

● WHITEPAPER · VAHA ENVIRONMENTAL SENSORS

Environmental sensors for VAHA: wiring, how readings reach the app, and two TVOC data-integrity gaps

AUTHORS

Ritesh Bonthalakoti

REVIEWER

Bharat Meda

VERSION

v1.0

DATE

2026-10-22

STATUS

Published

DISTRIBUTION

Public

CONTENTS

Abstract	6 Analysis and discussion
1 Introduction and objective	7 Limitations and threats to validity
2 Requirements and acceptance criteria	8 TRL assessment
3 System under test	9 Conclusions and next steps
4 Method	10 References
5 Results	Appendices

ABSTRACT

Abstract

VAHA stamps every voice note with the room's conditions when it was recorded: temperature, humidity, total volatile organic compounds (TVOC) and water flow. This paper describes how the three sensors are wired to the Arduino UNO Q, how a reading travels from them to the companion app, and checks the current build, as of 2026-08-17, for whether a bad reading can pass as a good one. For temperature and humidity it cannot: a failed read is sent as "nan" and stored as no value. For TVOC it can, in two ways. A failed I2C read returns a plain number, 7777 or 8888 plus an error code, which both Linux-side parsers store as a reading; run through the device server's code, both came back from its API unchanged and unflagged (measured). And a successful read is decoded without the sensor's checksum and with its status byte kept as the top 8 bits, so a non-zero status adds 16,777,216 ppb and a corrupted reply can yield a plausible wrong value (calculated). No sensor's accuracy and no delivery latency has been measured. The sensor path is at technology readiness level (TRL) 3.

KEY POINT

Key findings

- **TVOC failures stored as readings:** a failed read returns 7777 or 8888 + error code, stored and served as a real ppb value. The DHT22 path in the same function reports failure correctly.
- **TVOC decoded without its checksum:** the status byte is kept in the value and the CRC is never checked, so a set status flag or a corrupted reply gives a wrong reading with no sign of it.
- **Two parsers:** a capture's sensor snapshot is parsed in `main.py`; the live readings in `sensor_service.py`. Neither checks the TVOC value.
- **Not measured yet:** sensor accuracy, delivery latency, and the TVOC failure on real hardware.

SECTION 1

Introduction and objective

Every capture VAHA records carries a snapshot of the room alongside the audio: the project report describes each note as "stamped with water, temperature, humidity, and the TVOC reading" (reference 1). The snapshot is only as useful as its weakest reading, and a reading that is wrong without looking wrong is worse than a missing one, because nothing downstream can tell.

This paper answers three questions:

1. How are the sensors wired, and how does a reading get from each sensor to the app?
2. Can a failed or corrupted reading be told apart from a valid one, anywhere along the way?
3. Which accuracy and latency figures exist, and which have been measured?

In scope: the three sensors and their wiring; the microcontroller sketch; the bridge to the Linux side; both Linux-side parsers; how readings are stored with captures and served to the app; and the TVOC sensor's reply format as its datasheet gives it.

Out of scope: calibration of any sensor against a reference; long-term drift; the app's display of sensor data; and the sync of captures to the app, covered in the sync paper (reference 6).

SECTION 2

Requirements and acceptance criteria

The requirements come from the product README, the project report and the product's purpose. They were written down for this paper, after the code, so they state what the build is checked against, not acceptance criteria set before testing (section 8).

ID	Requirement	Criterion	Basis
R1	Readings delivered quickly	Sensor readings reach the app within a stated latency	README: "WebSocket Telemetry Latency: <10 ms latency for real-time sensor updates"
R2	Readings represent the room	Each reading is within the sensor's rated accuracy	Project report: every capture is stamped with the room's readings
R3	Failures and corrupt reads flagged	A failed or corrupted read is never stored or shown as a valid reading	Implied by R2; the DHT22 path already does this

SECTION 3

System under test

3.1 Sensors and wiring

Three sensors connect to the microcontroller side of the Arduino UNO Q (Figures 1 and 2, Table 3.1). The DHT22 gives temperature and humidity over a single-wire digital line. The AGS02MA gives TVOC over I2C; the sketch runs the bus at 20 kHz for it, within the sensor's 30 kHz limit (reference 2). The water-flow sensor gives a pulse per unit of water through a hall-effect switch, counted on an interrupt.

