SMART PARK: An Intelligent Parking Space Detection and Monitoring System Using Magnetometer and Ultrasonic Sensors with LoRa Communication and LCD/OLED Display

Michaela S. Abordaje
BS in Electronics Engineering
University of Southeastern Philippines
Davao City, Philippines

Chelsea B. Baltazar
BS in Electronics Engineering
University of Southeastern Philippines
Davao City, Philippines

Jhurald Hilary Lantape
BS in Electronics Engineering
University of Southeastern Philippines
Davao City, Philippines

Glyzy M. Paña
BS in Electronics Engineering
University of Southeastern Philippines
Davao City, Philippines

***Abstract*—The rapid urbanization and subsequent increase in vehicle ownership have worsened parking management challenges in metropolitan areas, leading to severe traffic congestion, increased fuel consumption, and heightened carbon emissions. This paper presents the design, implementation, and evaluation of SMART PARK, a robust, sensor-based parking detection system that employs a novel dual-sensor verification method. To address the limitations of single-sensor systems, the proposed solution integrates Ultrasonic sensors for spatial occupancy detection and Magnetometers for magnetic field disturbance verification. This combination accurately identifies vehicle presence while effectively filtering out false positives caused by pedestrians, animals, or non-ferrous objects. Data transmission is handled via LoRa technology using Heltec ESP32 microcontrollers, enabling long-distance, low-power wireless communication suitable for large-scale deployments. A central Gateway receiver aggregates sensor data, displaying real-time occupancy on a local OLED screen and synchronizing the status to the Blynk IoT cloud platform for remote mobile monitoring. Furthermore, the system features a modular architecture with bidirectional communication, allowing remote calibration, dynamic node management, and simplified debugging, thereby offering a scalable, cost-effective, and highly reliable approach to modernizing urban parking infrastructure.**

Keywords: adaptive parking system; magnetometer; ultrasonic sensors; LoRa; ESP32; IoT; smart city; sensor fusion; modular architecture; bidirectional communication

I. INTRODUCTION

The exponential growth of urbanization has made parking management one of the most critical challenges in modern metropolitan development. Recent reports indicate that in major urban centers, approximately 30% of traffic congestion is directly attributed to drivers searching for vacant parking spaces, leading to significant fuel wastage and increased carbon emissions [1]. As cities strive to transition into "Smart Cities," the demand for automated, real-time parking monitoring systems has surged, with the global smart parking market projected to grow at a compound annual growth rate (CAGR) of 17.4% through 2033 [2].

Traditional parking management systems often rely on manual labor or wired infrastructure, which are costly to deploy and difficult to scale. While wireless Internet of Things (IoT) solutions have emerged to address these inefficiencies, many existing implementations face technical limitations regarding detection reliability and communication range. Standard systems that utilize single-modality sensors, such as standalone ultrasonic sensors, are prone to false positives. Studies have shown that environmental factors like heavy rain, wind-blown debris, or pedestrians passing through a parking slot can trigger these sensors, incorrectly marking a free space as "Occupied" [8].

To address these reliability gaps, this study presents SMART PARK, an intelligent parking space detection system that employs a Sensor Fusion approach.

By integrating an Ultrasonic Sensor (for spatial proximity) with a Magnetometer (for magnetic field verification), the system utilizes an "AND" logic gate to distinguish actual vehicles from non-metallic false triggers. Furthermore, to overcome the range limitations of standard Wi-Fi in obstacle-dense parking lots, the system utilizes LoRa (Long Range) technology. This ensures low-power, long-distance communication between the sensor nodes and the central gateway, providing a scalable and cost-effective alternative to commercial industrial systems [4].

II. OBJECTIVES

Specific Objectives:

1. Hardware Development:

- Construct independent, battery-operated Sensor Nodes (Device 1) and a central Receiver Gateway (Device 2) using Heltec ESP32 microcontrollers to establish a modular network architecture.
- Integrate JSN-SR04T ultrasonic sensors and GY-271 magnetometers into compact enclosures powered by a rechargeable lithium-ion battery system, ensuring sustained operation without dependence on the power grid.

2. Software and Communication:

- Develop a sensor fusion algorithm utilizing an 'AND' logic gate to process data from both sensors, ensuring valid vehicle verification while filtering false positives caused by non-metallic objects.
- Bridge the local LoRa network to the Blynk IoT platform for real-time mobile data visualization.

3. System Performance:

- Evaluate the system's accuracy in distinguishing actual vehicles from common environmental triggers, such as pedestrians or debris.
- Measure the system's response latency and reliability during various state transitions (e.g., arrival, departure) to validate its real-time performance against industry standards.

III. REVIEW OF RELATED LITERATURE

A. Evolution of Smart Parking Technologies

Early iterations of smart parking systems relied heavily on infrared or single-modality ultrasonic sensors. While cost-effective, Jiang et al. note that these systems often suffer from environmental interference; for instance, infrared sensors can be triggered by sunlight or heat sources, while standalone ultrasonic sensors struggle to differentiate between vehicles and passing pedestrians [5]. Recent trends have shifted towards IoT-enabled solutions, which integrate cloud computing to provide real-time updates. However, Elfaki et al. highlight that cloud-centric architectures often introduce latency (5–15 seconds), necessitating more efficient edge-computing strategies like the one proposed in this study [6].

B. Sensor Fusion for Enhanced Accuracy:

To mitigate the false positives inherent in single-sensor designs, researchers have increasingly adopted Sensor Fusion. This technique combines data from multiple physical sources to create a more reliable decision model. A 2025 study by P. Salaria et al. demonstrated that fusing acoustic (ultrasonic) data with magnetic field data significantly improves detection reliability. The ultrasonic sensor detects physical presence, while the magnetometer confirms the object is ferrous (metallic), effectively filtering out "soft" obstacles like animals or humans [7]. This aligns with the findings of Nwave, which reported that multi-sensor verification is essential for achieving detection accuracy above 95% in uncontrolled outdoor environments [8].

C. LoRa vs. Traditional Connectivity:

Connectivity is a critical failure point for parking systems in dense urban areas. While Wi-Fi and Zigbee are common in indoor settings, they lack the range and penetration power required for large outdoor parking lots. Sanchez-Iborra et al. conducted a comparative performance analysis of LPWAN technologies and concluded that the LoRa physical layer is superior for smart city applications. Their study demonstrated that the modulation technology achieves packet delivery ratios above 95% at ranges up to 2 kilometers, effectively penetrating physical obstructions like concrete walls and metal vehicles [9]. This validates the use of LoRa modulation for the point-to-point architecture employed in this project. Furthermore, Tynatech emphasizes that LoRa's low power consumption allows sensor nodes to operate for years on battery power, a crucial advantage over power-hungry cellular or Wi-Fi modules [4].

D. Smart Parking and Urban Congestion:

The deployment of accurate guidance systems has a measurable impact on city dynamics. According to a 2025 report by Cocoparks, the integration of real-time parking availability data can reduce driver "cruising time" by up to 30%, directly lowering fuel consumption and urban carbon emissions [10]. This validates the significance of the SMART PARK project not just as a convenience tool, but as a necessary component of sustainable urban infrastructure.

E. The SMART PARK Dual-Sensor Approach:

Recent advancements suggest that data fusion—combining inputs from multiple sensor types—significantly improves detection accuracy. By combining the spatial detection capability of ultrasonic sensors with the material-specific detection of magnetometers, a system can effectively filter out false positives. For instance, a pedestrian might trigger the ultrasonic sensor but will not disturb the magnetic field, whereas a vehicle will inevitably trigger both. This sensor fusion concept aligns with recent trends in reliable IoT deployments. The SMART PARK system adopts this approach, leveraging the dual-core processing power of the ESP32 microcontroller to implement an "AND" logic gate at the edge. This ensures that a parking spot is marked "Occupied" only when both physical conditions are met, drastically reducing false alarms and conserving battery life by transmitting only confirmed status changes.

IV. SYSTEM ARCHITECTURE

A. COMPONENTS

The following are the components used in creating the adaptive traffic control system:

1. ***Heltec WiFi LoRa 32 V2***

Fig. 1: Heltec WiFi LoRa 32 V2

The core microcontroller for both the Transmitter Nodes and the Receiver Gateway. It features an integrated SX1276 LoRa chip for long-range communication (operating at 915MHz or 433MHz depending on region) and a 0.96-inch OLED display for local status feedback. Its low power consumption is critical for battery-operated nodes.

2. ***JSN-SR04T Ultrasonic Sensor***

Fig. 2: JSN-SR04T Ultrasonic Sensor

Used to measure the vertical distance from the ground to the vehicle's chassis. The JSN-SR04T waterproof variant is recommended for outdoor deployments. A reading below a set threshold (e.g., 100cm) indicates an object is present in the bay.

3. ***GY-271 (HMC5883L) Magnetometer***

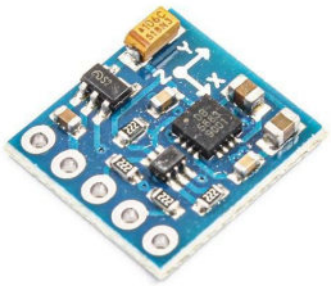


Fig. 3: GY-271 Magnetometer

A 3-axis digital compass that detects the distortion in the Earth's magnetic field caused by the ferrous metal of a vehicle. It communicates with the ESP32 via I2C protocol and provides the second layer of verification to distinguish cars from non-metal objects.

