0, histoX, histoY + 3 - int(float(histoH) *
(coldestLow - tempY) / (coldestLow - hottestHigh)));
      }
      saveFrame("mov#####.jpg");  // Save the image into a sequence for Movie Maker
    }
    ++i;
```

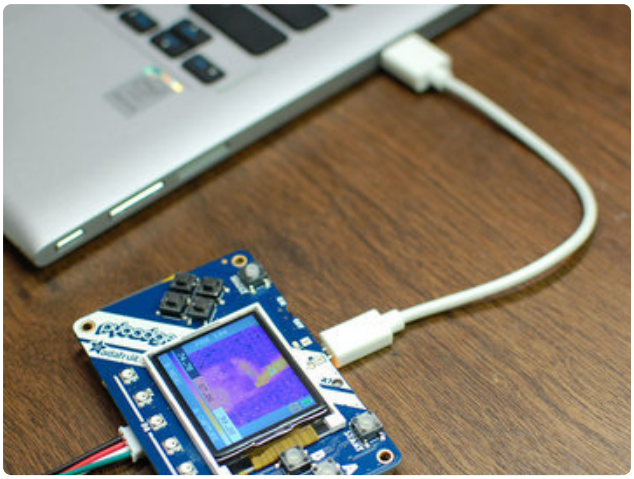
```
  }
}

void enlarge8bitColor() {  // Convert a small gray BMP array and plot an enlarged
colormapped version
  int x, y;

  if(lirpSmoothing) {           // Bilinear interpolation?
    for(y = 0; y &lt; 23; ++y) {   // Count the source pixels less one
      for(x = 0; x &lt; 31; ++x) {
        for(int yLirp = 0; yLirp &lt; 9; ++yLirp) {
          int corner0 = b[54 + ((32 * y + x) + 32) * 3] &amp; 0xFF;
          int corner1 = b[54 + ((32 * y + x) +  0) * 3] &amp; 0xFF;
          int pixLeft  = (corner0 * yLirp + corner1 * (8 - yLirp)) &gt;&gt; 3;  //
Lirp 1 endpoint from 2 L pixels,

          int corner2 = b[54 + ((32 * y + x) + 33) * 3] &amp; 0xFF;
          int corner3 = b[54 + ((32 * y + x) +  1) * 3] &amp; 0xFF;
          int pixRight = (corner2 * yLirp + corner3 * (8 - yLirp)) &gt;&gt; 3;  //
and the other from 2 R pixels

          for(int xLirp = 0; xLirp &lt; 9; ++xLirp) {
            int pixMid = (pixRight * xLirp + pixLeft * (8 - xLirp)) &gt;&gt;
3;         // Lirp between lirped endpoints, bilinear interp
            stroke(colorPal[4 * pixMid + 2] &amp; 0xFF, colorPal[4 * pixMid + 1]
&amp; 0xFF, colorPal[4 * pixMid + 0] &amp; 0xFF);
            point(offsetX + 4 + 8 * x + xLirp, offsetY + 188 - (8 * y +
yLirp));  // Draw a pixel, bottom up
          }
        }
      }
    }

    for(y = 0; y &lt; 192; ++y) {   // Pad out the empty side pixels
      stroke(get(offsetX + 4, offsetY + y));
      line(offsetX + 0, offsetY + y, offsetX + 3, offsetY + y);
      stroke(get(offsetX + 252, offsetY + y));
      line(offsetX + 253, offsetY + y, offsetX + 255, offsetY + y);
    }
    for(x = 0; x &lt; 256; ++x) {
      stroke(get(offsetX + x, offsetY + 4));
      line(offsetX + x, offsetY + 0, offsetX + x, offsetY + 3);
      stroke(get(offsetX + x, offsetY + 188));
      line(offsetX + x, offsetY + 189, offsetX + x, offsetY + 191);
    }
  } else {    // Plain square pixels
    noStroke();

    for(y = 0; y &lt; 24; ++y) {   // Count all source pixels
      for(x = 0; x &lt; 32; ++x) {
        int pixMid = b[54 + ((32 * y + x) +  0) * 3] &amp; 0xFF;
        fill(colorPal[4 * pixMid + 2] &amp; 0xFF, colorPal[4 * pixMid + 1] &amp;
0xFF, colorPal[4 * pixMid + 0] &amp; 0xFF);  // Get color from table
        rect(offsetX + 8 * x, offsetY + 8 * (23 - y), 8, 8);  // Draw a pixel,
bottom up
      }
    }
  }

  if(markersVisible) {  // Show the green marker crosses?
    stroke(0, 192, 0);  // Deep green

    y = ((b[2381] &amp; 0xff) &lt;&lt;  8) +  (b[2380] &amp; 0xff);   // Reassemble
16-bit addresses of cold / hot pixels
    line(offsetX + 8 * (y &amp; 31) + 1, offsetY + 188 - 8 * (y &gt;&gt; 5), offsetX
+ 8 * (y &amp; 31) + 7, offsetY + 188 - 8 * (y &gt;&gt; 5));
    line(offsetX + 8 * (y &amp; 31) + 4, offsetY + 185 - 8 * (y &gt;&gt; 5), offsetX
+ 8 * (y &amp; 31) + 4, offsetY + 191 - 8 * (y &gt;&gt; 5));
```