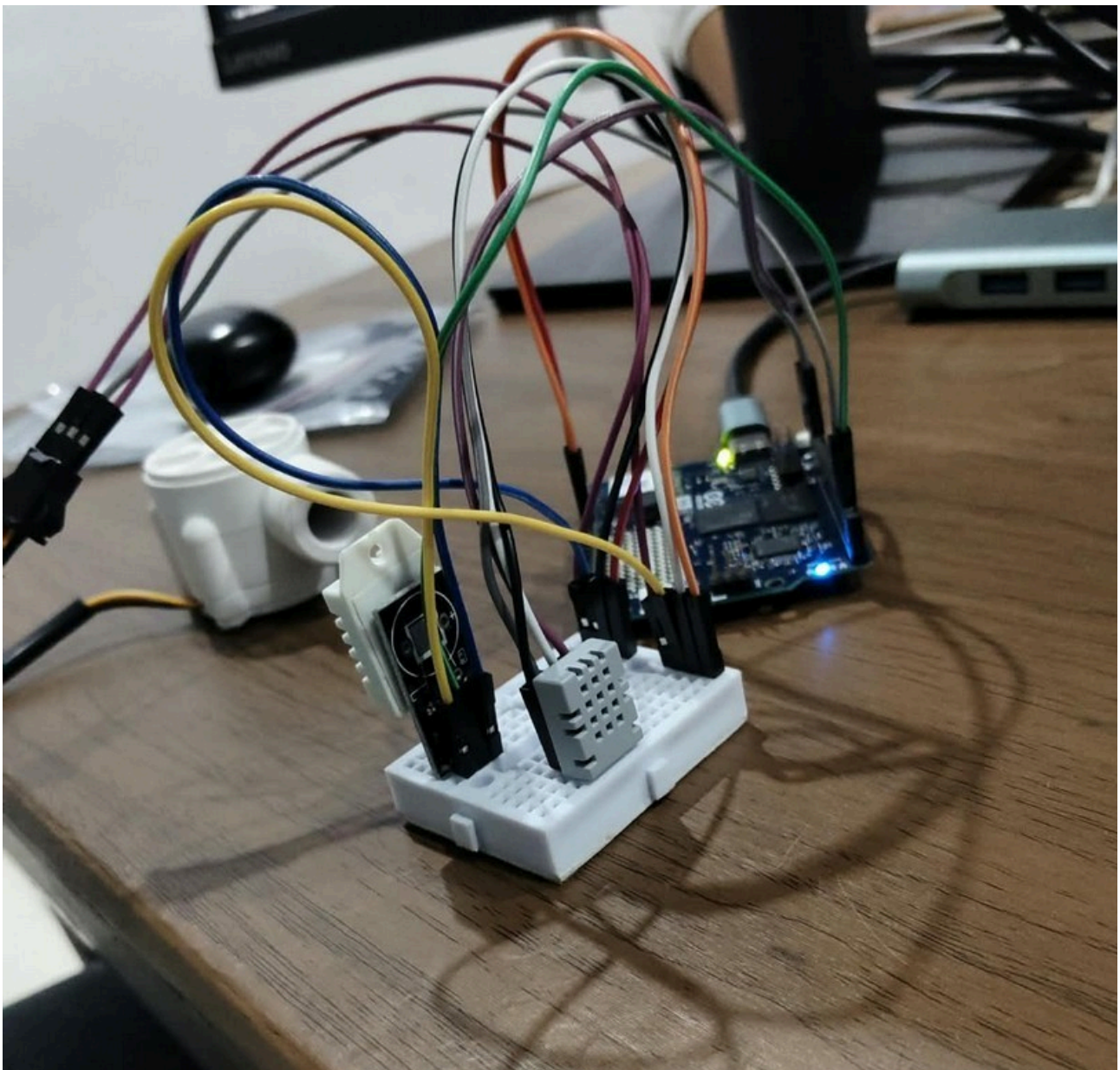


Figure 1 The three sensors on a breadboard during bring-up: the DHT22 and AGS02MA on the breadboard, the Arduino UNO Q at right and the flow sensor's white housing at upper left. From the VAHA repository (reference 1).

● SENSOR WIRING, AS BUILT

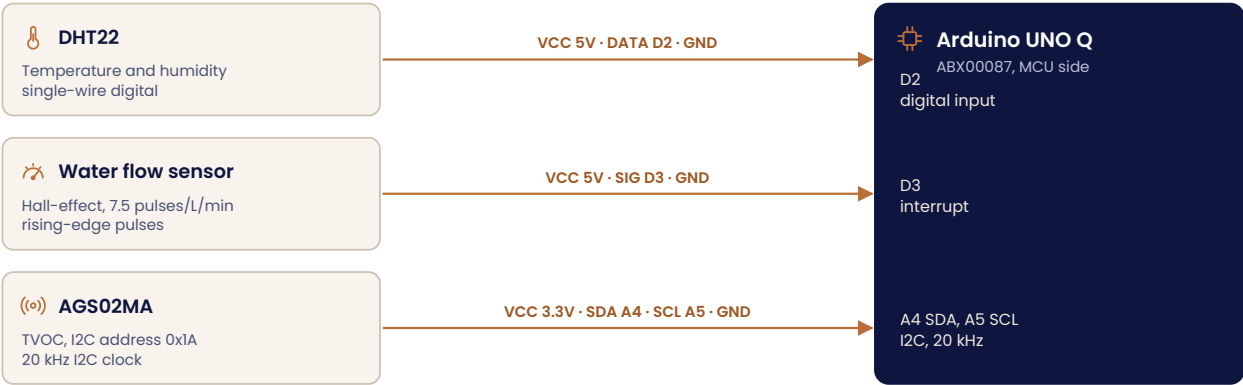


Figure 2 Sensor wiring, from the README's pin table and the pin definitions in `sketch.ino` (lines 5–8).

