

Here's a clear comparison between "OV cam" (typically OmniVision or similar MIPI camera modules) and a **USB camera** — what they are, how they differ, and where each makes sense.

📸 What They Are

OV Cam

- Short for a camera using an **OmniVision (OVxx)** sensor or similar module.
- Usually interfaces via **MIPI CSI-2 or parallel interfaces**.
- Found in embedded systems (Raspberry Pi, Jetson, smartphones, robotics).
- Requires a host board with the proper **camera interface connector**.

USB Cam

- A **plug-and-play** camera that communicates over **USB (UVC standard)**.
 - Works with almost any computer, SBC, or device with a USB port.
 - The interface handles both communication and power.
-

⚖️ Key Differences

Feature	OV Cam (MIPI / CSI)	USB Cam
Connection	Direct camera interface (CSI or parallel)	USB port
Plug-and-play?	✗ Not automatically recognized	✓ Generally automatic (UVC)
Power	Separate power (often 3.3V)	Powered over USB
Latency	Lower (direct link, less overhead)	Higher (USB stack overhead)
Bandwidth	Higher (up to many Gbps on CSI)	Limited by USB version (USB2 ~480Mbps; USB3 ~5Gbps)
CPU Load	Lower (DMA, direct memory access)	Higher (USB protocol processing)
Driver Support	Board/system must support sensor/driver	Broad OS support (Linux, Windows, macOS)
Image Quality	Depends solely on sensor optics	Depends on sensor + USB processing
Cost	Usually lower per image quality	Usually slightly higher

Pros & Cons

OV Cam Pros

- Low latency — great for real-time vision (robotics, drones).
- High performance — less CPU overhead.
- Flexible sensor selection — many resolutions & frame rates.
- Cost-effective — often cheaper in volume.

OV Cam Cons

- Harder to use — needs correct interface, drivers, power.
 - Not universal — only works on devices with camera connectors.
-

USB Cam Pros

- Universal compatibility — works on almost any device.
- Easy setup — plug & play with drivers built in (UVC).
- All-in-one — usually includes lens, mic, lighting.

USB Cam Cons

- Higher latency — less ideal for time-critical vision.
 - Limited bandwidth (USB2 especially).
 - Slightly more CPU overhead.
-

When To Use Which?

Use an OV cam if:

- You're doing **computer vision, robotics, embedded projects**.
- You need **low latency** and high throughput.
- You're comfortable with hardware interfacing.

Use a USB cam if:

- You want **simple setup** and broad compatibility.
 - You're on **PC/laptop** or just need video conferencing / casual streaming.
-

Example Uses

OV Cam

- Real-time object detection
- Robot navigation
- Machine vision
- Custom embedded boards

USB Cam

- Video calls
 - Desktop streaming
 - Basic surveillance
 - General PC use
-

⚙️ Quick Summary

- **OV cams** = sensor modules → best for *embedded and performance-centric* use.
 - **USB cams** = ready cameras → best for *plug-and-play and general purpose*.
-

If you want, tell me **your use case** (e.g., Raspberry Pi, laptop, robotics), and I can suggest which option and specific models would fit best!



ov 2640 vs 5640 vs 7725 vs 7740

Here's a detailed comparison between four popular OmniVision camera sensors — **OV2640**, **OV5640**, **OV7725**, and **OV7740** — to help you choose the right one for your project (e.g., ESP32/STM32/embedded vision).

📊 Sensor Comparison Overview

Sensor	Resolution	Frame Rate	Pixel Size	Interface	Key Strengths
Arducam OV2640 Camera Module	~2 MP (1600×1200)	~15 fps @ full	~2.2 µm	Parallel/DVP + SCCB	Low cost, JPEG hardware, easy MCU interfacing <small>cameramodule.c... +1</small>
OV5640 Camera for Xiao ESP32S3 Sense	~5 MP (2592×1944)	~15 fps @ full, 30 fps @ 1080p	~1.4 µm	DVP + MIPI CSI-2	Higher resolution, better image detail <small>cameramodule.c... +1</small>
OV7725 camera module	VGA (640×480)	~60fps VGA	~6 µm	DVP + SCCB	Excellent low-light & high frame rate <small>SparkFun +1</small>
OV7740	VGA (640×480)	~30fps VGA	~4.2 µm	DVP + SCCB	Very high light sensitivity, clean low-light image <small>OMNIVISION</small>

🧠 Detailed Breakdown

📷 1. OV2640 — Balanced 2 MP Module

- Resolution: ~2 MP (up to UXGA 1600×1200) [cameramodule.c...](#)
 - Frame Rates: ~15 fps at full res; ~30fps VGA; ~60fps CIF [cameramodule.c...](#)
 - Features:
 - Supports JPEG compression on the sensor, lowering host CPU load. [cameramodule.c...](#)
 - Good choice for ESP32-CAM and low-cost MCUs.
 - Lower bandwidth — easy to stream with small microcontrollers.
 - When to use: Microcontrollers with limited processing and need compressed output (e.g., ESP32).
- ❖ Great for hobbyist and IoT camera projects.
-

📸 2. OV5640 — Higher-Resolution 5 MP Sensor

- Resolution: ~5 MP (2592×1944) — much higher detail than OV2640. [cameramodule.c...](#)
- Frame Rates: ~15fps @ 5MP; ~30fps @ 1080p; possible higher fps at lower resolutions. [Arducam Blog](#)
- Features:
 - Supports YUV/RGB formats and some modules include JPEG/MJPEG. [Arducam Blog](#)
 - MIPI CSI-2 interface available on many boards (better bandwidth).
 - Better dynamic range and detail than OV2640.
- Drawbacks: Higher data throughput → needs faster MCU/host interface (e.g., SBC or MCU with MIPI support).