```
    y = ((b[2383] &amp;amp; 0xff) &amp;lt;&amp;lt;  8) +  (b[2382] &amp;amp; 0xff);
    line(offsetX + 8 * (y &amp;amp; 31) + 1, offsetY + 188 - 8 * (y &amp;gt;&amp;gt; 5), offsetX
+ 8 * (y &amp;amp; 31) + 7, offsetY + 188 - 8 * (y &amp;gt;&amp;gt; 5));
    line(offsetX + 8 * (y &amp;amp; 31) + 4, offsetY + 185 - 8 * (y &amp;gt;&amp;gt; 5), offsetX
+ 8 * (y &amp;amp; 31) + 4, offsetY + 191 - 8 * (y &amp;gt;&amp;gt; 5));

    y = 400;
    line(offsetX + 8 * (y &amp;amp; 31) + 1, offsetY + 188 - 8 * (y &amp;gt;&amp;gt; 5), offsetX
+ 8 * (y &amp;amp; 31) + 7, offsetY + 188 - 8 * (y &amp;gt;&amp;gt; 5));
    line(offsetX + 8 * (y &amp;amp; 31) + 4, offsetY + 185 - 8 * (y &amp;gt;&amp;gt; 5), offsetX
+ 8 * (y &amp;amp; 31) + 4, offsetY + 191 - 8 * (y &amp;gt;&amp;gt; 5));
  }
}

void loadColorTable(int choiceNum, int offset) {
  int i, x;

  switch(choiceNum) {
    case 1:     // Load 8-bit BMP color table with computed ironbow curves
      for(x = 0; x &lt; 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (1.02 - (fleX - 0.72) * (fleX - 0.72) * 1.96);
        fleG = (fleG &gt; 255.0) || (fleX &gt; 0.75) ? 255.0 : fleG;  // Truncate
curve
        i = (int)fleG;
        colorPal[offset + x * 4 + 2] = byte(i &amp;amp; 0xFF);    // Red vals

        fleG = fleX * fleX * 255.9;
        i = (int)fleG;
        colorPal[offset + x * 4 + 1] = byte(i &amp;amp; 0xFF);    // Grn vals

        fleG = 255.9 * (14.0 * (fleX * fleX * fleX) - 20.0 * (fleX * fleX) + 7.0 *
fleX);
        fleG = fleG &lt; 0.0 ? 0.0 : fleG;  // Truncate curve
        i = (int)fleG;
        colorPal[offset + x * 4 + 0] = byte(i &amp;amp; 0xFF);    // Blu vals
      }
      break;
    case 2:  // Compute quadratic "firebow" palette
      for(x = 0; x &lt; 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (1.00 - (fleX - 1.0) * (fleX - 1.0));
        i = (int)fleG;
        colorPal[offset + x * 4 + 2] = byte(i &amp;amp; 0xFF);    // Red vals

        fleG = fleX &lt; 0.25 ? 0.0 : (fleX - 0.25) * 1.3333 * 255.9;
        i = (int)fleG;
        colorPal[offset + x * 4 + 1] = byte(i &amp;amp; 0xFF);    // Grn vals

        fleG = fleX &lt; 0.5 ? 0.0 : (fleX - 0.5) * (fleX - 0.5) * 1023.9;
        i = (int)fleG;
        colorPal[offset + x * 4 + 0] = byte(i &amp;amp; 0xFF);    // Blu vals
      }
      break;
    case 3:  // Compute "alarm" palette
      for(x = 0; x &lt; 256; ++x) {
        float fleX = (float)x / 255.0;

        float fleG = 255.9 * (fleX &lt; 0.875 ? 1.00 - (fleX * 1.1428) : 1.0);
        i = (int)fleG;
        colorPal[offset + x * 4 + 2] = byte(i &amp;amp; 0xFF);    // Red vals

