

IoT-Enabled Smart Parking System Using ESP32 for Automated Gate Control and Slot Availability Monitoring

Nurul Alhikma Ramadani^{1*}, Aura Arung Karabiy SSP², Muh Yusuf³,
Zulkifli⁴

^{1,2,3,4}Electrical Engineering Education Study Program, Department of Electrical Engineering,
Universitas Negeri Makassar, Makassar, Indonesia

*Corresponding author : nurulalhikma09@gmail.com

ABSTRACT

Reliable parking information depends on the consistency between vehicle detection, gate actuation, and occupancy updates. This article presents an ESP32-based miniature parking system and examines the control logic documented in its project report. The prototype integrates two ultrasonic sensors, two servo-operated barriers, a 16 × 2 I2C display, and red and green indicators. Its firmware defines a five-vehicle capacity, a detection threshold below 5 cm, three distance samples per sensor, and a 3 s gate-open command. The source documentation reports successful vehicle detection, barrier operation, and local count display, but provides no repeated-trial dataset or timing measurements. Static analysis establishes that the counter is bounded by capacity checks, while exposing limitations not addressed by the functional demonstration. Occupancy changes before passage is confirmed, a stationary object can initiate repeated transactions, and invalid ultrasonic samples can bias the averaged distance. The entry-first decision structure also postpones exit handling when both requests qualify. These findings distinguish a functioning educational demonstrator from a validated parking installation. Although the platform is suitable for future network integration, the supplied firmware contains no internet communication implementation. A proposed improvement path combines event-based passage confirmation, nonblocking gate control, sample-validity handling, and traceable evaluation. The contribution is a reproducible description of the prototype and a code-grounded assessment of the conditions required for reliable occupancy tracking.

Keywords: smart parking; ESP32; ultrasonic sensing; occupancy tracking; embedded control

1. INTRODUCTION

A parking controller must do more than raise a barrier when an object is nearby. It must connect the detected event to a completed vehicle movement and update the available-space count accordingly. When these operations diverge, an apparently responsive interface can present an incorrect state: an aborted approach may consume a space, or a stationary object may be interpreted as several arrivals. For a small embedded system, the central engineering problem is therefore the consistency of detection, action, and accounting.

The present study examines the miniature smart-parking project documented by mikrokontroller project. The report provides a physical demonstrator, a wiring illustration, and a firmware listing, creating an opportunity to examine how its stated operation relates to the actual decision logic. This perspective retains the educational value of the project while defining precisely which conclusions the available evidence can support.

1.1 Related work and the role of local sensing

Smart-parking research covers vehicle detection, availability information, communication, and higher-level services. Lin et al. [1] survey smart-parking solutions, while Fahim et al. [2] compare systems across sensing, networking, and technological approaches. These perspectives show why the sensor node should be assessed within an entire information chain. A detection output is only one stage; useful parking information additionally requires a defensible interpretation of the observation and a consistent representation of occupancy.

Biyik et al. [3] review parking architectures and their technical development, emphasizing the relationship between sensing and system-level design. Channamallu et al. [4] further examine the

digitalization of parking services, including detection technologies, communication, and prediction. For the present work, these studies motivate a clear separation between local automation and connected services. Selecting a microcontroller with wireless capability does not, by itself, establish an implemented Internet of Things application.

At the service layer, Geng and Cassandras [5] address parking allocation and reservations through an optimization framework. Their work illustrates that allocation operates on a representation of resource availability. The present prototype does not perform optimization or reservations; it addresses an earlier local-control problem. Its occupancy variable must remain credible before that value can usefully support guidance, reservation, or a remote dashboard. This is a conceptual relationship, not a claim that the prototype implements or evaluates their algorithm.

1.2 Engineering gap and research questions

Within the supplied report, the central gap is between proximity-based triggering and passage-based counting. Two ultrasonic sensors observe the entrance and exit, but the code updates occupancy as soon as a threshold condition qualifies. A barrier then remains open for a fixed delay. The listing does not confirm that a vehicle crossed the gate or that the detection zone cleared before another event can be counted. These omissions are directly relevant to the reliability of a count-driven parking display.