❖ Best choice for higher-quality imaging and machine vision if your board supports it.

🎥 3. OV7725 — Fast VGA Sensor

- Resolution: 640×480 (VGA), ~0.3 MP. [SparkFun](#)
- Frame Rates: Up to 60fps @ VGA, even ~120fps in downscaled modes. [Cirkit Designer](#)
- Features:
 - Large pixels (~6 μm) → good sensitivity and low-light performance. [SparkFun](#)
 - On-board processing like auto exposure, gain, white balance. [OMNIVISION](#)
- Drawbacks: Much lower resolution — not suitable for detailed imaging.

❖ Use it for fast motion detection, robotics tracking, high fps needs.

🌙 4. OV7740 — VGA with Ultra-High Sensitivity

- Resolution: VGA (640×480). [OMNIVISION](#)
- Frame Rates: ~30fps @ VGA; ~60fps @ QVGA. [OMNIVISION](#)
- Features:
 - Exceptional low-light sensitivity thanks to OmniPixel3-HS tech. [OMNIVISION](#)
 - Very low noise images in poor lighting.
- Drawbacks: Still VGA resolution, modest speed compared to OV7725.

❖ Choose if low-light performance is critical and resolution isn't.

📌 Quick Recommendations by Use-Case

 If you want the most detail (higher res):

→ OV5640 — ##detail and flexibility (e.g., 1080p) cameramodule.c...

⚡ If you need low cost & MCU compatibility:

→ OV2640 — easy to interface and JPEG compressed cameramodule.c...

 If you need high fps for motion tracking:

→ OV7725 — fast VGA up to ~60 fps

 If low-light capability matters more than resolution:

→ OV7740 — top low-light sensitivity

Summary

- OV2640: 2 MP “sweet spot” — good for ESP32 & embedded MCU projects.
 - OV5640: Higher resolution — better images but heavier data.
 - OV7725: Fast, low-res sensor — great for motion/robot tracking.
 - OV7740: Low light specialist — classic VGA sensor with strong sensitivity.

If you want comparison results with *actual camera modules or boards* (e.g., frame rates quickly achievable on ESP32-S3 or Raspberry Pi), tell me your platform and I can tailor it!



for stm32

Here's a practical comparison of OV2640 vs OV5640 vs OV7725 vs OV7740 specifically for use with STM32 microcontrollers — focusing on compatibility, interface, performance, and what works best with STM32.

 Interfaces & STM32 Compatibility

All four sensors use a **parallel camera interface (DVP)** plus an SCCB/I²C control bus — this is important because:

STM32 can interface with them via:

- **FSMC / DCMI (Digital Camera Interface)** on many STM32 families (F4, F7, H7, L4+)
 - I²C/SCCB to configure sensor registers

 MIPI CSI-2 (like on some OV5640 modules) is *not* directly supported on STM32 — only parallel DVP works reliably.

So on STM32, you'll typically choose sensors/micro-modules that expose DVP parallel + SCCB signals.

Quick Feature Comparison (STM32 Context)

Sensor	Max Resolution	Typical Speed on STM32	Best Use-Case
OV2640	2 MP (1600×1200)	~5–15 fps @ XGA	General purpose, low cost
OV5640 (parallel)	5 MP (2592×1944)	~3–8 fps @ full	Higher detail if you can handle data
OV7725	VGA (640×480)	~30–60 fps	High frame rate / motion
OV7740	VGA (640×480)	~30 fps	Best low-light VGA

 These are *typical achievable* ranges on STM32 DCMI + DMA — actual fps depends on MCU clock, memory, and how you store/transfer frames.

Detailed Notes for STM32

OV2640 — The “Balanced” Choice

Pros

- Up to 2 MP — good detail without huge data.
- JPEG compression on-chip → STM32 stores/streams JPEG instead of raw — huge advantage if flash/RAM limited.
- Lower data throughput than 5 MP — easier to handle.

Cons

- Lower fps at full res vs VGA.
- Still needs efficient DMA + DCMI.

 **Best fit for:** STM32 applications with limited RAM/flash — web servers, basic vision, JPEG over UART/ethernet.

OV5640 — Higher Resolution

Pros

- Up to 5 MP — much detail for imaging.

Cons

- Big data volume — even at reduced resolution.

- Often **MIPI variants** which don't work with STM32 DCMI unless you find a parallel breakout.
 - Even parallel modules still push lots of data — saturates memory bus.
 - Limited fps at high resolution on STM32.
- ➡ **Use only if:** You *must* have high resolution and can handle slower fps + big memory transfers (e.g., pairing with external RAM + heavy DMA).
-

⚡ OV7725 — Fast VGA for Vision Tasks

Pros

- Very **high frame rate at VGA** (30–60+ fps) — ideal for motion tracking.
- Good sensitivity and clean signal.
- Lower resolution makes it much easier for STM32 to buffer & process.

Cons

- Only **VGA** — not great for detailed photography.
- ➡ **Best fit for:** Real-time vision where speed matters — tracking, robotics, gesture, optical flow.
-

🌙 OV7740 — VGA with Excellent Low-Light

Pros

- VGA resolution with **strong low-light performance**.
- Similar ease of use as OV7725 with better noise handling in dim scenes.

Cons

- Still **VGA** only, lower overall fps than OV7725 in some modes.
- ➡ **Best fit for:** Low-light embedded vision at VGA.
-

🧠 STM32 Families & What You Can Expect

MCU	Interface	Good Pairings
STM32F4 (e.g., F407 / F429)	DCMI + DMA + FSMC	OV7725, OV7740, OV2640 (JPEG)
STM32F7 / H7	Faster DCMI, better DMA	OV7725/OV7740 at high fps, OV2640/OV5640
STM32L4 / L4+	Some have DCMI	VGA sensors + JPEG
STM32G4 / G0	No DCMI	Not ideal — need parallel to SPI bridge

💡 To get the best frame rates, pair the sensor with **DCMI + DMA** and store frames to **external RAM (PSRAM)** if using higher resolutions.