Table 3.1: Pin connections (README "Hardware BOM & Wiring", reference 1)

Sensor	Module pin	Arduino UNO Q pin	Connection	Notes
DHT22	VCC / DATA / GND	5V / D2 / GND	Power / digital input / ground	Read on each request
Water flow sensor	VCC / SIG / GND	5V / D3 / GND	Power / interrupt / ground	Rising-edge pulses; 7.5 pulses per L/min (datasheet, as quoted in the README)
AGS02MA	VCC / SDA / SCL / GND	3.3V / A4 / A5 / GND	Power / I2C data / I2C clock / ground	I2C address 0x1A, 20 kHz clock

3.2 How a reading reaches the app

On the microcontroller, `sensors_get()` reads all three sensors and returns one string, `T:...:H:...:F:...:L:...:V:...:..`, over the board's router bridge to the Linux side (Figure 3, Table 3.2). Two Linux-side functions call it, and each has its own copy of the parser. At each capture, `main.py` reads the sensors once and stores the result with the capture, which the app later syncs. Separately, `sensor_service.py` reads them every 30 s and keeps the last 100 readings, about 50 minutes (calculated), for the live readings the app shows.

● FROM SENSOR TO APP: TWO PATHS ON THE LINUX SIDE

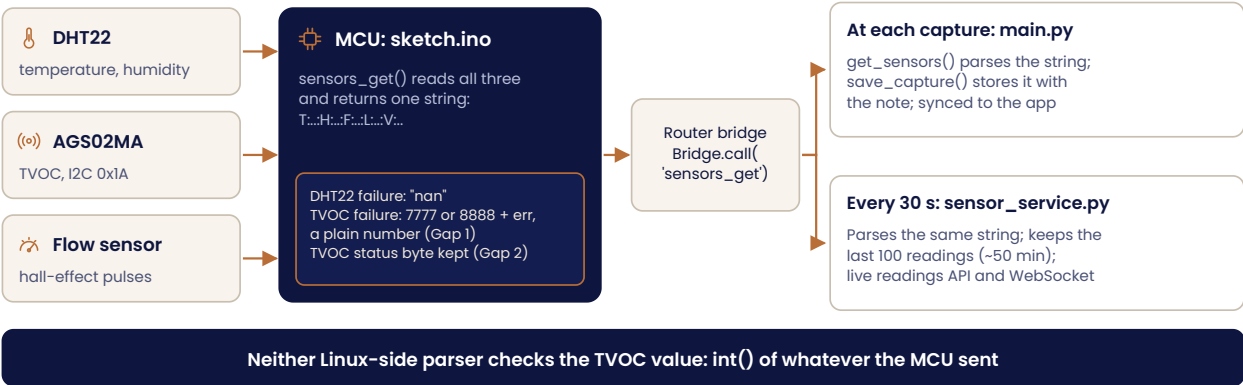


Figure 3 From sensor to app, as the code runs it. The two Linux-side paths parse the same string in the same way.

Table 3.2: A reading's path as implemented (source locations in Appendix A)

Step	What happens	Source
1	DHT22 temperature and humidity are read; a failed read (<code>isnan()</code>) is sent as the text <code>nan</code>	<code>sketch.ino</code> lines 53–54, 57–58
2	Flow pulses since the last call are converted to litres per minute and added to a running total	<code>sketch.ino</code> lines 44–52
3	<code>readTVOC()</code> asks the AGS02MA for register 0x00. On an I2C transmission error it returns <code>8888 + err</code> ; if fewer than 5 bytes arrive, <code>7777</code> ; otherwise bytes 0 to 3 as one 32-bit number. Byte 4 is never read into the value	<code>sketch.ino</code> lines 17–41
4	The values are joined into one string and returned over the bridge	<code>sketch.ino</code> lines 55–62
5	At each capture, <code>get_sensors()</code> parses the string: <code>nan</code> becomes no value; TVOC is converted with <code>int()</code> . The result is stored with the capture	<code>main.py</code> lines 300–316, 839–856; <code>capture_service.py</code> line 237
6	Every 30 s, <code>sensor_service.py</code> parses the string the same way and keeps the reading for the live readings API and WebSocket	<code>sensor_service.py</code> lines 30–53; <code>config.py</code> lines 41, 44

3.3 Technology stack

Layer	Technology
Sensors	DHT22 (single-wire), AGS02MA (I2C, address 0x1A), hall-effect flow sensor (interrupt)
Microcontroller	Arduino UNO Q (ABX00087), MCU side, <code>sketch.ino</code>
Bridge	<code>Arduino_RouterBridge</code> , MCU to Linux on the same board
Linux side	Python: <code>main.py</code> at each capture; <code>sensor_service.py</code> every 30 s
Delivery	Captures synced to the app; live readings over REST and WebSocket from the same server

