MLX90640 Quick Start — Detailed Write-Up

Mukanov Madiyar

February 2024

Contents

1 Overview

The **Melexis MLX90640** is a **32 × 24** (768 px) long-wave infrared (LWIR) thermal array with a digital **I²C** interface. It outputs an absolute temperature per pixel (°C) using on-board calibration data. It is commonly available as a breakout module with two optics variants:

- **D55** — ∼55° field-of-view (narrower, higher angular resolution).
- **D110** — ∼110° field-of-view (wider, covers more scene).

Typical effective frame rates in hobby libraries: ∼0.5–32 Hz (bus speed and host CPU dependent). The default 7-bit I²C address is `0x33`.

When to use MLX90640

- Thermal presence detection and people/animal localization.
- Low-resolution thermography for HVAC, electronics, or classroom demonstrations.
- Edge prototypes for safety/telehealth where rough temperature maps suffice.

Key properties (quick facts)

- **Array**: 32 × 24 (768 values per frame).
- **Interface**: I²C @ 100 kHz, 400 kHz; many hosts support up to 1 MHz for smoother refresh.
- **Logic/power**: 3.3 V logic; typical breakout powered from 3.3 V (check your board).
- **Modes**: "Chess" and "Interleaved" subpageing; library assembles full frames.
- **Units**: Outputs real temperatures (°C) after calibration.

2 Bill of Materials (BOM)

Item	Notes	Qty
MLX90640 breakout (D55 or D110)	Choose FOV per use-case.	1
Host MCU/SBC (Raspberry Pi / ESP32 / RP2040 / SAMD)	3.3 V logic recommended.	1
Jumper wires (SDA, SCL, 3V3, GND)	Keep short; twisted pairs help in noisy envs.	4
Optional: Level shifter	Needed if host I/O is 5 V only.	1
Optional: Small standoffs	Thermal isolation from warm PCBs/enclosures.	4

3 Electrical and Thermal Considerations

Power and logic levels

Most MLX90640 breakouts are 3.3 V devices. If using a 5 V microcontroller (e.g., classic Arduino Uno), add a **bi-directional I²C level shifter** on SDA/SCL. Verify whether your breakout already includes one.

Pull-ups and wiring

I^{2}C requires SDA/SCL pull-ups (often present on the breakout). Keep wires short, route away from motors/inductors, and avoid heat sources behind the sensor (board self-heating affects readings).

Thermal environment

Let the board **thermally settle** (tens of seconds) after power-up. Avoid touching the metal can. Highly reflective/shiny targets have low emissivity and read too low — use a piece of matte tape as a reference patch.

4 Pinout and Wiring

4.1 Common connections

VIN	3.3 V supply (some boards accept 5 V — check your board)
GND	Ground
SCL	I^{2}C clock
SDA	I^{2}C data

4.2 Raspberry Pi (3.3 V logic)

- VIN → 3V3, GND → GND, SCL → GPIO3 (SCL), SDA → GPIO2 (SDA).
- Enable I^{2}C (`raspi-config` → Interfaces).
- Verify with `i2cdetect -y 1` ⇒ `0x33` should appear.

4.3 ESP32 / RP2040 / SAMD

- Any I^{2}C-capable pins; set pin numbers in code.
- Start at 400 kHz; if wiring is solid, ∼1 MHz often works for higher FPS.

4.4 5 V Arduinos

- Prefer a 3.3 V board. Otherwise **use a proper level shifter** on SDA/SCL.

5 I^{2}C Bus Tuning (throughput vs. stability)

- Increase bus speed: 100 kHz → 400 kHz; advanced users may try 1 MHz.
- Reduce per-frame processing (interpolation, heavy plotting) inside the acquisition loop.
- Fix the color scale (e.g. 20–40 °C) for stable visuals; use autoscale only for exploration.

6 Python Quick Start (Raspberry Pi / Linux)

6.1 Install dependencies

```
python3 -m pip install --upgrade pip
python3 -m pip install adafruit-blinka adafruit-circuitpython-mlx90640 numpy
    matplotlib
# Enable I2C on your system and ensure the device shows up at 0x33:
# sudo apt install -y i2c-tools
# i2cdetect -y 1
```

