TELKOMNIKA Telecommunication Computing Electronics and Control
Vol. 24, No. 6, December 2026, pp. 2183~2195
ISSN: 1693-6930, DOI: 10.12928/TELKOMNIKA.v24i6.27902

An ESP32-based 2D LiDAR telemetry pipeline using binary UDP, TCP, and MQTT

Julian Camilo Forero Gutierrez[1], Andrés Leonardo Jutinico Alarcón[1], Henry B. Guerrero[1, 2]
[1]Electronic Engineering Program, Faculty of Engineering, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia
[2]Postdoctoral Researcher, BitPointer S.A.S, Villavicencio, Colombia

Article Info

Article history:

Received Feb 23, 2026
Revised Aug 17, 2026
Accepted Aug 28, 2026

Keywords:

Binary telemetry
ESP32
Message queuing telemetry transport
Transmission control protocol user datagram protocol
Two-dimensional light detection and ranging

ABSTRACT

Two-dimensional light detection and ranging (LiDAR) sensors are widely used in mobile robotics and monitoring, but streaming complete scan revolutions from resource-constrained microcontrollers over wireless fidelity (Wi-Fi) requires compact framing, transport-aware buffering, and reproducible instrumentation. This study presents a transport-independent telemetry framework for the Slamtec RPLidar A1 using an ESP32. The firmware acquires universal asynchronous receiver-transmitter (UART) measurements, detects revolution boundaries, packs angle-distance-quality points into a compact binary scan frame, and transmits complete scans through user datagram protocol (UDP)/binary (BIN), transmission control protocol (TCP)/BIN, or message queuing telemetry transport (MQTT)/BIN. A Python-based fast application programming interface (FastAPI) and Uvicorn service validates and reconstructs a canonical scan representation, exposes runtime counters, supports browser-based visualization, and records sessions for deterministic replay. Final operating parameters were selected through preliminary characterization of metadata encoding, motor duty cycle, quality retention, UDP fragmentation, and reconstruction stability. The official evaluation comprised 18 runs of approximately five minutes each under nominal and degraded-received signal strength indicator (RSSI) indoor Wi-Fi profiles. Across all protocol-profile combinations, all expected scan sequences were received, with 0.00% sequence loss, no duplicate sequences, no queue drops, and no source-timestamp inter-scan interval above 500 ms. The results demonstrate that the common binary architecture supports reproducible LiDAR scan telemetry with bounded host-process and embedded-resource use under the evaluated indoor conditions, without implying a universal protocol ranking.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Henry B. Guerrero
Electronic Engineering Program, Faculty of Engineering, Universidad Distrital Francisco José de Caldas
Bogotá, Colombia
Email: hbguerrero@ieee.org

1. INTRODUCTION

Two-dimensional (2D) light detection and ranging (LiDAR) sensors provide planar range measurements that are widely used in mobile robotics, obstacle detection, mapping, navigation, transportation, and field automation [1]–[6]. The broader LiDAR literature also includes automotive sensing, multispectral sensing, unmanned aerial vehicle (UAV) detection, construction-volume estimation, and spacecraft navigation [7]–[11]. For low-cost robotic platforms, educational prototypes, embedded perception

systems, and agricultural mobile robots, 2D LiDAR remains attractive because it provides useful geometric information while preserving moderate cost, power consumption, and integration complexity [6], [12]–[15].

In agricultural and mobile robotics, LiDAR has supported crop monitoring, precision agriculture, under-canopy navigation, crop-row following, greenhouse navigation, and field-oriented mobile robot operation [6], [12]–[18]. However, many of these systems rely on computer-based or operating-system-based processing architectures. In this context, related work by the authors has also investigated LiDAR-based crop-row navigation and low-cost agricultural robotic platforms [19], [20].

In addition to these computer-centered agricultural and mobile robotic systems, another area of research has investigated the integration of embedded or low-cost LiDAR using microcontroller operating system (micro-ROS) interfaces, Arduino-based robotic integration, microcontroller-based sensing platforms, STM32-based scanner implementations, and scanner characterization/calibration [5], [14], [15], [21], [22]. These studies illustrate various methods for connecting, processing, or characterizing low-cost LiDAR sensors; however, their primary objectives typically focus on mapping, robotic integration, sensor implementation, or calibration rather than directly evaluating complete wireless scan telemetry from a resource-constrained microcontroller.

Table 1 provides a summary of representative studies and situates the current work within the context of the telemetry layer. The comparison acknowledges that prior research pursued diverse objectives; rather than suggesting a direct one-to-one competition, it underscores that many studies focus on sensing, mapping, calibration, or robotic integration, whereas the comprehensive microcontroller-native scan telemetry pipeline is less frequently examined as the primary experimental subject.

Table 1. Positioning of the proposed research in relation to representative studies on low-cost, embedded, robotic, and transport-oriented LiDAR technologies

Work	Platform/context	Main focus	Telemetry treatment	Evaluation and reproducibility scope
Li *et al.* [5]	Mobile robot / micro-ROS	2D mapping and ROS-based integration	Micro-ROS middleware integration; no common scan-frame transport comparison	Reports mapping and middleware integration without a common user datagram protocol (UDP)/ transmission control protocol (TCP)/ message queuing telemetry transport (MQTT) telemetry evaluation
Hu *et al.* [15]	Portable embedded sensing platform	LiDAR integration with ARM M4 controller	Embedded sensing integration	Evaluates embedded sensing integration without transport comparison or deterministic replay
Yangyang and Qian [21]	STM32-based scanner	2D scanning LiDAR hardware/sensor implementation	Embedded scanner implementation without wireless scan-transport comparison	Reports scanner implementation characteristics rather than host reconstruction, replay, or transport-resource indicators
Ueyama *et al.* [14]	Educational autonomous mobile robot	Arduino and Hokuyo laser range finder integration	Arduino-based robot integration without binary wireless fidelity (Wi-Fi) scan-telemetry evaluation	Demonstrates robot integration rather than transport-level scan telemetry evaluation
Jafri *et al.* [22]	Multiple 2D laser scanners	Characterization and calibration	Sensor characterization and calibration rather than embedded wireless telemetry	Reports sensor-level calibration indicators without microcontroller-to-host scan-stream evaluation
Wirz *et al.* [23]	Teleoperated robots	Bidirectional transport protocol	Bidirectional robotic transport in a teleoperation context rather than a LiDAR scan-frame implementation	Evaluates robotic communication behavior without RPLidar A1–ESP32 framing, recording, or replay
This work	ESP32 + RPLidar A1 + FastAPI host	Complete 2D LiDAR scan telemetry	Common binary LS scan frame over UDP/ binary (BIN), TCP/BIN, and MQTT/BIN	Status observability, WebSocket viewer, recording/replay, CPU/RAM/heap/queue metrics, N1/N2 Wi-Fi profiles

As illustrated in Table 1, prior research has established the feasibility of integrating low-cost, embedded, robotic, calibration-oriented, and transport-oriented LiDAR systems. Nonetheless, these studies also identify a prevalent implementation pattern: the processes of LiDAR acquisition, processing, navigation, visualization, middleware integration, or telemetry analysis frequently depend on personal computers, onboard computers, industrial computers, ARM-based boards, and other computing units that utilize operating systems. Consequently, microcontroller-native wireless scan telemetry remains a practical gap

when complete LiDAR revolutions must be streamed from low-cost embedded hardware while preserving compact payloads, bounded buffering, host-side observability, and reproducible experimental records. Unlike prior embedded LiDAR studies centered on sensor integration, mapping, or calibration, this work treats complete wireless scan telemetry as the primary experimental object while maintaining a common scan representation and host workflow across three transport modes.