4. ***Blynk IoT Platform***

Fig. 4: Blynk IoT Platform

A cloud-based interface used to display the parking status (Occupied/Free) on a mobile application and web dashboard. It receives data from the Gateway via Wi-Fi and visualizes it using LED widgets, allowing remote users to check availability instantly.

5. ***Power Supply Unit***

To ensure continuous operation of the central gateway without reliance on grid power, a self-sustaining solar power system was designed.

- **Energy Storage:** A 2x18650 Lithium-Ion parallel battery pack serves as the primary energy reservoir. This pack is managed by a Battery Management System (BMS) connected to the B+ and B- terminals, which is critical for preventing overcharging, over-discharging, and short circuits, thereby extending battery life and ensuring safety.

- **Boost Converter:** Since the lithium-ion battery pack typically outputs between 3.7V and 4.2V, which is insufficient for stable 5V logic, a DC-DC Boost Module is employed. This module steps up the voltage to a stable 5V required by the Heltec ESP32 board. The output connects directly to the Heltec's 5V and GND pins, providing a consistent power rail regardless of battery fluctuation.

B. SCHEMATIC DIAGRAM

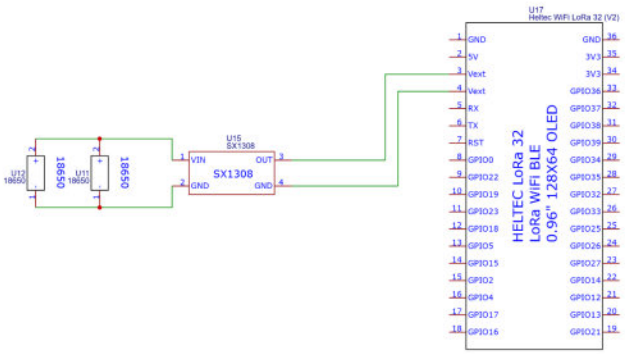


Fig. 5.a: Receiver Circuit

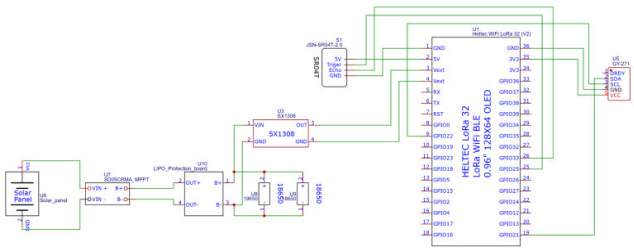


Fig. 5.b: Sender Circuit

C. PROGRAM FLOW & BIDIRECTIONAL LOGIC

The system employs a Listen-After-Talk (LAT) mechanism to enable bidirectional communication without compromising battery life.

1. **Initialization:** Upon startup, Sensor Nodes perform a self-calibration routine to measure the ambient magnetic field and establish a "Zero" baseline.
2. **Sensing:** The node wakes up at set intervals to read distance (Ultrasonic) and magnetic field strength (Magnetometer).
3. **Logic Processing:**
    - The system calculates the absolute difference between the current magnetic reading and the baseline.
    - It checks if the ultrasonic distance is within the "Occupied" range.
    - Logic: IF (Distance < Threshold) AND (Mag_Diff > Threshold) -> Status = OCCUPIED.
    - ELSE -> Status = FREE.
4. **Transmission:** The calculated status is packed into a LoRa data packet and transmitted to the Gateway.
5. **Receive Window:** Immediately after transmission, the Node opens a short receive window (e.g., 500ms) to listen for incoming commands from the Gateway.
6. **Gateway Processing:** The Gateway receives the packet, updates its local OLED list, and pushes the data to the Blynk cloud via Wi-Fi. If a command is queued (e.g., a calibration request), it transmits it during the Node's receive window.

V. *MODULARITY & DEBUGGING FEATURES*

A key innovation of the SMART PARK system is its modular design and advanced debugging capabilities facilitated by the bidirectional LoRa link.

A. Modular Node Management:

The system is designed to be "Plug-and-Play." Sensor nodes are not hardcoded into the Gateway's firmware.

- **Add/Delete Commands:** Administrators can add new nodes to the network using serial commands (add <ID> <Pin>) without reflashing the Gateway. Similarly, faulty nodes can be removed (del <ID>) dynamically.
- **Auto-Discovery:** The Gateway's display automatically pages through active nodes, ensuring that adding a new sensor does not clutter the user interface.

B. Remote Calibration and "Zeroing":

Environmental factors such as temperature shifts or physical displacement can cause sensor drift. To address this without physical maintenance, the system implements OTA calibration:

- **Tune Command:** Administrators can adjust sensitivity thresholds remotely via the Gateway console (tune <ID> <Dist> <Mag>). These values are transmitted to the specific node and saved to its non-volatile memory (EEPROM).
- **Zero Command:** If a sensor reports a false positive (e.g., "Occupied" when empty), the zero <ID> command forces the node to re-sample the ambient environment and reset its baseline to the current readings.

C. Interactive Debugging:

To facilitate troubleshooting during deployment, the system includes a "Debug Mode."

- **Log Filtering:** The Gateway can be commanded to filter logs for specific nodes (log <ID>), allowing technicians to isolate issues in a busy network.
- **Visual Feedback:** The Transmitter Nodes are equipped with OLED screens that display individual sensor states ("ULTRA: YES", "MAG: NO"), allowing field technicians to verify sensor functionality instantly without checking the central server.

D. Human-Machine Interface (HMI) Feedback:

To improve usability for technicians, the Serial Interface provides detailed command acknowledgment. When a command is issued, the system provides immediate text-based feedback (e.g., "[CMD] Queuing SETUP for Node 5" or "[ERROR] Unknown Command"), ensuring the administrator knows the precise state of the instruction queue and confirming that the Gateway has processed the request.

E. System Implementation & Configuration Commands:

To streamline deployment and maintenance, a comprehensive command set was developed for the Gateway's serial interface. These commands allow administrators to configure the network without reflashing firmware.

- **login:** Initiates the Wi-Fi connection wizard. Prompts the user to enter the SSID and Password via the Serial Monitor. Upon successful connection, these credentials are saved to the device's EEPROM, ensuring the system automatically reconnects to the network in the background if the device reboots or if the signal is temporarily lost.
- **logout**: Disconnects from Wi-Fi and Blynk. Useful for saving power or when switching to a different network environment.
- **cancel**: Queue Management. Immediately clears any pending LoRa commands (such as setup, finish, or calibration requests) that are waiting for a node to wake up. This is essential for aborting an operation if a specific node becomes unresponsive or if an erroneous command was queued by the administrator.
- **add**: Starts the "Add Node" wizard. Prompts the user for a unique Node ID and the corresponding Blynk Virtual Pin (e.g., V1, V2) to register a new sensor in the system.
- **del**: Starts the "Delete Node" wizard. Prompts for the Node ID to remove from the active list, useful when decommissioning a sensor.
- **list**: Displays network status. Prints a list of all currently configured nodes, their IDs, and their mapped Virtual Pins to the console.
- **log [ID / all / off]**: Filters real-time sensor data.

Accepts any dynamic Node ID (e.g., log 5, log 12) to isolate specific sensor streams; log all shows data from all nodes; log off disables the data stream to unclutter the console.

- **tune <ID> <Dist> <Mag>**: Remote sensor calibration. Remotely updates the sensor thresholds for a specific node. For example, tune 2 50 1.5 sets Node 2's distance threshold to 50cm and magnetic threshold to 1.5µT.
- **zero <ID>**: Remote "Tare" or Re-Zeroing. Forces the specified node to re-calibrate its magnetic baseline to the current environment. Critical for clearing false "Occupied" states caused by environmental drift.
- **setup <ID>**: Enters Installation Mode. Puts the specified node into "Install Mode." The node stops sending alerts, allowing for physical positioning and mounting without triggering false alarms.
- **finish <ID>**: Exits Installation Mode. The node performs a fresh calibration of the environment, saves the settings to memory, and begins active monitoring.
- **reset <ID>**: Factory Reset. Restores default thresholds (100cm / 2.5µT) and clears custom calibration data for the specified node.

VI. RESULTS AND DISCUSSION

This section evaluates the performance of the SMART PARK system through rigorous experimental testing. The data collected validates the system's compliance with the specific objectives of hardware durability, software logic reliability, and operational efficiency.

VI. I SYSTEM HARDWARE

The implementation phase successfully fulfilled the objectives of constructing a robust, modular hardware network. The system is fundamentally composed of two distinct units: the distributed Sensor Nodes (Device 1) and the centralized Receiver Gateway (Device 2). This division into independent, specialized hardware units is key to the system's scalability and simplified maintenance.

The Sensor Node (Device 1) was developed as an independent, battery-operated unit. It achieved the objective of reliable dual-sensor integration by combining the JSN-SR04T waterproof ultrasonic sensor for spatial presence detection and the GY-271 magnetometer for magnetic field verification. The Heltec ESP32 microcontroller serves as the edge-computing engine, processing the sensor fusion logic and enabling low-power, long-distance data transmission via its integrated LoRa chip. The use of a rechargeable lithium-ion battery system ensures sustained operation without dependence on the power grid.