● SECTION 4

Method

4.1 What was examined

Item	Details
Source	VAHA source repository (reference 1), as of its last change on 2026-08-17
Microcontroller	sketch/sketch.ino : readTVOC() , sensors_get()
Linux side	python/main.py get_sensors() ; python/server/services/sensor_service.py ; python/server/services/capture_service.py
Device	Arduino UNO Q (ABX00087), DHT22, AGS02MA, hall-effect flow sensor
Sensor documentation	AGS02MA datasheet (reference 2); Adafruit's AGS02MA library as a second reading of the reply format (reference 3)
Test machine	Development computer, Windows on AMD64, Python 3.12.10; fastapi 0.141.1, starlette 1.6.0, uvicorn 0.52.4, pydantic 2.13.4, python-dotenv 1.2.2, python-multipart 0.0.32, numpy 2.5.2, pinned in analysis/requirements.txt

4.2 Procedure

The scripts and data for this paper are published with it (reference 5). They sit in the analysis/ and data/ folders of the VAHA repository, and paths such as analysis/tvoc_decode_example.py below are relative to its root. They were added on 2026-10-22 without changing the code this paper examines.

1. Follow each sensor's reading through sensors_get() , and compare how the DHT22 and TVOC paths handle a failed read.
2. Read both Linux-side parsers for any check on the TVOC value.
3. Compare readTVOC() 's decoding with the AGS02MA's reply format in its datasheet and in Adafruit's library.
4. Use the repository history to date when each behaviour was introduced (Appendix B).
5. Run the device server's code unchanged on the test machine. Call save_capture() with three sensor readings, as main.py passes them after parsing: a normal TVOC value, and each of readTVOC() 's two failure values. Read each back through GET /api/v1/captures (analysis/tvoc_placeholder_propagation_test.py).
6. Decode example AGS02MA replies both as readTVOC() does and as the datasheet describes (analysis/tvoc_decode_example.py).

Not yet done: a TVOC failure induced on the device; the AGS02MA's raw replies logged during warm-up; accuracy against a reference for any sensor; and delivery latency.

SECTION 5

Results

5.1 Known gaps in the current build

CAUTION**Gap 1: TVOC read failures are stored as readings**

`readTVOC()` has two failure returns: `8888 + err` when the I2C transmission fails, and `7777` when fewer than 5 bytes come back. Both are ordinary numbers of the same type as a real reading (Listing 1). A few lines above, `sensors_get()` already handles a DHT22 failure properly: it sends `nan`, and both parsers turn that into no value. The TVOC value gets no such treatment: it is appended whatever it is, and both parsers convert it with `int()` (Listing 2). A TVOC failure therefore reaches the capture and the live readings as, for example, `7777 ppb`. This has been the case since the sketch was written on 2026-07-08.

Listing 1: Failure returns in `readTVOC()`, and how `sensors_get()` sends each value

```

sketch/sketch.ino
22     uint8_t err = Wire.endTransmission(false);
23     if (err != 0) {
24         Wire.setClock(100000);
25         return 8888 + err;
26     }
...
28     uint8_t received = Wire.requestFrom((uint8_t)AGS02MA_ADDR, (uint8_t)5, (uint8_t>true);
29     if (received < 5) {
30         Wire.setClock(100000);
31         return 7777;
32     }
...
55     uint32_t tvoc = readTVOC();
57     s += isnan(temp) ? "T:nan" : "T:" + String(temp, 1);
58     s += isnan(hum) ? "H:nan" : "H:" + String(hum, 1);
...
61     s += "V:" + String(tvoc);

```

Listing 2: The two parsers, temperature and TVOC lines

```

python/main.py
308     'temp':          float(data['T']) if data.get('T','nan')!='nan' else None,
312     'tvoc_ppb':      int(data.get('V',0))

python/server/services/sensor_service.py
45         'temp':          float(data['T']) if data.get('T', 'nan') != 'nan' else None,
49         'tvoc_ppb':      int(data.get('V', 0)),

```

CAUTION

Gap 2: TVOC replies are decoded without their checksum, and with the status byte in the value

The AGS02MA replies to a read of register 0x00 with 5 bytes: a status byte, the TVOC value in 24 bits, and a CRC checksum byte (reference 2). Adafruit's library reads it the same way: it rejects a reply whose CRC does not match and keeps only the low 24 bits (reference 3). `readTVOC()` builds its value from bytes 0 to 3, so the status byte becomes the top 8 bits, and it never checks byte 4 (Listing 3). Any non-zero status byte therefore adds at least 16,777,216 ppb to the reading, and a reply damaged on the bus is accepted as long as 5 bytes arrive (Table 5.1). The datasheet gives the status byte's bits: bits 7–4 reserved (0); bits 3–1 the data type `CH[2:0]`, 000 for ppb, the default after power-on; bit 0 `RDY`, 0 when the reading is ready and 1 when it is not, including during the datasheet's required 120 s warm-up after power-on (reference 2) — the obvious time for the status byte `readTVOC()` folds into the value to be non-zero. The one TVOC reading on record, 96 ppb after 6 minutes of uptime in the project report's app screenshot (reference 1), is plausible, so the status byte was zero then. The datasheet also asks for at least 1.5 s between reads, or `RDY` stays at 1 (reference 2); `main.py` and `sensor_service.py` read the sensor independently of each other, so nothing keeps their reads apart.

