BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

Latency and Reliability Evaluation of an HTTP-Controlled ESP8266 Wi-Fi Robot Car Using MIT App Inventor

Ryan Fikri*, Agariadne Dwinggo Samala, Thamrin, Delsina Faiza

Department of Electronics Engineering, Faculty of Engineering, Universitas Negeri Padang, Padang, Indonesia
Email: [1,*]ryanfikri@ft.unp.ac.id, [2]agariadne@ft.unp.ac.id, [3]thamrin@ft.unp.ac.id, [4]delsinafaiza@ft.unp.ac.id
Corresponding author email: ryanfikri@ft.unp.ac.id

Abstract−Low-cost Wi-Fi robot cars are frequently presented as functional prototypes, but many studies do not quantify whether motion commands remain reliable and responsive as wireless distance increases. This study therefore designed and experimentally evaluated an ESP8266-based differential-drive robot controlled by an Android application developed with MIT App Inventor. The proposed solution applies an HTTP command-mapping method in which each mobile-interface event is converted into a request, parsed by an ESP8266 web server, mapped to an L298N H-bridge state, and acknowledged after the control action is issued. The objective was to determine functional command accuracy and characterize the relationship among control distance, command-response latency, received signal strength indicator (RSSI), and communication reliability. Five commands forward, backward, left, right, and stop were tested in 150 functional trials, while 500 communication trials were conducted at 1, 5, 10, 15, and 20 m under indoor line-of-sight conditions. The robot correctly executed 148 of 150 functional commands, corresponding to 98.67% success. Distance-based reliability remained 100% at 1–5 m, decreased to 99% at 10 m and 97% at 15 m, and reached 92% at 20 m. Mean command-response latency increased from 88 to 248 ms while RSSI declined from −38 to −77 dBm. The main contribution is a reproducible, low-cost evaluation framework that links interface commands, HTTP communication, wireless quality, and physical motion execution. The results indicate that the platform is appropriate for responsive laboratory teleoperation within 15 m, while operation near 20 m requires stronger fail-safe and acknowledgement mechanisms.

Keywords: Internet of Things; Mobile Robot; ESP8266; MIT App Inventor; HTTP Control; Performance Analysis; Latency Evaluation

1. INTRODUCTION

The Internet of Things (IoT) has extended embedded systems from isolated controllers into networked physical devices that exchange data, receive commands, and coordinate actions through communication services. In mobile robotics, this integration links a human interface, a wireless channel, an embedded controller, power electronics, and electromechanical actuators. Recent studies describe connected robots as cyber-physical systems whose usefulness depends not only on mechanical construction but also on dependable communication and transparent software integration [1], [2]. Although advanced platforms increasingly employ edge intelligence, middleware, and autonomous navigation, educational laboratories and small-scale applications still require architectures that can be assembled, inspected, and maintained with limited resources. A low-cost robot car is useful in this context because the complete command path remains visible: an interface event generates a network message, the microcontroller interprets the message, and the motor driver produces a physical response.

Hands-on mobile robots are particularly valuable in electronics, computing, and mechatronics education because one artifact exposes learners to embedded programming, wireless networking, actuator control, power management, and system-level troubleshooting. Open IoT laboratories and multidisciplinary AGV or AMR activities have been reported to support experiential learning by connecting software decisions with observable physical processes [3], [4]. Recent reviews and empirical studies further emphasize that robotics laboratories, problem-based activities, and first-principle platforms can develop systems thinking when students are able to inspect rather than merely operate the technology [5], [6], [7]. Low-cost mobile platforms also offer a practical bridge from foundational actuation and communication to more advanced sensing, control, and autonomy [8], [9]. For beginners, however, this educational value depends on an interface that is accessible enough to permit experimentation without requiring a complete native-mobile development workflow.

The user interface therefore becomes a technical and pedagogical component of the system rather than a cosmetic addition. Conventional Android development requires familiarity with programming languages, software development kits, activity lifecycles, and asynchronous network handling. Visual programming environments reduce this entry barrier by representing application logic as events and connected blocks. A systematic review found that visual programming is widely used for end-user development in IoT, robotics, and education because it makes program structure more inspectable and lowers the initial coding burden [10]. MIT App Inventor follows this approach through screen components and block-based event handlers; it has also been used to create mobile interfaces that collect and exchange IoT data [11]. In a robot-control application, directional buttons, a speed slider, and an HTTP Web component can therefore be implemented without building a fully coded Android project, allowing novice users to focus on the relationship between the command and the robot response.

At the hardware and communication levels, the NodeMCU ESP8266 provides a low-cost microcontroller with integrated 2.4 GHz Wi-Fi and sufficient general-purpose input/output resources to operate a dual H-bridge driver. It can host a lightweight HTTP server, receive a motion request, and generate digital direction and pulse-width-modulation signals without an additional wireless module. A related NodeMCU robot-car study demonstrated the feasibility of smartphone-based Wi-Fi control but concentrated mainly on prototype construction [12]. Feasibility alone is insufficient for teleoperation because the operator experiences the combined effects of request processing, wireless propagation,

retransmission, signal degradation, and actuator response. For this reason, latency, RSSI, command success, and operating distance should be evaluated together. RSSI varies substantially across indoor positions [13], while long-range robot studies show that reliability and delay deteriorate as propagation conditions become more demanding [14].

Several communication approaches provide useful comparison points. Bluetooth-controlled mobile robots can achieve low short-range latency, but reported performance depends on the command modality and controller task. Gupta et al. reported approximately 110–140 ms for an Android-controlled mobile robot with manual and speech interaction [15], whereas a Bluetooth self-balancing platform achieved a lower mean latency of about 55 ms under shorter-range conditions [16]. LoRa-based teleoperation extends coverage to kilometer scale but accepts lower throughput and increasing packet loss near the range boundary [14]. Mixed-reality and ROS 2 interfaces offer richer monitoring and interaction, although they require more software infrastructure than an introductory platform [17]. These studies demonstrate that no single wireless approach is universally superior: the appropriate design depends on range, response time, interface complexity, and safety requirements. An ESP8266–HTTP architecture is attractive for instruction because each request can be directly inspected, but its practical range and response consistency must be quantified rather than assumed.