Implementing this transition presents significant challenges. A comprehensive 2D LiDAR revolution may encompass hundreds of angle–distance–quality measurements, necessitating that the acquisition unit maintain scan boundaries, encode measurements efficiently, prevent uncontrolled packet sizes and unnecessary internet protocol (IP)-level fragmentation, and ensure adequate observability to differentiate sensor, transport, buffering, and reconstruction events. In the absence of these mechanisms, issues such as jitter, packet loss, incomplete fragments, buffer pressure, and insufficient instrumentation may undermine the continuity and reproducibility of wireless scan stream data.

This study addresses this gap by presenting an ESP32-based telemetry pipeline for streaming complete RPLidar A1 2D scan revolutions over Wi-Fi using a compact binary format. The firmware reads LiDAR samples through universal asynchronous receiver-transmitter (UART), filters invalid measurements, detects revolution boundaries, packs each scan into a binary frame, and transmits the payload through UDP/BIN, TCP/BIN, or MQTT/BIN. On the host side, a Python FastAPI/Uvicorn service reconstructs the canonical scan representation, exposes runtime counters, forwards live scans to a browser-based WebSocket viewer, and supports recording and deterministic replay.

The main contributions of this study are fourfold. First, it defines a lightweight binary scan frame for ESP32-based 2D LiDAR telemetry, including sequence numbers, timestamps, point counts, and 5-byte polar measurement records. Second, it implements a common acquisition and host-processing pipeline that supports UDP/BIN, TCP/BIN, and MQTT/BIN without changing the scan data model. Third, it integrates observability, recording, and replay to support reproducible experiments using consistent metrics. Fourth, it evaluates the pipeline through 18 official runs under nominal and degraded-received signal strength indicator (RSSI) indoor Wi-Fi profiles, after preliminary characterization of metadata encoding, pulse-width modulation (PWM) duty cycle, quality-threshold behavior, UDP fragmentation, and reconstruction stability.

2. METHOD

This section describes the ESP32-based telemetry pipeline used to stream complete 2D LiDAR scan revolutions over Wi-Fi while varying only the transport mode among UDP/BIN, TCP/BIN, and MQTT/BIN.

2.1. System overview and hardware setup

The experimental setup comprised an RPLidar A1 2D scanning LiDAR, an ESP32 DevKit V1 development board, a host personal computer (PC), and a smartphone hotspot serving as a wireless access point. The RPLidar A1 functioned as a planar range sensor, delivering angular distance measurements via a transistor–transistor logic (TTL) UART interface at 115200 baud [24]. The ESP32 operated as the embedded acquisition and telemetry node, processing the LiDAR serial stream, managing the motor line through PWM, organizing valid samples by revolution, and transmitting each complete scan via Wi-Fi [25].

The LiDAR UART lines were interfaced with the ESP32 hardware serial interface, with the LiDAR TX line connected to the ESP32 RX input and the LiDAR RX line connected to the ESP32 TX output. A common ground reference was maintained between the sensor and the ESP32 board. The motor control line was driven from the ESP32 through a PWM output using a fixed duty cycle during the experimental campaigns. Figure 1 illustrates both the actual wiring configuration employed in the prototype and the logical connection summary, including the UART cross-connections, common ground, power input, PWM motor control path, and USB/Wi-Fi host link. The host service ran on a Windows 11 laptop with an AMD Ryzen 5 7520U processor at 2.80 GHz and 16 GB of RAM. The software stack included Python 3.13.7, FastAPI/Uvicorn for HTTP and WebSocket services, and a local Mosquitto broker for MQTT/BIN experiments. The service was launched with the command $python\ -m\ uvicorn\ lidar_service{:}app\ --host$ 0.0.0.0 $--port$ 8000 $--log-level\ info$. During operation, the browser viewer was accessed at localhost:8000, and the status, recording, and replay endpoints were used during the experimental procedure.

The embedded node used an ESP32 DevKit V1 development board mounted on an ACEBOTT expansion shield. The wireless access point was an Infinix NOTE 40 PRO smartphones configured as a 2.4 GHz Wi-Fi hotspot. Channel selection remained under the smartphone's automatic configuration and was not fixed or independently logged. The same hotspot was used for both wireless profiles; only its physical location was changed. Automatic network switching remained disabled throughout the official N1 and N2 campaigns.

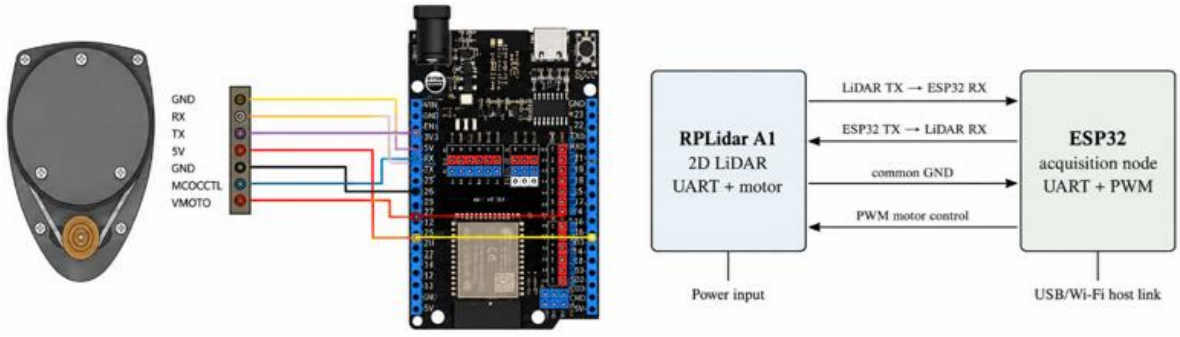


Figure 1. RPLidar A1–ESP32 wiring reference and logical connection summary

2.2. Embedded acquisition firmware

The ESP32 firmware performs the first stage of the LiDAR processing pipeline. It reads the RPLidar A1 serial stream, interprets measurement nodes, discards invalid or zero-distance samples, and preserves the quality indicator for each retained point. Each point is stored in a compact fixed-point format with angle, distance, and quality fields: angle_q6/64 gives the angle in degrees, dist_q2/4 gives the distance in millimeters, and quality occupies one byte. The quality threshold remains configurable for visualization or post-processing. A complete scan corresponds to one LiDAR revolution. The firmware detects revolution boundaries through angular wrap-around, generates a binary scan frame with the valid points, and attaches metadata such as the node identifier, sequence number, firmware timestamp, and point count. The same firmware architecture was used for UDP/BIN, TCP/BIN, and MQTT/BIN; only the configured transport path changed, isolating the transport effect while keeping the sensor, acquisition logic, binary representation, and host-side analysis constant.