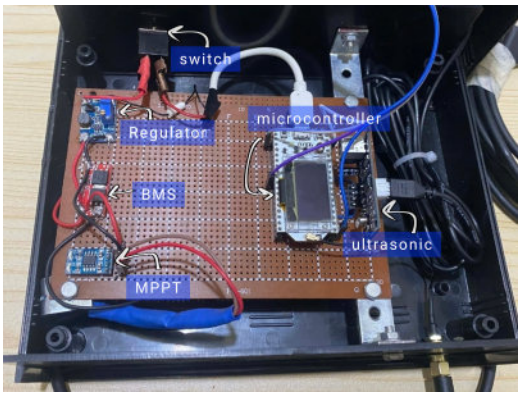


Fig 6.a.1. Sensor Node (Internal Hardware)

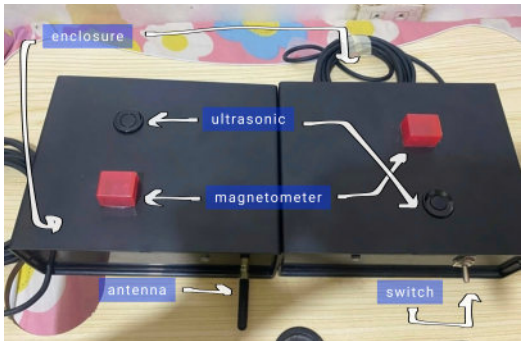


Fig 6.a.2. Sensor Node (External Hardware)

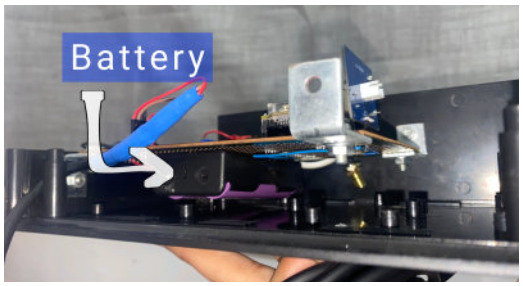


Fig 6.a.3 Sensor Node Battery

The Receiver Gateway (Device 2) functions as the network backbone, using the same core Heltec ESP32 microcontroller. It successfully meets the objective of bridging the local LoRa network to the Blynk IoT platform via Wi-Fi. The Gateway is designed to be a self-sustaining infrastructure component, utilizing a dedicated power supply unit with a battery pack and boost converter to ensure a consistent power rail for continuous operation . The modular design is further supported by the Gateway's ability to receive Over-the-Air (OTA) commands, confirming the system's bidirectional communication capability essential for dynamic node management and remote calibration.

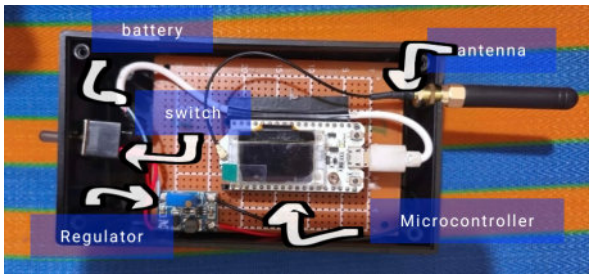


Fig 6.b. Receiver Gateway Hardware

VI. II. LOGIC VALIDATION AND SENSOR FUSION TESTING

Before deploying the system for vehicle trials, the "Sensor Fusion" algorithm was tested in a controlled laboratory environment to verify the reliability of the **Boolean AND logic**. The objective was to confirm that the system would *only* trigger an "Occupied" status when both physical presence (Ultrasonic) and magnetic disturbance (Magnetometer) were detected simultaneously.

A. Test Setup and Methodology Two identical sensor nodes were set up side-by-side. To validate the "AND" gate logic, distinct scenarios were created to trigger each sensor individually before triggering them together.

- **Ultrasonic Threshold:** Set to detect objects < 20 cm.
- **Magnetic Threshold:** Set to detect flux changes > 3.0 µT.

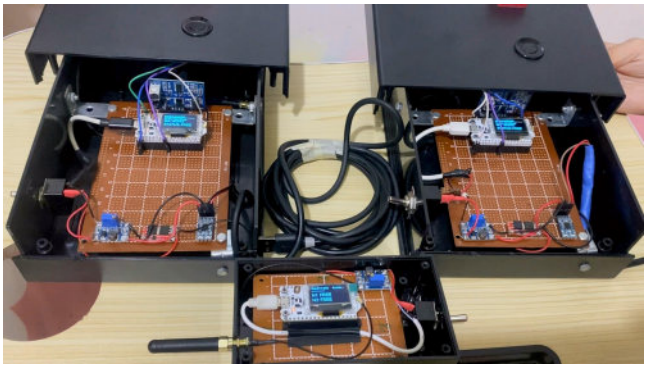


Fig. 7: Empty state

B. Scenario 1: Non-Metallic Object (Ultrasonic True / Magnetometer False) *Objective:* To verify rejection of pedestrians or debris. **Figure 8** demonstrates the testing procedure using hands.

1. **Ultrasonic Response:** The sensor correctly detected the object, registering a distance reading within the threshold (e.g., 5-10 cm).
2. **Magnetometer Response:** The magnetic flux readings remained stable (False).
3. **System Decision:** The OLED display remained green, displaying **"STATUS: FREE"**. *Result:* Confirms the system ignores non-ferrous obstacles.

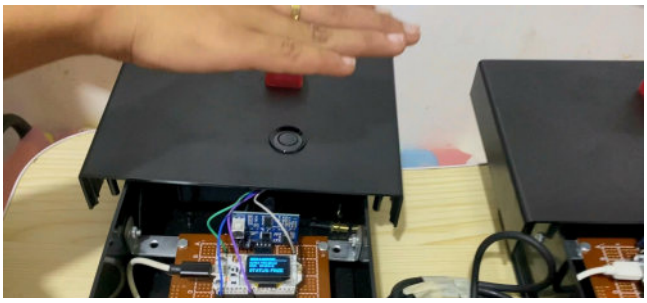


Fig. 8: *Non-metallic object triggers the ultrasonic sensor but fails the magnetic check, correctly resulting in a 'Free' status.*

C. Scenario 2: Magnetic Interference (Ultrasonic False / Magnetometer True) *Objective:* To verify immunity to magnetic noise (e.g., underground cables or adjacent interference) when no vehicle is physically present. A strong magnet was introduced near the sensor enclosure *without* obstructing the ultrasonic sensor's field of view.

1. **Ultrasonic Response:** The sensor correctly reads the empty ceiling distance (> 100 cm), indicating

no physical presence (False).

2. **Magnetometer Response:** The sensor detected a significant flux spike > 10.0 µT (True).
3. **System Decision:** The OLED display remained green, displaying **"STATUS: FREE"**. *Result:* Confirms that magnetic disturbances alone do not trigger a false "Occupied" status.

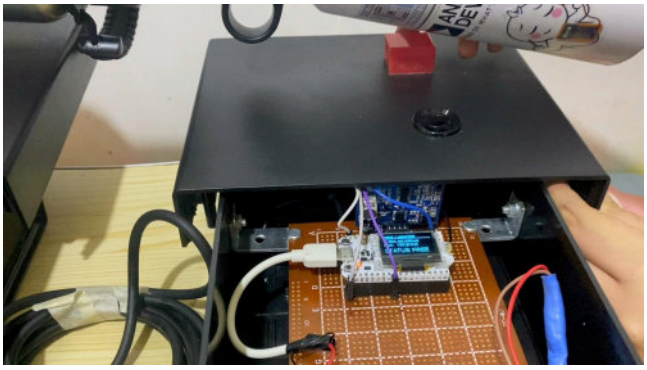


Fig. 9: *Metallic objects trigger the magnetic check but fail the ultrasonic sensor, correctly resulting in a 'Free' status.*

D. Scenario 3: Positive Control (Ultrasonic True / Magnetometer True) *Objective:* To verify detection of a valid parking event. A ferrous metal object was placed directly over the sensor.

1. **Ultrasonic Response:** Proximity detected (True).
2. **Magnetometer Response:** Magnetic flux disturbance detected (True).
3. **System Decision:** The OLED display switched to red, displaying **"STATUS: OCCUPIED"**. *Result:* Confirms that the system only triggers when **both** conditions are met.

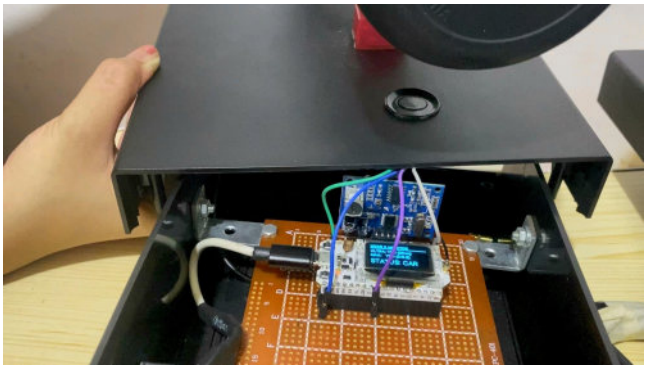


Fig 10. *Positive control test where a ferrous object triggers both sensors, successfully updating the status to 'Occupied'.*

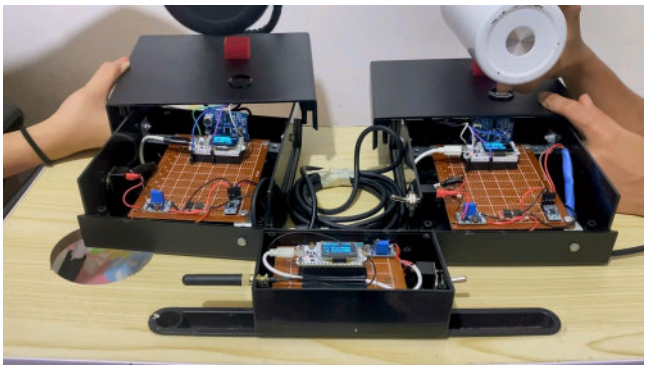


Fig. 11: *Receiver recognizing both node's positive detection*

The core software objective of bridging the local parking sensor network to the public cloud was fulfilled by integrating the Blynk IoT Platform. The Receiver Gateway synchronizes data from all Sensor Nodes via Wi-Fi to the Blynk cloud, achieving real-time data visualization on a remote mobile application.