        fleG = 255.9 * (fleX &lt; 0.875 ? 1.00 - (fleX * 1.1428) : (fleX - 0.875) *
8.0);
        i = (int)fleG;
        colorPal[offset + x * 4 + 1] = byte(i &amp;amp; 0xFF);    // Grn vals
```

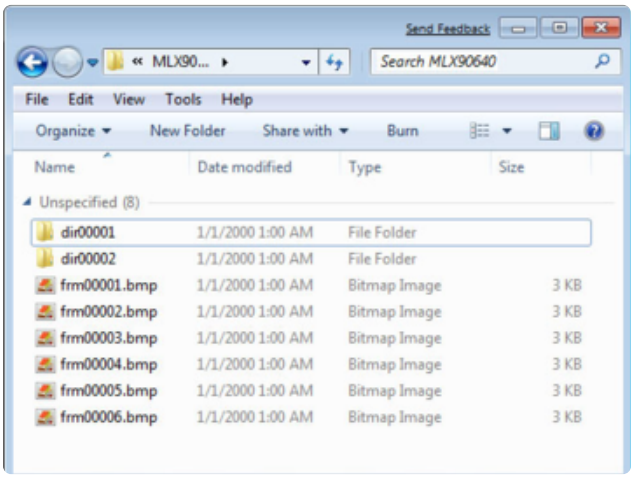
```
        fleG = 255.9 * (fleX &lt; 0.875 ? 1.00 - (fleX * 1.1428) : 0.0);
        i = (int)fleG;
        colorPal[offset + x * 4 + 0] = byte(i &amp; 0xFF);    // Blu vals
      }
      break;
    case 4:    // Grayscale, black hot
      for(x = 0; x &lt; 256; ++x) {
        colorPal[offset + x * 4 + 2] = byte(255 - x &amp; 0xFF);    // Red vals
        colorPal[offset + x * 4 + 1] = byte(255 - x &amp; 0xFF);    // Grn vals
        colorPal[offset + x * 4 + 0] = byte(255 - x &amp; 0xFF);    // Blu vals
      }
      break;
    default:    // Grayscale, white hot
      for(x = 0; x &lt; 256; ++x) {
        colorPal[offset + x * 4 + 2] = byte(x &amp; 0xFF);    // Red vals
        colorPal[offset + x * 4 + 1] = byte(x &amp; 0xFF);    // Grn vals
        colorPal[offset + x * 4 + 0] = byte(x &amp; 0xFF);    // Blu vals
      }
  }
}

// Rebuild a float from a fixed point decimal value encoded in 4 bytes
float expandFloat(byte m1, byte m2, byte e1, byte e2) {
  int fracPart;
  float floatPart;

  fracPart = ((e2 &amp; 0xff) &lt;&lt; 8) + (e1 &amp; 0xff);   // Reassemble 16-bit
value
  floatPart = (float)fracPart / 49152.0;         // Convert into fractional portion
of float
  fracPart = ((m2 &amp; 0xff) &lt;&lt; 8) + (m1 &amp; 0xff);   // Reassemble 16-bit
value
  return ((float)fracPart + floatPart) - 1000.0; // Complete reconstructing
original float
}

String[] listFileNames(String dir) {   // Return the filenames from a directory as
an array of Strings
  File file = new File(dir);