2.3. Binary scan-frame format

The binary frame reduces serialization overhead while preserving scan-level traceability. Each scan or fragment contains a fixed 18-byte LS header followed by a variable-length payload of packed LiDAR points. The header includes the protocol version, node identifier, sequence number, firmware timestamp, total point count, fragment index, fragment count, and payload length. Each point occupies five bytes, and all multibyte fields are encoded in little-endian order. Table 2 summarizes the header and point-record fields.

Table 2. Binary LS scan-frame structure used by the ESP32 LiDAR telemetry pipeline

Frame part	Field	Type	Size	Description or physical conversion
Header	$m0, m1$	uint8 each	2 bytes	Magic value LS
Header	ver	uint8	1 byte	Frame version
Header	$node_id$	uint8	1 byte	LiDAR node identifier
Header	seq	uint32	4 bytes	Scan sequence number
Header	t_ms	uint32	4 bytes	Firmware timestamp in milliseconds
Header	$npts$	uint16	2 bytes	Total points in the scan
Header	$frag_idx$	uint8	1 byte	Fragment index
Header	$frag_cnt$	uint8	1 byte	Total number of fragments
Header	$payload_len$	uint16	2 bytes	Payload length in bytes
Point record	$angle_q6$	uint16	2 bytes	$angle\ [deg] = angle_q6/64$
Point record	$dist_q2$	uint16	2 bytes	$distance\ [mm] = dist_q2/4$
Point record	$quality$	uint8	1 byte	LiDAR return quality indicator

The scan payload scales with the number of valid points because each point occupies 5 bytes. For example, 700 valid points require about 3500 bytes for point data, excluding the 18-byte scan header and transport-layer overhead. This compact format was selected to reduce payload size, parsing cost, and transmission overhead compared with text-based formats.

2.4. Transport modes

Three binary transport modes were evaluated with the same scan-frame representation: UDP/BIN, TCP/BIN, and MQTT/BIN. Figure 2 summarizes the data path from RPLidar A1 sampling and binary LS framing to host-side validation, WebSocket visualization, recording/replay, and status observability. Keeping

the acquisition procedure, scan data model, validation logic, and host services constant allowed the transport path to be varied in isolation. In UDP/BIN mode, the ESP32 divides each scan into application-level fragments. Each UDP datagram carries the LS header with the scan sequence number, fragment index, fragment count, and payload length. The host reconstructs scans using the node identifier and sequence number, and discards incomplete scans after the configured 600 ms reassembly timeout. This application-level fragmentation avoids relying on IP-layer fragmentation for complete scan messages, following general UDP usage guidelines [26].

In TCP/BIN mode, the ESP32 transmits each complete scan as a binary frame over a reliable byte stream. A transport-level length prefix separates individual LS frames within the TCP stream, and the host validates each LS scan frame. This mode provides ordered and reliable delivery at the transport layer, but may introduce connection-management overhead or backpressure under adverse network conditions.

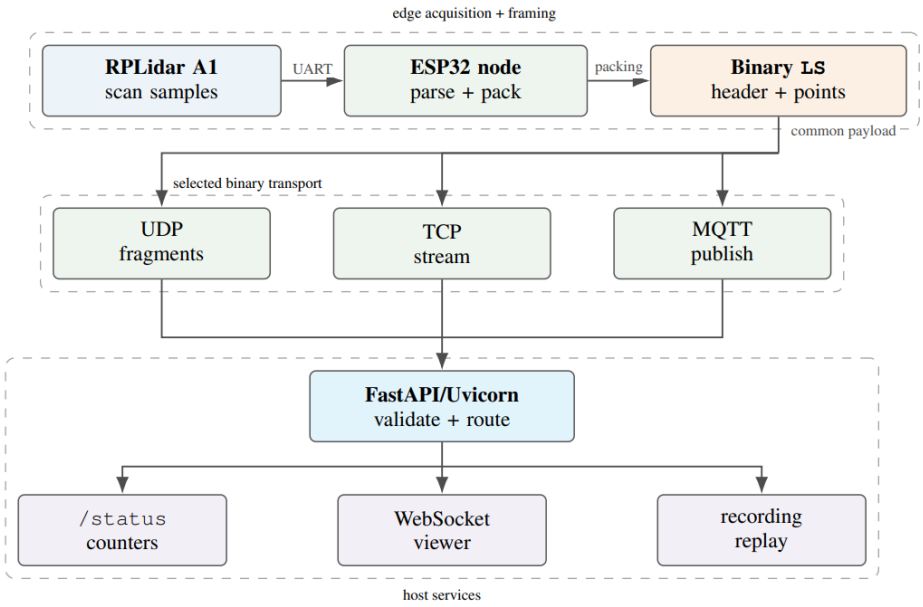


Figure 2. End-to-end architecture of ESP32-based LiDAR telemetry pipeline

In MQTT/BIN mode, the ESP32 publishes complete binary scan frames to a local Mosquitto broker running on the host PC. The host subscribes to the scan topic and validates the received payload using the same LS frame method used in the other modes. The MQTT scan topic was lidar/scan_bin, and ESP32 statistics were accessed through lidar/esp_stats. The MQTT/BIN experiments used quality of service (QoS) 0 with the broker on the same host PC.

2.5. Host service, web viewer, recording, and replay

The host service was implemented in Python using FastAPI and Uvicorn. UDP/BIN fragments were received on port 5005, TCP/BIN frames on port 5006, and MQTT/BIN frames through a local Mosquitto broker at 127.0.0.1:1883. The MQTT topics $lidar/scan_bin$ and $lidar/esp_stats$ carried scan frames and embedded statistics, respectively. After transport-specific validation or reassembly, each complete scan was converted into a canonical internal representation consisting of an 8-byte timestamp-and-point-count header followed by packed 5-byte point records. Runtime observability was provided through the /status endpoint, which reported transport counters, sequence continuity, UDP reassembly state, queue occupancy, recording/replay status, and ESP32 resource indicators. A binary WebSocket delivered live or replayed scans to the browser viewer. Figure 3 summarizes the host-side normalization, visualization, observability, recording, and deterministic-replay workflow. The same host configuration remained active throughout the 18 official runs. FastAPI/Uvicorn, the UDP and TCP listeners, the MQTT client, binary WebSocket viewer, runtime observability, and session recording were enabled in every session. Only the primary LiDAR scan transport changed among UDP/BIN, TCP/BIN, and MQTT/BIN; MQTT remained active as an auxiliary channel for ESP32 statistics during UDP/BIN and TCP/BIN runs. The viewer polled /status at approximately 1 Hz and was used only for live inspection.