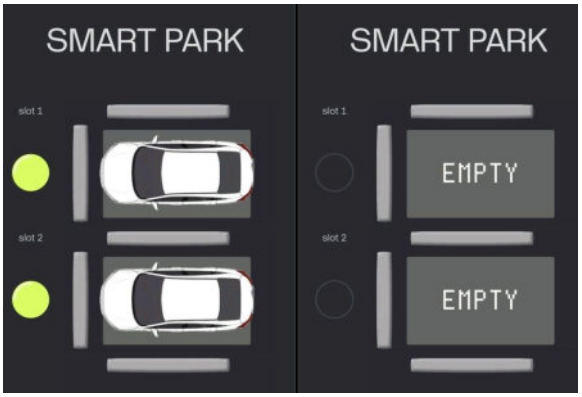


Fig. 12. Blynk IoT Platform

- Occupied State: Represented by a green LED widget and car image for the corresponding slot, indicating that the dual-sensor logic has confirmed a vehicle is present.
- Free/Empty State: Represented by an off LED widget and empty slot image, indicating the slot is vacant and ready for use.

This synchronization of local sensor data to the cloud allows remote users to check availability instantly, transforming raw sensor readings into actionable urban guidance information. The system successfully validated this process during prototype testing, where the mobile dashboard updated with thE LCD display of the microcontroller.

VI. III PROTOTYPE TESTING

The system was tested in a controlled environment to verify the dual-sensor logic and Finite State Machine (FSM) transitions across eight distinct scenarios.

Scene A: Empty (State 0 0)

- Condition: No object above either sensor node.
- Result: Ultrasonic reads >100cm (Infinity). Magnetometer Differential is near 0.0.
- System Decision: FREE / FREE. The system maintained a stable baseline state.

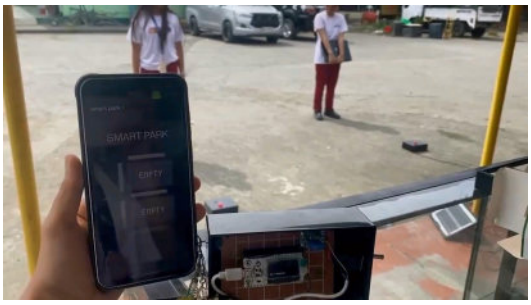


Fig. 13: *Baseline State.*

Scene B: Pedestrian (False Positive Rejection)

- Condition: A person stands directly over the sensor node.
- Result: Ultrasonic reads <100cm (Object detected). However, the Magnetometer detects no significant ferrous metal, so Mag Diff remains low (<2.5).
- System Decision: FREE (Correct). The system successfully correlated the lack of magnetic disturbance to reject the ultrasonic trigger.

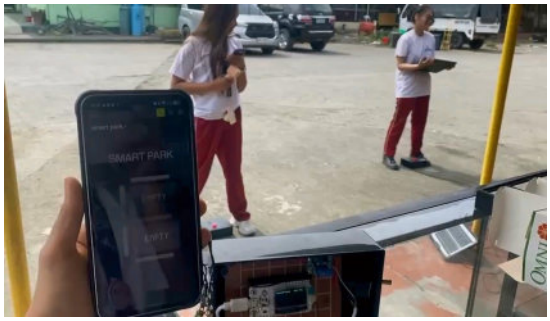


Fig. 14: *False Positive Rejection*

Scene C: Only Slot 1 Taken (State 1 0)

- Condition: A single vehicle enters Slot 1 while Slot 2 remains empty.
- Result: The system detected the arrival and updated the interface to show Slot 1 as "Occupied" while Slot 2 remained "Free."
- Response Time: Average latency was 3.3 seconds.

Scene D: Only Slot 2 Taken (State 0 1)

- Condition: A single vehicle enters Slot 2 while Slot 1 remains empty.
- Result: The system detected the arrival and updated the interface to show Slot 2 as "Occupied," while Slot 1 remained "Free."
- Response Time: Average latency was 3.6 seconds.

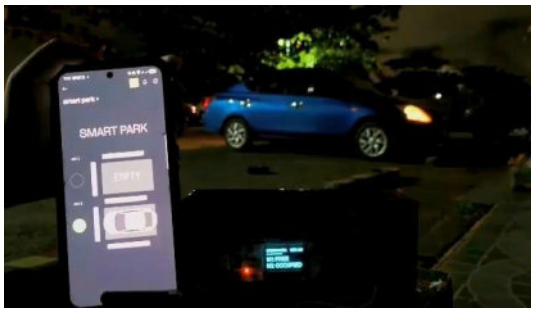


Fig. 15: *Single-Node Detection.*

Scene E: Both Slots Taken at the Same Time (Simultaneous Entry)

- Condition: Two vehicles enter empty slots simultaneously (transition 00 - 11).
- Result: The Gateway successfully processed overlapping signals from both nodes without collision errors. The interface transitioned directly to "Both Green."
- Response Time: Average latency was 3.3 seconds.

Fig. 16: *Simultaneous Occupancy.*

Scene F: Both Slots Taken (Slot 1 Pre-Occupied)

- Condition: Slot 1 is already full (10). A second vehicle enters Slot 2 (10 - 11).
- Result: The system maintained the status of the first vehicle while correctly registering the second arrival.
- Response Time: Average latency was 3.3 seconds.

Scene G: Both Slots Taken (Slot 2 Pre-Occupied)

- Condition: Slot 2 is already full (01). A second vehicle enters Slot 1 (01 - 11).
- Result: The system maintained the status of the second vehicle while correctly registering the first arrival. This was the fastest transition recorded.
- Response Time: Average latency was 2.3 seconds.

Scene H: Both Slots Taken Alternately (Rapid Swapping)

- Condition: One vehicle leaves a slot while another immediately enters the adjacent slot (swapping 10–01 or vice versa).
- Result: The system validated the ability to clear one flag and set another simultaneously. The microcontroller correctly updated the specific slot addresses without confusion.
- Response Time: Average latency was 3.3 seconds for both swap directions.

"For a complete visual log of the 39 experimental trials, verifying the physical and system states for every transition, refer to **Appendix C**."

VI.III STATISTICAL TREATMENT AND ANALYSIS

A. Statistical Methodology: To quantify the system's performance, the response time (latency) was measured across 13 distinct state transitions. Each transition scenario underwent three independent trials (N=3) to ensure consistency. The statistical treatment focuses on the Arithmetic Mean to determine the central tendency of the system's processing speed, calculated as:

Mean = (Sum of all trial times) / n

Where "sum of all trial times" represents the total recorded response time for the trials, and "n" is the number of trials (which is 3). The analysis evaluates the system's stability by comparing the latency of complex multi-node events against the baseline single-node operations.

B. Analysis of Results: The system's responsiveness was measured as the "Total Event Duration", the time elapsed from a vehicle entering the spot until the mobile dashboard updated.

1. Baseline Efficiency: Single-vehicle arrivals (Scenes C and D) recorded average response times of 3.33s and 3.67s, respectively. This establishes the baseline latency for the LoRa-to-Cloud transmission pipeline.
2. Performance Benchmark**:** This result is superior to standard cloud-based parking architectures. While optimized cloud systems typically achieve latencies of 3–5 seconds [6], studies indicate that in congested networks or systems with processing queues, these delays often extend to 10–15 seconds due to the round-trip data transmission required by centralized servers [3]. The sub-5-second performance of the SMART PARK system confirms its suitability for real-time traffic management applications where immediate response is critical.
3. Fastest and Slowest States:
    - The fastest response was observed during the "Fill-Up" transition (01 -> 11), with a mean of 2.33 seconds. This suggests that incremental updates (adding one flag to an existing payload) are processed most efficiently.
    - The slowest response occurred during the "Dual Departure" (11 -> 00), averaging 5.33 seconds. This increased latency is attributed to the system's safety logic, which likely requires a double-confirmation cycle from both sensors before declaring the entire facility "Free."