Three questions guide the analysis. First, how do the sensing, servo, and display functions interact in the documented implementation? Second, which guarantees follow from the control conditions, and which parking behaviors remain unverified? Third, what changes and evaluation evidence would be required to progress from a classroom demonstration to more reliable occupancy tracking? These questions are answerable from the available design and listing without inventing a measurement campaign.

1.3 Objective and contribution

The objective is to present the prototype as an English-language engineering article and critically examine its documented control behavior. The contribution comprises a traceable hardware and parameter description, explicit occupancy-transition rules, identification of code-level failure cases, and a proposed validation plan. The work does not introduce a new sensing principle or claim superiority over published parking systems. Its value lies in connecting observable prototype functions to the implementation details that determine counting reliability.

The analysis is deliberately bounded. The project report supplies no vehicle-level ground truth, repeated detection trials, network logs, power measurements, or full-scale deployment. Accordingly, this article reports qualitative source observations and analytical consequences of the firmware separately. Improvements to congestion, fuel use, labor requirements, or detection accuracy remain potential outcomes requiring controlled evaluation, rather than demonstrated results of this prototype.

2. RESEARCH METHODOLOGY

2.1 Prototype evidence and analytical method

The project uses a design-and-build approach comprising component selection, circuit assembly, programming, and demonstration. This article analyzes the report, its photographs, and its firmware. Static inspection follows execution order, variable updates, sensor filtering, and actuator commands. It does not constitute compilation, a new simulator run, or a hardware experiment. Quantities derived from the code are identified as configured values or analytical examples.

The PDF repeats the firmware listing, and the later copy contains a missing semicolon after a `digitalWrite` statement. The first complete listing is used as the behavioral reference. The duplicated text should not be concatenated into a single program because repeated declarations and functions would prevent an ordinary build. A final reproducible submission should preserve one clean source file and record the board package and library versions.

2.2 Hardware interfaces

Table 1. Connections specified by the supplied firmware.

Subsystem	Connection or setting	Role
Entry ultrasonic sensor	TRIG 12; ECHO 13	Approach detection
Exit ultrasonic sensor	TRIG 14; ECHO 23	Exit detection
Entry and exit servos	GPIO 18 and GPIO 19	Barrier commands
Red and green LEDs	GPIO 25 and GPIO 27	Transaction indication
LCD	I2C address 0x27; 16 × 2	Count and status display
Controller	ESP32 DevKit / Dev Module	Local sequential control

The Wire interface provides I2C communication with the display. Espressif documents board-dependent default SDA and SCL assignments [8]; the supplied code does not explicitly set these pins. They should therefore be recorded with the selected board definition. The ESP32Servo library supplies servo attachment and position commands [7]. The project sets both servo periods to 50 Hz and attaches them with pulse limits of 500–2400 microseconds; these are software settings rather than measured mechanical endpoints.

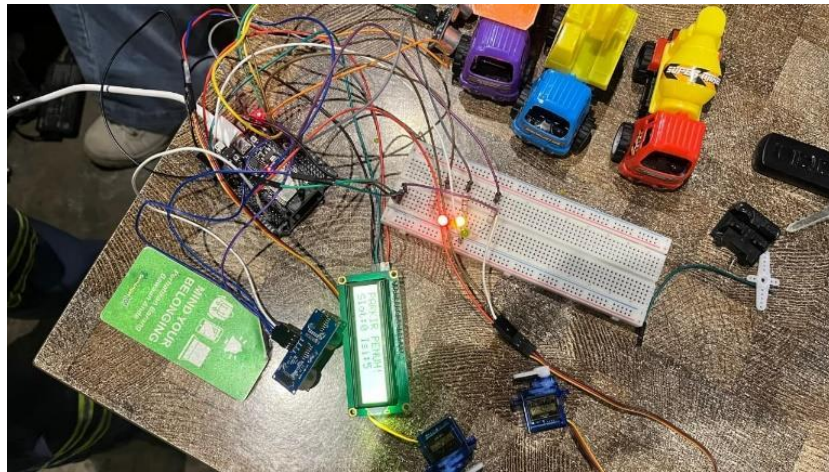


Figure 1. Breadboard assembly and miniature vehicles documented in the project report [1].