🧠 Summary — Which to Pick for STM32

📌 Easiest + Most Practical Overall

➡ OV2640

- Good detail with hardware JPEG
- Works well with STM32 webservers or lightweight apps

📌 Best for High FPS Real-Time Vision

➡ OV7725

- Fast VGA
- Great for motion tracking, robotics

📌 Best for Low-Light VGA

➡ OV7740

- Lower resolution but cleaner under dim lighting

📌 Highest Detail (Budget Caution)

➡ OV5640

- Only if you have memory and bandwidth to spare
- Preferably on STM32 with lots of RAM + fast bus

🛠️ Tips for STM32 Integration

- ✓ Use DMA + DCMI to avoid CPU blocking
- ✓ If using JPEG (OV2640), configure camera to output JPEG frames
- ✓ For raw data, ensure you have enough RAM or external SRAM/SDRAM
- ✓ Use libraries/drivers already adapted (e.g., CubeMX + community drivers)

If you tell me your **exact STM32 model**, I can suggest the **best sensor + wiring diagram + sample config steps** for your project!

□ ⌂ ↴ ⌃ ⌁ ...

I had a ov7740 cam divided from ip cam. pin is 24. can you let me know the pinout?

Here's what we *do* know about the **OV7740 camera sensor and typical 24-pin DVP modules**, plus guidance on the pinout — especially useful if you salvaged one from an IP camera and want to hook it up to an STM32 or similar.

dragonwake.com +1

📌 OV7740 Sensor Basic I/O (from the chip datasheet)

The raw OV7740 image sensor (CSP/COB die) exposes these *core signals* (not necessarily the exact 24-pin ribbon order on your module, but *logical signals* that must be mapped): uctronics.com

Power & Reset

- AVDD – Analog power (typically ~3.0–3.6 V)
- DOVDD – Digital I/O power (typically ~1.7–3.3 V)
- DVDD – Core/regulator power (~1.5 V internal)
- RESET_B – Reset (active low)
- PWDN – Power down mode control

Clock & Sync

- XVCLK – External clock input (master clock)
- PCLK – Pixel clock output
- VSYNC – Frame sync output
- HREF (*or Hsync/HREF*) – Line data enable

Parallel Data (DVP)

- DATA0 ... DATA9 – Parallel video data lines (often 8-bit or 10-bit selected mode)

Control Interface (SCCB/I²C)

- SIOC – SCCB/I²C clock
- SIOD – SCCB/I²C data

Other

- SFIN / Frame Sync In – optional frame strobe input on some modules

This is the core pin function set you'll need to match from your 24-pin cable. uctronics.com

💡 Typical 24-Pin Camera Module (FPC) – What the Pins Usually Represent

Most 24-pin DVP camera modules using OV7740 (or similar VGA imagers) map those sensor signals onto the 24 conductor ribbon in some order. The exact order *depends on the board design* — there is no single universal 24-pin standard that all cameras follow. ronghuayf.com

However, common assignments you should expect (and *look for on the module traces* or in vendor docs) include:

1. Supply / ground

- 3.3 V (or module's I/O power)
- GND
- Core/analog/reg power might be combined or separate pins

2. Control pins

- SCCB / I²C — SIOC (clock), SIOD (data)
- RESET — resets the camera

- PWDN — powers down the camera
3. Clock
 - XCLK — required input clock (often driven by MCU or oscillator)
 - PCLK — pixel clock output from camera
 4. Sync
 - VSYNC — frame start signal
 - HREF / Hsync — line valid
 5. Data bus
 - D0 ... D7 or D9 — parallel pixel data (8 or 10 bits)

Usually for STM32 you'll use 8-bit mode (D0–D7) since many camera ports expect that.

If your module uses 8-bit mode, *only D0–D7* are used in practice; extra pins might be NC or reserved.

uctronics.com

❗ Important: 24-Pin Cable Isn't a Standard Pinout Unless Provided

Unlike the OV7670/OV2640 connectors used on some Arduino/ESP32 boards, OV7740 modules inside IP cams often have:

- ◆ Custom pin ordering
- ◆ Mixed power rail pins
- ◆ Shielding pins
- ◆ Mechanical keys on the FFC cable

That means **two modules with 24 pins can have totally different pinouts even if the chip is the same.**

Reddit

So, the *exact pin order* for your cable can't be guaranteed without either:

- A **photo and part number** printed on the flex/connector, or
- Tracing the camera PCB traces back to labeled pads on the board it came from.

💡 How to Determine Your Exact 24-Pin Pinout

Method 1 — Look for Markings

Check the back or edges of the flex ribbon for:

- Model/part numbers
- Silkscreen labels near the connector

These often lead to datasheets or seller pinout tables.

Method 2 — Visual Trace / Continuity Test

Using a multimeter:

1. Identify **power rails** by continuity to large caps or regulator pins on original board.

2. Find ground pins by continuity to ground plane.
3. Identify I²C SDA/SCL by following tiny resistors/labels ("SIO", "SIOC", etc.).
4. Identify clock (XCLK) by checking which pin requires a clock from MCU to see activity.