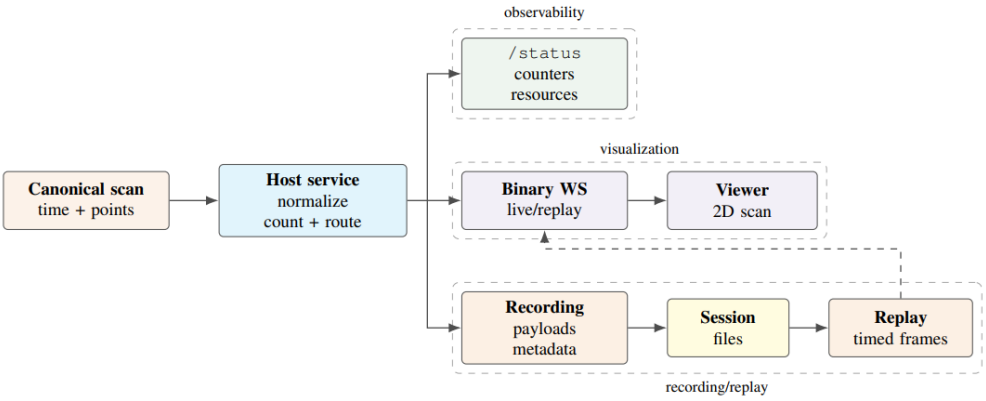


Figure 3. Host-side workflow for scan normalization, observability, visualization, recording, and deterministic replay

2.6. Preliminary characterization procedure

Preliminary experiments were conducted to select and freeze the operating parameters before the official N1/N2 campaign. These tests were used only for configuration and were not included in the primary transport comparison. The characterization comprised four stages. First, JSON and binary per-revolution metadata were compared to quantify serialization overhead; full-scan JSON streaming was not evaluated. Second, a PWM duty-cycle sweep was performed to select the motor operating point, with MQTT used only as an auxiliary command channel. Third, an offline qmin analysis evaluated point retention without applying an aggressive firmware-side quality filter. Finally, UDP packetization and reconstruction counters were examined to select the application-level fragmentation and timeout settings. The resulting acquisition, framing, buffering, quality-retention, and reconstruction parameters were fixed before the 18 official runs and remained unchanged across UDP/BIN, TCP/BIN, and MQTT/BIN.

2.7. Official experimental design

The official evaluation comprised 18 runs: two indoor Wi-Fi profiles, three transport modes, and three repetitions per protocol-profile combination. Each run lasted approximately five minutes. In N1, the smartphone hotspot, ESP32 LiDAR node, and host PC were positioned approximately 1 m apart. In N2, the ESP32 and host PC remained approximately 1 m apart, while the same hotspot was moved 9–10 m away with intervening walls. Table 3 summarizes the experimental design.

The same RPLidar A1, ESP32 node, host PC, smartphone hotspot, electrical mounting, power arrangement, sensor position and orientation, observed scene, and 55% motor duty cycle were maintained throughout the campaign. Before each recording, the system was allowed to stabilize for approximately 20 s. The hotspot served only the ESP32 and host PC, automatic network switching remained disabled, and no artificial background traffic was generated. The experiments were conducted in a stationary, moderately controlled indoor environment; temperature, humidity, Wi-Fi channel occupancy, and uncontrolled external interference were not independently measured. N2 therefore represents a lower-RSSI but operationally stable condition rather than a deliberately congested or lossy network. Its nine runs were interleaved in three MQTT–UDP–TCP cycles, although no formal randomization was applied to the complete campaign. The same automated session workflow recorded scan payloads, sequence metadata, host-process samples, and ESP32 statistics during every official run.

Table 3. Experimental design for the official Wi-Fi campaigns.

Profile	Wireless condition	Physical setup	Protocols	Repetitions
N1	Nominal indoor link	Hotspot, ESP32, and host PC approximately 1 m apart	UDP/BIN, TCP/BIN, MQTT/BIN	3 per protocol
N2	Degraded-RSSI indoor link	ESP32 and host PC approximately 1 m apart; hotspot 9–10 m away with walls in between	UDP/BIN, TCP/BIN, MQTT/BIN	3 per protocol

2.8. Metrics and aggregation procedure

The evaluation considered scan-sequence continuity, source-side scan-generation timing, wireless conditions, binary point payload, Python-process resource use, and ESP32 resource use. The indicators were

derived from the recorded sessions and were calculated independently for each official run before aggregation across repetitions.

Scan-sequence continuity was evaluated from received, expected, missing, and duplicate sequence numbers. For each run, the expected scan count was calculated as seq_max - seq_min + 1, whereas the received count corresponded to the number of unique sequence values stored during the recording. Duplicate scans were identified from repeated sequence values, and missing scans were calculated as the difference between the expected and unique received counts. Sequence numbering was not reset when recording started; therefore, a run could begin at an arbitrary sequence value. No sequence reset, counter wrap, or non-monotonic sequence was observed within the official runs. Loss percentage was calculated from the aggregated missing and expected counts.

Inter-scan intervals were calculated from the firmware-generated t_ms field stored with each received scan. This field is a 32-bit millisecond counter derived from the ESP32 millis() clock when an angular wrap was accepted as the boundary of a completed revolution. Consequently, the reported intervals characterize scan-generation timing at the embedded source, as preserved in the received sequence. They do not represent packet-arrival jitter, network latency, or end-to-end latency from LiDAR acquisition to browser rendering. Experimental-session duration was measured independently using the host monotonic clock through time.monotonic_ns(), while calendar time was used only to construct human-readable run identifiers.

Timing indicators included the run-level scan rate, mean inter-scan interval, 95th-percentile inter-scan interval, maximum inter-scan interval, and the number of intervals above 500 ms. The 500 ms threshold was used as a descriptive continuity indicator relative to the observed nominal scan-generation periods of approximately 340-405 ms. It was not treated as a hard real-time deadline and was independent of the 600 ms UDP reassembly timeout, which was used only to discard incomplete fragmented UDP scans.

Wireless conditions were characterized using the RSSI values reported by the ESP32. Payload behavior was evaluated from the mean number of points per scan and the corresponding point payload, calculated at five bytes per point. The reported point payload excludes the 18-byte LS header and all transport-layer overhead. Python/FastAPI process resource use was characterized using process CPU utilization and resident set size (RSS), with a nominal sampling period of 0.5 s. These indicators do not include the separate resource consumption of the browser, Mosquitto broker, operating system, or complete host computer. ESP32 statistics were published through the lidar/esp_stats MQTT topic at approximately 1 Hz and included RSSI, free heap, minimum heap, queue occupancy, and drop/failure counters.

For each protocol-profile combination, indicators were first calculated independently for each of the three runs. Reported mean values are arithmetic means of the three run-level estimates. The ± term associated with scan rate denotes the sample standard deviation across the three run-level scan rates. Run-wise P95 values were calculated independently and then averaged. Maximum intervals, minimum heap, and maximum queue occupancy represent the worst values observed across the three runs. RSSI ranges combine the lowest minimum and highest maximum observed across the repetitions. Event and frame counts were summed across runs, and loss percentage was calculated from the aggregated counts rather than by averaging run-level percentages.