2.3 Distance acquisition and decision variables

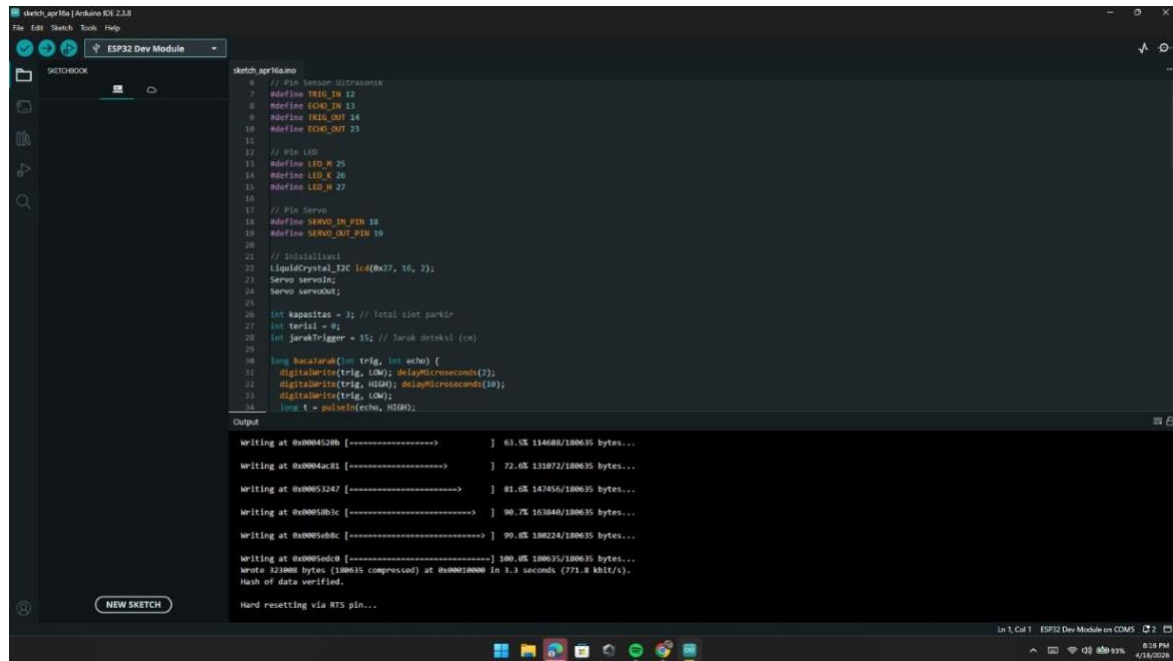
Each sensor reading contains three trigger-and-echo cycles. A 10 microsecond trigger pulse is followed by pulseIn with a 30,000 microsecond timeout. The measured duration t is converted using $d = 0.034t/2$, with distance in centimetres and duration in microseconds. This relation follows the code's assumed sound-speed factor and round-trip travel. Values are accumulated in an integer variable, averaged, and replaced with sentinel 999 if the resulting mean is nonpositive or greater than 400 cm.

The firmware defines capacity $C = 5$ and initial occupancy $n = 0$. The available count displayed on the LCD is $A = C - n$. An entry condition qualifies when the filtered entry distance is less than 5 cm and $n < C$; an exit qualifies when the exit distance is less than 5 cm and $n > 0$. Entry is evaluated first, and exit appears in an else-if branch. This sequence matters when both sensors simultaneously satisfy their distance conditions.