  if (file.isDirectory()) {
    String names[] = file.list();
    return names;
  } else    // It's not a directory
    return null;
}
```

Almost there!  Our next step is to bring in the thermal image data from the camera so we can do cool things with it.

# Getting the Pictures

The image data captured by your camera is written in flash storage, either in a removable microSD card, or in its built-in QSPI flash memory. A microSD card can be removed and accessed with a standard microSD reader. The onboard QSPI flash should appear as an external drive to a regular computer, when the board is connected by USB. Either way, go ahead and plug your device into the computer where your Processing code resides and get ready to copy some data.



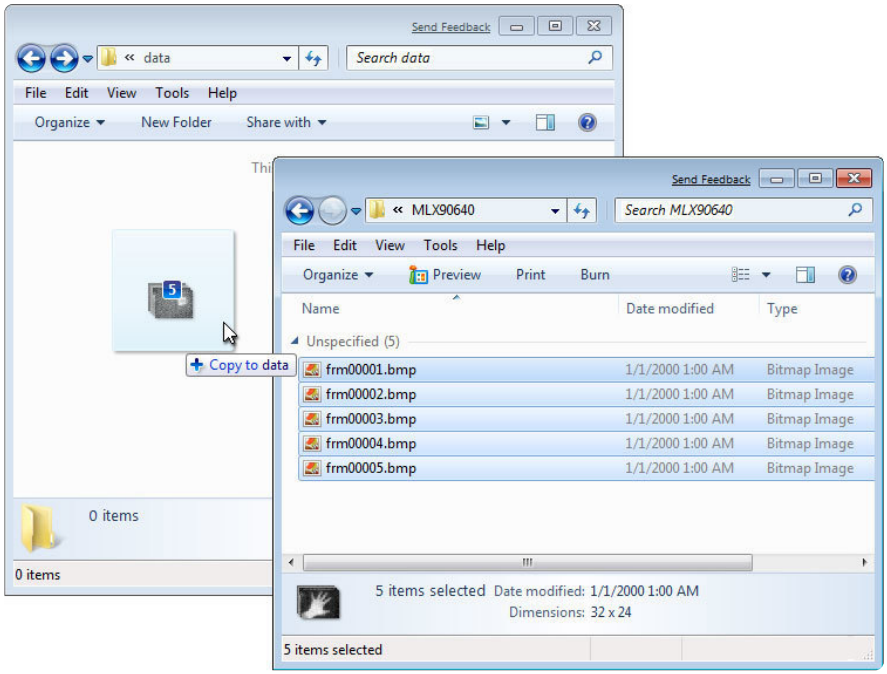

As you begin exploring the flash drive, you should see a folder in the root directory named MLX90640. This is where the camera sketch stores all its files. Open it and have a look.

Two things you might find in this folder are BMP files and more folders, all with names containing serial numbers. The folders should contain even more numbered BMP files. The BMP files are what we'll be copying and pasting, one group at a time. Any sequence of images is a good choice to start, either in the MLX90640 folder or in one of the dir##### folders. Decide on a group.

Since the camera sketch is active, it can capture more BMP images while connected, but the Arcada functions prevent new captures from being accessed while running. That's normal, and it prevents errors. Just restart the camera sketch to open your latest images.

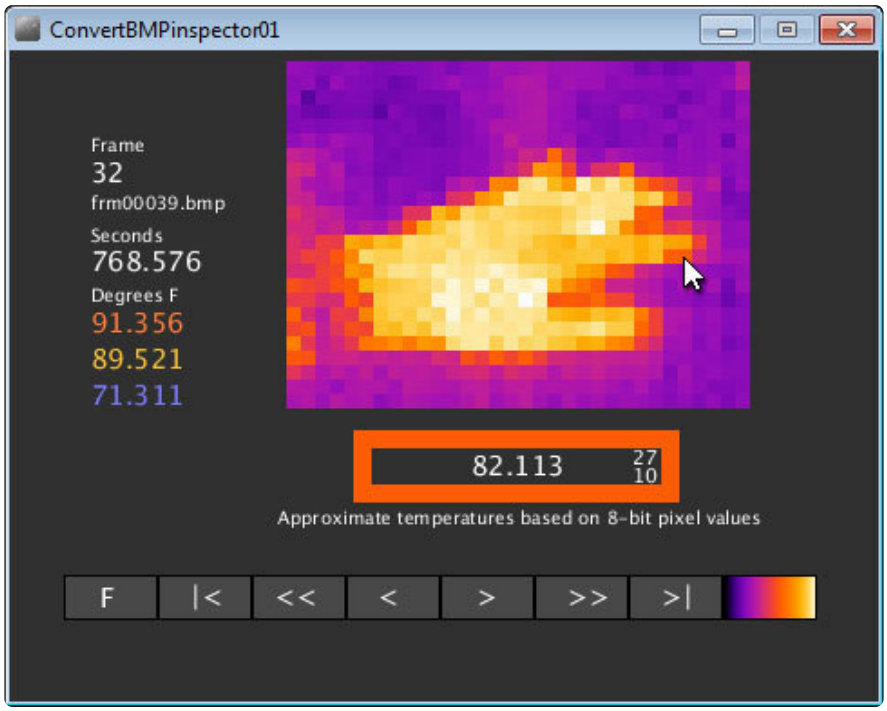
Now open the processing sketch called ConvertBMPinspector01, but don't run it (yet). Instead, open its home folder from the menu by selecting Sketch>Show sketch folder. Your operating system should present a file management window showing the filename of the sketch itself. Next to that, you will need to create another folder named data, if none already exists. Open that data folder, because that's where your BMP images should go. It's a simple matter of copy/paste, and a set of numbered files all named frm#####.bmp should be the only things pasted into the data folder.

(These steps will happen before using the other two Processing sketches, too.)




## Using ConvertBMPinspector01

Go ahead and run the ConvertBMPinspector01 sketch. You should see a window open showing some numeric values, a thermal image magnified and in color, and some control buttons beneath.




The screen is now yours to explore. The statistics on the left show file and temperature details like the filename, the frame count in a sequence, a time stamp showing elapsed seconds of runtime at capture, and three exact temperatures detected by the sensor (hottest pixel, coldest, and center pixel [16, 12]).