These indicators were selected to determine whether the common binary LiDAR telemetry pipeline could preserve scan-sequence continuity and bounded resource use under the evaluated indoor profiles. They were not intended to establish a universal ranking of UDP, TCP, and MQTT or to quantify end-to-end network latency. Because the campaign included three repetitions per protocol-profile combination and was designed as an engineering characterization rather than an inferential protocol-ranking study, no hypothesis tests were performed and no confidence intervals were reported. Comparisons among transport modes are therefore descriptive and are interpreted only within the evaluated experimental conditions.

3. RESULTS AND DISCUSSION

This section reports the preliminary parameter selection, the official continuity and resource results, and the practical interpretation of the three transport modes.

3.1. Preliminary characterization and parameter selection

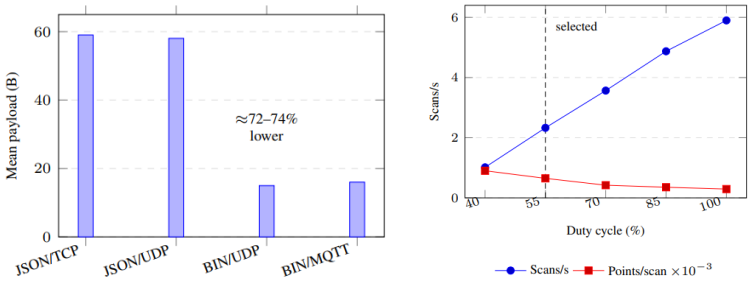
The first characterization addressed metadata encoding. Figure 4(a) shows that JSON metadata required approximately 58–59 bytes per revolution, while binary metadata required 15–16 bytes. This represents a 72–74% reduction and supports compact binary framing in the final scan pipeline. This comparison was limited to per-revolution metadata; full-scan JSON streaming was not implemented or evaluated.

The second characterization addressed the LiDAR motor PWM duty cycle. Figure 4(b) shows the expected trade-off: lower duty cycles retained more points per scan but reduced the scan refresh rate, while higher duty cycles increased scans per second but reduced point-cloud angular density. Points per scan were

scaled by 10−3 to compare this trend with scan refresh rate. MQTT was used only as an auxiliary channel for duty-cycle commands, not as the primary scan-streaming transport. A 55% duty cycle was selected as a practical compromise among refresh rate, points per scan, points per second, and reconstruction stability.

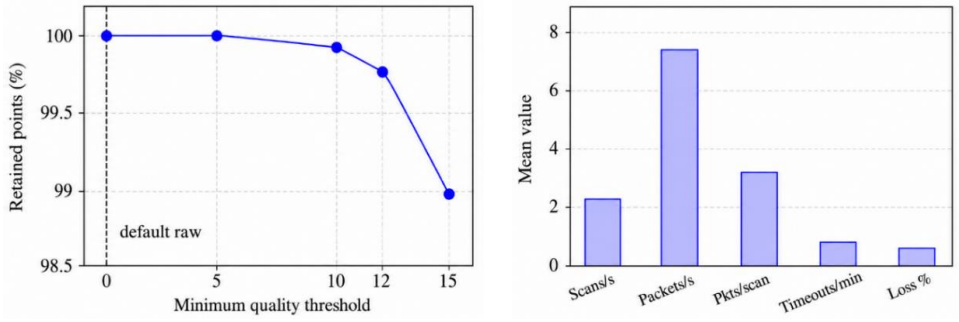
The quality-threshold behavior was evaluated offline. Figure 5(a) shows that thresholds between 10 and 15 retained approximately 99–100% of the points in the preliminary data. Therefore, the final system did not use an aggressive firmware-side quality filter. Instead, the firmware preserved the quality field for each point, and the default viewer/post-processing threshold was set to $qmin = 0$ to keep the raw telemetry. Figure 5(b) summarizes the initial UDP reconstruction indicators, including scan rate, packet rate, packets per scan, timeout drops, and estimated sequence loss. These results supported the final UDP/BIN configuration: application-level fragmentation with $PTS_PER_FRAG = 280$, a 600 ms reassembly timeout, and host-side counters for missing sequences, gaps, parse drops, timeout drops, and reconstructed scans.

The preliminary characterization supported the final operating configuration summarized in Table 4. Binary LS framing, a 55% motor duty cycle, $qmin = 0$, 280 points per UDP fragment, a latest-only UDP live policy, first-in-first-out (FIFO) application-level queueing for TCP/BIN and MQTT/BIN, and a 600 ms UDP reassembly timeout were fixed before the official campaign.



(a) (b)

Figure 4. Preliminary encoding and motor-speed characterization: (a) binary metadata reduced the per-revolution metadata payload by approximately 72–74% relative to JSON metadata and (b) a 55% PWM duty cycle was selected as the compromise between scan rate and points per scan



(a) (b)

Figure 5. Preliminary quality and UDP reconstruction characterization: (a) the evaluated quality thresholds retained approximately 99–100% of the points, supporting the use of $qmin = 0$ for raw telemetry preservation and (b) reconstruction indicators supported 280 points per UDP fragment and a 600 ms reassembly timeout

Table 4. Final parameter selection from preliminary characterization

Parameter	Final setting
Scan representation	Binary LS frame
Motor PWM duty cycle	55%
Default quality threshold	$qmin = 0$
UDP fragment capacity	280 points
UDP live policy	Latest-only
TCP/MQTT queue policy	FIFO
UDP reassembly timeout	600

3.2. Scan-sequence continuity and source-timestamp timing

The official campaign comprised 18 sessions: two Wi-Fi profiles, three transport modes, and three repetitions per protocol-profile combination. Table 5 reports the aggregated scan-sequence continuity in Table 5(a) and source-timestamp timing indicators in Table 5(b), calculated according to the procedure described in section 2.8. All six protocol-profile combinations preserved every expected sequence. No missing or duplicate sequence values were detected, resulting in 0.00% sequence loss for UDP/BIN, TCP/BIN, and MQTT/BIN under both N1 and N2. Mean scan-generation rate ranged from 2.48 to 2.59 Hz in N1 and from 2.90 to 2.94 Hz in N2. The corresponding mean source-timestamp intervals were 390.0–404.6 ms and 340.1–345.4 ms, respectively. The higher rate observed in N2 co-occurred with fewer points per scan, as shown in Table 6, and should not be interpreted as an improvement caused by the lower RSSI condition. No source-timestamp interval exceeded 500 ms; the maximum values ranged from 352 to 456 ms. Because these intervals were derived from the ESP32 scan-completion timestamps, they describe source-side revolution regularity and sequence continuity rather than packet-arrival jitter or end-to-end latency. The 500 ms threshold was a descriptive criterion relative to the observed scan periods and was independent of the 600 ms UDP reassembly timeout.

