

Virtual Instrumentation-Based Simulation and Monitoring of A Control Valve in A Water Flow System Using ESP32 and MATLAB

Ishayu Basu¹, Adhrit Ghosh², Ananya Misra³, Dr. Saravanakumar R⁴

Department Of Control And Automation School Of Electrical Engineering Vellore Institute Of Technology Vellore, India

Abstract—This paper presents a low-cost virtual instrument for observing the operation of a manually actuated control valve in a water flow system. Two digital vibration sensors are clamped to a pipe section near the valve and connected to the GPIO pins of an ESP32 microcontroller. The ESP32 samples both outputs, attaches a millisecond timestamp to every sample and streams the result as comma-separated text over a USB virtual serial port at 115200 baud. A MATLAB application, named AQUAGUARD, reads and parses the stream and presents a software front panel consisting of a status card for each sensor, two live strip charts over a 15-second moving window and a footer line that summarises the combined sensor state. The rationale is that flow through a partly open valve generates turbulence, which excites pipe-wall vibration that a pipe-mounted sensor can detect as a change of state. Because the sensors give only a binary output, the instrument reports a change in vibration state rather than a flow rate, and no leak diagnosis is claimed. With the valve closed, the dashboard consistently received Sensor 1 = LOW and Sensor 2 = HIGH, which confirmed correct acquisition, serial parsing and display. A simulation mode of the dashboard, driven by synthetic data, is also used to demonstrate the panel behaviour for every combination of sensor states and for pulsed inputs. We also identify a limitation of the present firmware, which transmits only when at least one sensor is HIGH and therefore leaves a stale reading on the panel when both sensors fall LOW, and we describe the correction. The paper documents the architecture, hardware set-up, firmware, software design, operating procedure and a roadmap towards analog vibration sensing, data logging and closed-loop PID valve control.

Index Terms—Virtual instrumentation, control valve, ESP32, MAT-LAB, flow-induced vibration, vibration sensor, serial data acquisition, pipeline monitoring.

I. INTRODUCTION

Water distribution, process plants, irrigation networks and building services all depend on control valves to start, stop and regulate the flow of liquid through pipes. In an industrial loop the control valve is the final control element: it receives a signal from a controller and changes the size of the flow opening so that a process variable such as flow, pressure, level or temperature is held near its set point [1]– [3]. Its behaviour therefore sits at the centre of process performance, and a valve that sticks, leaks, cavitates or fails to follow its command degrades the whole loop [4], [5].

Observing what a valve is doing does not always require a flow meter. Water passing a partly open restriction

becomes turbulent and the resulting pressure fluctuations excite the pipe wall [6]– [8]. The vibration is generally stronger when flow is higher and weaker when the valve is closed, so a vibration sensor attached to the outside of the pipe can indicate whether the flow condition has changed without breaking into the line. Vibration and acoustic signatures are also used for pipeline fault and leak detection [9]– [11], which makes a vibration-based valve monitor a natural entry point into condition monitoring of piping systems.

Traditionally, displaying and recording such measurements needed dedicated bench instruments. The idea of virtual instrumentation replaces much of that hardware with a general-purpose computer, acquisition hardware and software, so that the front panel and the processing are defined by the user [12], [13]. This lowers cost, makes the instrument easy to modify and suits teaching laboratories and

prototyping. At the same time, inexpensive microcontrollers with integrated connectivity, notably the ESP32, have become widely used for data acquisition and Internet of Things (IoT) sensing [14]–[19], and MATLAB graphical in-terfaces have been used successfully to display microcontroller sensor data in real time [20], [21].

This work combines these ideas into a small virtual instrument for a water flow rig. An ESP32 reads two digital vibration sen-sors mounted near a manually operated control valve and streams timestamped states to a PC, where a MATLAB dashboard shows the live sensor states and graphs while the valve is opened and closed. The aim is a clear, working acquisition-to-display chain that can later be extended, rather than a calibrated flow meter or a leak-detection product. The contributions of this paper are:

- the design of a low-cost ESP32–MATLAB virtual instrument for observing valve-induced vibration state in a pipe, together with a photograph of the assembled laboratory prototype;
- a documented serial data format and a MATLAB front panel with sensor cards, 15-s strip charts and a combined-state footer;
- an experimental verification of the acquisition, parsing and display chain with the valve closed;
- 4. a set of simulation-mode dashboard captures that demonstrate the panel response to all combinations of sensor states and to pulsed inputs; and
- 5. a critical analysis of the firmware and sensor limitations together with concrete corrective and future work.

The remainder of the paper is organised as follows. Section II re-views related work. Section III summarises the theory of control valves, flow-induced vibration and digital sensing. Section IV describes the system design, hardware set-up, firmware and soft-ware. Section V gives the experimental procedure, Section VI reports and discusses the results including the simulation-mode demonstrations, Section VII lists limitations and improvements, Section VIII outlines future work and Section IX concludes the paper.

II. RELATED WORK

1. Pipeline And Leak Monitoring

Surveys of pipeline fault detection classify methods into those based on hardware sensing (acoustic, vibration, fibre-optic, pres-sure) and those based on software or model-based analysis [9], [10]. For water mains in particular, reviews of inspection and condition-assessment technology [22], leakage management [23] and transient-based detection [24] show that non-intrusive sens-ing is attractive because it does not interrupt service. Vibration measurements on plastic water pipes have been used to detect leaks automatically [11]. Those systems typically use analog accelerometers and signal processing, whereas the present work deliberately uses simple digital vibration modules to demonstrate the monitoring chain at minimal cost.