2.4 Gate transaction and local display

For an accepted entry, the program increments n , updates the LCD, sets the green indicator, closes the exit barrier, commands the entry servo to 90°, waits 3 s, and returns that servo to 0°. The exit sequence decrements n and reverses the roles of the barriers. The code sets the red indicator after each transaction. A full-capacity message is written when n reaches five, and a 50 ms loop delay follows normal processing.

The status flag is set during a transaction and cleared afterwards. In the supplied single-threaded loop, blocking execution already serializes these operations. The flag does not create background monitoring, confirm passage, or ensure fairness between gates. The chosen angles are command targets; physical opening direction, travel time, and clearance depend on assembly and are not quantified by the report.



```

1 // Pin Sensor Ultrasonik
2 #define TRIG_IN 12
3 #define ECHO_IN 13
4 #define TRIG_OUT 14
5 #define ECHO_OUT 23
6
7 // Pin LED
8 #define LED_R 25
9 #define LED_G 26
10 #define LED_B 27
11
12 // Pin Servo
13 #define SERVO_IN_PIN 18
14 #define SERVO_OUT_PIN 19
15
16 // Inisialisasi
17 LiquidCrystal_I2C lcd(0x27, 16, 2);
18 Servo servoR;
19 Servo servoG;
20
21 // Inisialisasi
22 int kapasitas = 3; // total slot parkir
23 int serial = 9;
24 int jarakTrigger = 15; // Jarak deteksi (cm)
25
26 long bucarah(int trig, int echo) {
27   digitalWrite(trig, LOW); delayMicroseconds(2);
28   digitalWrite(trig, HIGH); delayMicroseconds(10);
29   digitalWrite(trig, LOW);
30   long t = pulseIn(echo, HIGH);
31 }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Output