The research gap addressed in this study is the limited quantitative evidence accompanying elementary ESP8266 robot-car implementations. Many prototype reports explain wiring and demonstrate forward, backward, left, right, and stop functions, yet do not connect those functions to repeated measurements of command success, response latency, RSSI, and control distance. Consequently, a prototype may appear responsive during a brief demonstration while producing delayed or missed commands under weaker signal conditions. The gap is distinct from autonomous-navigation research, which focuses on sensing, localization, path planning, or machine-learning control [18], [19], [20], [21]. Before those higher-level functions are added, the manual command layer must provide deterministic command mapping and a documented communication envelope. The present study therefore treats teleoperation reliability as the primary engineering problem rather than claiming a new autonomous algorithm.

This study aims to design and evaluate an HTTP-controlled differential-drive robot car using a NodeMCU ESP8266, an L298N motor driver, and MIT App Inventor. Its contributions are: (1) an explicit HTTP command-mapping method linking mobile events to H-bridge states; (2) a reproducible hardware–software architecture suitable for low-cost laboratories; (3) repeated functional and distance-based tests that jointly evaluate command success, command-response latency, and RSSI; and (4) an evidence-based operating recommendation supported by comparison with recent connected-robot studies. The novelty lies in the integrated performance evaluation of an accessible low-code interface and a low-cost Wi-Fi control layer, not merely in assembling common components. This positioning makes the study useful both as an instructional implementation and as a baseline for future comparisons with WebSocket, MQTT, sensor feedback, and autonomous functions.

2. RESEARCH METHODOLOGY

2.1 Research Framework and Scope

This study used an engineering design and experimental-evaluation framework to transform a low-cost robot-car prototype into a testable connected robotic system. The object of study was a four-wheel differential-drive platform controlled from an Android smartphone through a local 2.4 GHz Wi-Fi connection. The research scope covered requirement definition, hardware integration, HTTP command mapping, firmware and interface implementation, functional verification, and communication-performance testing. Autonomous navigation, obstacle avoidance, localization, and learning outcomes were excluded because the primary objective was to establish whether the basic manual-control layer remained accurate and responsive over increasing distance. This boundary is consistent with first-principle robotics approaches that validate physical, electrical, embedded, and control layers before adding perception and planning functions[6], [9]. The work was conducted in the Electronics Engineering laboratory at Universitas Negeri Padang. The system was considered operational when every interface command mapped to the intended motor state, the robot maintained a Wi-Fi connection across the test distances, and the communication outcomes could be expressed through repeatable success, latency, and RSSI measurements.

2.2 Research Stages and HTTP Command-Mapping Method

The study followed seven sequential stages, as introduced in Figure 1. First, the functional requirements were defined as five deterministic commands forward, backward, left, right, and stop together with adjustable motor power through pulse-width modulation (PWM). Second, the NodeMCU ESP8266, L298N driver, four geared DC motors, battery pack, and acrylic chassis were integrated. Third, the HTTP command-mapping method was specified. Each button event in MIT App Inventor generated an HTTP request containing a command token; the ESP8266 server parsed the token, selected the corresponding H-bridge state, applied the PWM value, and returned an acknowledgement. Fourth, the mobile interface and embedded firmware were implemented. Fifth, short-range functional verification checked whether the requested and observed motions agreed. Sixth, repeated communication trials were conducted at five distances. Finally, the results were analyzed through command success rate, mean and standard-deviation latency, RSSI trends, confidence intervals, and aggregate-level correlation. This staged approach separates construction errors from communication degradation and

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

makes the solution process visible rather than presenting the prototype as a single undifferentiated assembly.

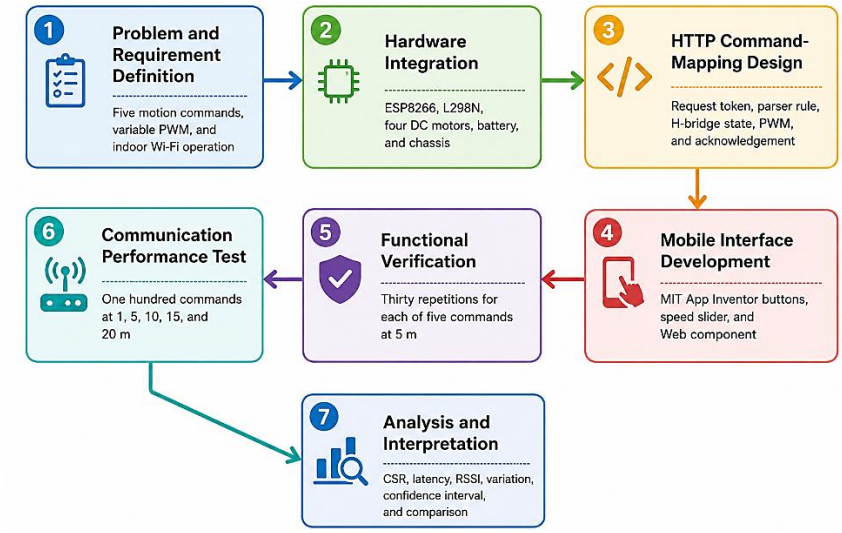

Figure 1. Research and Experimental Evaluation Stages

2.3 Hardware Architecture and Pin Mapping

The proposed architecture is introduced in Figure 2. The Android smartphone provides the human–machine interface, while the ESP8266 acts simultaneously as the local Wi-Fi endpoint, HTTP web server, and motor-control processor. After receiving a request, the controller generates GPIO direction states and PWM enable signals for the L298N. The motor driver supplies bidirectional current to two motor groups, one on each side of the chassis, thereby producing differential-drive motion. A common electrical ground is required between the logic and motor subsystems. The architecture deliberately avoids an external cloud broker so that command processing remains local, inspectable, and independent of internet availability. This design choice supports the low-cost and reproducible character of the study while preserving a measurable request–acknowledgement path.