Method 3 — Compare to Known OV7740 Pin Tables

You can use the *sensor's logical pin descriptions* (above) as a reference to match signals once you've identified some pins. [uctronics.com](#)

STM32 Integration Notes

If your goal is to interface with an STM32 via its DCMI (parallel camera interface):

- ✓ You'll need XCLK from your STM32 (often a timer PWM output)
- ✓ Connect PCLK, VSYNC, HREF, D0–D7 to DCMI pins
- ✓ Use I²C (SIOC/SIOD) for camera register init/config via SCCB
- ✓ Connect RESET and PWDN to available GPIOs

This is essentially the same set of signals listed from the OV7740 datasheet. [uctronics.com](#)

Summary

Signal Type	Likely Pin on 24-Pin
Power & Ground	AVDD/DOVDD/GND
MCU Clock In	XCLK
Camera Output Clock	PCLK
Frame/Line Sync	VSYNC / HREF
Data Bus	D0–D7 (parallel output)
Control Bus	SCCB/I ² C: SIOC + SIOD
Control GPIOs	RESET, PWDN

These are the **logical signals** your 24-pin module must map — but the **exact 24-pin order varies by manufacturer** and won't be correct without tracing or documentation. [chinax.com](#)

If you can provide a photo of the flex ribbon and any printed numbers on the camera board, I can help you identify which pin is which and map it to STM32 signals!

"Lens Type: 650 nm IR Cut" means the camera lens includes an **infrared-cut (IR-cut) filter** that blocks light above ~650 nanometers.

What that means in practice

- Visible light is roughly 400–700 nm.
- Infrared (IR) starts just above visible red (~700 nm).
- A 650 nm IR-cut filter blocks **deep red and infrared light**, letting mainly visible colors through.

Why it's used

- Accurate colors (daylight) 

Image sensors are very sensitive to IR. Without an IR-cut filter, IR light washes out colors (skin looks grayish, foliage looks wrong).

- Sharper images

IR light focuses differently than visible light; blocking it reduces blur.

Trade-offs

-  Worse low-light / night performance
Since IR is blocked, the camera can't use IR illumination (e.g., 850 nm IR LEDs).
-  Not suitable for night-vision with IR LEDs

Compare with other common options

Lens / Filter	What it does	Typical use
650 nm IR-cut	Blocks red+IR early	Accurate color, indoor/daylight
700–720 nm IR-cut	Blocks IR but keeps more red	General color cameras
No IR-cut (IR-pass)	Allows IR	Night vision, IR imaging
IR-cut switch (ICR)	Day: block IR / Night: allow IR	Day-night security cameras

For your camera / STM32 project

- If you want true color images in daylight → 650 nm IR-cut is good 
- If you want night vision with IR LEDs → do NOT use 650 nm IR-cut  (use no-IR-cut or an ICR lens)

If you want, tell me what lighting conditions (day only, indoor, night with IR LEDs) and which sensor (OV7740, etc.), and I'll recommend the best lens/filter combo.

can I detect the temperature with this cam?

Short answer:  No — not with this camera.

Your OV7740 with a 650 nm IR-cut lens is a visible-light camera, not a thermal sensor.

Why it can't measure temperature

1 Wrong wavelength range

- OV7740 detects visible light (~400–700 nm)
- Temperature (thermal radiation) from objects at room/body temperature is emitted mainly in **long-wave infrared (LWIR)**:
 - 8–14 μm (8000–14000 nm)

 That is $\sim 20\times$ longer wavelength than what your camera can see.

2 IR-cut filter blocks heat radiation

- Your 650 nm IR-cut lens:
 - Blocks **near-IR**
 - Completely blocks **thermal IR**
- Even without the IR-cut, the **sensor itself cannot detect heat**

3 CMOS cameras ≠ thermal cameras

Camera type	Can detect heat?	Example
OV7740 / OV2640	 No	Visible cameras
IR-modified CMOS	 (only reflections)	Night-vision
Thermal camera	 Yes	FLIR, MLX90640

Thermal cameras use **microbolometers**, not CMOS pixels.

! Common misunderstanding

You **cannot** infer real temperature from:

- brightness
- color

- red channel intensity
- near-IR images

Those only show **reflected light**, not emitted heat.

What you *can* do instead

Option 1: Use a real thermal sensor (recommended)

For STM32:

Sensor	Type	Resolution	Notes
MLX90640	Thermal array	32×24	Cheap, I ² C
MLX90641	Thermal array	16×12	Very low cost
FLIR Lepton	Thermal camera	80×60 / 160×120	Best quality

These directly measure temperature (°C).

Option 2: Use OV7740 only for *indirect* detection

You can:

- Detect **hot objects** visually (glowing metal, flames)
- Detect **presence** (people, movement)

But not measure temperature.

Summary

- OV7740 + 650 nm IR-cut lens → cannot detect temperature
 - Removing IR-cut → still cannot measure heat
 - Thermal sensor required → MLX90640 / FLIR
-

If you tell me:

- your **temperature range** (human / industrial / fire)
- **accuracy needed**
- **budget**

I can recommend the **best thermal sensor + STM32 wiring + code approach** 

For **human temperature detection**, here's the clear, correct answer and your best options 👇

✗ Your OV7740 camera cannot measure human temperature

Even for humans:

- Human body temperature (~36–37 °C) emits thermal radiation at 8–14 μm
- OV7740 + 650 nm IR-cut only sees visible light
- Result: NO temperature data, not even approximate

So:

You cannot measure human temperature with this camera

✓ What you SHOULD use for human temperature (with STM32)

🟡 MLX90640 (Best balance)

- 32×24 thermal pixels
- Measures actual temperature (°C)
- Accuracy: ±1 °C (can reach ±0.5 °C with calibration)
- Interface: I²C
- Cost: \$ \$ (much cheaper than FLIR)

✓ Widely used in fever detection, presence sensing, thermal imaging

🟡 MLX90641 (Lower cost)

- 16×12 pixels
- Same principle, lower resolution
- Good for presence + rough temperature

🟡 FLIR Lepton

- 80×60 or 160×120
- Medical-grade potential
- Expensive, SPI interface

❖ Recommended setup (human temp)

Best practical combo

nginx

STM32 + MLX90640 + OV7740

 Copy code

- MLX90640 → measures temperature
- OV7740 → detects face / person / position
- Combine both → accurate + smart system

This is how **real fever-screening systems** work.