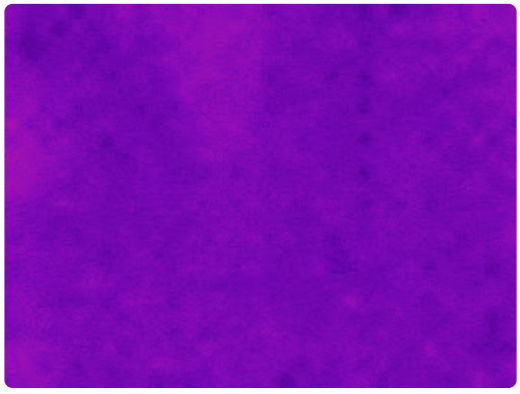
With the pointer you can hover over the image itself and make the color and approximate temperature of any pixel appear in a rectangular frame below. The XY location of the pixel is shown as well. Whereas the left-side temperatures are shown exactly as measured, the interactive pixel temperatures are approximate, since they're recomputed from 8-bit color values. You can see this by comparing pixel[16,12]. The temperatures returned are not the same, but pretty close.

Sometimes images are captured using a preset color range, but actual temps can fall outside that range. The display will indicate colors that are maxed out (or "minned" out) with > or < symbols. The left-side numbers are not affected by this.

The control buttons at the bottom are pretty straightforward. The leftmost toggles between temperatures displayed in Fahrenheit or Celsius. The rightmost will cycle between several false-color palettes to enhance detail visibility. The buttons in between are for stepping through a sequence of image files in small or large steps. The keyboard can manage these controls as well, using the **,** **.** **<** **>** **m** / keys. (The sketch can be used to inspect a single image, but a sequence of images is more fun.)

# Putting It in Motion

Recording a sequence of thermal images over time means time-based analyses are possible, but the fun part is seeing your pictures **move**! Let's convert a sequence of BMP images into a colorful motion GIF like this one.

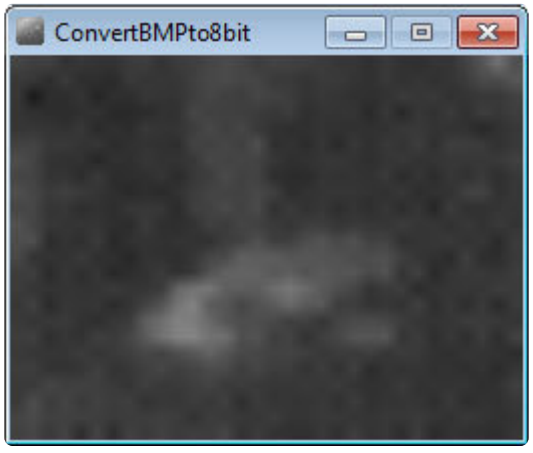


## Using ConvertBMPto8bit

This sketch will begin just like the last one. Open the sketch in Processing but don't run it. Use Sketch>Show sketch folder to open the home folder, and either locate or create a data folder inside it. Choose a sequence of BMP images from your camera's flash storage and copy/paste it into the data folder. Then run ConvertBMPto8bit.

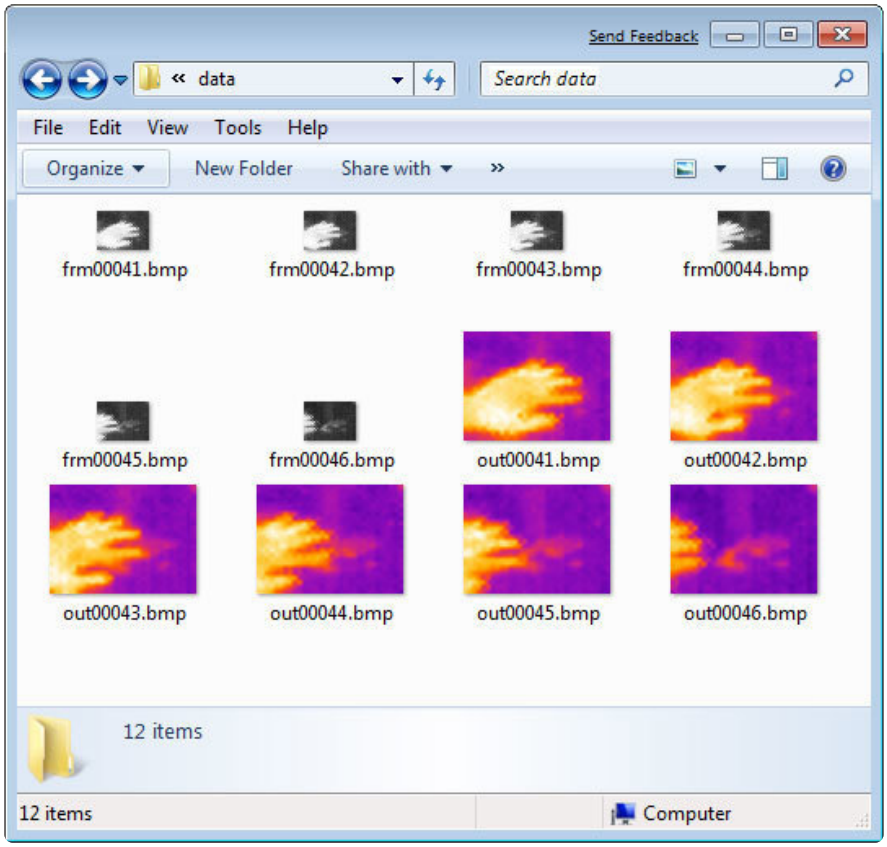
Did anything happen? Some text appeared in the output part of the main window at the bottom, then one gray thermal image appeared in a mini window, and nothing else changed after that. Did it break?

Nope. That's exactly what's supposed to happen. You can stop the sketch.




Here's what the sketch does. It reads each tiny gray BMP image, enlarges and smooths the pixels with bilinear interpolation, colors the image from a palette of 256,

and saves it in the data folder as a new 8-bit BMP file with the new color scheme. The folder should now contain twice as many BMP files as before.



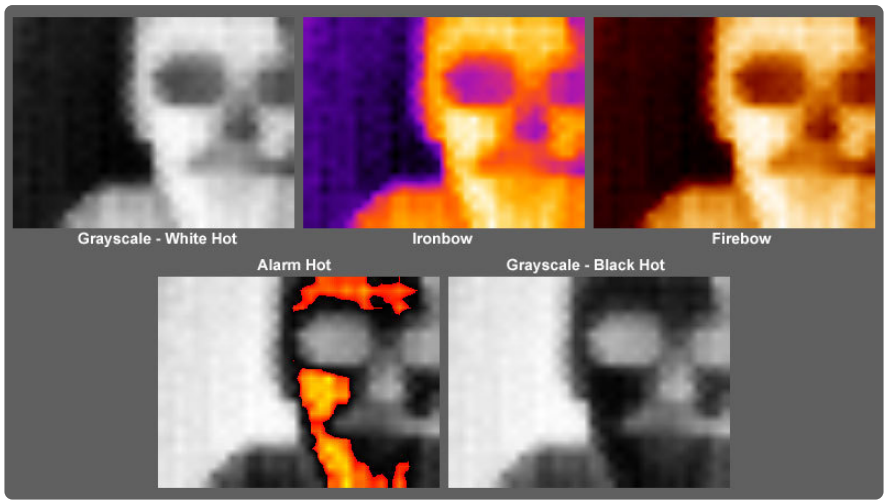

## Modifying the Colors

Have a look at some of those new BMP images. If the color palette doesn't work for you, ConvertBMPto8bit has other options. Look for these lines in the Processing window.