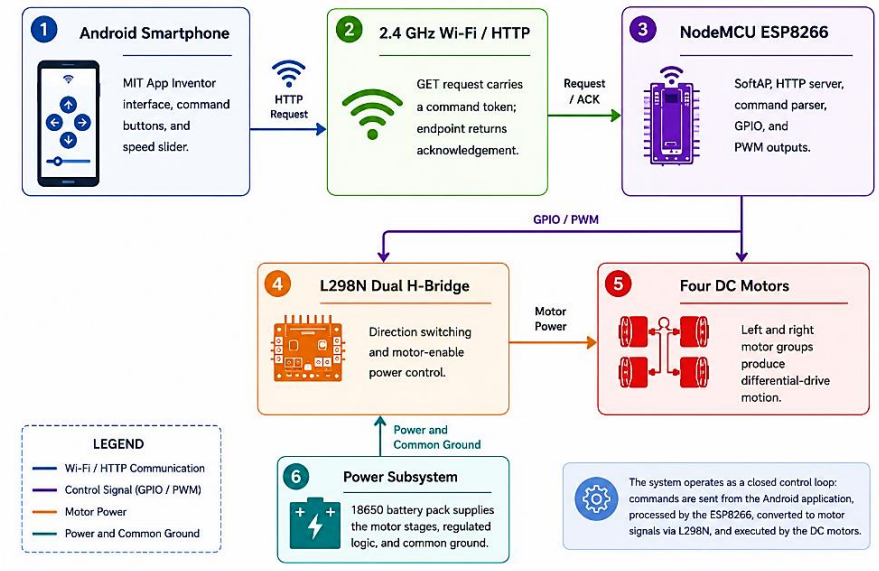

Figure 2. Hardware and HTTP Communication Architecture

As illustrated in Figure 2, the control path and power path are distinct but interdependent. The smartphone and ESP8266 exchange low-power network messages, whereas the L298N and battery deliver the current required by the motors. This separation helps identify whether a failed motion originates from the wireless request, the command parser, the logic signals, or the motor-power stage. It also supports future protocol comparisons without changing the mechanical platform.

Table 1 lists the components used to implement the architecture. Component selection prioritized availability, direct compatibility with the Arduino development environment, and sufficient functionality for a laboratory teleoperation platform. The ESP8266 provides the integrated Wi-Fi interface, the L298N supplies two bidirectional motor channels, and MIT App Inventor implements the low-code mobile interface.

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

Table 1. Main Hardware and Software Components

Component	Specification	Primary function
NodeMCU ESP8266 V3	ESP8266 SoC; 2.4 GHz Wi-Fi	Hosts the local HTTP server and generates motor-control signals
L298N motor driver	Dual H-bridge; two enable channels	Controls motor direction and PWM power for left and right groups
DC geared motors	Four low-voltage geared motors	Produce four-wheel differential-drive motion
Battery pack	Rechargeable 18650 cells	Supplies the motor and controller circuits
Android smartphone	Wi-Fi-enabled device	Runs the MIT App Inventor control application
Arduino IDE	ESP8266 firmware environment	Compiles and uploads the embedded program
MIT App Inventor	Block-based Android development environment	Implements command buttons, speed control, and HTTP requests

Table 1 confirms that the proposed system can be reproduced with commonly available modules and free development tools. Each component has a single identifiable role, which reduces hidden dependencies and facilitates troubleshooting. The four motors are connected as left and right groups so that the L298N can control the chassis through two H-bridge channels rather than four independent drivers.

Table 2. L298N and NodeMCU ESP8266 Pin Mapping

L298N pin	NodeMCU pin	Control role
ENA	D5	PWM enable for motor group A
ENB	D6	PWM enable for motor group B
IN1	D8	Direction input A1
IN2	D7	Direction input A2
IN3	D4	Direction input B1
IN4	D3	Direction input B2

Table 2 defines the electrical mapping used by the HTTP command table. D5 and D6 provide PWM enable signals for the two motor groups, while D8, D7, D4, and D3 determine H-bridge polarity. This mapping must remain identical in the firmware and physical wiring; otherwise, a valid network command can still produce an incorrect direction. After the mapping was verified, the complete wiring arrangement was documented in Figure 3.

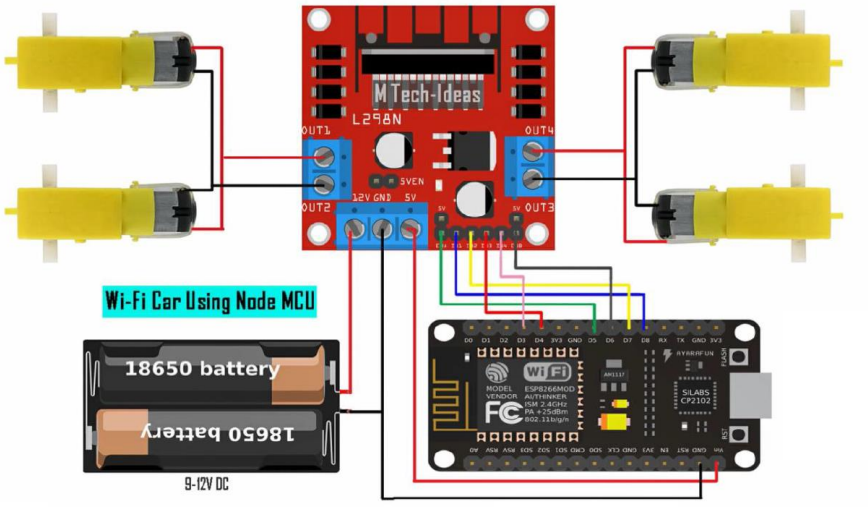

Figure 3. NodeMCU, L298N, Motor, and Battery Wiring Arrangement

Figure 3 provides the implementation-level connection reference for the prototype. The two motors on each side are connected to the same driver channel, the battery supplies the motor stage, and the NodeMCU shares the reference ground with the L298N. The grayscale redraw improves print readability while retaining the wire paths required to reproduce
the assembly.

2.4 Embedded Firmware and Mobile Interface Implementation

The embedded solution applies a deterministic HTTP command-mapping procedure. After startup, the firmware configures the motor pins as outputs, sets both enable channels to a safe stopped state, initializes the ESP8266 network,

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

and starts the HTTP server. For each incoming request, the parser checks the command token against the permitted set {forward, backward, left, right, stop}. A valid token selects a predefined combination of IN1–IN4 states, after which the current PWM value is applied to ENA and ENB. Forward and backward commands drive both motor groups with matching polarity; left and right commands apply opposite or asymmetric states to create differential steering; and stop disables the motor outputs. Invalid or incomplete requests invoke the stop state rather than retaining an undefined command. The endpoint then returns an acknowledgement used to calculate command-response latency. This procedure is a rule-based command-mapping method, not an autonomous navigation algorithm. Its advantage is deterministic behavior: for a given token and PWM value, the expected output state is known in advance and can be verified directly at the GPIO and motor levels.

The mobile side was implemented with MIT App Inventor using directional buttons, a central stop button, a speed slider, and the Web component. Each button event constructed the ESP8266 endpoint URL and sent the corresponding HTTP request. The speed slider updated the PWM parameter independently from direction selection. Figure 4 introduces the two interface states used during operation: connection to the robot access point and the command screen. The simplified interface was selected to reduce touch ambiguity and to keep the relationship among button event, URL, command token, and physical action visible to novice users, consistent with the accessibility advantages of visual programming [9], [11].