Table 5(a). Scan-sequence continuity and scan rate during the official campaign

Profile	Protocol	Runs	Rx/expected	Missing	Duplicates	Loss (%)	Scan rate (Hz)
N1	UDP/BIN	3	2461/2461	0	0	0.00	2.55 ± 0.22
N1	TCP/BIN	3	2501/2501	0	0	0.00	2.48 ± 0.22
N1	MQTT/BIN	3	2537/2537	0	0	0.00	2.59 ± 0.30
N2	UDP/BIN	3	2741/2741	0	0	0.00	2.94 ± 0.07
N2	TCP/BIN	3	2734/2734	0	0	0.00	2.94 ± 0.05
N2	MQTT/BIN	3	2696/2696	0	0	0.00	2.90 ± 0.07

Table 5(b). Source-timestamp scan-generation intervals during the official campaign

Profile	Protocol	Mean Δt (ms)	Mean run-wise P95 Δt (ms)	Maximum Δt (ms)	Intervals >500 ms
N1	UDP/BIN	394.6	397.7	437	0
N1	TCP/BIN	404.6	408.0	436	0
N1	MQTT/BIN	390.0	393.7	456	0
N2	UDP/BIN	340.1	343.7	363	0
N2	TCP/BIN	340.3	343.0	352	0
N2	MQTT/BIN	345.4	348.3	357	0

Note: counts are aggregated across three runs. Scan rate is reported as the mean ± sample standard deviation of the run-level estimates. Mean Δt and P95 Δt are means of the corresponding run-level values, whereas maximum Δt is the worst observed value. Intervals were derived from the ESP32 t_ms field and do not represent network or end-to-end latency.

3.3. Wireless conditions, point payload, and resource utilization

Table 6(a) summarizes the wireless and point-payload indicators, Table 6(b) summarizes Python-process, and ESP32 resource indicators. Mean RSSI was approximately 29–31 dB lower in N2 than in N1, confirming that N2 represented the weaker-signal profile. Complete scan-sequence continuity was nevertheless preserved under both profiles. This result does not imply that RSSI reduction is harmless in general because channel occupancy, uncontrolled interference, and external network traffic were not independently measured, and no deliberate congestion or packet-loss injection was applied.

Mean point count ranged from 711.4 to 742.7 points per scan in N1 and from 606.1 to 616.4 in N2, corresponding to point payloads of 3.48–3.63 KiB and 2.97–3.02 KiB, respectively. These values exclude the LS header and transport overhead. The lower point counts in N2 co-occurred with the higher source-side scan rates, but the experiments do not establish a causal relationship with the wireless profile.

Table 6(a). Wireless and point-payload indicators measured during the official campaigns

Profile	Protocol	RSSI mean [min,max] (dBm)	Mean points/scan	Mean point payload/scan (KiB)
N1	UDP/BIN	-43.3 [-66, -29]	723.8	3.54
N1	TCP/BIN	-43.3 [-63, -21]	742.7	3.63
N1	MQTT/BIN	-42.7 [-65, -18]	711.4	3.48
N2	UDP/BIN	-71.8 [-77, -65]	606.1	2.97
N2	TCP/BIN	-71.1 [-77, -66]	606.5	2.97
N2	MQTT/BIN	-73.6 [-84, -67]	616.4	3.02

Table 6(b). Python-process and ESP32 resource indicators measured during the official campaigns

Profile	Protocol	Python/FastAPI process			ESP32			
		CPU mean (%)	Mean run-wise CPU P95 (%)	RSS mean (MB)	Mean free heap (kB)	Minimum heap (kB)	Max queue	Drops/fails
N1	UDP/BIN	4.19	8.03	64.7	151.0	134.6	1	0
N1	TCP/BIN	4.06	8.03	61.6	130.9	107.7	1	0
N1	MQTT/BIN	4.04	8.00	57.7	131.2	116.4	1	0
N2	UDP/BIN	3.76	6.10	53.0	153.0	143.5	1	0
N2	TCP/BIN	3.74	6.10	49.2	130.1	117.7	1	0
N2	MQTT/BIN	3.80	7.97	56.4	131.2	119.7	1	0

Note: RSSI mean, point count, point payload, CPU, RSS, and mean free heap are aggregated from run-level estimates. Mean point payload equals five bytes per point and excludes the 18-byte LS header and transport overhead. CPU and RSS correspond only to the Python/FastAPI process. Minimum heap and maximum queue are worst-case values across the three runs, whereas drops/fails are aggregated event counts.

Mean Python/FastAPI process CPU utilization remained between 3.74% and 4.19%, with mean run-wise P95 values no greater than 8.03%. Process RSS ranged from 49.2 to 64.7 MB. On the ESP32, minimum heap remained above 107.7 kB, maximum queue depth was 1, and no queue drops or transport failures were recorded. These indicators show that the implemented workload operated with available heap headroom and bounded queue occupancy under the official stationary indoor conditions. They characterize the evaluated implementation, not universal host or embedded-resource requirements for the three protocols.

3.4. Protocol-level interpretation and practical scope

Transport selection affects delivery semantics, framing, buffering, and the handling of missing or delayed data [23]. Although UDP/BIN, TCP/BIN, and MQTT/BIN preserved every expected sequence in the official campaigns, the experiments were not designed to identify a universally superior protocol. UDP/BIN used application-level fragmentation and a latest-only live-scan policy, preventing pending outdated scans from accumulating. However, every fragment was required for complete reconstruction, and the implemented path provided no retransmission, redundancy, or forward-error correction. Such recovery could be added using reliable UDP mechanisms [27], but it was outside the present scope. TCP/BIN used length-prefixed frames and FIFO transmission over a reliable ordered stream, simplifying host reconstruction; under loss or congestion, however, retransmission and flow control may delay newer data [28]. MQTT/BIN used complete frames, FIFO transmission, a local broker, and QoS 0. Its publish/subscribe model supports monitoring-oriented extensions, but QoS 0 provided no MQTT-level acknowledgment or retransmission, and remote brokers, other QoS levels, and multiple subscribers were not evaluated.

Table 7 summarizes these implementation-specific trade-offs. The principal result is that a common LS frame, acquisition process, canonical host representation, recording workflow, and observability structure supported datagram, byte-stream, and broker-based transports without changing the scan data model. The interpretation remains limited to one ESP32/RPLidar A1 node, a local host and broker, stationary indoor profiles, and no deliberate traffic impairment. Multi-node deployment would require timestamp alignment, coordinated recording, and contention analysis. External deployment would additionally require security controls and long-duration validation. ROS 2 or micro-ROS interoperability remains a future extension.