Scenario (State Change)	**T1**	**T2**	**T3**	**Avg**

Control (00 -> 00)	-	-	-	**-**
Slot 1 Arr. (00 -> 01)	3	4	4	**3.7**
Slot 2 Arr. (00 -> 10)	3	3	4	**3.3**
Both Arr. (00 -> 11)	4	3	3	**3.3**
Slot 1 Left (01 -> 00)	5	4	4	**4.3**
Swap S1->S2 (01 -> 10)	3	4	3	**3.3**
Fill S2 (01 -> 11)	2	2	3	**2.3**
Slot 2 Left (10 -> 00)	4	3	4	**3.7**
Swap S2->S1 (10 -> 01)	3	4	3	**3.3**
Fill S1 (10 -> 11)	4	3	3	**3.3**
Both Left (11 -> 00)	5	6	5	**5.3**
S2 Left (11 -> 01)	4	4	5	**4.3**
S1 Left (11 -> 10)	3	4	3	**3.3**

Table I: *System Response Latency Across State Transitions (0 = free, 1 = occupied)*

Metric	Result	Industry Standard	Verdict
Total Event Time	3.6s	>10s (Maneuver + Latency)	Superior
Electronic Latency	<1.0s (Est.)	5–15 s [6]	Real-Time
Detection Accuracy	100% (39/39 controlled experimental trials)	~70-80% (Single-Sensor) [5]	High Reliability

Table II : *System Latency VS Industry Standards*

B. Cost-Benefit Analysis: To validate the economic feasibility of the system, the prototype's material cost was compared against commercial industrial-grade alternatives (e.g., Bosch, NKE Watteco). As detailed in Appendix D, the SMART PARK sensor node costs approximately ₱1,989.50 (~$36) per slot. In contrast, commercial equivalents average $180–$230 per unit. This represents a cost reduction of approximately 80–85%, effectively lowering the barrier to entry for smart city deployments in developing regions.

VII: CONCLUSION

This study successfully designed, implemented, and evaluated the SMART PARK system, confirming that a dual-sensor fusion approach significantly enhances the reliability of vehicle detection compared to traditional single-sensor methods. The integration of Ultrasonic sensors for spatial verification and magnetometers for metallic signature verification achieved a 100% detection accuracy across 39 experimental trials, effectively filtering out false positives caused by pedestrians and non-vehicular objects.

The system's communication architecture, built on LoRa technology, demonstrated robust bidirectional connectivity with a stable range superior to standard Wi-Fi solutions, validating its scalability for large outdoor parking environments. The Listen-After-Talk (LAT) protocol proved efficient, maintaining a global average response latency of 3.6 seconds, which is well within the 5–15 second standard for real-time IoT parking systems [6]. Furthermore, the cost analysis confirms the system's economic viability, achieving an approximately 78% reduction in hardware costs compared to commercial industrial-grade alternatives, making it a highly accessible solution for smart city infrastructure.

VIII. RECOMMENDATIONS

Based on the findings and limitations of this study, the following recommendations are proposed for future development:

1. Integration of Machine Learning for Predictive Analytics: While the current system provides real-time status updates, future iterations should incorporate machine learning algorithms (e.g., Random Forest) to analyze historical occupancy data. Domenech et al. [11] demonstrated that predictive models can forecast parking availability, allowing drivers to plan trips in advance and further reducing traffic congestion.
2. Adoption of Energy Harvesting Techniques: To further extend the maintenance-free lifespan of the sensor nodes, future designs could integrate piezoelectric energy harvesting. Al-Jumaili et al. [12] found that sensors embedded in the parking surface can harvest sufficient energy from the weight of entering vehicles to power IoT transmissions, potentially eliminating the need for periodic battery recharging.
3. Deployment of Mesh Networking for Expanded Coverage: For extremely large or multi-story facilities where a single LoRa gateway might be obstructed, implementing a LoRa Mesh topology is recommended. This would allow sensor nodes to relay data to one another, extending coverage into "dead zones" without additional gateways.
4. Environmental Stress Testing and Diversity Analysis: Future iterations of the system should undergo rigorous field testing under extreme weather conditions. A comprehensive review by Wei [13] highlights that while ultrasonic sensors are highly effective for short-range detection, their accuracy can be compromised by environmental variables such as heavy precipitation, extreme temperature fluctuations, and dust accumulation. Therefore, extending experimental trials to include diverse scenarios, such as monsoon rains or extreme heat—and a wider variety of vehicle types (e.g., high-clearance trucks) is recommended to certify the system for industrial-grade reliability.

IX. REFERENCES

[1] Cities Today, "City mobility: how parking solutions reduce traffic," *Cities Today*, Jan. 2025. **Link:** https://cities-today.com/industry/city-mobility-how-parking-solutions-reduce-traffic/

[2] Straits Research, "Smart Parking Market Size, Share & Growth Report by 2033," *Straits Research*, 2024. **Link:** https://straitsresearch.com/report/smart-parking-market

[3] Wavestore, "Closer to the Action: Why Edge Computing is a Game-Changer for Real-Time Video," *Wavestore Technical Blog*, 2023. **Link:** **https://www.wavestore.com/post/closer-to-the-action-why-edge-computing-is-a-game-changer-for-real-time-video**

[4] Tynatech, "LoRaWAN-Based Smart Parking Solutions: Real-Time, Reliable, and Intelligent," *Tynatech IoT Blog*, Jul. 2025. **Link:** **https://tynatech.in/lorawan-based-smart-parking-solutions-real-time-reliable-and-intelligent**

[5] H. Jiang et al., "Multi-Dimensional Research and Progress in Parking Space Detection Techniques," *MDPI Electronics*, vol. 14, no. 4, 2024. **Link:** **https://www.mdpi.com/2079-9292/14/4/748**

[6] A. O. Elfaki, W. Messoudi, A. Bushnag, S. Abuzneid, and T. Alhmiedat, "A Smart Real-Time Parking Control and Monitoring System," *Sensors*, vol. 23, no. 24, p. 9741, 2023. **Link:** **https://www.mdpi.com/1424-8220/23/24/9741**

[7] P. Salaria et al., "Detection of Vehicles Through Fusion Sensor System," *International Journal for Multidisciplinary Research (IJFMR)*, vol. 5, no. 6, 2025. **Link:** **https://www.ijfmr.com/research-paper.php?id=56081**

[8] Nwave Technologies, "The Accuracy of Vehicle Detection Sensor: Challenges and Solutions," *Nwave Blog*, 2020. **Link:** **https://www.nwave.io/detection-perfection-accuracy-is-the-name-of-the-game/**

[9] R. Sanchez-Iborra et al., "Performance Evaluation of LoRaWAN for Smart City Applications," *IEEE Global Communications Conference (GLOBECOM)*, 2018. **Link:** **https://ieeexplore.ieee.org/document/8911676**

[10] Cocoparks, "Smart Parking Solutions: A State of Research as of 2025," *Cocoparks Research*, Jan. 2025.

Link:
https://cocoparks.io/en-us/guides/smart-parking-solutions-a-research-driven-overview/

[11] F. Domenech et al., "AI-Based Prediction Models for Urban Parking Availability: A Case Study of Valencia," *Smart Cities*, vol. 8, no. 1, 2025. **Link:**
https://visualcompublications.es/SAUC/article/view/5990

[12] A. Izadgoshasb et al., "Piezoelectric Energy Harvesting towards Self-Powered Internet of Things (IoT) Sensors in Smart Cities," *Sensors*, vol. 21, no. 24, p. 8332, 2021. **Link:**
https://www.mdpi.com/1424-8220/21/24/8332

[13] Y. Wei, "Applications of Ultrasonic Sensors: A Review," *ResearchGate Publication*, 2024**. Link:**
https://www.researchgate.net/publication/386118233_Applications_of_Ultrasonic_Sensors_A_Review