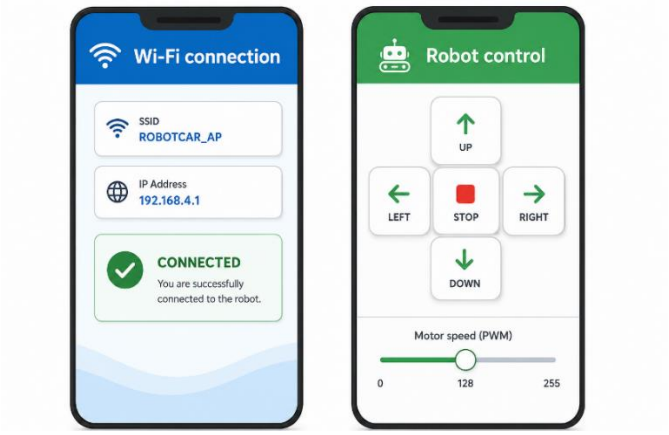

Figure 4. Wi-Fi Connection and MIT App Inventor Control Interface

Figure 4 shows that network configuration and motion control are separated into clear interface functions. The fixed local address reduces setup complexity, while the large command buttons and centrally located stop control prioritize unambiguous manual operation. The interface intentionally avoids decorative elements that do not contribute to command generation or status recognition.

2.5 Experimental Setup and Evaluation Metrics

Functional testing evaluated the five movement commands at a nominal smartphone-to-robot distance of 5 m on a flat laboratory floor. Each command was issued 30 times, producing 150 trials. A trial was classified as successful when the observed wheel motion matched the requested direction without an incorrect intermediate action and the response occurred within one second. The same assembled prototype, firmware build, mobile interface, battery configuration, floor surface, and device orientation were maintained throughout the sequence. Commands were distributed across the test session rather than evaluating only a single continuous movement, thereby requiring repeated transitions among motor states. The functional test was intended to verify the complete command chain; it did not measure trajectory error or turning angle because the platform did not include wheel encoders or external motion tracking.

Communication performance was tested under indoor line-of-sight conditions at 1, 5, 10, 15, and 20 m. At each distance, 100 HTTP commands were transmitted using a balanced sequence of the five motion instructions, resulting in 500 trials. Command-response latency was defined as the interval between request transmission by the mobile application and receipt of the ESP8266 endpoint acknowledgement. It therefore represents interface-to-controller response, not the time until measurable chassis displacement. RSSI was recorded to characterize the wireless condition associated with each distance. The same network configuration and approximate antenna orientation were maintained, and no intentional obstacle was introduced. A command was counted as successful when the request was acknowledged and the intended motion state was observed. These controls do not eliminate all indoor interference, but they reduce avoidable variation and allow the distance trend to be interpreted within the tested laboratory environment.

Table 3. Experimental Scenarios and Evaluation Metrics

Test	Trials	Controlled/independent condition	Measured output and decision rule
Functional command test	30 × 5 commands	Command type; 5 m; flat floor	Correct observed motion within 1 s; count and CSR

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

Test	Trials	Controlled/independent condition	Measured output and decision rule
Distance reliability test	100 commands × 5 distances	Distance: 1, 5, 10, 15, 20 m; line of sight	Acknowledged correct motion; success rate and Wilson interval
Latency and RSSI test	Same 500 distance trials	Distance and recorded RSSI	Request-to-acknowledgement mean latency and standard deviation
Integrated operation test	Continuous manual sequence	Combined hardware–software operation	Stable connection, correct motor actuation, and stop response

Table 3 connects each research question to a repeatable test and measurable output. The functional test answers whether the command table is correct, while the distance test evaluates whether that correctness is maintained as the signal weakens. The latency and RSSI records provide diagnostic information that cannot be obtained from success rate alone, because a command may still arrive successfully while becoming slower or less predictable.

Command success rate (CSR) was calculated with Equation (1), where Ns is the number of correctly executed commands and Nt is the total number of transmitted commands.

$$CSR = \left(\frac{N_s}{N_t}\right) \times 100\% \quad (1)$$

Mean command-response latency and its sample standard deviation were calculated with Equation (2), where Li is the latency of trial i, $\bar{L}$ is the mean latency, and n is the number of successful latency observations.

$$\bar{L} = \left(\frac{1}{n}\right) \sum L_i \quad (2)$$

The analysis was primarily descriptive because the experiment characterized one prototype in one controlled environment. Wilson 95% confidence intervals were calculated for binomial success proportions. Pearson coefficients were also calculated across the five aggregate distance points to explore the relationships among distance, RSSI, latency, and success; because n = 5 at this level, these coefficients were interpreted as trend indicators rather than population estimates. The operating recommendation was based jointly on command success, mean latency, latency variability, and signal strength. No claim was extended beyond the tested hardware, application, and indoor line-of-sight configuration.

3. RESULTS AND DISCUSSION

3.1 Execution of the HTTP Method and Prototype Implementation

Implementation followed the stages shown in Figure 1. The first verification point was the motor-command table: each HTTP token was issued while the chassis was raised so that wheel direction could be inspected without translational movement. After the polarity of both motor groups matched the defined table, the chassis was placed on the floor and the same commands were repeated under load. The completed prototype integrated the NodeMCU, L298N driver, battery pack, wiring, and four motors on an acrylic chassis, as shown in Figure 5. The controller and driver remained physically accessible, which simplified inspection of the logic and power paths. During operation, the smartphone connected to the robot network, the interface transmitted a command request, the ESP8266 parser selected the corresponding H-bridge state, and the server returned an acknowledgement. This sequence confirmed that the proposed method solved the stated problem through a transparent chain of rule-based decisions rather than through hidden middleware or cloud processing.