Table 7. Practical interpretation of the evaluated binary transport modes

Transport mode	Implemented behavior and practical advantage	Main limitation in this study	Suitable use context
UDP/BIN	Connectionless datagrams with application-level fragmentation and host-side reassembly. The latest-only live-scan policy prevents pending outdated scans from accumulating.	A missing fragment prevents reconstruction of the complete scan. The implementation did not include retransmission, redundancy, or forward-error correction, and transport latency was not measured.	Freshness-oriented local telemetry on stable links where incomplete scans may be discarded and explicit reconstruction counters are available.
TCP/BIN	Length-prefixed LS frames over a reliable and ordered byte stream with FIFO transmission. Complete-frame delimitation and host reconstruction are simplified.	Retransmission, flow control, and ordered delivery may delay newer data under loss or congestion. Controlled impairment, backpressure, and end-to-end latency were not evaluated.	Continuous ordered scan delivery when completeness and sequence preservation are prioritized.
MQTT/BIN	Complete LS frames published through a local broker using QoS 0 and FIFO transmission. The publish/subscribe structure decouples the embedded publisher from monitoring or logging consumers.	QoS 0 provides no MQTT-level acknowledgment or retransmission. Results are limited to a local broker and the tested subscriber load; other QoS levels and broker placements were not evaluated.	Broker-based monitoring, logging, IoT integration, and future multi-client architectures after validation under the intended broker and network conditions.

4. CONCLUSION

This study developed and evaluated a transport-independent telemetry framework for streaming complete RPLidar A1 scan revolutions from an ESP32 over Wi-Fi. The framework combines a compact binary LS scan representation, UDP/BIN, TCP/BIN, and MQTT/BIN transport paths, canonical host-side reconstruction, runtime observability, browser visualization, session recording, and deterministic replay. Its principal contribution is a common and reproducible embedded-to-host architecture that enables multiple transport alternatives to be evaluated without changing the LiDAR acquisition procedure or scan data model.

The official evaluation comprised 18 approximately five-minute runs under nominal and degraded-RSSI indoor Wi-Fi profiles. Every expected scan sequence was received in all protocol-profile combinations, resulting in 0.00% sequence loss, with no duplicate sequences, queue drops, or transport failures. No ESP32 source-timestamp inter-scan interval exceeded 500 ms; however, these intervals characterize scan-generation regularity and do not represent packet-arrival jitter or end-to-end latency. The Python/FastAPI process operated with moderate CPU and memory use, while the ESP32 maintained available heap headroom and a maximum queue depth of one. These findings demonstrate the feasibility of the common binary telemetry architecture under the evaluated stationary indoor conditions, without establishing a universal ranking of UDP, TCP, and MQTT.

The study was limited to a single ESP32/RPLidar A1 node, a local host and MQTT broker with QoS 0, and wireless profiles without deliberate congestion, mobility, or packet-loss injection. Future work should evaluate controlled network impairments, synchronized multi-node deployments, remote brokers and alternative MQTT QoS levels, end-to-end timestamping, long-duration stability, security mechanisms, and ROS 2 or micro-ROS interoperability.

ACKNOWLEDGMENTS

The authors acknowledge the support of the Ministry of Science, Technology, and Innovation of Colombia (MINCIENCIAS) and the Francisco José de Caldas Fund through Call 934 of 2023, research contract No. 332-2023, project 99209. The authors also thank the LASER Research Group at Universidad Distrital Francisco José de Caldas for its academic guidance, technical discussions, and support during the development of this study. The authors extend special appreciation to BitPointer S.A.S., the beneficiary of project 99209, for providing institutional support and facilitating the conditions required for the research activities. The authors also thank its manager, Eng. Juan Fajardo Barrero, and administrative director, lawyer Alexandra Reyes Agudelo, for their support, collaboration, and dedication to this initiative.

FUNDING INFORMATION

This work was supported by the Ministry of Science, Technology, and Innovation of Colombia (MINCIENCIAS) and the Francisco José de Caldas Fund under Call 934 of 2023, research contract No. 332-2023, project 99209.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Julian Camilo Forero Gutierrez	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓			
Andrés Leonardo Jutinico Alarcón	✓	✓			✓	✓	✓			✓		✓	✓	
Henry B. Guerrero	✓	✓		✓	✓	✓			✓	✓		✓	✓	✓

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors declare no conflict of interest.

DATA AVAILABILITY

The firmware, host-side scripts, web viewer, and supporting materials are available at https://github.com/fjuliancamilo-sys/LIDAR-ESP32. Processed metrics and experimental recordings are available from the corresponding author upon reasonable request.

REFERENCES

[1] N. Li *et al.*, "A Progress Review on Solid-State LiDAR and Nanophotonics-Based LiDAR Sensors (Laser Photonics Rev. 16(11)/2022)," *Laser & Photonics Reviews*, vol. 16, no. 11, Nov. 2022, doi: 10.1002/lpor.202270057.

[2] A. N. Catapang and M. Ramos, "Obstacle detection using a 2D LIDAR system for an Autonomous Vehicle," in *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2016, pp. 441–445, doi: 10.1109/iccsce.2016.7893614.

[3] S. A. Gargoum and K. El Basyouny, "A literature synthesis of LiDAR applications in transportation: feature extraction and geometric assessments of highways," *GIScience & Remote Sensing*, vol. 56, no. 6, pp. 864–893, Feb. 2019, doi: 10.1080/15481603.2019.1581475.

[4] X. Yang, X. Lin, W. Yao, H. Ma, J. Zheng, and B. Ma, "A Robust LiDAR SLAM Method for Underground Coal Mine Robot with Degenerated Scene Compensation," *Remote Sensing*, vol. 15, no. 1, p. 186, Dec. 2022, doi: 10.3390/rs15010186.

[5] B. Li, Z. Ma, and Y. Zhao, "2D Mapping of Mobile Robot Based on micro-ROS," *Journal of Physics: Conference Series*, vol. 2402, no. 1, p. 12030, Dec. 2022, doi: 10.1088/1742-6596/2402/1/012030.

[6] S. M. Farhan, J. Yin, Z. Chen, and M. S. Memon, "A Comprehensive Review of LiDAR Applications in Crop Management for Precision Agriculture," *Sensors*, vol. 24, no. 16, p. 5409, Aug. 2024, doi: 10.3390/s24165409.

[7] H. Holzhüter, J. Bödewadt, S. Bayesteh, A. Aschinger, and H. Blume, "Technical concepts of automotive LiDAR sensors: a review," *Optical Engineering*, vol. 62, no. 03, Jan. 2023, doi: 10.1117/1.oe.62.3.031213.

[8] N. Takhtkeshha, G. Mandlburger, F. Remondino, and J. Hyyppä, "Multispectral Light Detection and Ranging Technology and Applications: A Review," *Sensors*, vol. 24, no. 5, p. 1669, Mar. 2024, doi: 10.3390/s24051669.

[9] U. Seidaliyeva *et al.*, "LiDAR Technology for UAV Detection: From Fundamentals and Operational Principles to Advanced Detection and Classification Techniques," *Sensors*, vol. 25, no. 9, p. 2757, Apr. 2025, doi: 10.3390/s25092757.

[10] I. Niskanen, G. Duan, T. Kolli, M. Immonen, T. Makkonen, and R. Heikkilä, "Volume measurement of soil in excavator buckets by solid-state 3D LiDAR," *Construction Robotics*, vol. 9, no. 2, 2025, doi: 10.1007/s41693-025-00166-w.

[11] J. A. Christian and S. Cryan, "A Survey of LIDAR Technology and its Use in Spacecraft Relative Navigation," *AIAA Guidance, Navigation, and Control (GNC) Conference*, Aug. 15, 2013. doi: 10.2514/6.2013-4641.