Important medical reality

If this is for **fever screening**:

- Absolute accuracy is hard
- Needs:
 - Distance control (30–50 cm)
 - Ambient temperature compensation
 - Face region selection (forehead / inner canthus)
- Not legally “medical-grade” without certification

But for **engineering / hobby / screening projects** → totally fine.

STM32 wiring example (MLX90640)

- VCC → 3.3 V
- GND → GND
- SDA / SCL → STM32 I²C
- Frame rate: 2–16 Hz (configurable)

Final verdict

- OV7740  cannot measure human temperature
- MLX90640  best choice for humans
- Combine camera + thermal = professional approach

If you want, tell me:

- your STM32 model
- distance to human
- budget

I'll give you **exact sensor, schematic, and STM32 code flow** 

Thanks — with STM32F407 and a distance of 10–15 meters, here is the honest, engineering-correct answer



✗ Human temperature detection at 10–15 m is NOT feasible

(using MLX90640, MLX90641, FLIR Lepton, or any small thermal sensor)

Why it does NOT work (physics limit)

Human body temperature measurement relies on **thermal radiation (8–14 µm)**.

At 10–15 m:

- Human face becomes **too small in the thermal sensor's field of view**
 - Thermal energy is **diluted by distance**
 - Ambient air absorbs & distorts IR
 - Accuracy collapses ($> \pm 5\text{--}10^\circ\text{C}$ error)
-

▀ Let's look at the numbers (important)

MLX90640 example (best low-cost thermal)

- Resolution: 32×24
- FOV: ~55°
- At 10 m:
 - One pixel $\approx 30\text{--}40\text{ cm}$
 - Human face $\approx 1\text{--}2\text{ pixels only}$

- ➡ You cannot isolate forehead or skin
 - ➡ You get an **average blob**, not temperature
-

🚫 What will NOT work at 10–15 m

Device	Result
MLX90640	✗ useless at that distance
FLIR Lepton	✗ still too low resolution
OV7740 (any lens)	✗ no temperature

Device	Result
IR thermometer modules	✗ require < 1 m

What IS possible at 10–15 m

1 Human detection / presence (NOT temperature)

With your OV7740 + STM32F407:

- Detect:
 - Human presence
 - Movement
 - Face / body shape (basic)
- ✗ No temperature

This is **fine and realistic**.

2 Fever screening architecture (REAL systems)

Professional systems do this:

kotlin

```
Long distance → detect human
Short distance → measure temperature
```

 Copy code

Real workflow:

1. Camera detects person at 10–15 m
2. Person walks closer (0.3–1 m)
3. Thermal sensor measures temperature

This is how **airports & hospitals** work.

If you MUST measure temperature at 10–15 m

You need **industrial / military-grade equipment**:

Device	Cost
High-resolution LWIR camera (640×480)	\$5,000–\$15,000
Motorized zoom LWIR lens	\$2,000+
Calibration blackbody	Required

! STM32F407 is not used in these systems.

Best practical solution for YOU (recommended)

Hybrid system (realistic & works)

Use what you already have + add one sensor

Task	Sensor
Detect human at 10–15 m	OV7740
Measure temperature at < 1 m	MLX90640
Control & logic	STM32F407

STM32F407 suitability

STM32F407 is perfect for:

- OV7740 via DCMI
- MLX90640 via I²C
- Decision logic
- Display / communication

Final verdict (very important)

- Human temperature at 10–15 m: impossible with small sensors
- Human detection at 10–15 m: yes
- Temperature measurement < 1 m: yes
- Hybrid approach = correct engineering

If you want, I can:

- Design a full block diagram
- Give exact STM32 pin mapping
- Suggest lens FOV for OV7740
- Provide distance-based logic flow

Just tell me

< 2/2 > ...

Below is a clear, practical, STM32F407-focused guide to detect human temperature at ~1 m using MLX90640. This is the **correct way** to do it.

✓ What MLX90640 Can Do at 1 m

- Measures true skin temperature (°C)
- Works well at 0.3–1.0 m
- Accuracy: ±1 °C (can improve with calibration)
- Resolution: 32×24 thermal pixels

At 1 m, a human face covers **multiple pixels**, which is the key requirement.

🔌 Hardware Connection (STM32F407)

MLX90640 Pins → STM32F407

MLX90640	STM32F407
VDD	3.3 V
GND	GND
SDA	I2C1_SDA (PB7)
SCL	I2C1_SCL (PB6)
ADDR (if present)	GND or VDD

- ◆ Use **pull-up resistors** (4.7 kΩ) on SDA & SCL
 - ◆ Use I2C speed = 400 kHz
-

⚙️ STM32 Configuration (CubeMX)

Enable:

- I2C (Fast mode 400 kHz)
- Floating point (FPU)
- Enough RAM (~20 KB free)

Clock:

- CPU ≥ 84 MHz (168 MHz preferred)
-

📦 MLX90640 Software Stack (Important)

You **must** use the official Melexis MLX90640 API.