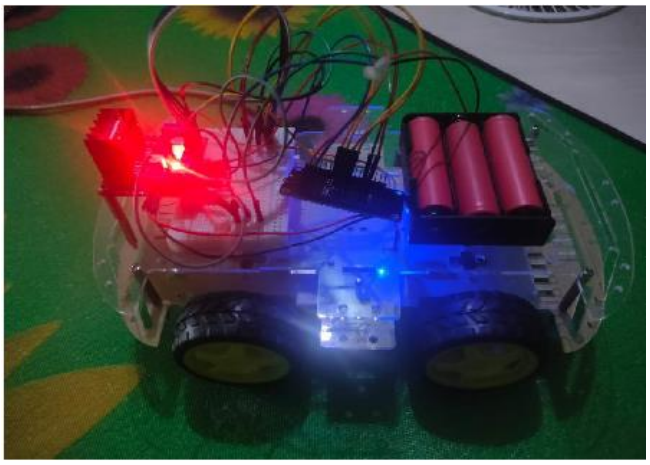

Figure 5. Implemented ESP8266-Based Robot Car Prototype

Figure 5 demonstrates that the architecture was realized as a compact, functional platform rather than evaluated only through simulation. The visible separation of the battery, controller, and driver is useful for instruction because

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

voltage supply, GPIO mapping, and motor outputs can be examined independently. The smartphone interface connected to the local endpoint and exposed large directional controls, a central stop command, and an independent speed slider. This arrangement is less sophisticated than mixed-reality or ROS 2 interfaces [17], but it is appropriate for the study objective: the operator can directly associate a touch event with an HTTP request and a physical motor response. The prototype also provides a modular baseline. Communication protocols, sensors, or encoders can be changed without replacing the chassis and motor driver, which aligns with recent low-cost and first-principle robotics platforms intended for laboratory experimentation [6], [8], [9]

3.2 Functional Command Performance

Table 4 presents the repeated functional-command results. The robot correctly executed 148 of 150 trials, yielding an overall success rate of 98.67%. The Wilson 95% confidence interval for the overall proportion was approximately 95.27–99.63%, indicating that the observed performance was consistently high within the test sequence while still acknowledging uncertainty from the finite sample. Forward, backward, and stop achieved 30 successful executions each. Left and right each produced 29 successful executions, so the two failures occurred during steering transitions rather than during straight movement or stopping. The data therefore support the correctness of the HTTP parser and command table, but they also show that identical network delivery does not guarantee identical electromechanical behavior across command types.

The slightly lower steering result is technically plausible because a turn depends on the simultaneous response of the two motor groups. Small differences in gearbox friction, wheel loading, starting torque, or battery voltage can delay one side even when the control token is received correctly. Straight movement applies matched states to both sides, whereas a differential turn produces a larger sensitivity to motor asymmetry. The 96.67% left- and right-turn proportions each have a wide Wilson interval of approximately 83.33–99.41% because only 30 trials were conducted per command. For this reason, the command-specific values should be interpreted as evidence of a pattern rather than as precise population estimates. A future encoder-based experiment could distinguish a delayed mechanical start from a lost or incorrectly parsed command.

The stop command achieved 100% success in the functional test, which is important because it provides the operator's immediate software-level safety action. Nevertheless, a successful stop button does not constitute a complete fail-safe system. The prototype currently depends on the command reaching the controller; loss of connectivity after a movement command could allow the last state to persist. A more robust firmware implementation should add a dead-man timer that disables ENA and ENB when no valid command or heartbeat is received within a defined interval. Sequence numbers and explicit acknowledgement status would further distinguish a delayed command from a duplicated or lost request. Compared with systems that include speech recognition or closed-loop balancing [15], [16], the present command vocabulary is simpler, so its high functional success should be interpreted as validation of deterministic button control rather than evidence of superiority over more complex tasks.

Table 4. Functional Command Test Results

Command	Trials	Successful executions	Success rate (%)
Forward	30	30	100.00
Backward	30	30	100.00
Left	30	29	96.67
Right	30	29	96.67
Stop	30	30	100.00
Overall	150	148	98.67

Table 4 confirms that all five required actions were available and that the total failure count was limited to two steering trials. Because the distance test at 5 m used a separate 100-command sequence and obtained 100% success, the two datasets are not contradictory; they represent different repetitions and emphasize that occasional electromechanical or communication variation can occur even under the same nominal distance. Reporting both tests provides a more transparent view than selecting only the better sequence.

3.3 Distance, Reliability, Latency, and Signal Strength

The distance-based results in Table 5 show a progressive reduction in communication quality. Command success remained 100% at 1 and 5 m, decreased slightly to 99% at 10 m and 97% at 15 m, and declined more clearly to 92% at 20 m. The corresponding Wilson 95% intervals were approximately 96.30–100%, 94.55–99.82%, 91.55–98.97%, and 85.00–95.89% for the 100%, 99%, 97%, and 92% observations, respectively. The intervals overlap at neighboring distances, but the monotonic pattern across the five test points indicates that the range boundary affects reliability. An exploratory correlation between distance and success across the five aggregate points was $r = -0.907$. Given the small aggregate sample, this value is used only to summarize the observed trend.

Latency changed more strongly than success at the intermediate distances. Mean command-response latency increased from 88 ± 12 ms at 1 m to 103 ± 15 ms at 5 m, 126 ± 18 ms at 10 m, 169 ± 27 ms at 15 m, and 248 ± 49 ms at 20 m. Thus, the link became slower before it became frequently unsuccessful. The standard deviation increased by more than four times between 1 and 20 m, showing that predictability deteriorated together with the mean. For manual steering,

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

this variation is operationally important: an operator can compensate for a stable delay more easily than for a response that changes from command to command. Across the five distance points, distance and mean latency were strongly positively associated (r = 0.958), supporting the interpretation that the edge of the tested range imposed a growing response penalty.

RSSI declined from −38 dBm at 1 m to −49, −59, −68, and −77 dBm as distance increased. The aggregate relationship between RSSI and latency was strongly negative (r = −0.935): more negative signal levels were associated with longer response times. RSSI and success showed a positive trend (r = 0.875), although the small number of aggregate points limits formal inference. These observations are consistent with indoor Wi-Fi research showing that signal fingerprints vary across positions [6]. They also explain why distance alone should not be treated as a universal specification. Antenna orientation, nearby networks, walls, reflective surfaces, and access-point characteristics can alter RSSI at the same geometric distance. The reported measurements therefore define the tested laboratory profile rather than a guaranteed range for every ESP8266 installation.

The combined evidence supports 15 m as the practical operating boundary for the present configuration. At that point, success remained 97%, mean latency was 169 ms, and RSSI was −68 dBm. At 20 m, success fell to 92%, mean latency approached a quarter of a second, and the 49 ms standard deviation indicated substantially less predictable response. A 92% command success rate may be acceptable for a supervised demonstration, but it is inadequate for safety-critical teleoperation because a lost steering or stop command can have a disproportionate consequence. The result therefore justifies protocol-level improvements such as a persistent WebSocket or MQTT connection, acknowledgement visualization, retransmission rules, command sequence numbers, and a watchdog. The current HTTP approach remains valuable as a transparent baseline against which those alternatives can be evaluated.