```
// "paletteChoice" selects a false color palette:
// 0 == Grayscale, white hot
// 1 == Ironbow
// 2 == Firebow
// 3 == Hot alarm
// 4 == Grayscale, black hot
int paletteChoice = 1;
```

The last line sets the variable `paletteChoice` to 1, but four other options are available. Two are grayscales, one is a dark-to-light gradient in warm tones, and "Hot alarm" is a light-to-dark grayscale that switches to a red-yellow highlight at the top, emphasizing the hottest colors. Try different values for the variable and see which colors you prefer.

Coders familiar with Processing can even modify the function named `loadColorTable()` and generate their own custom palettes.




Once you're happy with your new BMP images, they can be uploaded to an online GIF creating site like ezgif.com (https://adafru.it/JDa), where they can be assembled into a motion GIF, and tweaked for size, speed, compression, and special effects. Then you can send or post your hot motion images. Dazzle your friends with your thermal fire dance moves!

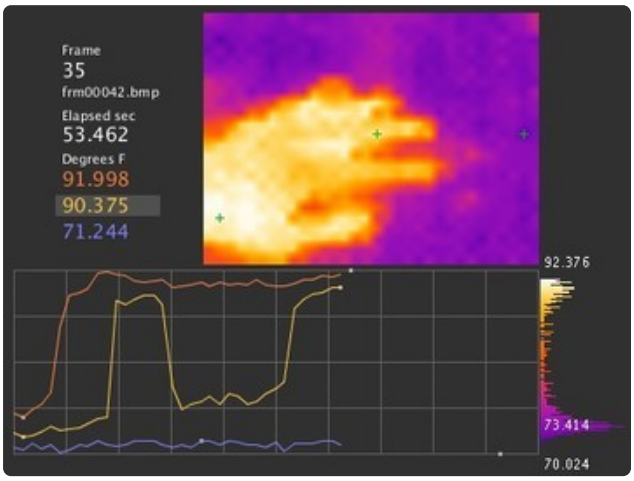
## Using ConvertBMPtoSeq01

Here's where your thermal images get all science-y. Creating time-based graphs and statistics in colorful motion, ready for detailed examination, it makes me feel like Mister Spock.

You know the drill by now, right? Open the sketch and paste your downloaded BMPs into its data folder before running. You got it. This time the images flash by in the graphical window as the sketch does its job. What's it doing? It's enlarging and coloring the images as before, then placing numbers and graphs around them for added coolness.

This sketch, unlike the others, can't process just one BMP file. Two is the minimum. Also, random files that aren't from a recorded sequence don't turn out well. You can try them, but they produce tangled graphs.

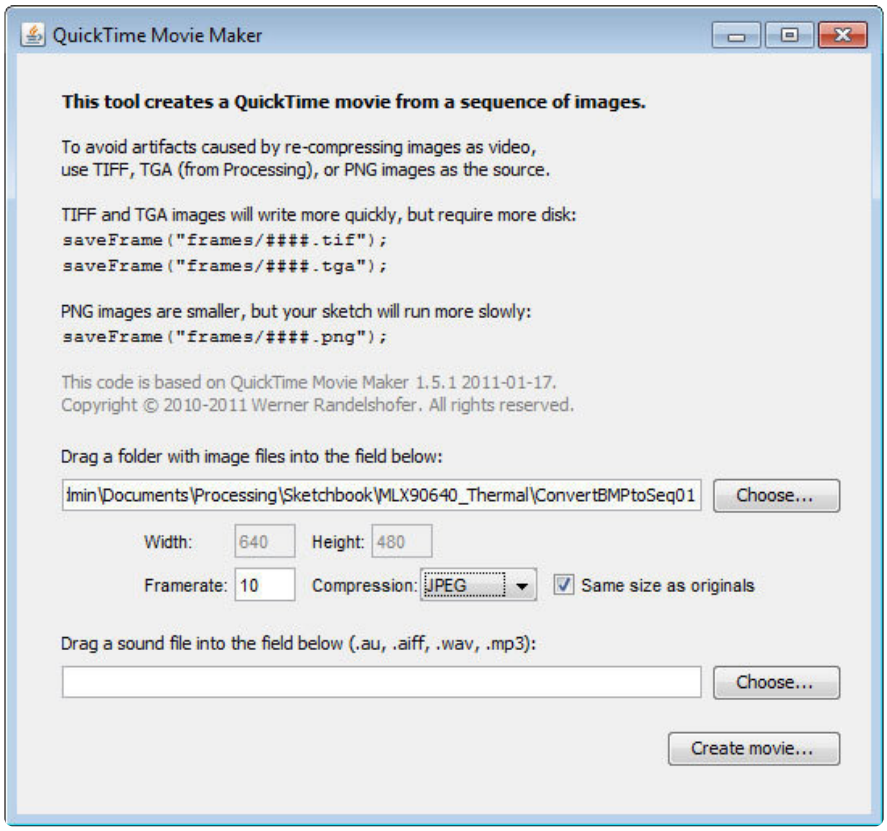
As each image appears, it gets saved as a JPG file in the sketch folder, (not in the data folder but alongside it). The sequence of source BMPs yields a sequence of JPGs. The sketch freezes at the end of the sequence with the last image visible. You may then stop the sketch (but don't close it yet).




Open one of the mov#####.jpg files in an image viewer and look it over. The lines and colors carry real information about the temperatures in your pictures, but let's make them into a QuickTime movie next, and take a closer look afterward.

## Images into Movies

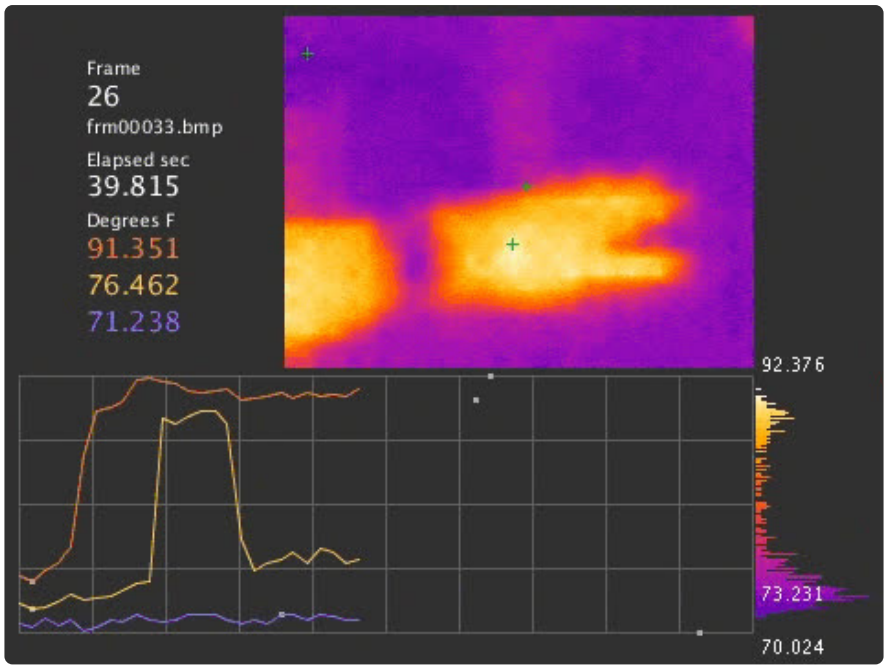