```

Writing at 0x0041200 [=====] 63.5% 134688/180635 bytes...
Writing at 0x0004ac81 [=====] 72.6% 133872/180635 bytes...
Writing at 0x00051247 [=====] 81.0% 147456/180635 bytes...
Writing at 0x00051b1c [=====] 90.7% 163840/180635 bytes...
Writing at 0x00051b1c [=====] 99.0% 180224/180635 bytes...
Writing at 0x00051b1c [=====] 100.0% 180635/180635 bytes...
Wrote 180635 bytes (180635 compressed) at 0x000018000 in 3.3 seconds (771.8 kbit/s).
Hash of data verified.
Hard resetting via RTS pin...

```

Figure 2. Firmware upload in Arduino IDE, reproduced from the report. This development snapshot shows different parameter values; the complete printed listing is the basis of the analysis.

3. RESULTS AND DISCUSSION

3.1 Documented prototype and evidence limits

The project report describes successful object detection, gate opening and closing, and updates on the local LCD [1]. Figure 3 documents the completed miniature. These observations support the existence of a demonstrator integrating sensing, actuation, and presentation. They do not establish a measured detection rate, count accuracy, or response-time distribution. No denominator of attempted vehicle movements is available from which to compute a success percentage.



Figure 3. Completed parking miniature reproduced from the project report.

Although the source uses the term IoT, the listed includes and program operations show local control only. The firmware contains no Wi-Fi initialization, remote endpoint, message publication, or cloud display implementation. Network capability is therefore treated as an extension opportunity. The LCD is the documented interface, and no dashboard image or remote performance result is introduced.

3.2 Occupancy bounds and transition behavior

Table 2. Analytical transitions implied by the firmware, not measured trials

Condition	Selected action	Next count
Entry below 5 cm; $n < 5$	Entry gate cycle	$n + 1$
No accepted entry; exit below 5 cm; $n > 0$	Exit gate cycle	$n - 1$
Entry only; $n = 5$	Entry blocked	5
Exit only; $n = 0$	Exit ignored	0
Both qualify; $0 < n < 5$	Entry takes precedence	$n + 1$

Starting from $n = 0$, the conditional increments and decrements keep the software variable within 0–5. This boundedness is useful but weaker than correctness: the number can remain inside the allowed range while disagreeing with actual occupancy. Table 2 therefore expresses a property of the update rules, not proof that every vehicle is counted exactly once.

3.3 Repeated detection and premature counting

Occupancy changes before the gate-open delay, so the counter records an accepted approach rather than confirmed passage. If an approaching vehicle retreats after detection, the increment is not reversed. If the same object remains within the entry threshold after the cycle finishes, the next loop can accept it again while capacity remains. This follows from level-based triggering: there is no requirement for a clear zone before a new transaction. It is an analytical failure case, not a failure frequency measured on the miniature.

A similar issue affects exits. A persistent exit detection can repeatedly reduce n until zero, while an undetected exit leaves the internal count too high. Because there is no per-space sensing, such errors accumulate in the single occupancy variable. Restarting the program resets n to zero regardless of the physical state. Recovery therefore requires a defined initialization or reconciliation procedure rather than assuming that a software reset implies an empty parking area.

3.4 Sample validity and threshold interpretation

Arduino documents that `pulseIn` returns zero when a complete pulse is not obtained within the timeout [6]. The supplied filter does not exclude those values before averaging. For example, three duration results of 600, 0, and 0 microseconds contribute approximately 10, 0, and 0 cm after integer conversion. The returned mean is 3 cm, which qualifies under the 5 cm trigger even though the one valid observation is about 10 cm. This constructed calculation reproduces the listed arithmetic; it is not an acquired sensor record.

Three timeouts produce a zero mean and are converted to sentinel 999, so that particular pattern does not trigger entry. The problem is mixed validity, where one or two missing readings can pull the mean downward. Averaging also does not guarantee stability: a minimum valid-sample count and explicit detection-state criteria are needed. A revised filter should classify missing pulses separately, then estimate distance only from valid observations.

3.5 Blocking delay and competing gate requests

The `delay(3000)` call prevents the main loop from making further sensor decisions during the commanded open interval. The nominal three-second dwell implies a theoretical ceiling below 20 serialized transactions per minute once acquisition and other overheads are included. This is a code-derived bound, not measured throughput, and does not include a realistic vehicle passage time. Background pulse generation may continue, but it does not make the decision loop concurrent.

When both requests qualify at an intermediate occupancy, entry is selected first. Under persistent entry requests, exits can be postponed; at full capacity, the entry guard fails and an eligible exit can proceed. A fairness policy is not defined. Furthermore, the fixed closing delay does not inspect barrier clearance before closing. The miniature therefore demonstrates timed gate movement without validating an obstruction-aware barrier controller.

3.6 Indicator semantics and proposed control improvements