Listing 3: How `readTVOC()` decodes a reply

```
sketch/sketch.ino
33   uint8_t data[5];
34   for (int i = 0; i < 5; i++) data[i] = Wire.read();
35   Wire.setClock(100000);
36   uint32_t ppb = ((uint32_t)data[0] << 24) |
37                 ((uint32_t)data[1] << 16) |
38                 ((uint32_t)data[2] << 8) |
39                 (uint32_t)data[3];
40   return ppb;
```

Table 5.1: Example replies decoded both ways (calculated, `analysis/tvoc_decode_example.py`; constructed replies, not captured from a sensor)

Reply (hex)	What it is	<code>readTVOC()</code> reports	Datasheet layout gives
00 00 00 60 6C	96 ppb, status 0x00	96 ppb	96 ppb
01 00 00 60 F7	96 ppb, status 0x01	16,777,312 ppb	96 ppb, status 0x01
00 00 00 20 6C	96 ppb reply with one bit flipped on the bus	32 ppb	Rejected: CRC mismatch

The last row is the harder case to catch: a damaged reply produces a plausible number.

Table 5.2: Gaps summary

Gap	Where	Since	Consequence
TVOC failures stored as readings	<code>sketch.ino</code> lines 25, 31, 61; <code>main.py</code> line 312; <code>sensor_service.py</code> line 49	2026–07–08	A failed TVOC read is stored with the capture and shown live as a reading, such as 7777 ppb

Gap	Where	Since	Consequence
TVOC decoded without CRC, status byte kept	sketch.ino lines 33–40	2026–07–08	A set status flag adds at least 16,777,216 ppb; a damaged reply can give a plausible wrong value

5.2 What the server does with a failure value

The test starts where `main.py` hands the parsed reading to `save_capture()`, with the same arguments, and reads the capture back from the API. It shows what storage and the API do with a failure value; what the parsers do is shown by their code (Listing 2).

Table 5.3: TVOC values read back from the device server's API (measured, one run per case, `data/raw/tvoc_placeholder_propagation_2026-10-22.csv`)

Case	TVOC given to <code>save_capture()</code>	<code>voc_parts_per_billion</code> from the API	Any flag in the response
Normal reading	142	142	None
Short read (7777)	7777	7777	None
Transmission error, <code>err = 3 (8888 + 3)</code>	8891	8891	None

All three came back exactly as given. Nothing in the response marks the second and third as anything but measurements.

SECTION 6

Analysis and discussion

6.1 Results against the requirements

Table 6.1: Requirement conformance

ID	Criterion	Result	Status
R1	Readings reach the app within a stated latency	The README's under 10 ms has no recorded method; not measured for this paper	Not measured
R2	Each reading within the sensor's rated accuracy	No sensor compared with a reference; Gap 2 can make TVOC readings wrong	Not measured
R3	A failed or corrupted read is never stored as valid	TVOC failures and damaged replies are stored as readings (Gaps 1 and 2; Table 5.3)	Not met

6.2 Why the TVOC gaps matter

Each note keeps its sensor snapshot for good, and the TVOC values in it look like any other number. Nothing in a capture, in the API, or in a Notion export of the note (`main.py` line 328) separates 7777 ppb from a real 7777 ppb, and a damaged reply such as the 32 ppb in Table 5.1 is indistinguishable even in principle without the CRC. A user looking back at air quality across their notes, or any analysis of that history, inherits these values silently.

The pattern that would have prevented Gap 1 sits three lines above it: the DHT22 values are checked with `isnan()` and sent as `nan`, and both parsers already turn `nan` into no value. The TVOC value never got the same treatment, and because the function looks handled, part of it being handled, the omission is easy to miss.

6.3 Closing the gaps

The sketch below is written against the current code and has not been built or tested. On the microcontroller, `readTVOC()` reports success separately from the value, checks the CRC and keeps only the 24-bit reading; a failure is sent as `nan`, as for the DHT22:

```
// Returns true only for a complete reply with a matching CRC and a ready status
bool readTVOC(uint32_t &ppb) {
    // ... request register 0x00 as now; return false on err != 0 or received < 5
    uint8_t d[5];
    for (int i = 0; i < 5; i++) d[i] = Wire.read();
    if (crc8(d, 4) != d[4]) return false;           // CRC-8, polynomial 0x31, initial 0xFF (helper to
add)
    if (d[0] != 0) return false;                   // status: RDY (bit 0) 1 = not ready; CH[2:0] (bits
3-1) not 000 = not ppb; bits 7-4 reserved
    ppb = ((uint32_t)d[1] << 16) | ((uint32_t)d[2] << 8) | d[3];
    return true;
}

// in sensors_get()
uint32_t tvoc;
s += readTVOC(tvoc) ? ":V:" + String(tvoc) : ":V:nan";
```

On the Linux side, both parsers treat TVOC as they already treat temperature, and should share one function rather than two copies:

```
'tvoc_ppb': int(data['V']) if data.get('V', 'nan') != 'nan' else None,
```

A test that sends a reply with a wrong CRC, and one with no reply, and expects no TVOC value in the capture, keeps the fix from being undone.