X. APPENDICES

APPENDIX A: TRANSMITTER NODE CODE

```
/*
  SMART PARK - Transmitter Node (Isolation Mode)
  Hardware: Heltec WiFi LoRa 32 V2
*/

#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>
#include <EEPROM.h>
#include "SSD1306Wire.h"

// --- CONFIGURATION ---
#define NODE_ID 1
#define BAND    915E6
#define MAG_ADDR 0x1E      // Genuine HMC5883L Address

// --- PINS ---
#define SS_PIN   18
#define RST_PIN  14
#define DIO0_PIN 26
#define TRIG_PIN 25
#define ECHO_PIN 33

// --- OLED PINS (Bus 0) ---
#define OLED_SDA 4
#define OLED_SCL 15
#define OLED_RST 16

// --- SENSOR PINS (Bus 1) ---
#define SENS_SDA 21
#define SENS_SCL 22

// --- DEFAULTS ---
const int DEFAULT_DIST = 100;
const float DEFAULT_MAG_THRESH = 3.0;

// --- VARIABLES ---
int distThreshold = DEFAULT_DIST;
float magThreshold = DEFAULT_MAG_THRESH;
bool isSetupMode = false;

float magBaseline = 0.0;
SSD1306Wire display(0x3c, OLED_SDA, OLED_SCL);

// --- CREATE SECONDARY BUS ---
TwoWire MagBus = TwoWire(1); // Use Hardware I2C Channel 1

unsigned long lastSendTime = 0;
const long sendInterval = 2000;

// --- MEMORY ADDRESSES ---
#define EEPROM_SIZE 64
#define ADDR_BASELINE 0
#define ADDR_DIST 10
#define ADDR_MAG_THRESH 20

// --- DIRECT I2C DRIVER (USING MagBus) ---
void setMagMode() {
  MagBus.beginTransmission(MAG_ADDR);
  MagBus.write(0x02); // Mode Register
  MagBus.write(0x00); // Continuous-Measurement Mode
  MagBus.endTransmission();
  delay(5);
}

float readRawMag() {
  // Ensure we are in continuous mode
  setMagMode();

  MagBus.beginTransmission(MAG_ADDR);
  MagBus.write(0x03); // Point to Data X MSB
  MagBus.endTransmission();

   // Request 6 bytes from the SENSOR BUS, not the main wire
  MagBus.requestFrom(MAG_ADDR, 6);
  if(MagBus.available() >= 6) {
    int16_t x = MagBus.read() << 8 | MagBus.read();
    int16_t z = MagBus.read() << 8 | MagBus.read();
    int16_t y = MagBus.read() << 8 | MagBus.read();

    // Scale by 10 for easier reading
      return sqrt(sq((float)x) + sq((float)y) + sq((float)z)) / 10.0;
  }
  return 0.0;
}

void setup() {
  Serial.begin(115200);

  // 1. Init OLED (Uses Standard Wire Bus 0)
  pinMode(OLED_RST, OUTPUT);
```

```
  digitalWrite(OLED_RST, LOW); delay(20); digitalWrite(OLED_RST, HIGH);
  display.init();
  display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, "Booting Node " + String(NODE_ID) + "...");
  display.display();

  // 2. Init Ultrasonic
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // 3. Init LoRa
  SPI.begin(5, 19, 27, 18);
  LoRa.setPins(SS_PIN, RST_PIN, DIO0_PIN);
  if (!LoRa.begin(BAND)) {
    display.drawString(0, 10, "LoRa Error!"); display.display();
    while (1);
  }
  LoRa.setSyncWord(0xF3);

  // 4. Init Magnetometer (ON SECONDARY BUS)
  Serial.println("Starting Sensor Bus...");
  MagBus.begin(SENS_SDA, SENS_SCL, 100000); // Start Bus 1 on 21/22
  setMagMode();

  // Test Read
  float test = readRawMag();
  Serial.print("Test Read: "); Serial.println(test);
  if (test == 0.0) {
    Serial.println("SENSOR FAILURE: Still 0.0");
    display.drawString(0, 20, "Sensor Failed (0)");
    display.display();
  }

  // 5. Memory Load & FORCE CALIBRATION
  EEPROM.begin(EEPROM_SIZE);

  // We force calibration once here to make sure we don't load 0.0 from memory
  Serial.println("Calibrating...");
  calibrateNow();
  saveAllSettings();

  // Load Thresholds
  int savedDist;
  float savedMagThresh;
  EEPROM.get(ADDR_DIST, savedDist);
  EEPROM.get(ADDR_MAG_THRESH, savedMagThresh);

  if (savedDist > 0 && savedDist < 500) distThreshold = savedDist;
  else distThreshold = DEFAULT_DIST;

  if (!isnan(savedMagThresh) && savedMagThresh > 0.0) magThreshold = savedMagThresh;
  else magThreshold = DEFAULT_MAG_THRESH;

  Serial.println("--- NODE STARTED ---");
  updateScreenStatus(0, 0, false, false);
}

void loop() {
  listenForCommands(isSetupMode ? 100 : 500);

  if (isSetupMode) {
    display.clear();
    display.setFont(ArialMT_Plain_16);
    display.drawString(10, 20, "SETUP MODE");
    display.setFont(ArialMT_Plain_10);
    display.drawString(15, 40, "Mount sensor now...");
    display.display();
    return;
  }

  unsigned long currentMillis = millis();
  if (currentMillis - lastSendTime >= sendInterval) {
    lastSendTime = currentMillis;

    // 1. Read Sensors
    long duration, distance;
    digitalWrite(TRIG_PIN, LOW); delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH); delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    duration = pulseIn(ECHO_PIN, HIGH, 30000);
    distance = (duration == 0) ? 999 : (duration * 0.034 / 2);

    float currentMag = readRawMag();
    float magDiff = abs(currentMag - magBaseline);

    // Debug print
    Serial.print("Raw Mag: "); Serial.print(currentMag);
    Serial.print(" | Diff: "); Serial.println(magDiff);

    // 2. Logic Checks
```

```
  bool ultraDetected = (distance < distThreshold);
  bool magDetected = (magDiff > magThreshold);
  bool occupied = ultraDetected && magDetected;

  // 3. Update OLED
    updateScreenStatus(distance, magDiff, ultraDetected, magDetected);

  // 4. Send Packet
        String packet = String(NODE_ID) + ":" + String(occupied ? 1 : 0) + ":" +
            String(distance) + ":" + String(magDiff);

  LoRa.beginPacket();
  LoRa.print(packet);
  LoRa.endPacket();
  Serial.print("Sent: "); Serial.println(packet);
 }
}

void updateScreenStatus(int dist, float magVal, bool ultraYes, bool magYes) {
 display.clear();

 display.setFont(ArialMT_Plain_10);
 display.drawString(0, 0, "NODE " + String(NODE_ID) + " MONITOR");
 display.drawLine(0, 12, 128, 12);

 // Ultra Status
  String uStr = "ULTRA: " + String(ultraYes ? "YES" : "NO") + " (" + String(dist) + "cm)";
 display.drawString(0, 16, uStr);

 // Mag Status
  String mStr = "MAG:   " + String(magYes ? "YES" : "NO") + " (D:" + String(magVal, 1) + ")";
 display.drawString(0, 28, mStr);

 // Final Status
 display.setFont(ArialMT_Plain_16);
  String status = (ultraYes && magYes) ? "CAR" : "FREE";
 display.drawString(0, 45, "STATUS: " + status);

 display.display();
}

void listenForCommands(int durationMs) {
 long start = millis();
 while (millis() - start < durationMs) {
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
   String incoming = "";
            while (LoRa.available()) { incoming += (char)LoRa.read(); }

    if (incoming.startsWith("SET:")) {
     int first = incoming.indexOf(':');
     int second = incoming.indexOf(':', first + 1);
     int third = incoming.indexOf(':', second + 1);
     String idStr = incoming.substring(first+1, second);

     if (idStr.toInt() == NODE_ID) {
          distThreshold = incoming.substring(second+1, third).toInt();
                                magThreshold = incoming.substring(third+1).toFloat();
       saveAllSettings();
       sendAck("SAVED_TUNE");
     }
    }
    else if (incoming.startsWith("ZERO:")) {
     int first = incoming.indexOf(':');
     String idStr = incoming.substring(first+1);
     if (idStr.toInt() == NODE_ID) {
       calibrateNow();
       saveAllSettings();
       sendAck("SAVED_ZERO");
     }
    }
    else if (incoming.startsWith("CMD:SETUP")) {
     int lastDiv = incoming.lastIndexOf(':');
     String idStr = incoming.substring(lastDiv+1);
     if (idStr.toInt() == NODE_ID) {
       isSetupMode = true;
       sendAck("SETUP_ON");
     }
    }
    else if (incoming.startsWith("CMD:FINISH")) {
     int lastDiv = incoming.lastIndexOf(':');
     String idStr = incoming.substring(lastDiv+1);
     if (idStr.toInt() == NODE_ID) {
       calibrateNow();
       saveAllSettings();
       isSetupMode = false;
       sendAck("SETUP_DONE_SAVED");
     }
    }
    else if (incoming.startsWith("CMD:RESET")) {
```

```
        int lastDiv = incoming.lastIndexOf(':');
        String idStr = incoming.substring(lastDiv+1);
        if (idStr.toInt() == NODE_ID) {
           distThreshold = DEFAULT_DIST;
           magThreshold = DEFAULT_MAG_THRESH;
           calibrateNow();
           saveAllSettings();
           isSetupMode = false;
           sendAck("RESET_DONE");
        }
      }
    }
  }
}

void calibrateNow() {
  display.clear();
  display.setFont(ArialMT_Plain_10);
  display.drawString(20, 20, "Calibrating...");
  display.display();

  float total = 0;
  // Use MagBus specifically!
  for(int i=0; i<20; i++) {
    total += readRawMag(); delay(20);
  }
  magBaseline = total / 20;
}

void saveAllSettings() {
  EEPROM.put(ADDR_BASELINE, magBaseline);
  EEPROM.put(ADDR_DIST, distThreshold);
  EEPROM.put(ADDR_MAG_THRESH, magThreshold);
  EEPROM.commit();
}

void sendAck(String msg) {
   delay(50);
    String ackPacket = "ACK:" + String(NODE_ID) + ":" +
msg;
   LoRa.beginPacket();
   LoRa.print(ackPacket);
   LoRa.endPacket();
}
```