Required files:

- mlx90640_api.c
- mlx90640_api.h
- mlx90640_params.h

(Official Melexis library – widely used)

⌚ Measurement Flow (Critical)

1 Read EEPROM (once)

C

```
MLX90640_DumpEE(0x33, eeData);
MLX90640_ExtractParameters(eeData, &params);
```

 Copy code

2 Set frame rate (recommended: 8 Hz)

C

```
MLX90640_SetRefreshRate(0x33, MLX90640_REFRESH_8_HZ);
```

 Copy code

3 Read a frame

C

```
MLX90640_GetFrameData(0x33, frameData);
```

 Copy code

4 Calculate temperature

C

```
float Ta = MLX90640_GetTa(frameData, &params);
MLX90640_CalculateTo(frameData, &params, 0.95, Ta, pixelTemp);
```

 Copy code

- pixelTemp[768] → 32×24 temperature map
 - 0.95 = human skin emissivity
-



Human Temperature Extraction (VERY IMPORTANT)

- Do NOT average all pixels
- Extract face region only

Practical approach at 1 m:

1. Identify warmest cluster (face)
2. Select center 5×5 pixels
3. Take maximum or median value

c

```
float maxTemp = 0;  
for(int i = 0; i < 768; i++) {  
    if(pixelTemp[i] > maxTemp)  
        maxTemp = pixelTemp[i];  
}
```

 Copy code

- Use max or top 3 average → best for forehead detection

Temperature Calibration (Required)

Raw skin temp \neq body temp

Simple compensation:

c

```
bodyTemp = skinTemp + 2.0; // empirical offset
```

 Copy code

Typical offsets:

- Indoor: +1.5 to +2.5 °C
- Stable ambient needed

Distance Rules (Strict)

Distance	Works?
0.3–0.5 m	   BEST
0.5–1.0 m	  OK
>1.5 m	 Unreliable

- Keep distance constant

Environmental Requirements

To get usable results:

- Indoor environment
 - No direct sunlight
 - No wind / AC blowing on face
 - Person stationary ~2 sec
-

💡 Recommended System Logic

css

OV7740 detects face →

Person moves **to 1 m** →

MLX90640 measures →

STM32 validates →

Display / transmit

 Copy code

🚫 Common Mistakes

- ✗ Using average temperature
 - ✗ Measuring through glass
 - ✗ Changing distance
 - ✗ Outdoor sunlight
 - ✗ Expecting medical accuracy
-

✓ Final Result You Can Expect

- Resolution: 32×24 thermal image
 - Accuracy: ±1 °C
 - Reliable **fever screening**
 - Perfect for **STM32F407**
-

If you want next:

- ✓ Exact CubeMX settings
- ✓ Full STM32 code example
- ✓ Face-region pixel mapping
- ✓ UART / LCD output

Tell me 

      ...

any other thermal cams produced in china?

Here are **thermal imaging and temperature-sensing camera modules** you can consider (including some **China-produced or easily sourced options**) if you want something beyond just the MLX90640, especially for use with microcontrollers like your STM32F407:



[MLX90640 IR Thermal Imaging Module](#)

\$69.12 • Newegg.com - LvZhengKeji...

[MLX90640 Thermal Camera Breakout](#)

\$70.95 • PiShop USA + others

[Grove Thermal Imaging Camera MLX90641](#)

\$55.99 • Seeed Studio + others

Mid-Level Thermal Modules (good for hobby/embedded)

- [MLX90640 IR Thermal Imaging Module](#) — 32×24 thermal array; China-sourced variants exist and are affordable (~\$30–\$70). Good balance of detail and microcontroller compatibility. SmartBuy
- [MLX90640 Thermal Camera Breakout](#) — another breakout board for the same sensor with good software ecosystem.
- [Grove Thermal Imaging Camera MLX90641](#) — slightly different MLX906xx variant with I²C interface and similar integration ease.

Higher-Resolution / More Capable Thermal Cameras

(These are more powerful than a simple 32×24 array and often come from Chinese manufacturers or suppliers.)

- [Waveshare 80×62 Thermal Imaging Camera](#) — 80×62 pixel thermal array module; better spatial detail than MLX90640. Spotpear
- [ESP32-S3 Thermal Camera Module \(80×62\)](#) — similar sensor but geared toward embedded use with USB/Type-C output.
- [Waveshare Long-Wave IR Thermal Imaging Module](#) — another 80×62 variant with a standard 40-pin camera interface (easier integration with custom boards).

Lower-Resolution / Budget Options

- [Official AMG8833 IR Thermal Camera](#) — 8×8 thermal array. Not very detailed but very inexpensive for basic temperature mapping.
 - [Grove MLX90621 Thermal Sensor](#) — lower resolution array (16×4) useful for simple thermal detection rather than imaging.
-

-China Thermal Camera Sensors & Modules

Aside from common hobby sensors like MLX90640/41, there are [more advanced thermal cores made in China](#) — often used in industrial and surveillance gear:

-CN Chinese Thermal Core Manufacturers

- [InfiRay \(IRay\)](#) – A major Chinese maker of tiny thermal imagers used in consumer and industrial thermal cameras. Wikipedia
- [Domestic focal plane arrays & thermal imaging solutions](#) – Companies such as those making modules like the 80×62 sensors or larger core modules for drones and security systems. Spotpear +1
- [Higher-end indigenous cores](#) like LGC6122 (640×512) or other uncooled IR FPAs from Chinese manufacturers (used in night vision, vehicle systems) exist, though they're often sold to OEMs, not as breakout boards. Yiqi Service...

These advanced products may be [expensive and targeted at OEM/system integrators](#) rather than hobbyists, but they show that a domestic Chinese thermal sensor ecosystem exists beyond just MLX906xx parts.

