HC-SR04 Ultrasonic Distance Sensor with ESP32

OmArTronics Sensor Lab · Project 01 · Workbench Guide

A printable companion to the full tutorial. Everything you need at the bench, nothing you only need on the sofa.

Quick facts

Sensor	HC-SR04 ultrasonic distance module (ElecFreaks design, many clones)
Principle	Time-of-flight of a 40 kHz ultrasonic burst
Interface	Digital — 10 us Trigger pulse in, Echo pulse-width out
Supply	5 V DC, about 15 mA
Logic	Trig accepts 3.3 V drive · **Echo outputs 5 V and must be divided**
Range	2 – 400 cm (datasheet) · 0.3 cm resolution · ±3 mm best case
Beam	About 15 degree cone
Cycle time	60 ms minimum between pings
Size	45 × 20 × 15 mm
MCU used here	ESP32-WROOM-32, Trig on GPIO 5, Echo on GPIO 18

1. How it works, in 150 words

The module fires eight cycles of 40 kHz sound out of its transmitter can when you pull Trig high for 10 microseconds. That burst travels through the air, reflects off whatever is in front of it, and returns to the receiver can. The module then holds the Echo pin high for exactly as long as the round trip took. Your microcontroller times that pulse and does the arithmetic.

The word that matters is *round*: the sound flew to the target and back, so the measured time covers twice the distance you want.

Equation

```
distance_cm = (echo_time_us × 0.0343) / 2
```

`0.0343` is the speed of sound in cm per microsecond at 20 °C. The datasheet's shortcut `distance_cm = echo_time_us / 58` is the same thing, because 1 / (0.0343 / 2) = 58.3.

Worked example — Echo stays high for 1160 us: 1160 × 0.0343 = 39.79 cm of total travel, ÷ 2 = **19.9 cm**. Cross-check: 1160 / 58 = 20.0 cm.

Speed of sound rises about 0.6 m/s per °C, so an uncompensated reading drifts roughly 0.17 % per degree away from 20 °C.

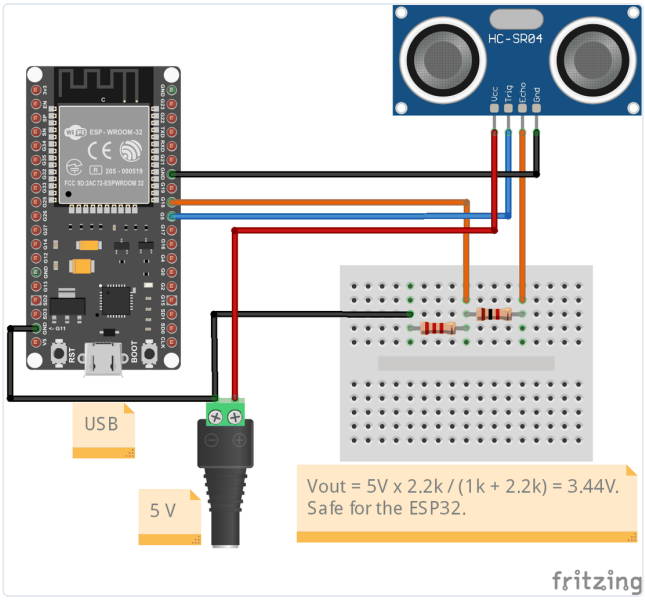
2. Bill of materials

Component	Model / Value	Qty	Purpose
ESP32 DevKit	ESP32-WROOM-32	1	Times the Echo pulse, serves the dashboard
Ultrasonic sensor	HC-SR04	1	Fires the burst, returns the Echo pulse
Resistor	1 kOhm, ¼ W	1	R1 of the Echo divider (Echo → node)
Resistor	2.2 kOhm, ¼ W	1	R2 of the Echo divider (node → GND)
Breadboard	400 or 830 tie-point	1	Solderless wiring
Jumper wires	M-M / M-F	~6	Connections

About 6–9 EUR in total, most of it the ESP32. The HC-SR04 itself is 1–2 EUR.

On the resistor values: 2 kOhm is not an E12 standard value, so most kits do not have one. 1 kOhm / 2.2 kOhm gives 3.44 V on the node. If you want to stay under the 3.3 V rail, 2.2 kOhm / 3.3 kOhm gives 3.0 V and both values are in every kit.