6.2 Minimal live heatmap (Matplotlib)

Listing 1: Live heatmap with auto-scaling (set fixed vmin/vmax for consistent viewing).

```
import time, board, busio
import numpy as np
import matplotlib.pyplot as plt
import adafruit_mlx90640 as mlx

# I2C at 400kHz; try 1_000_000 for higher throughput if your setup allows
i2c = busio.I2C(board.SCL, board.SDA, frequency=400_000)
cam = mlx.MLX90640(i2c)

# Choose a refresh rate your host can handle (0.5..64 Hz enumerations may exist)
cam.refresh_rate = mlx.RefreshRate.REFRESH_8_HZ

W, H = 32, 24
frame = [0.0] * (W * H)

plt.ion()
fig = plt.figure(figsize=(6, 4))
im = plt.imshow(np.zeros((H, W)), interpolation="nearest",
                vmin=20, vmax=40) # fix window for stability
plt.colorbar(label="Temperature (C)")

while True:
    try:
        cam.getFrame(frame) # fills 'frame' with 768 float temps in C
        data = np.array(frame).reshape(H, W) # shape to 2D
        im.set_data(data)
        # Optional: im.set_clim(vmin=data.min(), vmax=data.max()) # auto-window
        plt.pause(0.001)
    except KeyboardInterrupt:
        break
```

6.3 Practical examples (Python)

A) Save frames to CSV for logging/telemetry

```
import csv, time
# After 'cam' init ...
with open("mlx90640_log.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["t_epoch"] + [f"p{i}" for i in range(32*24)])
    for _ in range(100): # capture 100 frames
        cam.getFrame(frame)
        writer.writerow([time.time()] + frame)
        time.sleep(0.05) # throttle (~20 Hz target)
```

B) Hotspot detection (max-pixel location)

```
import numpy as np
cam.getFrame(frame)
arr = np.array(frame).reshape(24, 32)
idx = np.argmax(arr)
y, x = divmod(idx, 32)
```

```
print(f"Hotspot at (x={x}, y={y}) = {arr[y, x]:.2f} C")
```

C) Orientation fixes (rotate/flip to match your lens mount)

```
# Rotate 90 clockwise and flip horizontally if your mounting needs it:
arr = np.rot90(arr, k=-1) # -1 means CW; +1 would be CCW
arr = np.fliplr(arr) # left-right flip
```

7 Arduino / C++ Quick Start

7.1 Install libraries

Use Library Manager to install:

- `Adafruit MLX90640`
- `Adafruit BusIO`

7.2 Minimal read and basic telemetry

Listing 2: Read full frame and print a few stats at 115200 baud.

```
#include <Wire.h>
#include <Adafruit_MLX90640.h>

Adafruit_MLX90640 mlx;
float frame[32*24]; // 768 pixels

void setup() {
  Serial.begin(115200);
  Wire.begin();
  Wire.setClock(400000); // I2C @ 400 kHz

  if (!mlx.begin(0x33, &Wire)) {
    Serial.println("MLX90640 not found at 0x33!");
    while (1) delay(10);
  }
  mlx.setMode(MLX90640_INTERLEAVED); // or MLX90640_CHESS
  mlx.setResolution(MLX90640_ADC_18BIT);
  mlx.setRefreshRate(MLX90640_8_HZ); // pick a stable rate for your board
}

void loop() {
  if (mlx.getFrame(frame)) {
    float tmin = frame[0], tmax = frame[0];
    for (int i = 1; i < 32*24; ++i) {
      if (frame[i] < tmin) tmin = frame[i];
      if (frame[i] > tmax) tmax = frame[i];
    }
    Serial.print("Min: "); Serial.print(tmin, 2);
    Serial.print(" Max: "); Serial.println(tmax, 2);
  }
}
```

7.3 Practical examples (Arduino)

A) Hotspot pixel coordinates

```
int hotspotX = 0, hotspotY = 0;
float hotspotT = -1000.0;

if (mlx.getFrame(frame)) {
  for (int y = 0; y < 24; ++y) {
    for (int x = 0; x < 32; ++x) {
      float t = frame[y*32 + x];
      if (t > hotspotT) {
        hotspotT = t; hotspotX = x; hotspotY = y;
      }
    }
  }
  Serial.print("Hotspot @("); Serial.print(hotspotX);
  Serial.print(", "); Serial.print(hotspotY);
  Serial.print(") = "); Serial.print(hotspotT, 2); Serial.println(" C");
}
```

B) Fixed windowing (map to 0..255 for serial/LED gradients)

```
float vmin = 20.0, vmax = 40.0; // indoor window
uint8_t pixels8[32*24];

if (mlx.getFrame(frame)) {
  for (int i = 0; i < 32*24; ++i) {
    float t = frame[i];
    float n = (t - vmin) / (vmax - vmin); // 0..1
    if (n < 0) n = 0; if (n > 1) n = 1;
    pixels8[i] = (uint8_t)(n * 255.0f + 0.5f);
  }
  // send pixels8 over Serial / SPI display, etc.
}
```

8 Data Flow and Modes (Under the Hood)

MLX90640 uses two subpages to assemble a full frame. Libraries abstract this into a single 32×24 array. You can usually choose between:

- **Interleaved**: alternating pixel groups per subframe
- **Chess**: checkerboard sampling pattern

Higher refresh rates require higher I^2C throughput and more CPU. If frames appear noisy or stale, reduce the refresh rate or increase bus speed.