Choosing the Right Module

👉 For microcontroller use (STM32F407):

- [Best thermal resolution / easiest integration](#): MLX90640 / MLX90641 variants
- [Upgrade over MLX90640](#): 80×62 sensors (better detail, similar I²C/SPI interface)
- [Smallest / cheapest option](#): AMG8833 / MLX90621 (lower resolution, but simple and cheap)

👉 For projects needing real thermal *images*:

- 80×62 (or higher) is recommended; 32×24 may feel coarse at a distance.

👉 For high-end imaging (industrial / long range):

- Consider modules with larger sensors (e.g., 256×192 or 640×512) from more specialized Chinese suppliers — these are not necessarily plug-and-play but can be sourced through industrial channels.

Huanyu Vision

Quick Integration Notes

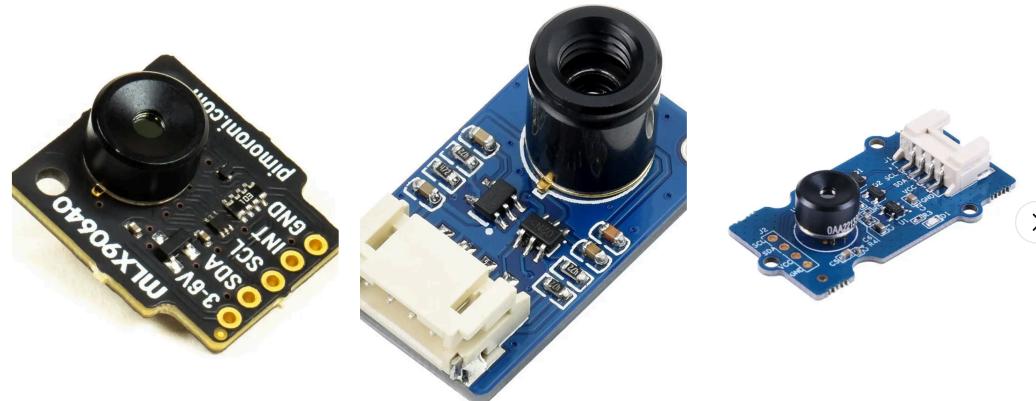
- Most of these modules communicate via I²C (for thermal data/config) and sometimes SPI (for bulk reads), which STM32F407 can handle.
- For larger arrays (80×62 and above), data rate increases — make sure to optimize I²C or use modules with faster interfaces.

If you tell me your target resolution, max budget, and how close the subject will be, I can recommend the best specific module and wiring plan for your STM32F407 project!



how is the price?

Here are **current price ranges** for thermal camera modules you could use with your STM32F407 (especially for detecting human temperature at close range). Prices vary based on **resolution, brand, and seller**, and many modules are **Chinese-sourced or generic imports**:



💡 Typical standalone MLX90640 chip-only units (boardless) can be even cheaper (~\$25–\$35), but they require breakout & proper wiring to use reliably. SmartBuy

Lower-Resolution (8×8 AMG8833)

Good for simple temperature detection but much lower detail:

- [AMG8833 IR Thermal Camera Sensor](#) – ~\$49.95
- [AMG8833 Thermal Imaging Camera Module](#) – ~\$54

These provide basic heat blob detection and average temperatures per small grid (8×8) — okay for simple human presence/temp detection but **much less spatial accuracy** than MLX90640.

Price Summary (Approx)

Sensor Type	Typical Price (each)
32×24 MLX90640 module	\$50–\$90
AMG8833 8×8 thermal module	\$40–\$60
Bare MLX90640 sensor only	\$25–\$35 (needs breakout) <small>SmartBuy</small>

Prices fluctuate with seller, shipping, and stock — the cheaper end often comes from generic Chinese sellers (AliExpress / eBay), while SparkFun/Adafruit tend to be slightly higher but with better support.

Tips for Buying

For STM32F407 projects:

- MLX90640 (~\$50–\$90) is the *sweet spot* — enough resolution for face/forehead temp at ~0.5–1 m.
- AMG8833 (~\$40–\$60) is cheaper but low res (~8×8) — better for presence or rough temp, not detailed thermal maps.

Check sellers for:

- Field of view (FOV) — wider FOV means larger area coverage but lower per-pixel spatial resolution at distance. SmartBuy
- Frame rate and interface (usually I²C).
- Included hardware and documentation.

If you tell me **your budget** and **how you want to use the thermal data** (just forehead temp vs a thermal map), I can recommend the **best specific module and how to wire/code it on STM32F407** 

To detect a person using an OV camera (OV7740 / OV2640, etc.) with STM32, it's important to be clear first:

- ! OV cameras are normal RGB sensors — they do NOT detect temperature or humans directly.
- They only capture visible light images.
- Person detection must be done using image processing or ML.

Below are practical methods, from simplest to advanced, that actually work on STM32F407.

1 Motion-based Human Detection (MOST PRACTICAL on STM32F407)

Idea

Detect movement, then assume moving objects of human-like size = person.

How it works

1. Capture grayscale frame (QQVGA or QVGA)
2. Compare with previous frame
3. Count changed pixels
4. If area > threshold → person detected

Requirements

- OV7740 / OV2640
- Low resolution (160×120 or 320×240)
- STM32F407 + external SRAM recommended

Algorithm

text

Frame N - Frame N-1 → Difference

Threshold → Binary image

Morphological filter

Blob size check → Human

 Copy code

Pros

- Fast
- No ML
- Works on STM32F407

Cons

- Cannot detect static person
- False positives (pets, shadows)

2 Shape / Size-based Detection (Better Accuracy)

Idea

Detect **blobs** and check if they match human proportions.