SECTION 7

Limitations and threats to validity

- **No TVOC failure on real hardware.** The gaps are read from the code and the datasheet. Table 5.3 shows what the server does with a failure value once one exists, not that the AGS02MA produces one under a given fault.
- **Status byte not observed.** Whether and when the AGS02MA sets its status byte has not been logged; Gap 2's effect depends on it.
- **Decode examples are constructed.** Table 5.1's replies are built from the datasheet's layout, not captured from the sensor.
- **Server tested off the device.** Table 5.3 ran on a development computer, one run per case, starting after the parser.

- **No accuracy data.** No sensor has been compared with a reference instrument; accuracy figures are the manufacturers' only.
- **Latency not measured.** The README's under 10 ms has no recorded method, sample size or spread.
- **Flow rig informal.** The bench rig in Figure 6 has been used to check that the flow sensor counts, not to calibrate it.
- **Requirements set after the code.** R1 to R3 come from the README, the project report and the product's purpose.

SECTION 8

TRL assessment

TECHNOLOGY READINESS: WHERE THIS WORK SITS

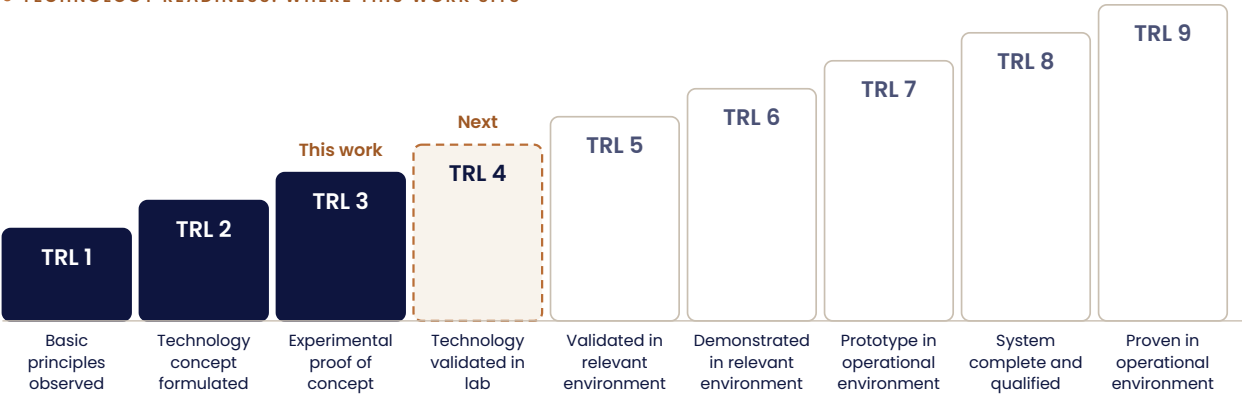


Figure 4 Where this work sits on the EU Horizon TRL scale. Definitions from the Dharanova publishing guidelines (EU Horizon TRL scale, ISO 16290 criteria, reference 4).

TRL demonstrated	Below TRL 4. The evidence supports an experimental proof of concept (EU Horizon TRL 3) for the sensor path
Integration level	Component: sensors, sketch, bridge and Linux-side parsing
Fidelity	Prototype: real sensors, real firmware, real server code
Environment	Lab bench; server code on a development computer
Evidence	Sections 3 and 5: the code, the datasheet, and the server test
Why not TRL 4	TRL 4 requires validation in the lab against defined criteria. No accuracy or latency limits are set, nothing has been measured against them, and R3 is not met
Gaps to TRL 4	Set limits first, then close each requirement and test it, as shown in Figure 5
Gaps beyond TRL 4	TRL 5 needs the relevant environment defined (a candidate: a bathroom, with shower humidity, fan noise and cleaning products) and the sensors tested there