2. Control Valve Behaviour And Diagnostics

Control valve terminology, flow capacity and sizing are standard-ised [25], [26], and the practical selection and operation of valves is documented in manufacturer and handbook literature [2], [3]. Hydrodynamic noise and cavitation are known to generate strong broadband vibration downstream of a restriction [27], [28], and abrupt valve motion produces pressure transients [29], [30]. In loop diagnostics, valve stiction has been modelled and detected from process data [4], [5]. Those studies work from controller signals, while a vibration sensor offers a complementary, directly physical indication at the valve.

3. Microcontroller Acquisition And Virtual Instruments

Microcontroller readout of resistive and other sensors has been studied extensively [31]– [34], with attention to the quality of the acquired data, including bias, drift and outliers [35]. The ESP32 specifically has been characterised for data acquisition and synchronised sensor networks [14]– [17], [36] and has been applied to energy, temperature and environmental mon-itoring [19], [37]– [39]. On the PC side, MATLAB App De-signer and related graphical tools have been used for sensor visualisation and even SCADA-style supervision [20], [21], and simulators such as Wokwi support early prototyping of embedded firmware [40]. The broader context is the move towards connected, software-defined instrumentation under the IoT and Industry 4.0

paradigms [18], [41]. Our contribution sits in this space: a minimal, fully documented ESP32–MATLAB instrument applied to valve-induced vibration

III. THEORETICAL BACKGROUND

1. Control Valve And Flow Capacity

A control valve regulates flow by changing the area of the open-ing through which the fluid passes. When closed the flow stops, and as the valve opens the flow rate rises. For a liquid, the volu-metric flow through a valve is commonly related to the pressure drop by the flow coefficient C_v [3], [26]:

vs

$$Q = C_v \sqrt{\frac{\Delta P}{G_f}}, \quad (1)$$

where Q is the flow rate, ΔP the pressure drop across the valve and G_f the specific gravity of the liquid. The coefficient C_v itself depends on the valve position, and the inherent characteris-tic (linear, equal-percentage or quick-opening) defines how C_v varies with opening [1], [2]. In the pipe, continuity gives $Q = Av$, with A the cross-sectional area and v the mean velocity [42].

In a closed-loop process, a controller compares the measured variable with a set point and drives the valve through an actuator, commonly using proportional–integral–derivative (PID) action[43]– [46].

In the present rig the valve is operated by hand, so the instrument acts as the measurement and display part of a loop that could later be closed.

2. Flow-Induced Vibration

Whether the flow in a pipe is laminar or turbulent is indicated by the Reynolds number,

$$Re = \frac{\rho v D}{\mu}, \quad (2)$$

where ρ is the fluid density, D the pipe diameter and μ the dynamic viscosity [8], [42]. Flow past a partly open valve is a bluff-body-like obstruction that produces separation, vortices and wide-band pressure fluctuations. Where shedding is periodic, its frequency scales with the Strouhal relation $f_s = St v/d$, with d a characteristic obstruction dimension [6]. These fluctuations load the pipe wall, and the structural response depends on the coupling between fluid and

pipe [7]. Cavitation, if it occurs, adds intense high-frequency content [27], [28]. In simple terms, higher flow gives stronger vibration, and a closed valve gives weaker vibration, although ambient machine vibration, pump pulsation and sensor mounting also contribute.

3. Digital Vibration Sensing

Vibration is normally measured with accelerometers, including micromachined (MEMS) devices [47], [48]. The low-cost sen-sors used here are digital modules whose output is HIGH or LOW depending on whether the sensed vibration crosses an on-board threshold. If $a(t)$ is the vibration signal and θ the threshold, the binary output can be modelled as ded firmware [40]. The broader context is the move towards connected, software-defined instrumentation under the IoT and Industry 4.0 paradigms [18], [41]. Our contribution sits in this

$$s[k] = \begin{cases} 1, & |a(kT_s)| > \theta, \\ 0, & \text{otherwise,} \end{cases} \quad (3)$$

with T_s the sampling interval. Such a sensor reveals a change in vibration state but gives no amplitude, frequency content or flow

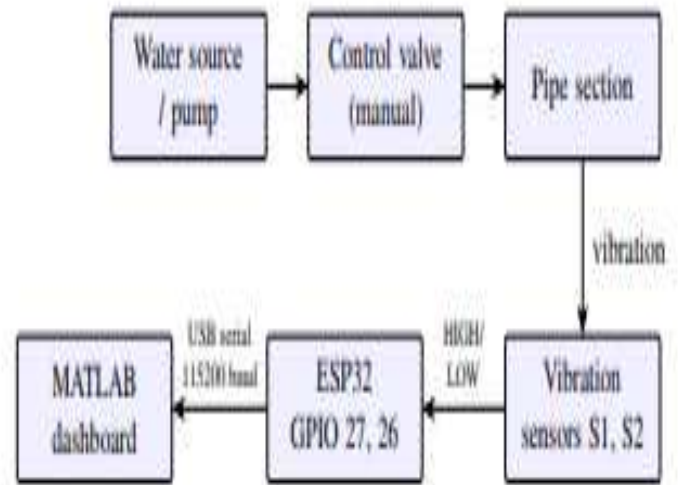


Fig. 1. Block diagram of the valve-monitoring virtual instrument.