Steps

1. Convert to grayscale
2. Background subtraction
3. Connected-component labeling
4. Check:
 - Height > width
 - Area > threshold
 - Vertical symmetry

Human-like heuristics

c

```
if (area > MIN_AREA &&
    height > width &&
    height/width > 1.3)
{
    person_detected = 1;
}
```

 Copy code

Pros

- Still MCU-friendly
- Better than motion only

Cons

- Still heuristic-based
- Limited accuracy

3 Face Detection (LIMITED on STM32F407)

Classical method

- Haar Cascade (OpenCV style)

Reality check

- Too heavy for STM32F407
- Needs large RAM + CPU

 Not recommended unless:

- Very low resolution (96×96)

- Very slow frame rate
 - Optimized fixed-point code
-

4 ML Person Detection (EDGE AI)

Feasible models

- Tiny CNN
- MobileNet-0.25 (very small)

Frameworks

- TensorFlow Lite Micro
- ST X-CUBE-AI

Requirements

- Image $\leq 96 \times 96$
- Grayscale
- External RAM

Performance

Item	Value
FPS	1–3
Accuracy	Medium
Complexity	High

Pipeline

text

OV Cam → Resize → Normalize → CNN → Person / No person

 Copy code

5 Best Practical Setup (RECOMMENDED)

Hardware

- OV7740 / OV2640
- STM32F407
- External SRAM
- Optional: PIR sensor (for trigger)

Software Stack

text

PIR triggers →
OV frame capture →
Motion + blob detection →
Confirm person

 Copy code

This hybrid approach is **fast, cheap, and reliable.**

6 Important Reality Check

Task	OV Cam
Human temperature	 Impossible
Thermal detection	 Impossible
Person detection	 Yes (vision-based)
Night detection	 Needs IR LEDs + no IR-cut

🔥 If You Want HIGH Accuracy

Use:

- OV camera → Raspberry Pi / Jetson
OR
- Thermal cam (MLX90640) → STM32 for presence
- Combine both → best results

⭐ Recommendation for YOU

Since you already mentioned:

- STM32F407
- OV cam
- 1–10 m range

 Use motion + blob detection

 Add IR LEDs + remove IR-cut for night

 Use MLX90640 if temperature or presence is needed

If you want, I can give you:

-  Exact STM32 C code for motion detection
-  OV7740 camera config
-  Memory layout for F407

- Simple ML model for person detection

Just tell me which one you want.



just a simple detection. detect area of paper clip in a board.
background is black and paper is brown or dark gray

Got it

This is much simpler than human detection and is perfectly doable on STM32F407 with an OV camera.

You have:

- Black background
- Paper / paper clip area = brown or dark gray
- Goal: detect the area (blob size)

No ML needed

Just thresholding + blob area

Recommended Simple Method (FAST & RELIABLE)

Core idea

Convert image → grayscale → threshold → count pixels

Black background ≈ low pixel values

Paper ≈ higher pixel values

Camera Settings (VERY IMPORTANT)

Resolution

Use QQVGA (160×120) or QVGA (320×240)

Camera output

- Grayscale (Y channel) if supported
- Or RGB565 → convert to grayscale

Lock exposure

To avoid brightness changes:

- Disable auto exposure
- Disable auto white balance

1 Grayscale Conversion

If RGB565:

C

```
uint8_t gray = (r*30 + g*59 + b*11) / 100;
```

 Copy code

Or faster:

C

```
gray = g; // good enough
```

 Copy code

2 Thresholding (Key Step)

Choose a threshold **above black** and **below paper**

Example:

C

```
#define THRESHOLD 60 // tune this
```

 Copy code

Binary image:

C

```
if (gray > THRESHOLD)
    bin = 1; // paper
else
    bin = 0; // background
```

 Copy code

3 Noise Filtering (Optional but recommended)

Remove single-pixel noise:

- 3×3 majority filter
- Or ignore blobs smaller than X pixels

4 Area Detection (Simple Pixel Count)

Simplest version (works well)

C

```
uint32_t area = 0;

for (int i = 0; i < WIDTH*HEIGHT; i++)
{
    if (binary[i])
        area++;
}
```

[Copy code](#)

Convert to real area

```
c

real_area_mm2 = area * pixel_area_mm2;
```

[Copy code](#)

5 Blob Detection (Optional – Better Accuracy)

If you want **only** the paper, not dust:

1. Flood-fill / connected component
2. Select largest blob
3. Measure its area

Pseudo:

```
c

find_all_blobs();
largest_blob = max(area);
```

[Copy code](#)

6 Example STM32-Friendly Pseudocode

```
c

uint32_t detect_paper_area(uint8_t *frame)
{
    uint32_t area = 0;

    for (int y = 0; y < H; y++)
    {
        for (int x = 0; x < W; x++)
        {
            uint8_t gray = frame[y*W + x];

            if (gray > THRESHOLD)
                area++;
    }
}
```

[Copy code](#)

```
    }  
    return area;  
}
```

7 Calibration (DO THIS ONCE)

1. Capture empty black board → record max pixel value
2. Capture paper → record min pixel value
3. Threshold = midpoint

text

 Copy code

```
black ≈ 10–30  
paper ≈ 70–120  
threshold ≈ 50
```

🎯 Expected Performance

Item	Value
FPS	10–30
RAM	< 30 KB
CPU	Very low
Accuracy	Very high (controlled lighting)

⚠ Common Mistakes

- Auto exposure ON ✗
 - Shiny surface (reflections) ✗
 - Uneven lighting ✗
-

✓ BEST SETUP FOR YOU

- OV7740 / OV2640
 - Grayscale output
 - Fixed LED lighting
 - Black matte background
-

If you want, I can give you:

- Exact OV7740 register settings
- Full STM32F407 C code
- Blob detection (connected component) code
- Area-to-mm calibration math

Just tell me what you want next.