● PATH FROM THIS WORK TO TRL 4

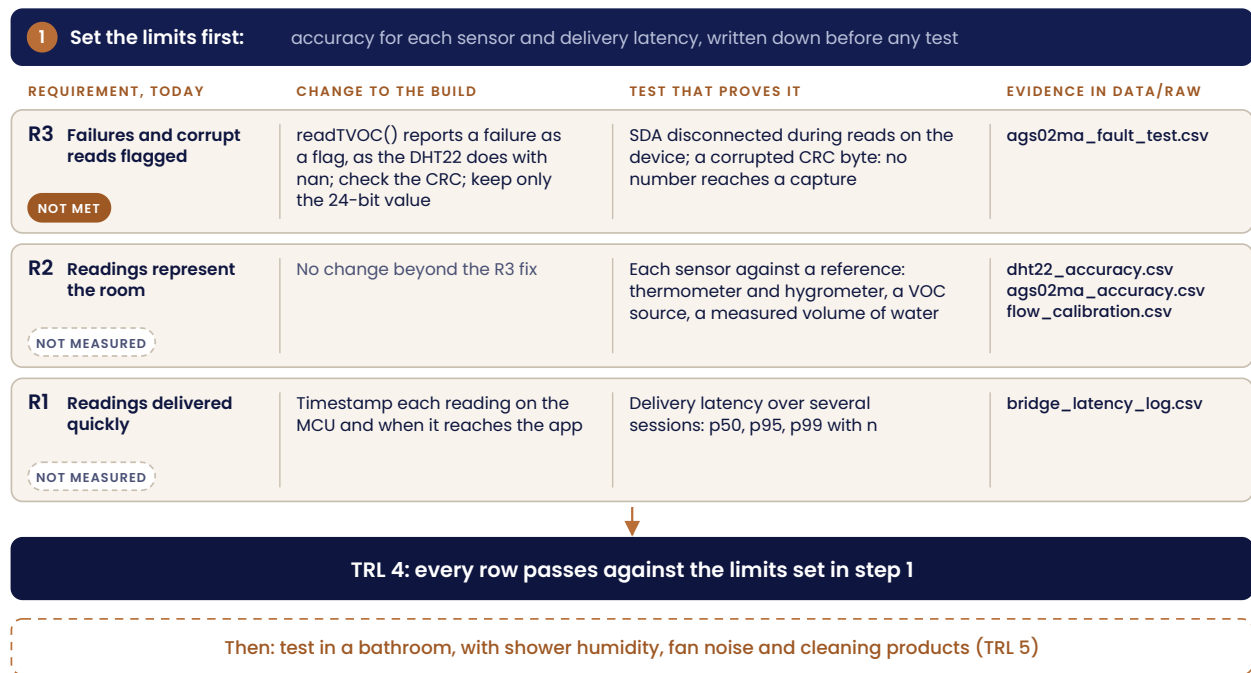


Figure 5 Path from this work to TRL 4, one row per requirement in priority order: its status today (Table 6.1), the change to the build, the test that shows it is met, and the data the test leaves in `data/raw`.

● SECTION 9

Conclusions and next steps

The sensors are wired and integrated, and their readings reach every capture and the app's live view. Temperature and humidity failures are handled; TVOC failures and damaged replies are not, and are stored as readings. Neither accuracy nor delivery latency has been measured. The sensor path is a proof of concept, below TRL 4.

Next steps, in the order of Figure 5:

- 1. Set limits** for each sensor's accuracy and for delivery latency.
- 2. Fix** `readTVOC()` as in section 6.3, and have both parsers treat a TVOC `nan` as no value, from one shared function.
- 3. Fault the AGS02MA on the device:** disconnect SDA during reads, and confirm that no TVOC value reaches a capture. Log the raw 5-byte replies from power-on through warm-up to see when the status byte is set.
- 4. Compare each sensor with a reference:** a thermometer and hygrometer for the DHT22, a known VOC source for the AGS02MA, and a measured volume of water for the flow sensor.
- 5. Measure delivery latency:** timestamp each reading on the microcontroller and on arrival in the app, over several sessions, and report p50, p95 and p99 with n.
- 6. Publish the results as a new version** of this paper.

SECTION 10

References

1. Dharanova, VAHA source repository, <https://github.com/dharanova-code/vaha>. README sections "Hardware BOM & Wiring" and "Performance Metrics"; project report `docs/VAHA Report.pdf`; source files as listed in Appendix A.
2. Aosong (ASAIR), *AGS02MA TVOC sensor datasheet*, June 2022 edition (no separate revision number given): reply format of register 0x00, status byte (§3.2), CRC (§3.7), warm-up time (Table 1), I2C clock limit (§3) and minimum interval between reads (§3.8).
3. Adafruit Industries, *Adafruit_AGS02MA* Arduino library, https://github.com/adafruit/Adafruit_AGS02MA, `getTVOC()` and `_readReg()`.
4. ISO 16290:2013, *Space systems: Definition of the Technology Readiness Levels (TRLs) and their criteria of assessment*.
5. Scripts and data for this paper, in the VAHA source repository (reference 1): `analysis/` and `data/`.
6. Dharanova, *Device-to-app sync for VAHA: architecture, and the integrity and security of the current build*, v1.0, 2026-10-22.

APPENDICES

Appendices

A. Source locations

All paths are in the VAHA source repository (reference 1). Line numbers refer to the repository as of 2026-08-17.