rate [49], [50]. A useful derived quantity is the duty cycle over a window of N samples, $D = 1 \sum_N s[k]$, which

approximates how often the threshold is exceeded. Sampling must respect the Nyquist–Shannon criterion for the dynamics of interest [51]– [53]. Mechanical or relay-type sensors may also chatter, and contact bounce is conventionally

4. ESP32 Digital Input and Serial Link

The ESP32 is a dual-core microcontroller with integrated Wi-Fi and Bluetooth and numerous general-purpose input/output (GPIO) pins [55], [56]. A pin configured as an input is read with `digitalRead()`, which returns HIGH (1) or LOW (0) according to the pin voltage. The Arduino core for the ESP32 exposes this and the `millis()` timer, which returns the milliseconds elapsed since start-up [36], [57]. The board is programmed and powered over USB, where a USB-to-UART bridge enumerates as a virtual COM port on the PC [58].

5. Virtual Instrumentation

A virtual instrument consists of acquisition hardware, a computer and software that implements the front panel and the signal processing [12], [13]. In this work the ESP32 is the acquisition hardware and MATLAB provides the front panel. MATLAB's serialport interface reads text lines terminated by a line feed, and `animatedline` appends points to a plot efficiently, which suits live strip charts [59], [60].

IV. SYSTEM DESIGN

1. Architecture

The system has four functional blocks, shown in Fig. 1: (i) the physical process, comprising a water source or pump, a pipe section and a manually operated control valve; (ii) the sensing stage with two digital vibration sensors on the pipe; (iii) the acquisition stage, an ESP32 that samples and timestamps the sensor outputs; and (iv) the virtual instrument, a MATLAB dashboard that receives, parses and displays the data over USB serial.

2. Hardware Table I lists the apparatus. Sensor 1 output is wired to GPIO 27 and Sensor 2 output to GPIO 26 (Table II); both sensors take power and ground from the ESP32 supply pins, and the USB cable provides power and the serial link. Wiring is made on a breadboard with jumper wires so that the layout can be changed easily. The assembled laboratory prototype is shown in Fig. 2: the laptop on the bench runs the AQUAGUARD dashboard, the ESP32 and breadboard wiring sit beside the transparent vessel in the centre of the picture, and the sensor leads (red and white

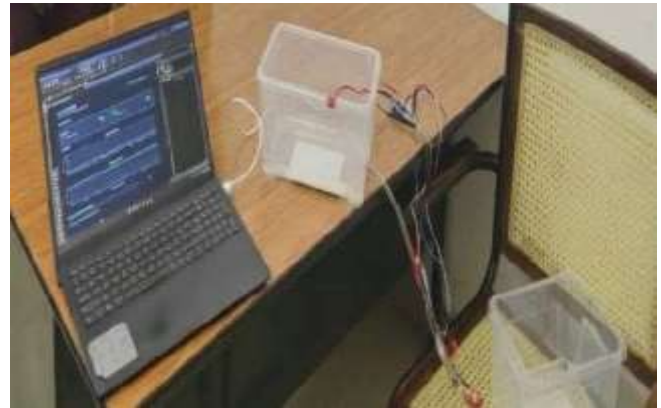


Fig. 2. Photograph of the laboratory prototype. The laptop (left) runs the AQUAGUARD MATLAB dashboard; the ESP32 and breadboard wiring are placed beside the transparent vessel on the bench, and the sensor leads are routed to the vessels of the water flow test set-up. A single USB cable provides power and the serial link to the PC

Table I
Apparatus And Software

No.	Item	Purpose
1	ESP32 development board	Reads sensors and streams data over USB serial
2	Two digital vibration sensors	Detect pipe vibration; digital HIGH/LOW output
3	Manual control valve	Starts, stops and regulates flow
4	Pipe section, water source/pump	Water flow system under test
5	Breadboard, jumper wires	Electrical connections
6	USB cable	Power and serial link to PC
7	PC with MATLAB	Acquisition, display, front panel
8	Arduino IDE	Programming the ESP32

jumper wires) run from the board to the test vessels of the water set-up. The USB cable visible on the bench carries both power and the serial data stream between the ESP32 and the PC.

3. Firmware

The firmware (Listing 1) was written in the Arduino IDE. In setup() the serial port is opened at 115200 baud, both pins are set as inputs and a header line time_ms,sensor1,sensor2 is printed. In loop() both pins are read and, when at least one is HIGH, a line containing the millis() timestamp and the two states is sent; a 5-ms delay sets the loop period, giving a maximum sampling rate near 200 Hz

Listing 1. ESP32 firmware (Arduino IDE) as used in the experiment

```
1 const int SENSOR_1 = 27;
2 const int SENSOR_2 = 26;
3
4 void setup() {
5   Serial.begin(115200);
6   pinMode(SENSOR_1, INPUT);
7   pinMode(SENSOR_2, INPUT);
8   Serial.println("time_ms,sensor1,sensor2");
9 }
10
11 void loop() {
12   int state1 = digitalRead(SENSOR_1);
13   int state2 = digitalRead(SENSOR_2);
14
15   // Send data only when at least one sensor
   is HIGH
16   if (state1 == HIGH || state2 == HIGH) {
17     Serial.print(millis());
18     Serial.print(",");
19     Serial.print(state1 == HIGH ? 1 : 0);
20     Serial.print(",");
21     Serial.println(state2 == HIGH ? 1 : 0);
22   }
23   delay(5);
24 }
```