The Processing language has a QuickTime movie maker in its collection of tools (https://adafru.it/JDb). You can open it by finding it under the Tools menu. This brings up an interface window. Here's one that shows the settings I used for the handprint movie.




The directions don't recommend using a JPG sequence as a source for creating QuickTimes, but I do so anyway, and I find the quality acceptable. You can modify the sketch to output TIFF or PNG images if you want. You be the judge.

In the interface box there's a text field where the path of the folder holding the JPG images is selected. The path to your sketch folder is what goes there, and you can use the Choose... button to find it. The other options are yours to try. See what works best for you.

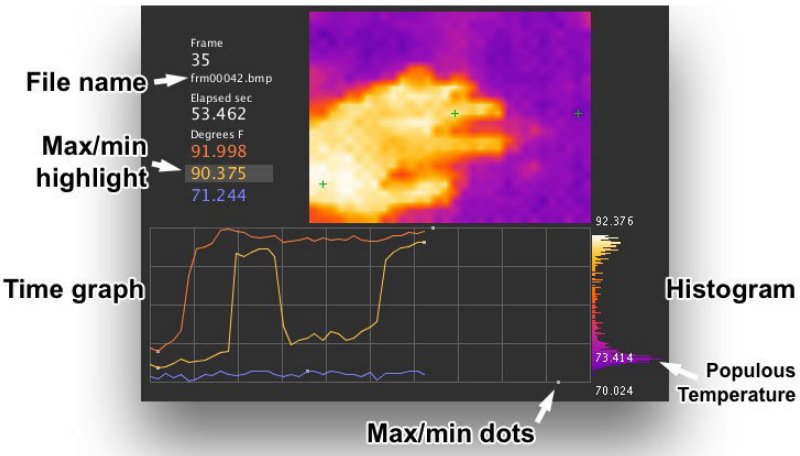
When you're ready, click Create movie..., enter a name for the file, and watch it work. (The movie you create may end up with a TMP file alongside it. It's no longer needed, and you can trash it.) When finished, a QuickTime player can open your new movie and show it in glorious motion. You're a thermal cinematographer!




## Digging the Data

A QuickTime player can easily present the movie one frame at a time for closer inspection. This is where those extra bytes added to each recorded BMP deliver their information. Pause your movie on an interesting frame and take another look.

There's the colorized image to the right of the column of numeric values, and below them are two graphs, a basic value over time graph and a histogram.




The three number values colored red/gold/blue show our three favorite temperatures, the hottest/center/coldest for each frame. The graph shows how those values were changing over time. Six small dots mark the highest and lowest points on each line, and a brief gray box will highlight the number at each moment when a dot is reached. This makes finding the hottest or coldest thing in your movie easy to do.

The histogram shows the population count of the 8-bit pixel values in the image, and the most populous pixel value gets converted back into a temperature and overprinted on the graph. The whole thing is auto-scaled so no values go off the charts. (Saying that readings are "off the charts" sounds so dramatic. Sorry about disallowing this.)

## Modify Your Movies

ConvertBMPtoSeq01 has a small chunk of code near the beginning much like the one described earlier. It allows different false-color palettes to be used for color output, just like ConvertBMPto8bit.