Table A.1: Files and lines cited

File	Lines	What is there
<code>sketch/sketch.ino</code>	5–8	Pins and the AGS02MA's I2C address
	17–41	<code>readTVOC()</code> : failure returns at 25 and 31; decoding at 33–40
	43–63	<code>sensors_get()</code> : DHT22 <code>nan</code> handling at 57–58; TVOC appended at 61
<code>python/main.py</code>	300–316	<code>get_sensors()</code> : parser used for each capture; TVOC <code>int()</code> at 312
	328	TVOC written into the Notion export line
	839–856	Sensors read and stored with each capture
<code>python/server/services/sensor_service.py</code>	30–53	Parser for live readings; TVOC <code>int()</code> at 49
<code>python/server/services/capture_service.py</code>	237	TVOC stored as <code>voc_parts_per_billion</code>

File	Lines	What is there
python/server/config.py	41, 44	Live readings kept: 100; poll interval: 30 s
README.md	198–221	Hardware BOM and pin table
	249	"WebSocket Telemetry Latency: <10 ms"

B. When each behaviour was introduced

Table B.1: Sensor path history, from the repository

Date	Change
2026-07-08	Sketch written, with <code>readTV0C()</code> 's failure returns and decoding and the DHT22 <code>nan</code> handling as they are today
2026-07-24	I2C stop bit and flow-interrupt handling changed and reverted several times to resolve lock-ups; by the end of the day the sketch was back to its 2026-07-08 state
2026-07-27	Sketch file renamed for the Arduino command-line tools, then renamed back; no change to its code
2026-08-17	Last change to the repository; the sketch is unchanged since 2026-07-08

C. Figures quoted in the product documentation

Table C.1: Reported figures (method unknown)

Metric	Value	Source
WebSocket telemetry latency	under 10 ms	README line 249 (reference 1)
Flow sensor pulse rate	7.5 pulses per L/min	Datasheet, as quoted in the README
DHT22, AGS02MA accuracy	Not verified	Manufacturers' specifications only

No script, log or note in the repository states how the latency was measured, over how many readings, or between which two points.

D. Hardware, as built

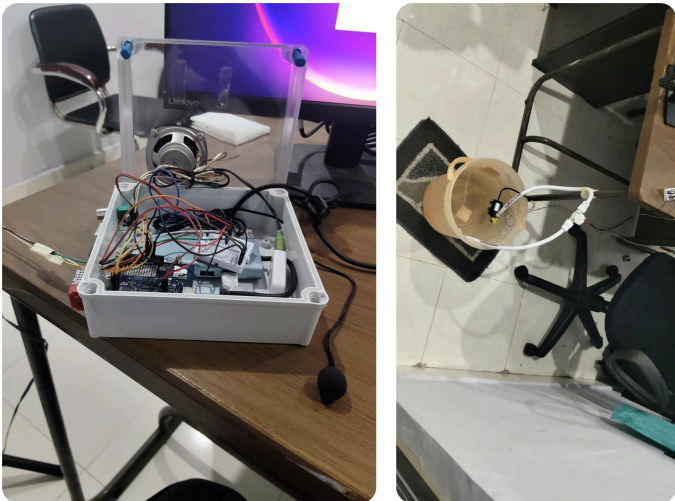


Figure 6 The assembled hardware in its enclosure (Arduino UNO Q, breadboard, speaker and sensor wiring), and the flow sensor's bench rig, a bucket and tubing used to check that the sensor counts (section 7). From the VAHA repository (reference 1).

E. Revision history

Version	Date	Author	Changes
v0.1	2026-10-19	Ritesh Bonthalakoti	First draft
v0.2	2026-10-22	Ritesh Bonthalakoti	Rebuilt on the publication template; TVOC failure values run through the device server; DHT22 and TVOC handling compared; photos, TRL assessment and diagrams added
v0.3	2026-10-22	Ritesh Bonthalakoti	Added Gap 2: TVOC decoded without CRC and with the status byte, from the AGS02MA datasheet. Corrected: captures are parsed in <code>main.py</code> , live readings in <code>sensor_service.py</code> ; the README's under 10 ms is WebSocket telemetry latency, not bridge latency; the sketch has not changed since 2026-07-08. R1 and R2 restated as not measured. Logic simulation of <code>readTVOC()</code> dropped as evidence. Code screenshots replaced by listings; diagrams drawn by script; section 6.3 on closing the gaps added; internal R&D log removed as a source
v1.0	2026-10-22	Ritesh Bonthalakoti	First published version. Package versions, AGS02MA status-byte bits and datasheet edition recorded; datasheet's minimum interval between reads added to Gap 2. Reviewed by Bharat Meda



We build intelligence into everyday objects.

● HOW TO CITE

Ritesh Bonthalakoti. *Environmental sensors for VAHA: wiring, how readings reach the app, and two TVOC data-integrity gaps*. Dharanova whitepaper, v1.0, 2026.

dharanova.com · info@dharanova.com