4. Serial Data Format and Throughput

Each transmitted line has the form t,s1,s2, for example 96720,0,1, where t is the time in milliseconds since ESP32 start-up and s1, s2 are the sensor states. A line is at most about 14 characters including the line terminator. At 115200 baud with 10 bits per character (8N1 framing) the link carries about 11.5 kB/s. Streaming at the maximum loop rate of 200 lines/s therefore needs about $200 \times 14 = 2.8$ kB/s, roughly 24 % of the link capacity, so the link is not a bottleneck.

E. MATLAB Dashboard

The dashboard script (named leak_detection_dashboard.m in the project) is a script-based figure built from uicontrol, uipanel and axes objects. It opens serialport("COM6",115200), sets the line-feed terminator and flushes the buffer. A dark theme is defined through colour constants for background, panel, grid, text and accent colours. The front panel (Fig. 3) comprises:

- a header showing the application name AQUAGUARD, the subtitle "Pipeline Intelligence / Live Sensor Monitoring" and the connection string ESP32 | COM6 | 115200 BAUD;
- two sensor cards, each showing the sensor name, the GPIO pin and the current state (HIGH in green, LOW in grey, WAITING before data arrive);
- two strip charts built with animatedline, with a vertical axis restricted to the two states LOW and HIGH and a horizontal axis showing elapsed time in seconds over a moving 15-s window; and
- a footer that reports the combined state (for instance SENSOR 01 LOW / SENSOR 02 HIGH) and the time of the last sample. The footer purposely reports states only: the text "Digital monitoring only | Leak not yet confirmed" reminds the user that the, panel does not make a leak diagnosis.

The core acquisition loop is shown in Listing 2. While the figure handle is valid, the script checks whether bytes are available, reads one line, trims it and converts it with sscanf. Only if exactly three numbers are returned is the line accepted, which rejects the header line and any partial line produced at start-up. The time is converted from milliseconds to seconds, points are added to both lines, the cards and footer are updated, and the x-limits are set to $[\max(0, r-15), r]$ with $r = \max(15, t)$ so that the window slides once the elapsed time exceeds 15 s. drawnow limitrate keeps the interface responsive. When the window is closed the loop ends and the serial object is cleared so that the COM port is released. Fig. 4 summarises this flow

```
1 esp32 = serialport("COM6", 115200);
2 configureTerminator(esp32, "LF");
flush(esp32);
3 windowSeconds = 15;
4 while ishandle(fig)
5 if esp32.NumBytesAvailable > 0
```



Fig. 3. AQUAGUARD MATLAB dashboard captured with the valve closed. Sensor 01 (GPIO 27) reads LOW and Sensor 02 (GPIO 26) reads HIGH; the last sample is at 96.72

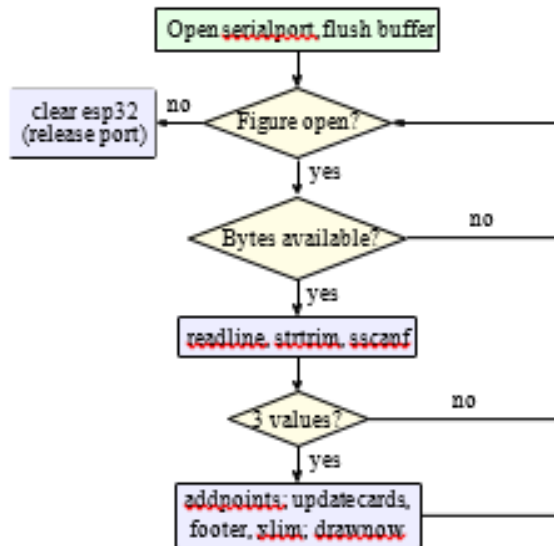


Fig. 4. Simplified flow of the MATLAB acquisition loop. Rejected or absent data return control to the loop test; closing the figure exits the loop and releases the serial port

```
6 dataLine = strtrim(readline(esp32));
7 values = sscanf(dataLine, '%f%f%f');
8 if numel(values) == 3
9 t = values(1)/1000; s1 = values(2); s2 =
values(3);
10 addpoints(h1, t, s1); addpoints(h2, t,
s2);
11 % update cards: HIGH -> green, LOW ->
muted
12 % update footer with combined state +
last sample
13 rightEdge = max(windowSeconds, t);
14 leftEdge = max(0, rightEdge -
windowSeconds);
15 xlim(ax1, [leftEdge rightEdge]);
16 xlim(ax2, [leftEdge rightEdge]);
17 drawnow limitrate;
18 end
19 else
20 pause(0.01); drawnow limitrate;
21 end
22 end
23 clear esp32; % release the serial port
```

6. Simulation Mode

For development and demonstration, the dashboard can also be run in a simulation mode in which synthetic sensor states replace the live serial stream, so that the card, chart, window-sliding and colour-coding logic can be exercised without the water rig. In this mode the connection string in the header is replaced by an amber banner reading SIMULATION MODE | SYNTHETIC DATA, so that a simulated capture cannot be mistaken for a mea-surement (Section VI-E).