APPENDIX B: RECEIVER GATEWAY CODE

```
/*
     SMART PARK - Receiver Gateway (Persistent
Connection)
  Hardware: Heltec WiFi LoRa 32 V2

  UPDATES:
    - FIXED: "log [random text]" bug (now rejects invalid
inputs).
  - ADDED "cancel" command.
  - Header displays "Reconn..." instead of splash screen.
  - Added "Feedback Details" to ALL commands.
  - Fixed Display to show ANY node ID in Log status.
*/

#define BLYNK_TEMPLATE_ID "TMPL6-ntrYU9F"
#define BLYNK_TEMPLATE_NAME "smart park"
#define                              BLYNK_AUTH_TOKEN
"fSVSO6ww02UVGUzpaLgHYasEJAkaPARw"

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <SPI.h>
#include <LoRa.h>
#include <Wire.h>
#include <EEPROM.h>
#include "SSD1306Wire.h"

// --- PINS ---
#define SS_PIN   18
#define RST_PIN  14
#define DIO0_PIN 26
#define BAND 915E6
#define OLED_SDA 4
#define OLED_SCL 15
#define OLED_RST 16

SSD1306Wire display(0x3c, OLED_SDA, OLED_SCL);

// --- SETTINGS ---
#define MAX_NODES 20
#define SCROLL_DELAY 3000
#define EEPROM_SIZE 128 // Bytes for SSID/Pass

struct ParkNode {
  int id;
  int blynkPin;
  bool occupied;
  bool active;
};

ParkNode nodes[MAX_NODES];

// --- GLOBALS ---
int logFilter = 0;
bool isBlynkActive = false;
bool shouldAutoReconnect = false;
String savedSSID = "";
String savedPass = "";
```

```
String pendingPayload = "";
int pendingTarget = 0;
bool waitingForAck = false;
unsigned long ackStartTime = 0;

int scrollPage = 0;
unsigned long lastScrollTime = 0;
unsigned long lastWifiCheck = 0;

void setup() {
  Serial.begin(115200);
  EEPROM.begin(EEPROM_SIZE);

  // 1. OLED Init
  pinMode(OLED_RST, OUTPUT);
          digitalWrite(OLED_RST,      LOW);      delay(20);
digitalWrite(OLED_RST, HIGH);
  display.init();
  display.flipScreenVertically();

  // 2. LoRa Init
  SPI.begin(5, 19, 27, 18);
  LoRa.setPins(SS_PIN, RST_PIN, DIO0_PIN);
   if (!LoRa.begin(BAND)) { Serial.println("LoRa Failed!");
while(1); }
  LoRa.setSyncWord(0xF3);
  Blynk.config(BLYNK_AUTH_TOKEN);

  // 3. Init Nodes
  nodes[0] = {1, 1, false, true};
  nodes[1] = {2, 0, false, true};
  for(int i=2; i<MAX_NODES; i++) nodes[i].active = false;

  // 4. LOAD WIFI
  loadCredentials();


Serial.println("================================
==========");
  Serial.println("       SMART PARK SYSTEM READY");

Serial.println("================================
==========");

  if (savedSSID.length() > 1) {
    shouldAutoReconnect = true;
    Serial.println("Found saved WiFi: " + savedSSID);
  }

  updateDisplay();
}

void loop() {
  // 1. WiFi Management (Non-Blocking)
  if (shouldAutoReconnect) {
    checkConnection();
    if (WiFi.status() == WL_CONNECTED) {
      Blynk.run();
    }
  }

  // 2. Scroll Check
  if (millis() - lastScrollTime > SCROLL_DELAY) {
    lastScrollTime = millis();
    scrollPage++;
    updateDisplay();
  }

  // 3. Serial Commands (Always Active)
  if (Serial.available()) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    parseCommand(cmd);
  }

  // 4. LoRa Receiver
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    String incoming = "";
              while (LoRa.available())  {  incoming  +=
(char)LoRa.read(); }
    parseData(incoming);
  }

  // 5. Ack Timeout
  if (waitingForAck && millis() - ackStartTime > 4000) {
    Serial.println("\n[TIMEOUT] Node did not reply.");
    waitingForAck = false;
    pendingPayload = "";
    pendingTarget = 0;
  }
}

// --- WIFI RECONNECT LOGIC (PERSISTENT) ---
void checkConnection() {
  if (millis() - lastWifiCheck > 10000) {
    lastWifiCheck = millis();

    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("\n[WiFi] Connecting to " + savedSSID +
"...");
      Serial.println("(Type 'logout' to stop retrying)");

      updateDisplay();

      WiFi.disconnect();
      WiFi.begin(savedSSID.c_str(), savedPass.c_str());

    } else {
      if (!isBlynkActive) {
        Serial.println("[WiFi] Online! Syncing Blynk...");
        Blynk.connect();
        isBlynkActive = true;
        updateDisplay();
```

```
    }
   }
  }
}

void updateDisplay() {
  display.clear();

  // HEADER
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, "SMART PARK");

  if (shouldAutoReconnect) {
   if (WiFi.status() == WL_CONNECTED) {
    display.drawString(80, 0, "WiFi: ON");
   } else {
    display.drawString(80, 0, "Reconn...");
   }
  } else {
   display.drawString(80, 0, "Offline");
  }

  display.drawString(0, 10, "----------------");

  // LOG STATUS
  if (logFilter == 99) {
   display.drawString(80, 14, "[LOG: A]");
  } else if (logFilter > 0) {
    display.drawString(80, 14, "[LOG: " + String(logFilter)
+ "]");
  }

  // LIST LOGIC
  int activeIndices[MAX_NODES];
  int activeCount = 0;
  for(int i=0; i<MAX_NODES; i++) {
   if(nodes[i].active) activeIndices[activeCount++] = i;
  }

  if (activeCount == 0) {
   display.setFont(ArialMT_Plain_16);
   display.drawString(20, 30, "NO NODES");
   display.display();
   return;
  }

  int totalPages = (activeCount + 1) / 2;
  if (totalPages < 1) totalPages = 1;
  scrollPage = scrollPage % totalPages;

  display.setFont(ArialMT_Plain_16);
  int startIdx = scrollPage * 2;

  if (startIdx < activeCount) {
   int i = activeIndices[startIdx];
       String line = "N" + String(nodes[i].id) + ": " +
String(nodes[i].occupied ? "OCCUPIED" : "FREE");
   display.drawString(0, 25, line);
  }

  if (startIdx + 1 < activeCount) {
   int i = activeIndices[startIdx + 1];
       String line = "N" + String(nodes[i].id) + ": " +
String(nodes[i].occupied ? "OCCUPIED" : "FREE");
   display.drawString(0, 45, line);
  }

  if (totalPages > 1) {
   display.setFont(ArialMT_Plain_10);
     display.drawString(110, 54, "P" + String(scrollPage +
1));
  }

  display.display();
}

// --- MEMORY FUNCTIONS ---
void saveCredentials() {
  for (int i = 0; i < EEPROM_SIZE; i++) EEPROM.write(i,
0);
  int addr = 0;
  EEPROM.writeString(addr, savedSSID);
  addr += savedSSID.length() + 1;
  EEPROM.writeString(addr, savedPass);
  EEPROM.commit();
  Serial.println("[Memory] WiFi Credentials Saved.");
}

void loadCredentials() {
  int addr = 0;
  savedSSID = EEPROM.readString(addr);
  addr += savedSSID.length() + 1;
  savedPass = EEPROM.readString(addr);

   if(savedSSID.length() > 32 || savedSSID.length() < 1)
savedSSID = "";
  if(savedPass.length() > 64) savedPass = "";
}

void clearCredentials() {
  for (int i = 0; i < EEPROM_SIZE; i++) EEPROM.write(i,
0);
  EEPROM.commit();
  savedSSID = "";
  savedPass = "";
  Serial.println("[Memory] Credentials Cleared.");
}

// --- COMMANDS ---
void performLogin() {
  Serial.println("\n[CMD] Login Sequence Initiated");
  Serial.println(">> Enter SSID:");
  while(!Serial.available()) delay(10);
  savedSSID = Serial.readStringUntil('\n');
```

```
  savedSSID.trim();
  Serial.println("Target SSID: [" + savedSSID + "]");

  Serial.println(">> Enter Pass:");
  while(!Serial.available()) delay(10);
  savedPass = Serial.readStringUntil('\n');
  savedPass.trim();
  Serial.println("Target Pass: [******]");

  Serial.println("Connecting to " + savedSSID + "...");
  shouldAutoReconnect = true;
  updateDisplay();

  WiFi.begin(savedSSID.c_str(), savedPass.c_str());

  int k=0;
  while(WiFi.status()!=WL_CONNECTED && k<20) {
    delay(400); Serial.print("."); k++;
  }

  if(WiFi.status()==WL_CONNECTED) {
    Blynk.connect();
    isBlynkActive=true;
    saveCredentials();
    Serial.println("\nOnline. Credentials Saved.");
  } else {
    Serial.println("\nFailed. Will retry in background.");
    saveCredentials();
  }
  updateDisplay();
}

void performLogout() {
  Serial.println("\n[CMD] Logout Initiated...");
  Blynk.disconnect();
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);

  isBlynkActive = false;
  shouldAutoReconnect = false;
  clearCredentials();

     Serial.println("[SUCCESS] Logged out & Network Forgotten.");
  delay(100);
  updateDisplay();
}

// --- NEW FUNCTION: CANCEL ---
void performCancel() {
  Serial.println("\n[CMD] Cancel Initiated...");

  if (pendingTarget == 0 && !waitingForAck) {
    Serial.println("[INFO] No pending commands found.");
    return;
  }

   // Feedback for queued commands (waiting for node to wake up)
  if (pendingTarget != 0) {
     Serial.println(" - Removed queued command for Node " + String(pendingTarget));
  }

  // Feedback for waiting for ACK (waiting for reply)
  if (waitingForAck) {
    Serial.println(" - Stopped waiting for ACK.");
  }

  // Clear everything
  pendingPayload = "";
  pendingTarget = 0;
  waitingForAck = false;

     Serial.println("[SUCCESS] All pending operations cleared.");
}

void parseCommand(String cmd) {
  // 1. Single Word Commands
  if (cmd == "login") performLogin();
  else if (cmd == "logout") performLogout();
  else if (cmd == "cancel") performCancel();
  else if (cmd == "add") performAddNode();
  else if (cmd == "del") performDelNode();
  else if (cmd == "list") printNodeList();

  // 2. Log Command
  else if (cmd.startsWith("log")) {
    int space = cmd.indexOf(' ');
    String arg = (space > 0) ? cmd.substring(space+1) : "";

    if(arg == "all") {
       logFilter = 99;
       Serial.println("[CMD] Log Filter: ALL Nodes");
    }
    else if(arg == "off") {
       logFilter = 0;
       Serial.println("[CMD] Log Filter: OFF");
    }
    else {
       // --- FIX: Ensure input is a valid positive number ---
       int val = arg.toInt();
       if (val > 0) {
         logFilter = val;
         Serial.print("[CMD] Log Filter: Node ");
         Serial.println(logFilter);
       } else {
           Serial.println("[ERROR] Invalid Log ID. Use 'log [ID]', 'log all', or 'log off'.");
       }
    }
    updateDisplay();
  }
```

```
  // 3. Complex Commands
  else {
    int space = cmd.indexOf(' ');

    if (space == -1) {
          Serial.println("[ERROR] Unknown Command: " +
cmd);
      return;
    }

    String type = cmd.substring(0, space);
    String args = cmd.substring(space + 1);
    int targetID = args.toInt();

    if (type == "setup") {
      pendingPayload = "CMD:SETUP";
      pendingTarget = targetID;
          Serial.print("[CMD] Queuing SETUP for Node ");
Serial.println(targetID);
    }
    else if (type == "finish") {
      // FIX: Force send immediately to wake up silent Node
      String packet = "CMD:FINISH:" + String(targetID);

      LoRa.beginPacket();
      LoRa.print(packet);
      LoRa.endPacket();

      Serial.println("[CMD] FORCE SENT 'FINISH' to Node
" + String(targetID));

      // Manually start waiting for reply (bypass queue)
      pendingTarget = 0;
      waitingForAck = true;
      ackStartTime = millis();
      LoRa.receive();
    }
    else if (type == "reset") {
      pendingPayload = "CMD:RESET";
      pendingTarget = targetID;
          Serial.print("[CMD] Queuing RESET for Node ");
Serial.println(targetID);
    }
    else if (type == "zero") {
      pendingPayload = "ZERO";
      pendingTarget = targetID;
        Serial.print("[CMD] Queuing RE-ZERO for Node ");
Serial.println(targetID);
    }
    else if (type == "tune") {
      int s2 = args.indexOf(' ');
      int s3 = args.lastIndexOf(' ');
      if (s2 > 0 && s3 > 0) {
        String dist = args.substring(s2+1, s3);
        String mag = args.substring(s3+1);
        targetID = args.substring(0, s2).toInt();
        pendingPayload = "SET:" + dist + ":" + mag;
        pendingTarget = targetID;
              Serial.print("[CMD] Queued TUNE Node ");
Serial.println(targetID);
      } else {
        Serial.println("[ERROR] Invalid Tune Parameters");
      }
    }
    else {
      Serial.println("[ERROR] Unknown Command Type: "
+ type);
    }
  }
}

void performAddNode() {
  Serial.println("\n[CMD] Add Node Sequence Initiated");
  Serial.println(">> Enter Node ID:");
  while(!Serial.available()) delay(10);
  int newID = Serial.readStringUntil('\n').toInt();

  Serial.println(">> Enter Blynk V-Pin:");
  while(!Serial.available()) delay(10);
  int newPin = Serial.readStringUntil('\n').toInt();

  bool saved = false;
  for(int i=0; i<MAX_NODES; i++) {
    if(nodes[i].active && nodes[i].id == newID) {
      Serial.println("[ERROR] ID Exists!"); return;
    }
    if(!nodes[i].active) {
      nodes[i] = {newID, newPin, false, true};
      saved = true;
            Serial.println("[SUCCESS] Added Node " +
String(newID));
      break;
    }
  }
  if(!saved) Serial.println("[ERROR] List Full");
  updateDisplay();
}

void performDelNode() {
       Serial.println("\n[CMD] Delete Node Sequence
Initiated");
  Serial.println(">> Enter ID to Delete:");
  while(!Serial.available()) delay(10);
  int targetID = Serial.readStringUntil('\n').toInt();

  for(int i=0; i<MAX_NODES; i++) {
    if(nodes[i].active && nodes[i].id == targetID) {
      nodes[i].active = false;
            Serial.println("[SUCCESS] Deleted Node " +
String(targetID));
      updateDisplay();
      return;
    }
```