```
// Change the following values to customize the output images.
// "paletteChoice" selects a false color palette:
// 0 == Grayscale, white hot
// 1 == Ironbow
// 2 == Firebow
// 3 == Hot alarm
// 4 == Grayscale, black hot
int paletteChoice = 1;
boolean markersVisible = true, celsiusFlag = false, lirpSmoothing = true;
```

The three Boolean values that come next can also modify your output. The options can hide the green crosses marking the extreme pixels, display the temperatures in Celsius, and disable the bilinear smoothing of the image pixels.

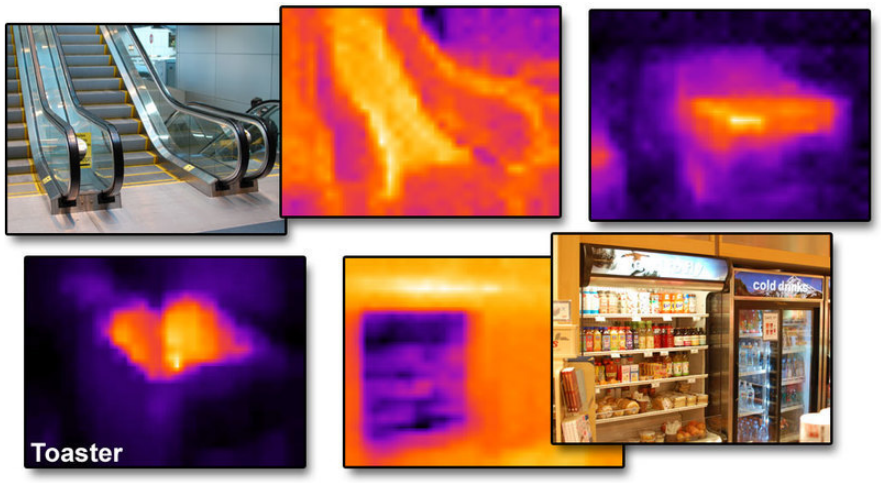
Above this code block are even more predefined values you could change, values that control the positions of various elements in the images. You could define your own arrangement of the onscreen data display, if you wish. The world of thermal movies is yours to explore!

---

# Where Do We Go from Here?

This project has already provided me with hours of fun and learning, but it's just a basic tool, waiting for creative people to expand it. What can it become?

I can tell you my plans. I plan to add in-camera pixel smoothing, like the Processing sketches offer. I plan to store the camera settings in a config file, and allow it to start up with its last settings remembered. I plan to learn how to control the emissivity value to sense more accurate temperatures from surfaces that don't radiate heat well, like thin sheets and shiny metals. I plan to look into the Adafruit libraries for Arcada devices and for the MLX90640, and see if I can speed up some of the camera functions. In short, my plans are to improve what's already working.




What could you plan? Could you add an ESP32 FeatherWing (http://adafru.it/4264) and make a wireless thermal webcam? Could you send image data over USB serial to a regular computer for a big-screen thermal cam as a science exhibit (https://adafru.it/JDd), or a thermal photo booth? A thermal camera security sentry, or baby guardian? The creativity of the many outweighs the creativity of the one.

Thanks for taking time to explore my little project, and if you do build your own thermal camera and begin capturing interesting images and movies, come to Show and Tell (https://adafru.it/DX7) and share what you've done. Inspire others! Have a grand adventure.

Hallelujah!