Table III

OBSERVED SENSOR STATES VERSUS VALVE POSITION

Valve Position	S1 (Gpio 27)	S2 (Gpio 26)	Foote r Sage	Mes-
Fully closed	LOW	HIGH	S1 LOW HIGH	/ S2
Partly open	TBD	TBD	TBD	
Fully open	TBD	TBD	TBD	
Partly closed again	TBD	TBD	TBD	

V. EXPERIMENTAL PROCEDURE

The experiment was carried out in the following order.

- The vibration sensors were attached to the pipe near the control valve. Sensor 1 output was wired to GPIO 27, Sensor 2 output to GPIO 26, and sensor power and ground to the ESP32 supply and ground pins.

- The ESP32 was connected by USB and the COM port (COM6 here) was identified in Device Manager.
- The ESP32 board and port were selected in the Arduino IDE and the sketch of Listing 1 was uploaded.
- The Arduino Serial Monitor was closed so that the COM port was free for MATLAB.
- The dashboard script was saved and portName was set to the correct port if it differed.
- The script was run; the window opened with both cards showing WAITING.
- With the valve fully closed, the sensor states were observed.
- The valve was gradually opened to start flow, and any change in the cards and graphs was observed.
- The valve was then partly closed and the readings compared with the fully open case, and the observations recorded against valve position.
- The dashboard window was closed to end the program and release the serial port.

VI. RESULTS AND DISCUSSION

1. Closed-Valve Observation

With the valve fully closed and the dashboard running, the status cards settled to Sensor 01 = LOW and Sensor 02 = HIGH, as seen in Fig. 3. The strip charts show a flat LOW trace for Sensor 01 and a flat HIGH trace for Sensor 02 over the displayed window of 82–96 s, and the footer reports SENSOR 01 LOW / SENSOR 02 HIGH | Last sample: 96.72 s. This confirms that the whole chain works: the ESP32 reads the GPIO pins, timestamps and transmits the data, MATLAB receives and parses the lines, and the cards, charts, time axis and footer are updated consistently. The sliding window also behaved as designed, with the axis range following the elapsed time.

Table III summarises the observations by valve position. Only the closed-valve row was verified from the captured run reported here; the remaining rows are to be completed from the recorded trials.

2. Interpretation

Two points need care. First, the closed-valve state is a verification of the measurement chain, not of the flow physics. Since the sensors are digital, the result shows that the panel faithfully reproduces whatever logic level

the sensors provide. Second, Sensor 02 reads HIGH although the valve is closed, which is not what a simple “vibration only when flowing” picture would suggest. Several explanations are plausible and are not distinguished by the present data: the sensor threshold may be set low enough to trip on ambient vibration or residual water motion; the module’s output polarity may be active-low or idle-high; the two sensors may be mounted with different coupling to the pipe; or pump or building vibration may be reaching the pipe. This difference between two nominally identical sensors is itself useful information, because it shows that each digital channel must be characterised and its threshold adjusted before its state can be associated with valve position [35], [48], [49].

3. Effect Of The Transmit-Only-When-High Policy

The firmware sends a line only when at least one sensor is HIGH. When both sensors drop LOW, nothing is transmitted, so the dashboard does not receive the transition: the cards and charts keep showing the last received values, and the animatedline traces are not extended. The panel can therefore display a stale state, and the time axis only advances when new data arrive. A related consequence is that LOW intervals appear in a chart only if one sensor remains HIGH. This behaviour is a design limitation rather than a hardware fault and is corrected by transmitting continuously (Section VII). The caution is general for instrumentation: the quality of displayed data depends as much on the acquisition policy as on the sensor [35], [50].

4. Timing And Resolution

The timestamp resolution is 1 ms, and with the 5-ms loop delay the maximum sampling rate is near 200 Hz, giving a Nyquist limit of about 100 Hz [51], [52]. Because the sensor output is binary and the dashboard is intended to show state changes over seconds, this is adequate for the present purpose. It would not be adequate for resolving vibration spectra, which for turbulent pipe flow and cavitation can extend to kilohertz [6], [27], [28]. Fast pipe-pressure transients caused by rapid valve closure also evolve faster than the sampling interval used here [29], [30], so the present instrument should be regarded as a slow state monitor.

5. Simulation-Mode Dashboard Demonstrations

To show how the front panel responds to every combination of sensor states, the dashboard was also run in simulation mode, in which synthetic digital signals are plotted in place of the serial stream. Four captures taken in chronological order are shown in Fig. 5 and summarised in Table IV. The banner SIMULATION MODE | SYNTHETIC DATA is visible in every frame, so these figures illustrate the behaviour of the software and are not mea-surements of the water rig.

Four observations follow from these frames. First, the two channels are handled independently: Sensor 01 alone, Sensor 02 alone and both sensors together are all displayed correctly, and



(a) 0–15 s window: Sensor 01 rises to HIGH at about 3 s while Sensor 02 stays LOW.



(b) 3–18 s window: Sensor 01 HIGH for 3–6 s, Sensor 02 HIGH for 10–13 s, both HIGH from about 17 s.