9 Emissivity, Reflections, and Accuracy

9.1 Why shiny objects read "too cold"

Thermal cameras assume a default emissivity (often around 0.95). Shiny metals have low emissivity and reflect ambient IR, leading to underestimated temperatures. Use a small piece of matte electrical tape as a reference spot to get a reliable reading.

9.2 Tips

- Avoid direct view of very hot/cold reflective surfaces.
- Let the sensor **warm up and settle**.
- If your library exposes an emissivity parameter, set it to match your target; otherwise, rely on reference patches.

10 Performance Tuning Checklist

- **I²C speed**: use 400 kHz or higher if wiring permits.
- **Refresh rate**: start at 4–8 Hz; increase slowly while watching CPU load/dropped frames.
- **Processing**: avoid heavy interpolation inside the capture loop; render first, post-process later.
- **Thermal isolation**: mount on standoffs; avoid hot SBC components behind the sensor.
- **Visualization**: fix color window (e.g. 20–40 °C); only autoscale during exploration.

11 Troubleshooting

I²C scan shows nothing

- Check 3.3 V power, ground, SDA/SCL pins, and cable continuity.
- Ensure I²C enabled in OS/firmware; confirm pull-ups.
- Verify address `0x33` is not blocked by another device.

NaN/garbled temperatures

- Refresh rate too high for current bus speed/CPU.
- Incomplete subpage reads — slow the rate or increase I²C clock.

All pixels shifted/rotated

- Correct for lens/mount orientation in software (rotate/flip).

Looks “too hot” or “too cold”

- Allow settle time; consider emissivity effects and reflections.
- Use a matte-tape reference spot.

12 Verification and Bench Tests

Fast sanity checks

1. Point at your palm/forehead: expect ∼32–36 °C on the hottest pixel cluster.
2. Cup a warm mug: watch gradient and hotspot move with the mug.
3. Open a window: verify cooler background; observe overall histogram shift.

Repeatability

Capture 100 frames from a static scene and print min/max/std. dev. to gauge noise; verify no sudden spikes (indicative of bus errors or subpage desync).

13 Raspberry Pi Notes

Enable and tune I²C

```
sudo raspi-config # Interface Options -> I2C -> Enable
sudo apt install -y i2c-tools
i2cdetect -y 1 # expect 0x33

# Optional: increase I2C baudrate by adding to /boot/config.txt
# dtparam=i2c_arm=on,i2c_arm_baudrate=400000
```

14 FAQ

Q: Can I run at >32 Hz? Pragmatically, the array size + I²C bandwidth limits you. Some hosts manage higher effective rates with fast I²C and tight loops, but expect diminishing returns.

Q: Can I get absolute accuracy like a spot IR thermometer? The MLX90640 is a low-resolution imager; expect decent relative accuracy and good repeatability after warm-up. For critical single-spot accuracy, use a calibrated IR thermometer or validate with reference patches.

Q: How do I stabilize visuals for demos? Fix the color window (e.g. 20–40 °C), avoid autoscale, and mount the board away from heat sources.

15 Appendix A — Glossary

- **Emissivity**: surface effectiveness in emitting IR compared to a perfect blackbody.
- **I²C**: two-wire digital bus (SDA, SCL) with pull-up resistors.
- **Subpage**: half-frame sample that the library merges into a full 32×24 frame.

16 Appendix B — Minimal API Cheatsheet

Python (CircuitPython driver)

`mlx.MLX90640(i2c)`	Create camera instance.
`cam.refresh_rate = ...`	Select enum per library (e.g. 4 Hz, 8 Hz).
`cam.getFrame(list)`	Fill list of 768 floats (°C).

Arduino (Adafruit driver)

`mlx.begin(0x33, &Wire)`	Init, returns bool.
`mlx.setMode(...)`	`MLX90640_INTERLEAVED` or `MLX90640_CHESS`.
`mlx.setRefreshRate(...)`	e.g. `MLX90640_8_HZ`.
`mlx.getFrame(float* f)`	True if frame captured; 768 temps in °C.