[12] V. A. H. Higuti, A. E. B. Velasquez, D. V. Magalhaes, M. Becker, and G. Chowdhary, "Under canopy light detection and ranging-based autonomous navigation," *Journal of Field Robotics*, vol. 36, no. 3, pp. 547–567, Dec. 2018, doi: 10.1002/rob.21852.

[13] P. Escandón-Panchana, G. Herrera-Franco, M. Jaya-Montalvo, and S. Martínez-Cuevas, "Geomatic tools used in the management of agricultural activities: a systematic review," *Environment, Development and Sustainability*, vol. 27, no. 7, pp. 15275–15309, Mar. 2024, doi: 10.1007/s10668-024-04576-8.

[14] Y. Ueyama, T. Sago, T. Kurihara, and M. Harada, "An Inexpensive Autonomous Mobile Robot for Undergraduate Education: Integration of Arduino and Hokuyo Laser Range Finders," *IEEE Access*, vol. 10, pp. 79029–79040, 2022, doi: 10.1109/access.2022.3194162.

[15] B. Hu, S. Hsiung, and M. Kersey, "Design a Portable Sensing Platform with a Lidar and TI ARM M4 Controller," *2019 ASEE Annual Conference & Exposition Proceedings*, doi: 10.18260/1-2--32588.

[16] A. Abanay, L. Masmoudi, M. El Ansari, J. Gonzalez-Jimenez, and F.-A. Moreno, "LIDAR-based autonomous navigation method for an agricultural mobile robot in strawberry greenhouse: AgriEco Robot," *AIMS Electronics and Electrical Engineering*, vol. 6, no. 3, pp. 317–328, 2022, doi: 10.3934/electreng.2022019.

[17] H. Guo *et al.*, "Research on autonomous navigation system of greenhouse electric crawler tractor based on LiDAR," *Frontiers in Plant Science*, vol. 15, May 2024, doi: 10.3389/fpls.2024.1377269.

[18] E.-T. Baek and D.-Y. Im, "ROS-Based Unmanned Mobile Robot Platform for Agriculture," *Applied Sciences*, vol. 12, no. 9, p. 4335, Apr. 2022, doi: 10.3390/app12094335.

[19] A. E. B. Velasquez *et al.*, "Reactive navigation system based on H∞ control system and LiDAR readings on corn crops," *Precision Agriculture*, vol. 21, no. 2, pp. 349–368, 2019, doi: 10.1007/s11119-019-09672-8.

[20] H. B Guerrero, D. C. Montenegro, and A. L. Jutinico, "Reactive fuzzy control and field validation of a low-cost autonomous tracked robot for plantain crop row following," *Engineering Research Express*, vol. 8, no. 4, p. 45227, Feb. 2026, doi: 10.1088/2631-8695/ae458e.

[21] W. Yangyang and K.-Y. Qian, "2-D Scanning LiDAR with MEMS Mirror and STM32," *IOP Conference Series: Materials Science and Engineering*, vol. 563, no. 3, p. 32051, 2019, doi: 10.1088/1757-899x/563/3/032051.

[22] S. R. un N. Jafri, S. Shamim, S. M. Faraz, A. Ahmed, S. M. Yasir, and J. Iqbal, "Characterization and calibration of multiple 2D laser scanners," *PLoS ONE*, vol. 17, no. 7, p. e0272063, Jul. 2022, doi: 10.1371/journal.pone.0272063.

[23] R. Wirz, R. Marin, M. Ferre, J. Barrio, J. M. Claver, and J. Ortego, "Bidirectional Transport Protocol for Teleoperated Robots," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 9, pp. 3772–3781, 2009, doi: 10.1109/tie.2009.2025291.

[24] Seeed Studio, "Slamtec LiDAR Documentation Hub: A1 Series LiDAR datasheet, user manual, SDK, and 3D models," 2025.

[25] N. Cameron, "ESP32 microcontroller features," in *Electronics Projects with the ESP8266 and ESP32*, Apress, 2020, pp. 641–682, doi: 10.1007/978-1-4842-6336-5_22.

[26] L. Eggert, G. Fairhurst, and G. Shepherd, "UDP Usage Guidelines," *RFC Editor*, Mar. 2017. doi: 10.17487/rfc8085.

[27] M.-H. Wang, L.-W. Chen, P.-W. Chi, and C.-L. Lei, "SDUDP: A Reliable UDP-Based Transmission Protocol Over SDN," *IEEE Access*, vol. 5, pp. 5904–5916, 2017, doi: 10.1109/access.2017.2693376.

[28] A. Simpson, M. Alshaali, W. Tu, and M. R. Asghar, "Quick UDP Internet Connections and Transmission Control Protocol in unsafe networks: A comparative analysis," *IET Smart Cities*, vol. 6, no. 4, pp. 351–360, May 2024, doi: 10.1049/smc2.12083.

BIOGRAPHIES OF AUTHORS



Julian Camilo Forero Gutierrez is an undergraduate student pursuing the B.Eng. degree in Electronic Engineering from Universidad Distrital Francisco José de Caldas, Bogotá, Colombia. His academic and technical interests include embedded systems, the internet of things, real-time telemetry, communication protocols, control, and automation. His work focuses on the end-to-end integration of sensing devices, microcontroller-based acquisition, wireless data transmission, host-side services, and visualization tools for monitoring and analyzing the experiments. In the present study, he contributed to the embedded firmware, host-side implementation, experimental data acquisition, data curation, visualization, and technical analysis of the ESP32-based 2D LiDAR telemetry pipeline. He can be contacted at email: jucforerog@udistrital.edu.co.



Andrés Leonardo Jutinico Alarcón received the B.Eng. degree in Electronic Engineering from Universidad Distrital Francisco José de Caldas, Bogotá, Colombia, in 2005, the M.Sc. degree in Industrial Automation from the National University of Colombia in 2012, and the Ph.D. degree in Mechanical Engineering from the University of São Paulo, Brazil, in 2019. His current research interests include control systems, assistive and rehabilitation robotics, sensor fusion, robotic perception, and embedded technologies for assistive applications. In the present study, he contributed to the conceptualization, methodology, formal analysis, investigation, resources, supervision, and manuscript review. He can be contacted at email: aljutinicoa@udistrital.edu.co.



Henry B. Guerrero received the B.Eng. degree in Electronic Engineering from Universidad de los Llanos, Villavicencio, Colombia, in 2004, and a Ph.D. degree in Mechanical Engineering from the University of São Paulo, Brazil, in 2016. He has conducted postdoctoral research for BitPointer S.A.S. in Villavicencio, Colombia, in collaboration with the Ministry of Science, Technology, and Innovation of Colombia. His current research interests include control systems, agricultural mobile robotics, embedded sensing, autonomous navigation, and robotic platforms for field applications. In the present study, he contributed to the conceptualization, methodology, validation, supervision, project administration, funding acquisition, and manuscript review. He can be contacted at email: hbguerrero@ieee.org.