(c) 15–29 s window: both sensors HIGH for 17–20 s and then LOW; a 1-Hz pulse train starts on Sensor 01 at 25 s.



(d) 24–38 s window: Sensor 01 pulses end at about 30.5 s; a pulse train starts on Sensor 02 at 35 s.

Fig. 5. AQUAGUARD dashboard captures in simulation mode (synthetic data). The sequence shows the independent response of the two sensor cards and strip charts and the sliding 15-s window.

TABLE IV Synthetic Test Sequence Observed in Fig. 5

Time (s)	S1 (GPIO 27)	S2 (GPIO 26)	Cards
0–3	LOW	LOW	L / L
3–6	HIGH	LOW	H / L
10–13	LOW	HIGH	L / H
17–20	HIGH	HIGH	H / H
25–30.5	1-Hz pulses	LOW	toggles
35–39	LOW	1-Hz pulses	toggles

the card colour follows the state (grey LOW, green HIGH), as in Fig. 5(a) and (b). Second, the pulsed segments in Fig. 5(c) and (d) have a period of about 1 s

with the signal HIGH for about 0.5 s, so the duty cycle D defined in Section III-C is about 0.5 in those intervals; this is the kind of pattern a fluctuating vibration source would produce on a real digital sensor. Third, the horizontal axis starts at 0–15 s and then slides as [max(0, $r-15$), r], giving the 3–18, 15–29 and 24–38 s windows seen in the captures, which confirms the window logic of Listing 2. Fourth, when both inputs return LOW (for example after 20 s in Fig. 5(c)) the traces continue as flat LOW lines and the cards return to LOW. This is the behaviour expected of a continuous-transmission stream and not of the transmit-only-when-HIGH firmware of Listing 1, which would leave the last HIGH value on the panel (Section VI-C); the captures therefore also illustrate what the corrected firmware of Section VII-B should deliver. These captures verify the display logic only. They do not show any relation between valve position and sensor state, and the open and partly open rows of Table III remain to be completed from the recorded trials.

VII. LIMITATIONS AND PROPOSED IMPROVEMENTS

Limitations

- *Binary sensing.* The sensors give no amplitude or spectrum and cannot measure flow rate; the instrument reports a state change only.
- *Conditional transmission.* Data are sent only when at least one sensor is HIGH, which can leave a stale display.
- *Time base.* Time is relative to ESP32 start-up, with no absolute date/time stamp or logging to file.
- *No calibration or flow reference.* No reference flow meter was used, so the sensor state cannot yet be related to a flow value through Eq. (1).
- *Manual valve.* Valve position is set by hand and is not recorded automatically.
- *Single operating condition verified.* The captured result from the physical rig covers the closed-valve case; the other captures in this paper use synthetic data.

A. Corrections

Continuous transmission is obtained by removing the conditional so that every loop iteration sends a line:

```
1 void loop() {  
2   Serial.print(millis()); Serial.print(",");  
3   Serial.print(digitalRead(SENSOR_1)); Serial.print(",");  
4   Serial.println(digitalRead(SENSOR_2));  
5   delay(5);  
6 }
```

With this change the dashboard receives both HIGH and LOW states and the charts show true transitions. Further low-cost refinements are a short software debounce or majority vote over several samples to suppress chatter [53], [54], logging of each parsed row with a PC timestamp to a CSV file, and a status indicator that turns amber if no line has arrived for a set time, which would flag a lost link and distinguish it from a genuine LOW state.

6. Future Scope

- Analog vibration sensing. Replacing the digital modules with an analog or MEMS accelerometer [47], [48] would allow amplitude, RMS and frequency-domain features to be computed on the ESP32 or in MATLAB and related to valve opening and flow.
- Flow reference and calibration. Adding a flow sensor would permit calibration of vibration features against flow, supporting a quantitative flow estimate and checks against valve sizing relations [3], [26].
- Data logging and analytics. Timestamped logging would support later analysis of data quality, drift and outliers [35], and features could be tested with pattern recognition for fault and leak indication [9]–[11].
- Closed-loop control. A motorised actuator driven by the ESP32, with a PID controller around a measured flow or pressure, would turn the monitor into a control rig [1], [43], [44]. Valve stiction and response tests could then be explored [4], [5].
- Wireless and cloud operation. The ESP32's Wi-Fi interface could replace the USB cable and forward data to a remote dashboard, in line with IoT-based monitoring practice [18], [19], [38], [39]. A web or MATLAB App Designer interface could add session control and export [20], [21]. Firmware and interface could also be prototyped in a simulator before deployment [40].

VIII. CONCLUSION

A virtual instrument for observing control valve operation in a water flow system was designed and built from an ESP32, two digital vibration sensors and a MATLAB dashboard. The ESP32 streams timestamped sensor states at 115200 baud and the dash-board shows sensor cards, 15-s strip charts and a combined-state footer in real time. With the valve closed, the panel displayed Sensor 1 LOW and Sensor 2 HIGH, confirming correct acquisition, parsing and display, and simulation-mode captures with synthetic data demonstrated the panel response to all combinations of sensor states and to pulsed inputs. The instrument indicates a change in vibration state and not flow rate, and the present firmware's transmit-only-when-HIGH policy can leave a stale reading; both limitations have clear remedies. The architecture provides a low-cost basis for analog vibration sensing, data logging and closed-loop PID valve control, and for teaching virtual instrumentation and process measurement.