3. Wiring



HC-SR04 pin	ESP32 pin	Wire	Note
VCC	5V / VIN	red	Full 5 V — on 3.3 V the range shrinks and the signal gets noisy
Trig	GPIO 5	blue	3.3 V drive from the ESP32 is enough
Echo	divider node → GPIO 18	yellow	5 V output — **must** be divided first
GND	GND	black	Common ground, connect this first

The divider

```
Echo ──[ R1 = 1 kOhm ]──┬──[ R2 = 2.2 kOhm ]── GND
                        │
                        └── ESP32 GPIO 18
```

V(node) = 5 V × R2 / (R1 + R2) = 5 × 2.2 / 3.2 = **3.44 V**

Order of assembly: USB unplugged → GND → VCC → Trig → build the divider → node to GPIO 18 → check VCC/GND are not swapped → plug in USB.

Safety notes

- Never wire Echo straight to a GPIO. The divider is not optional; 5 V on a 3.3 V pin degrades it over time.
- Do not use GPIO 6–11 — the ESP32 wires them to its flash chip. GPIO 5 and 18 are safe general-purpose pins.
- Give the sensor a real 5 V supply. USB power through the ESP32's 5V/VIN pin is fine.
- Aim at a flat surface of at least 0.5 m² for a clean echo.

---

4. Test sketch — full listing

`firmware/hc-sr04_test/hc-sr04_test.ino` · Arduino IDE, Board = **ESP32 Dev Module**, Serial Monitor at **115200 baud**.

```
/*
 * HC-SR04 Ultrasonic Distance — Test Sketch
 * OmArTronics · Project 01 · 2026-07-25 · MIT License
 *
 * Prints distance in cm over Serial at 115200 baud.
 * Wiring: Trig -> GPIO 5, Echo -> GPIO 18 through a 1k/2.2k divider (5V -> 3.44V),
 *         VCC -> 5V, GND -> GND.  See wiring.md.
 *
 * Single file — just open in the Arduino IDE and upload. Change pins below.
 */

// ===== Settings (edit these) =====
#define TRIG_PIN            5       // ESP32 GPIO 5  -> HC-SR04 Trig
#define ECHO_PIN            18      // ESP32 GPIO 18 <- Echo via 1k/2.2k divider
#define READING_INTERVAL_MS 60      // datasheet suggests >= 60 ms between pings
#define ECHO_TIMEOUT_US     25000   // ~430 cm round trip; pulseIn gives up after this
#define SERIAL_BAUD         115200
#define SOUND_CM_PER_US     0.0343f // speed of sound at 20 C, cm per microsecond
#define DIST_MIN_CM         2.0f    // datasheet minimum
```

```
#define DIST_MAX_CM         400.0f  // datasheet maximum
#define CALIBRATION_OFFSET  0.0f    // cm added after conversion (tune vs a ruler)
#define CALIBRATION_SCALE   1.0f    // multiply reading (1.0 = no change)
// ================================

// One measurement. Returns distance in cm, or NAN if the reading is invalid.
float readDistanceCm() {
  // 10 us trigger pulse tells the module to fire an 8-cycle 40 kHz burst.
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // pulseIn measures how long Echo stays HIGH, in microseconds.
  // It returns 0 if no echo arrives before the timeout.
  unsigned long echo_us = pulseIn(ECHO_PIN, HIGH, ECHO_TIMEOUT_US);
  if (echo_us == 0) return NAN;                 // timeout: nothing in range

  // Convert time of flight to distance (divide by 2 for the round trip).
  float cm = (echo_us * SOUND_CM_PER_US) / 2.0f;
  cm = cm * CALIBRATION_SCALE + CALIBRATION_OFFSET;

  // Reject readings outside the datasheet range.
  if (cm < DIST_MIN_CM || cm > DIST_MAX_CM) return NAN;
  return cm;
}

void setup() {
  Serial.begin(SERIAL_BAUD);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  digitalWrite(TRIG_PIN, LOW);
  delay(50);
  Serial.println(F("HC-SR04 test ready. Distance in cm:"));
}

void loop() {
  float cm = readDistanceCm();
  if (isnan(cm)) {
    Serial.println(F("-- out of range / no echo --"));
  } else {
    Serial.print(cm, 1);
    Serial.println(F(" cm"));
  }
  delay(READING_INTERVAL_MS);
}
```

Expected Serial output, hand moving toward the sensor:

```
HC-SR04 test ready. Distance in cm:
34.2 cm
28.9 cm
19.9 cm
11.3 cm
-- out of range / no echo --
```

If this is wrong, the wiring is wrong. No web server will fix it.

