Motion detection without the magic

A deterministic detector on a ten-dollar board, and a headline number that did not survive being checked

Munawar Kazmi • munawarkazmi.com

A plain-language guide to **esp32-cam-motion-detector**, firmware from a commercial prototype.

Code and demo: github.com/munawarkazmi/esp32-cam-motion-detector

---

A camera that notices things, and nothing else

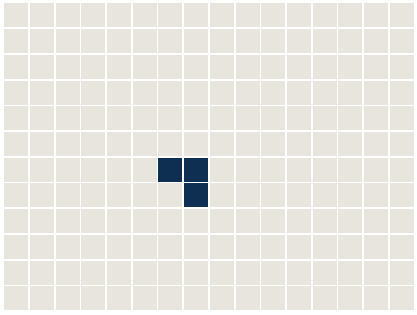
This is firmware for a very small, very cheap camera board, roughly the size of a large postage stamp and costing about the price of a sandwich. Its entire job is to notice when something in front of it moves, switch on a light, and say so.

There is no artificial intelligence in it. No neural network, no machine learning library, no cloud service. It does arithmetic on numbers, and the arithmetic is simple enough to explain completely in the next two paragraphs.

How it actually works

The camera takes a picture, 800 pixels wide and 600 tall, in black and white. Working with half a million individual pixels on a board this small would be slow, so the first thing the firmware does is throw most of that detail away.

It carves the image into squares of 50 by 50 pixels, and reduces each square to a single number: the average brightness inside it. That leaves a grid of 16 across by 12 down, 192 numbers in total, which is a manageable summary of what the camera can see.



The whole picture reduced to 192 brightness values. Three of them changed enough between frames to count; two would already have been enough to raise the alarm.

Then it takes another picture and does the same thing, and compares the two grids number by number. If a square's average brightness has changed by more than 40 percent, that square counts as changed. If two or more squares have changed, something moved, and the board lights its LED and announces it.

That is the whole detector. Averaging, subtracting, and counting.

The two thresholds are what make it usable rather than maddening. Requiring a 40 percent change stops it reacting to the gentle flicker that every cheap camera sensor produces. Requiring

two squares rather than one stops a single speck of noise in a single square from triggering an alert. Together they distinguish a person walking past from a camera quietly being a camera.

Why not use AI for this?

It is a fair question, since almost everything is sold with AI attached to it now. For this particular job, arithmetic wins on four counts.

- **It fits.** The board has a very small amount of memory. A neural network capable of recognising things would struggle to fit alongside the camera driver, and would leave nothing spare.
- **It is fast enough** to keep up with the camera without help.
- **It is predictable.** Identical inputs always produce an identical answer. When it triggers on something you did not expect, you can work out exactly why: which squares changed, and by how much.
- **It can be explained to a customer** in one sentence, which matters more in a commercial product than it does in a research paper.

A model would be the right answer to a harder question, such as whether the moving thing is a person, a cat or a curtain. This device does not answer that question and does not pretend to.

Where it came from

This firmware is from a real commercial prototype, built for an electronics company in 2025 as part of an intrusion detector. The full product alerted over a long-range radio mesh network, so that units in a field or a warehouse could report back without wiring or wi-fi. That radio component remains the company's property and is not part of this public release.

What is public is the detection firmware itself, and a short video of the device running, so anyone can see the thing work rather than take a description on trust.

The number that had to go

Now the part that makes this repository unusual, and it is the reason it is worth writing about at all.

An earlier version of this project's front page claimed a 93.8 percent detection accuracy, measured over more than 100 hours of field trials, along with a 60 percent reduction in memory usage. It also included files presented as evidence from those trials.

Going back through the repository later, none of it held up:

- The file labelled as a field log contained no log entries.
- The document that supposedly calculated the accuracy used a formula that does not produce the number it stated.
- The firmware printed the figure "93.8%" because that figure was typed directly into the code, not because anything had been measured.

- The built-in self-test, which appeared to validate the detector, compared its detections against a fixed clock schedule rather than against any record of what had actually happened in front of the camera.

The project was old and those figures could no longer be substantiated, so they were withdrawn. The retraction now sits at the top of the repository's front page, above everything else, rather than being quietly deleted.

> The uncomfortable detail is the hardcoded number. Firmware printing a fixed accuracy figure looks, to a casual reader, exactly like firmware reporting a measurement. That is precisely why the claim needed removing rather than softening.

What the repository claims now

Nothing quantitative. There is no accuracy figure, no detection rate, no memory saving, and no performance claim anywhere in it.

What it does have is a green badge indicating that automated checks pass, and the front page states exactly what that badge means: this code compiles for the intended board, as committed, on every change. Not that it works well. Not that it is accurate. That it builds.

The device does report numbers while running, every thirty seconds: how many frames it has processed, how many detections it has made, how fast its loop is going, and how much memory is free. Those are things it genuinely measures about itself, as opposed to a verdict on how good it is.

And the front page sets out what would have to exist before any accuracy figure appeared there again: the raw event logs, the method used to establish what actually happened in front of the camera, and the exact firmware version that produced the numbers. That is a deliberately high bar, and it is the same one applied across the author's other projects, where the numbers are published together with the code and data that generate them.

Why this matters more than the detector does

The technical content here is modest, and honestly stated: a simple, robust, well-suited piece of embedded firmware that does one job without pretending to do more.

The interesting content is the correction. Unverifiable numbers are extremely common in engineering portfolios and product pages, and they are almost never challenged, because checking somebody's claimed accuracy figure requires their data, and their data is usually not there. Nobody would have caught this one either.

> The claim was removed because it could not be supported, not because anyone complained. That is the entire point.

A repository that says "this builds, here is exactly what it does, and here is what I removed because I could not prove it" is more useful than one asserting a number nobody can check.

The firmware, the detection code, and the demo video are public. This guide is part of a wider set covering projects where the numbers are published alongside the code and data that produce them.

Repository: github.com/munawarkazmi/esp32-cam-motion-detector