The narrative associates green and red indicators with vacant and occupied slots, but the listing uses green during a gate transaction and red afterwards. There are no per-slot sensor variables or five independent slot indicators in the shown program. The LEDs should be described as transaction-state indicators unless the implementation is revised. The numeric LCD count is an aggregate estimate, not a map identifying which individual space is occupied.

A proposed event-based controller would distinguish approach, authorization, passage confirmation, clearance, and reset to idle. Counting should commit once a validated passage event occurs, with the detection zone required to clear before rearming. Independent gate state machines and elapsed-time checks can replace the blocking delay. These changes are recommendations and have not been implemented or tested as part of the supplied project.

Table 3. Proposed evaluation cases and records required for validation

Test case	Purpose	Evidence to retain
Single entry and exit	Check basic updates	Video ground truth and count log
Object remains near sensor	Check one event per passage	Detection and rearm timestamps
Vehicle approaches then retreats	Check count commitment	Passage confirmation and count
Both gates requested	Check arbitration	Request order and service delay
Missing and mixed echoes	Check invalid-sample handling	Individual pulse durations
Reset with occupied spaces	Check recovery	Actual occupancy and restored state

For repeated trials, count error should compare the displayed value with independent ground truth after each completed movement. Event precision and recall should identify false counts and missed passages separately. Report sample size, movement conditions, and timing boundaries before summarizing performance. Tests should also check electrical interfaces, actuator supply behavior, and physical clearance, rather than relying only on a successful software branch.

3.7 Network extension and research positioning

A later IoT extension could publish timestamped occupancy and gate-state events through MQTT, whose publish-subscribe behavior is specified by OASIS [9]. The local controller should retain its operation when the network is unavailable, while the remote display should expose data age and connection status. Neither messaging reliability nor remote latency can be inferred from this local demonstration. Network integration must be evaluated as an additional subsystem.

Relative to the broader parking literature [3]–[5], the prototype addresses a narrow embedded-control layer. Its main insight is that a bounded counter and visible barrier movement are insufficient to establish reliable occupancy tracking. A defensible next study would test event confirmation and recovery against the existing threshold-and-delay logic using the same ground-truth movement sequences. Only that comparison could quantify improvement.

4. CONCLUSION AND RECOMMENDATIONS

The ESP32 prototype integrates two ultrasonic approach sensors, two servo gates, and a local display for a configured capacity of five vehicles. The source reports a working miniature, while static analysis identifies the conditions under which its count may diverge from physical occupancy. The key limitations are counting before passage confirmation, re-triggering by persistent objects, mixed-validity distance averaging, blocking gate dwell, and entry-first arbitration. These are properties or risks inferred from the listing, not measured error rates.

The next development stage should prioritize passage confirmation, explicit rearming, nonblocking control, and occupancy recovery. A repeated-trial protocol with independent ground truth is required before claiming accuracy or operational benefits. Internet connectivity and remote monitoring should be presented as future extensions until their implementation and measurements are available.

5. ACKNOWLEDGEMENTS

This article is based on the Microcontroller Systems project report, Electrical Engineering Education Study Program, Universitas Negeri Makassar.

CONFLICT OF INTEREST

A conflict-of-interest declaration was not supplied and requires author confirmation before submission.

REFERENCES

- [1] T. Lin, H. Rivano, and F. Le Mouél, "A Survey of Smart Parking Solutions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 12, pp. 3229–3253, 2017, doi: 10.1109/TITS.2017.2685143.
- [2] A. Fahim, M. Hasan, and M. A. Chowdhury, "Smart parking systems: comprehensive review based on various aspects," *Heliyon*, vol. 7, no. 5, e07050, 2021, doi: 10.1016/j.heliyon.2021.e07050.
- [3] C. Biyik et al., "Smart Parking Systems: Reviewing the Literature, Architecture and Ways Forward," *Smart Cities*, vol. 4, no. 2, pp. 623–642, 2021, doi: 10.3390/smartcities4020032.
- [4] S. S. Channamallu, S. Kermanshachi, J. M. Rosenberger, and A. Pamidimukkala, "Smart parking systems: A comprehensive review of digitalization of parking services," *Green Energy and Intelligent Transportation*, vol. 5, no. 1, 100293, 2026, doi: 10.1016/j.geits.2025.100293.
- [5] Y. Geng and C. G. Cassandras, "New 'Smart Parking' System Based on Resource Allocation and Reservations," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1129–1139, 2013, doi: 10.1109/TITS.2013.2252428.
- [6] Arduino, "pulseIn," Language Reference. <https://docs.arduino.cc/language-reference/en/functions/advanced-io/pulseIn/>. Accessed Sep. 14, 2026.
- [7] ESP32Servo maintainers, "ESP32Servo: Arduino-compatible servo library for the ESP32." <https://github.com/madhephaestus/ESP32Servo>. Accessed Sep. 14, 2026.
- [8] Espressif Systems, "I2C," Arduino-ESP32 documentation. <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/i2c.html>. Accessed Sep. 14, 2026.
- [9] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, Eds., MQTT Version 5.0, OASIS Standard, Mar. 7, 2019. <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.