Acknowledgment

The authors thank the School of Electrical Engineering, Vellore Institute of Technology, for laboratory facilities and guidance.

REFERENCES

1. D. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. Doyle III, *Process Dynamics and Control*, 4th ed. Hoboken, NJ, USA: Wiley, 2016.
2. B. G. Lipták, *Instrument Engineers' Handbook: Process Control and Optimization*, 4th ed. Boca Raton, FL, USA: CRC Press, 2005.
3. Emerson Automation Solutions, *Control Valve Handbook*, 5th ed. Emerson, 2017.
4. M. A. A. S. Choudhury, S. L. Shah, and N. F. Thornhill, "Modelling valve stiction," *Control Eng. Pract.*, vol. 13, no. 5, pp. 641–658, 2005.
5. M. Jelali and B. Huang, *Detection and Diagnosis of Stiction in Control Loops*. London, U.K.: Springer, 2010.
6. R. D. Blevins, *Flow-Induced Vibration*, 2nd ed. New York, NY, USA: Van Nostrand Reinhold, 1990.
7. M. P. Paidoussis, *Fluid–Structure Interactions: Slender Structures and Axial Flow*. London, U.K.: Academic Press, 1998, vol. 1.
8. S. B. Pope, *Turbulent Flows*. Cambridge, U.K.: Cambridge Univ. Press, 2000.
9. S. Datta and S. Sarkar, "A review on different pipeline fault detection methods," *J. Loss Prev. Process Ind.*, vol. 41, pp. 97–106, 2016.
10. M. A. Adegboye, W.-K. Fung, and A. Karnik, "Recent advances in pipeline monitoring and oil leakage detection technologies: Principles and approaches," *Sensors*, vol. 19, no. 11, p. 2548, 2019.
11. A. Martini, M. Troncossi, and A. Rivola, "Automatic leak detection in buried plastic pipes of water supply networks by means of vibration measurements," *Shock Vib.*, vol. 2015, 2015.
12. H. Goldberg, "What is virtual instrumentation?" *IEEE Instrum. Meas. Mag.*, vol. 3, no. 4, pp. 10–13, Dec. 2000.
13. J. Travis and J. Kring, *LabVIEW for Everyone*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2006.
14. A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things," in *Proc. Internet Technol. Appl. (ITA)*, Wrexham, U.K., 2017, pp. 143–148.
15. M. Babiuch, P. Foltýnek, and P. Smutný, "Using the ESP32 microcontroller for data processing," in *Proc. 20th Int. Carpathian Control Conf. (ICCC)*, 2019.
16. V.-K. Tran, B.-T. Thai, H. Pham, V.-K. Nguyen, and V.-K. Nguyen, "A proposed approach to utilizing ESP32 microcontroller for data acquisition," *J. Eng. Technol. Sci.*, vol. 56, no. 4, pp. 474–488, 2024.
17. M. J. Espinosa-Gavira, A. Agüera-Pérez, J. C. Palomares-Salas, J. M. Sierra-Fernández, P. Remigio-Carmona, and J. J. G. de-la Rosa, "Characterization and performance evaluation of ESP32 for real-time synchronized sensor networks," *Procedia Comput. Sci.*, vol. 237, pp. 261–268, 2024.
18. J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision,

- architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.
19. G. Mois, S. Folea, and T. Sanislav, "Analysis of three IoT-based wireless sensors for environmental monitoring," *IEEE Trans. Instrum. Meas.*, vol. 66, no. 8, pp. 2056–2064, 2017.
20. M. Ibarra, M. J. A. Calpeño, and R. E. Grande, "Electronic instrumentation for the acquisition and visualization of data from multiple sensors in a graphical user interface in MATLAB," *Rev. Tecnol. Inf.*, vol. 11, no. 31, pp. 1–9, 2024.
21. A. F. Saransari, F. Dehghanisanij, and M. T. Iqbal, "Design and implementation of SCADA architecture based on MATLAB app designer for a hybrid power system," *J. Electron. Electr. Eng.*, vol. 3, no. 2, pp. 476–489, 2024.
22. Z. Liu and Y. Kleiner, "State of the art review of inspection technologies for condition assessment of water pipes," *Measurement*, vol. 46, no. 1, pp. 1–15, 2013.
23. R. Puust, Z. Kapelan, D. A. Savic, and T. Koppel, "A review of methods for leakage management in pipe networks," *Urban Water J.*, vol. 7, no. 1, pp. 25–45, 2010.
24. A. F. Colombo, P. Lee, and B. W. Karney, "A selective literature review of transient-based leak detection methods," *J. Hydro-environ. Res.*, vol. 2, no. 4, pp. 212–227, 2009.
25. International Electrotechnical Commission, *Industrial-process control valves – Part 1: Control valve terminology and general considerations*, IEC 60534-1.
26. ISA, *Industrial-Process Control Valves – Part 2-1: Flow capacity – Sizing equations for fluid flow under installed conditions*, ANSI/ISA 75.01.01.
27. International Electrotechnical Commission, *Industrial-process control valves – Part 8-4: Noise considerations – Prediction of noise generated by hydrodynamic flow*, IEC 60534-8-4.
28. C. E. Brennen, *Cavitation and Bubble Dynamics*. New York, NY, USA: Oxford Univ. Press, 1995.
29. E. B. Wylie and V. L. Streeter, *Fluid Transients in Systems*. Englewood Cliffs, NJ, USA: Prentice Hall, 1993.
30. M. H. Chaudhry, *Applied Hydraulic Transients*, 3rd ed. New York, NY, USA: Springer, 2014.
31. M. Šimić, "Microcontroller-based readout of resistive sensors," *IFAC-PapersOnLine*, vol. 55, no. 4, pp. 242–247, 2022.
32. Z. Czaja, "Simple measurement method for resistive sensors based on ADCs of microcontrollers," *IEEE Sensors J.*, vol. 24, no. 3, pp. 2996–3003, 2024.
33. S. Kaunkid, A. Aurasopon, W. Khamsen, C. Takeang, N. Piladaeng, and J. Lloret, "Resistive measurement method for MQ sensors based on ADCs of microcontrollers," *IEEE Access*, vol. 12, pp. 144 364–144 376, 2024.
34. A. S. Fiorillo, C. D. Critello, and S. A. Pullano, "Theory, technology and applications of piezoresistive sensors: A review," *Sens. Actuators A, Phys.*, vol. 281, pp. 156–175, 2018.
35. H. Y. Teh, A. W. Kempa-Liehr, and K. I.-K. Wang, "Sensor data quality: A systematic review," *J. Big Data*, vol. 7, no. 11, 2020.
36. N. Kolban, *Kolban's Book on ESP* Leanpub, 2018.
37. A.-C. Gheorghe, E. Stan, and I. Udrioiu, "Electricity consumption measurement system using ESP32," *Sci. Bull. Elect. Eng. Fac.*, vol. 21, no. 2, pp. 23–26, 2021/2022.
38. Pratikto, M. Arman, M. A. Falahuddin, R. A. Pratama, R. Z. Fitriani, and Y.-D. Kuan, "Temperature data acquisition system for showcase refrigerator based on ESP32 and online remote display," *Sens. Mater.*, vol. 35, no. 11, pp. 3687–3696, 2023.
39. S. Pasika and S. T. Gandla, "Smart water quality monitoring system with cost-effective using IoT," *Heliyon*, vol. 6, no. 7, p. e04096, 2020.
40. S. M. Tsyrlunyk, V. M. Tkachuck, and M. O. Tsyrlunyk, "Prototyping IoT project at WOKWI service," in *Proc. VNTU*, 2022.
41. H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, 2014.
42. Y. A. Çengel and J. M. Cimbala, *Fluid Mechanics: Fundamentals and Applications*, 4th ed. New York, NY, USA: McGraw-Hill, 2018.
43. K. J. Åström and T. Hägglund, *PID Controllers: Theory, Design, and Tuning*, 2nd ed. Research Triangle Park, NC, USA: ISA, 1995.

44. K. Ogata, Modern Control Engineering, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.
45. N. S. Nise, Control Systems Engineering, 7th ed. Hoboken, NJ, USA: Wiley, 2015.
46. R. C. Dorf and R. H. Bishop, Modern Control Systems, 13th ed. Boston, MA, USA: Pearson, 2017.
47. N. Yazdi, F. Ayazi, and K. Najafi, "Micromachined inertial sensors," *Proc. IEEE*, vol. 86, no. 8, pp. 1640–1659, 1998.
48. J. Fraden, Handbook of Modern Sensors: Physics, Designs, and Applications, 5th ed. Cham, Switzerland: Springer, 2016.
49. J. G. Webster and H. Eren, Measurement, Instrumentation, and Sensors Handbook, 2nd ed. Boca Raton, FL, USA: CRC Press, 2014.
50. A. S. Morris and R. Langari, Measurement and Instrumentation: Theory and Application, 2nd ed. Oxford, U.K.: Academic Press, 2016.
51. H. Nyquist, "Certain topics in telegraph transmission theory," *Trans. AIEE*, vol. 47, no. 2, pp. 617–644, 1928.
52. C. E. Shannon, "Communication in the presence of noise," *Proc. IRE*, vol. 37, no. 1, pp. 10–21, 1949.
53. A. V. Oppenheim and R. W. Schaffer, Discrete-Time Signal Processing, 3rd ed. Upper Saddle River, NJ, USA: Pearson, 2010.
54. J. Ganssle, "A guide to debouncing," The Ganssle Group, 2004.
55. Espressif Systems, ESP32 Series Datasheet, Espressif Systems, [Online]. Available: <https://www.espressif.com>.
56. 56ESP32 Technical Reference Manual, Espressif Systems, [Online]. Available: <https://www.espressif.com>.
57. "Arduino core for the ESP32," GitHub repository, [Online]. Available: <https://github.com/espressif/arduino-esp32>.
58. USB Implementers Forum, Universal Serial Bus Specification, Revision 2.0, 2000.
59. The MathWorks, Inc., "MATLAB serial port interface (serialport)," Documentation, [Online]. Available: <https://www.mathworks.com/help/matlab/>.
60. —, "MATLAB graphics: animated lines and realtime plotting," Documentation, [Online].

Available: <https://www.mathworks.com/help/matlab/>.