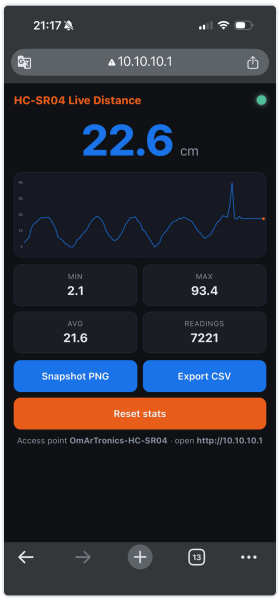
---

5. Live web dashboard — five steps

`firmware/hc-sr04_plotter/hc-sr04_plotter.ino` · needs the **ArduinoJson** library.

1. Open the plotter sketch and upload it. One file, one upload — the dashboard is compiled into the sketch as `PROGMEM`, so there is no filesystem, no plugin and no second upload.

2. On your phone, join the WiFi network `OmArTronics-HC-SR04` (password `ultrasonic`).
3. Open `http://10.10.10.1` in the browser. Plain `http`, not `https`.
4. Read the live value, the rolling chart of the last 100 readings, and the min / max / average / count cards. The page polls five times a second.
5. Use the buttons: **snapshot** saves the chart as PNG, **export** downloads the buffered readings as CSV, **reset** clears the statistics on the ESP32.



What the sensor actually did

Measured on this build, from the CSV export and dashboard statistics — not from the datasheet.

Observation	Measured
Motionless wall at ~89 cm, 24 readings in 1.55 s	88.7 – 89.6 cm · median 89.1 · σ 1.8 mm · **9 mm peak-to-peak**
Close range, 2.4 s window at ~8 cm	mean 7.67 cm · 6 mm spread
Single-sample outlier in a 100-sample export	22.3 → **32.7** → 23.5 cm, target unmoved
Session minimum	2.1 cm (partly the sketch's own `DIST_MIN_CM` filter)
Session maximum	93.4 cm (an earlier run peaked at 101.4 cm)
Serial sample spacing	median 78 ms against a configured 60 ms
CSV sample spacing	median 205 ms — that file holds what the *browser polled*

The 9 mm spread is three times the resolution the datasheet advertises, on a target that never moved. Quoting an HC-SR04 reading to two decimals is fiction. **If your project acts on distance, take a median of three or five readings.**

Not measured here, predicted from the datasheet: soft or angled targets scatter the echo and drop out; the ~15° cone reports the nearest object anywhere in it; the fixed 0.0343 constant drifts with temperature; two modules firing at once can hear each other.

6. Troubleshooting

Symptom	Likely cause	Fix
Always `-- out of range / no echo --`	Trig/Echo on the wrong pins, or the divider node never reaches GPIO 18	Re-check GPIO 5 and GPIO 18; check continuity from the R1/R2 junction to GPIO 18
Readings are 0 or nonsense	VCC on 3.3 V, or VCC and GND swapped	Move VCC to 5V/VIN; verify a common ground
Values jump several cm on a still target	Single-sample noise; soft or tilted target	Median-filter 3–5 samples; aim square at a flat, hard surface
Occasional lone spike (e.g. 32.7 cm)	One false echo	Reject samples that differ from the running median by more than a threshold
Nothing below ~2 cm	Transducer still ringing from its own burst, plus the `DIST_MIN_CM` filter	Expected behaviour — design around the dead zone
`ArduinoJson.h: No such file or directory`	Library not installed	Library Manager → install **ArduinoJson**
Upload fails, port busy	Serial Monitor holds the COM port	Close the Serial Monitor, replug, upload again
Phone on the AP but the page will not load	Phone fell back to mobile data, or `https` was typed	Turn mobile data off for the test; use `http://10.10.10.1`

7. Files and links

File	What it is
`firmware/hc-sr04_test/`	Serial test sketch (section 4)
`firmware/hc-sr04_plotter/`	Live web-plotter, dashboard included, one upload
`firmware/hc-sr04_plotter/web_src/`	The dashboard as editable HTML / CSS / JS
`electronics/schematic.svg`	Wiring diagram with the Echo divider
`electronics/wiring.md`	Pin table, divider maths, assembly order
`cad/OmBlock HC-SR04.stl`	3D-printable sensor holder for the OmGrid
`animation/hc-sr04_animation.html`	Interactive time-of-flight animation

- **Full tutorial:** https://omartronics.com/hc-sr04-ultrasonic-sensor-esp32/
- **Datasheet:** https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
- **Next in the series:** PIR HC-SR501 — motion without a single sound wave.