Table 5. Distance-Based Communication Results

Distance (m)	Commands	Successful	Success (%)	Latency, mean ± SD (ms)	RSSI (dBm)
1	100	100	100	88 ± 12	−38
5	100	100	100	103 ± 15	−49
10	100	99	99	126 ± 18	−59
15	100	97	97	169 ± 27	−68
20	100	92	92	248 ± 49	−77

Table 5 integrates the four indicators required to interpret communication performance. Success rate alone would suggest that the robot remained usable at 20 m, whereas latency and variation show that operation was already less responsive and less predictable. Figure 6 visualizes the reliability trend, allowing the sharper decline between 15 and 20 m to be distinguished from the relatively stable performance at shorter distances.

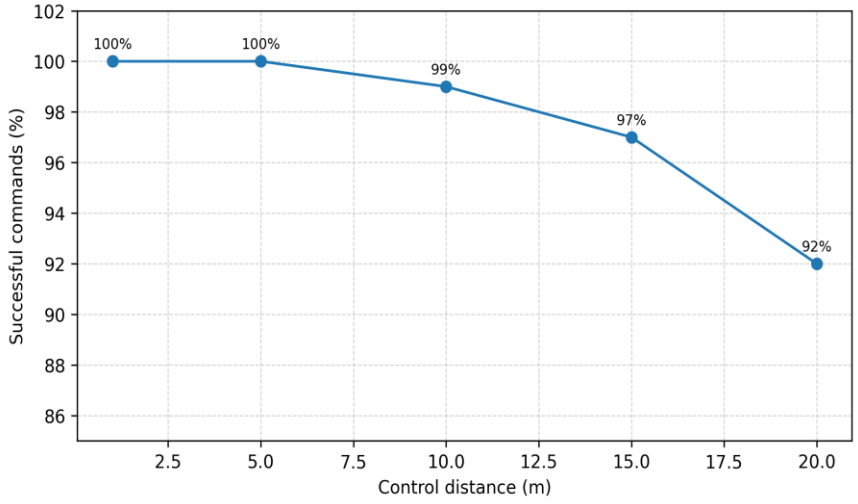

Figure 6. Command Success Rate at Different Control Distances

Figure 6 shows that reliability was essentially unchanged through 10 m and remained above 95% at 15 m. The final test point produced the largest single-step reduction, from 97% to 92%. This pattern supports an operating recommendation based on a margin from the measured boundary rather than on the maximum distance at which any command was received. The latency profile is presented separately in Figure 7 because response time and reliability convey different aspects of teleoperation quality. Error bars represent the standard deviation reported for each distance.

Figure 7 demonstrates a nonlinear increase in both mean latency and dispersion. The rise from 15 to 20 m was 79 ms, larger than any earlier interval, while the standard deviation increased from 27 to 49 ms. The endpoint acknowledgement therefore became not only slower but also more variable near the tested range limit, reinforcing the recommendation to keep routine laboratory operation within 15 m.

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 4, June 2026 | Hal 1655-1665
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i4.1091

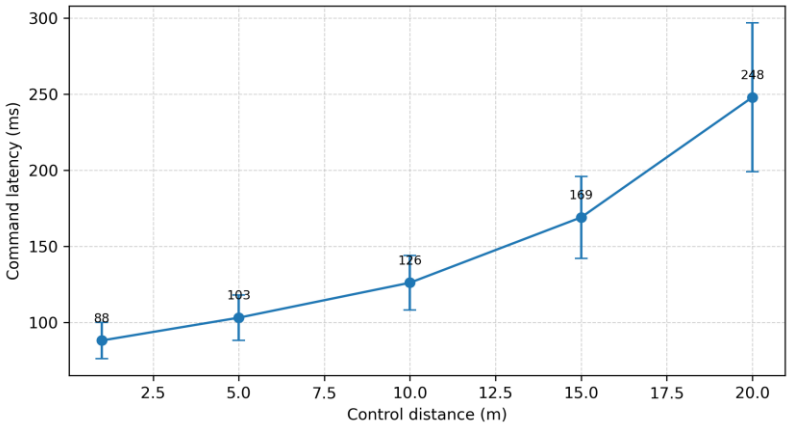

Figure 7. Mean Command-Response Latency and Variation by Distance

3.4 Interaction of Communication, Control, and Hardware

The observed performance results from the interaction of three subsystems. The communication layer determines when the request reaches the server and when the acknowledgement returns. The control layer parses the token and sets the GPIO and PWM outputs. The electromechanical layer determines when wheel torque becomes sufficient to move the chassis. The measured latency represents the first two layers but not the full time to physical displacement. This distinction helps explain why a command can be acknowledged while a turn appears delayed. The L298N introduces voltage loss, low-cost DC motors have unequal dead zones, and static friction varies with wheel loading. At low PWM values, one motor group may start before the other, producing curvature or a delayed differential turn. Instrumenting the command-receipt timestamp and adding wheel encoders would allow network, firmware, and actuation latency to be separated in future work.

Power behavior is another potential source of variability. Starting four motors simultaneously produces a transient current demand that can reduce battery voltage. If the logic supply and motor supply are not adequately regulated or decoupled, a voltage drop can affect torque and, in severe cases, controller stability. The prototype used a common ground as required by the driver interface, but future revisions should add a dedicated regulated logic supply, bulk capacitance near the motor driver, and battery-voltage logging. These changes would improve repeatability across charge states. The present study does not quantify current consumption, speed, endurance, straight-line error, or turning angle, so the claims are appropriately restricted to command execution and communication response.

Despite these limitations, the modular architecture offers a useful experimental progression. Students can first verify deterministic HTTP control, then compare HTTP with MQTT or WebSocket, add an ultrasonic sensor, instrument wheel speed, or implement autonomous navigation without discarding the base platform. Recent research emphasizes that sensing, path planning, and learning-based control are central to autonomous mobile robots [18], [19], [20], [21], but those functions depend on a stable actuation and communication foundation. The current system therefore occupies the physical, electrical, embedded, and basic control levels of a broader robotics curriculum [6]. This limited but explicit scope is a strength for introductory work because it prevents higher-level algorithms from masking wiring, power, or communication faults.

3.5 Comparison with Related Connected-Robot Studies

The performance of the proposed system was compared with selected connected mobile-robot studies representing Wi-Fi, Bluetooth, LoRa, and middleware-based control architectures. The comparison was intended to clarify differences in communication range, latency, interface complexity, and application scope rather than to rank the systems directly, because each platform was developed for a different operational objective.

The NodeMCU-based robot car reported by Siddesh et al. [12] is the closest comparison in terms of hardware architecture. Both studies employ an ESP8266-class controller, a smartphone interface, and an H-bridge motor driver for wireless motion control. However, the earlier work primarily demonstrated prototype functionality and provided limited quantitative evidence regarding communication performance. The present study extends that implementation-oriented approach by conducting repeated functional and distance-based trials and by jointly analyzing command success, latency, latency variation, RSSI, and operating distance. Therefore, its contribution lies not only in constructing a Wi-Fi-controlled robot but also in defining a measurable communication envelope for laboratory teleoperation.

Short-range Bluetooth systems provide another relevant comparison. The Android-controlled ESP32 robot reported in [15] achieved command latencies of approximately 110–140 ms over distances of 1–10 m and more than 92% command accuracy. Under the same distance interval, the proposed ESP8266 platform produced mean latencies of 88–126 ms and maintained command success between 99% and 100%. These results indicate broadly comparable responsiveness, although the two systems differ in processor architecture, wireless protocol, and command modality. The self-balancing robot presented in [16] achieved lower latency, approximately 38–72 ms with a mean of about 55 ms. Nevertheless, that platform operated under a shorter-range Bluetooth configuration and incorporated inertial feedback

and closed-loop stabilization. Its latency values should therefore not be interpreted as a direct benchmark for the present HTTP-based teleoperation system.

A different trade-off is demonstrated by the LoRa agricultural teleoperation platform in [14], which achieved communication over distances of several kilometers but experienced increasing packet loss near the range boundary. This result reflects the typical compromise between coverage and responsiveness: LoRa is appropriate for long-range, low-data-rate operation, whereas Wi-Fi and Bluetooth are more suitable for short-range control requiring faster operator feedback. In the present study, the ESP8266–HTTP architecture remained most dependable within 15 m, where command success was 97–100% and mean latency remained between 88 and 169 ms. At 20 m, reliability decreased to 92% and mean latency increased to 248 ms, confirming that the system is better suited to indoor laboratory operation than to long-range field teleoperation.

More advanced interfaces also provide useful contextual comparisons. Mixed-reality and ROS 2 systems such as that reported in [17] support richer visualization, monitoring, and robot interaction, but they require more capable computing hardware, middleware configuration, and software integration. Similarly, low-code connected-robot frameworks [2] reduce implementation complexity through higher-level abstractions. MIT App Inventor offers a more accessible alternative for a single embedded endpoint because users can directly inspect the relationship among interface events, HTTP requests, GPIO states, and motor responses. This transparency is particularly valuable in introductory robotics and IoT laboratories, where understanding the complete control chain is often more important than providing advanced autonomy or visualization.

Overall, the proposed system does not outperform every related platform in range, latency, or interface sophistication. Its principal advantage is the combination of low hardware cost, accessible mobile development, deterministic HTTP command mapping, and quantitative communication evaluation. The study therefore provides a reproducible baseline for educational teleoperation and for future comparisons involving WebSocket, MQTT, encoder feedback, obstacle sensing, or autonomous navigation. This positioning is consistent with recent educational-robotics research emphasizing inspectable hardware, progressive system integration, and first-principle learning [2], [6], [7], [8], [10], [22], [23].

3.6 Limitations and Validity of the Findings

The findings should be interpreted within five limitations. First, the reported tests characterize one prototype, one mobile interface, and one indoor line-of-sight environment; different smartphones, ESP8266 modules, interference levels, or antenna orientations can change the results. Second, latency was measured to the endpoint acknowledgement and therefore excludes the exact onset of wheel motion. Third, the robot had no encoders or external tracking, so trajectory accuracy, turning angle, and speed consistency were not quantified. Fourth, the aggregate correlations used only five distance points and are descriptive indicators rather than general statistical models. Fifth, the platform is manually teleoperated and should not be described as autonomous. The educational relevance is supported by the transparent design and related literature, but no learner study was conducted, so no claim is made about measured learning gains.

Future work should archive trial-level timestamps, acknowledgement status, RSSI, battery voltage, and encoder data across multiple sessions. Obstacle and interference conditions should be tested separately from pure distance. A controlled comparison of HTTP, WebSocket, and MQTT on the same hardware would identify protocol overhead and response consistency, while a dead-man timer and hardware emergency stop would strengthen safety. Sensors and path-planning methods can then be added after the communication baseline has been validated. These extensions would preserve the low-cost character of the prototype while increasing its suitability for rigorous connected-robot research.

4. CONCLUSION

The HTTP command-mapping method successfully linked MIT App Inventor events to deterministic ESP8266 and L298N motor states on a four-wheel robot car. Across 150 functional trials, 148 commands were executed correctly, giving 98.67% overall success, while the stop command was successful in all 30 repetitions. The 500 distance trials showed that communication remained most dependable within 15 m: success was 97–100%, mean command-response latency was 88–169 ms, and RSSI ranged from −38 to −68 dBm. At 20 m, success declined to 92%, latency increased to 248 ± 49 ms, and RSSI reached −77 dBm, indicating reduced responsiveness and predictability. The study therefore establishes a measurable operating envelope for an accessible, low-cost Wi-Fi teleoperation platform and demonstrates that command reliability should be interpreted together with latency variation and signal strength. The architecture is suitable for laboratory instruction and as a baseline for protocol comparison, but it requires a watchdog, explicit acknowledgement feedback, improved power regulation, and physical-motion instrumentation before use in more demanding applications. These findings also establish clear priorities for the next design iteration: preserve the accessible low-code interface, strengthen loss-of-connection safety, separate network latency from actuation latency, and validate performance across repeated sessions and obstructed indoor conditions.

REFERENCES

[1] A. Biswas and H.-C. Wang, “Autonomous vehicles enabled by the integration of IoT, edge intelligence, 5G, and blockchain,” Sensors, vol. 23, no. 4, Art. no. 1963, Feb. 2023, doi: 10.3390/s23041963.

[2] R. Brouzos, K. Panayiotou, E. Tsardoulias, and A. Symeonidis, "A low-code approach for connected robots," Journal of Intelligent & Robotic Systems, vol. 108, Art. no. 28, Jun. 2023, doi: 10.1007/s10846-023-01861-y.

[3] A. Awouda, E. Traini, M. Asranov, and P. Chiabert, "Bloom's IoT taxonomy towards an effective Industry 4.0 education: Case study on open-source IoT laboratory," Education and Information Technologies, vol. 29, pp. 15043–15065, Jan. 2024, doi: 10.1007/s10639-024-12468-7.

[4] Á. Cservenák and J. Husár, "A multidisciplinary learning model using AGV and AMR for Industry 4.0/5.0 laboratory courses: A study," Applied Sciences, vol. 14, no. 17, Art. no. 7965, Sep. 2024, doi: 10.3390/app14177965.

[5] R. Barradas, J. A. Lencastre, S. P. Soares, and A. Valente, "Arduino-based mobile robotics for fostering computational thinking development: An empirical study with elementary school students using problem-based learning across Europe," Robotics, vol. 13, no. 11, Art. no. 159, Oct. 2024, doi: 10.3390/robotics13110159.

[6] M. Ryalat, N. Almtireen, G. Al-refai, H. Elmoaqet, and N. Rawashdeh, "Research and education in robotics: A comprehensive review, trends, challenges, and future directions," Journal of Sensor and Actuator Networks, vol. 14, no. 4, Art. no. 76, Jul. 2025, doi: 10.3390/jsan14040076.

[7] B. Van Scoy, P. Jamieson, and V. Chidurala, "On first-principle robot building in undergraduate robotics education in the Robotic System Levels model," Robotics, vol. 14, no. 6, Art. no. 70, May 2025, doi: 10.3390/robotics14060070.

[8] M. Pekarcikova, P. Trebuna, M. Kliment, J. Kronova, and M. Matiscsak, "Educational robotics for Industry 4.0 and 5.0 with Wlkata Mirobot in laboratory process modelling," Machines, vol. 13, no. 9, Art. no. 753, Aug. 2025, doi: 10.3390/machines13090753.

[9] A. Kunz Cechinel, C. E. Soares, S. G. Pfleger, L. L. G. A. De Oliveira, E. Américo de Andrade, C. Damo Bertoli, C. R. De Rolt, E. R. De Pieri, P. D. M. Plentz, and J. Röning, "Mobile Robot + IoT: Project of sustainable technology for sanitizing broiler poultry litter," Sensors, vol. 24, no. 10, Art. no. 3049, May 2024, doi: 10.3390/s24103049.

[10] C. Márquez-Sánchez, J. Sandoval-Gutiérrez, and D. L. Martínez-Vázquez, "Construction of an educational prototype of a differential wheeled mobile robot," Hardware, vol. 4, no. 1, Art. no. 2, Jan. 2026, doi: 10.3390/hardware4010002.

[11] K. Żyła, K. Chwaleba, and D. Choma, "Evaluating usability and accessibility of visual programming tools for novice programmers The case of App Inventor, Scratch, and StarLogo," Applied Sciences, vol. 14, no. 21, Art. no. 9887, Oct. 2024, doi: 10.3390/app14219887.

[12] P. V. Nandankar, V. Kansal, G. S. Rekha, S. Jain, H. Patil, and D. Vekariya, "Design and development of a miniature home automation system using IoT and Android application," in Proc. 2024 7th International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, Apr. 24–26, 2024, pp. 1824–1829, doi: 10.1109/ICICT60155.2024.10544782.

[13] G. K. Siddesh, R. K. Patel, S. Maitra, S. Bhattacharya, S. Moosa, and P. Pavan, "Robotic car using NodeMCU ESP8266 Wi-Fi module," in Proc. 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, Mar. 17–18, 2023, pp. 1439–1443, doi: 10.1109/ICACCS57279.2023.10113098.

[14] H. Abacı and A. Ç. Seçkin, "Mobile robot positioning with wireless fidelity fingerprinting and explainable artificial intelligence," Sensors, vol. 24, no. 24, Art. no. 7943, Dec. 2024, doi: 10.3390/s24247943.

[15] R. R. Shamshiri, E. Navas, V. Dworak, T. Schütte, C. Weltzien, and F. A. Auat Cheein, "Internet of robotic things with a local LoRa network for teleoperation of an agricultural mobile robot using a digital shadow," Discover Applied Sciences, vol. 6, Art. no. 414, Jul. 2024, doi: 10.1007/s42452-024-06106-7.

[16] S. Gupta, U. Mamodiya, and A. J. A. Al-Gburi, "Speech recognition-based wireless control system for mobile robotics: Design, implementation, and analysis," Automation, vol. 6, no. 3, Art. no. 25, Jun. 2025, doi: 10.3390/automation6030025.

[17] S. Gupta, K. Ray, and S. Kaiser, "Self-balancing mobile robot with Bluetooth control: Design, implementation, and performance analysis," Automation, vol. 6, no. 3, Art. no. 42, Sep. 2025, doi: 10.3390/automation6030042.

[18] D. Janecký, E. Kučera, O. Haffner, E. Výchlopeňová, and D. Rosinová, "Using mixed reality for control and monitoring of robot model based on Robot Operating System 2," Electronics, vol. 13, no. 17, Art. no. 3554, Sep. 2024, doi: 10.3390/electronics13173554.

[19] A. Loganathan and N. S. Ahmad, "A systematic review on recent advances in autonomous mobile robot navigation," Engineering Science and Technology, an International Journal, vol. 40, Art. no. 101343, Apr. 2023, doi: 10.1016/j.jestch.2023.101343.

[20] H. Qin, S. Shao, T. Wang, X. Yu, Y. Jiang, and Z. Cao, "Review of autonomous path planning algorithms for mobile robots," Drones, vol. 7, no. 3, Art. no. 211, Mar. 2023, doi: 10.3390/drones7030211.

[21] Y. Liu, S. Wang, Y. Xie, T. Xiong, and M. Wu, "A review of sensing technologies for indoor autonomous mobile robots," Sensors, vol. 24, no. 4, Art. no. 1222, Feb. 2024, doi: 10.3390/s24041222.

[22] M. Rybczak, N. Popowniak, and A. Lazarowska, "A survey of machine learning approaches for mobile robot control," Robotics, vol. 13, no. 1, Art. no. 12, Jan. 2024, doi: 10.3390/robotics13010012.

[23] S. Yilmaz, C. Akay, and F. Kaysi, "Design and implementation of a low-cost IoT-based robotic arm for product feeding and sorting in manufacturing lines," Electronics, vol. 14, no. 24, Art. no. 4801, Dec. 2025, doi: 10.3390/electronics14244801.