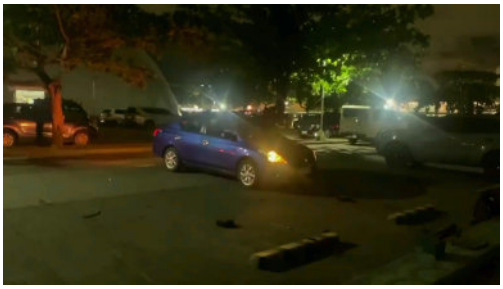
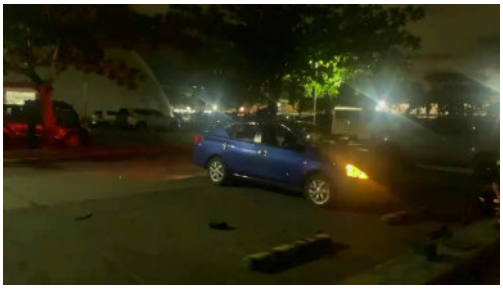
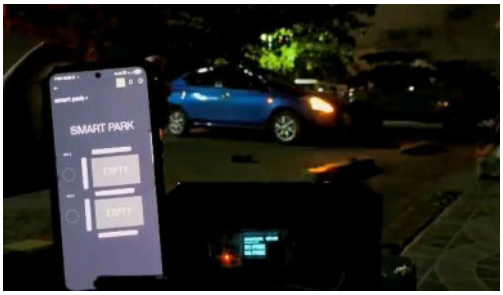
```
  }
  Serial.println("[ERROR] Node Not Found.");
}

void printNodeList() {
  Serial.println("\n[CMD] Listing Active Nodes...");
  Serial.println("--- NODES ---");
  for(int i=0; i<MAX_NODES; i++) {
    if(nodes[i].active) {
            Serial.println("Slot " + String(i) + ": ID " +
String(nodes[i].id) + " (V" + String(nodes[i].blynkPin) +
")");
    }
  }
  Serial.println("-------------");
}

void parseData(String packet) {
  if (packet.startsWith("ACK:")) {
                Serial.println("\n[SUCCESS]    Reply:    "    +
packet.substring(4));
    waitingForAck = false; pendingTarget = 0; return;
  }

  int firstDiv = packet.indexOf(':');
  if (firstDiv == -1) return;
  String idStr = packet.substring(0, firstDiv);
  int nodeID = idStr.toInt();

   if (pendingTarget != 0 && pendingTarget == nodeID)
sendCommand(nodeID);

  int secondDiv = packet.indexOf(':', firstDiv + 1);
  int thirdDiv = packet.indexOf(':', secondDiv + 1);
      String statusStr = packet.substring(firstDiv + 1,
secondDiv);
     String distStr = packet.substring(secondDiv + 1,
thirdDiv);
  String magStr = packet.substring(thirdDiv + 1);

    bool isOccupied = (statusStr == "1" || statusStr ==
"OCCUPIED");
  bool nodeFound = false;

  for(int i=0; i<MAX_NODES; i++) {
    if (nodes[i].active && nodes[i].id == nodeID) {
      nodes[i].occupied = isOccupied;
      nodeFound = true;
                                        if        (isBlynkActive)
Blynk.virtualWrite(nodes[i].blynkPin, isOccupied ? 1 : 0);
      break;
    }
  }

  updateDisplay();

  if (logFilter == 99 || logFilter == nodeID) {
        if (!nodeFound) Serial.print("[UNKNOWN "); else
Serial.print("[");
     Serial.print("NODE " + String(nodeID) + "] ");
       Serial.println("Dist:" + distStr + " Mag:" + magStr +
(isOccupied ? " CAR" : " FREE"));
  }
}

void sendCommand(int targetID) {
  delay(15);
    String fullPacket = (pendingPayload == "ZERO" ?
"ZERO:" : pendingPayload + ":") + String(targetID);
             LoRa.beginPacket();       LoRa.print(fullPacket);
LoRa.endPacket();
      Serial.println(">> SENT CMD to Node " +
String(targetID));
  pendingTarget = 0; waitingForAck = true; ackStartTime =
millis(); LoRa.receive();
}
```

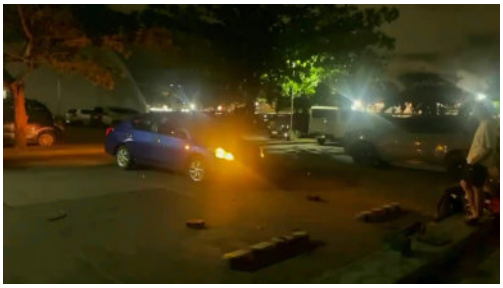
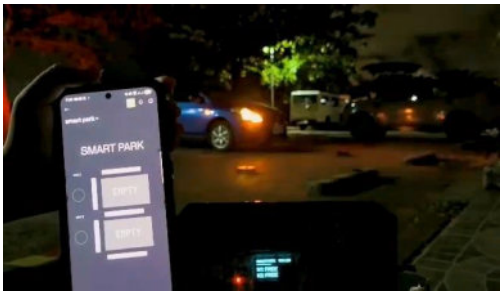
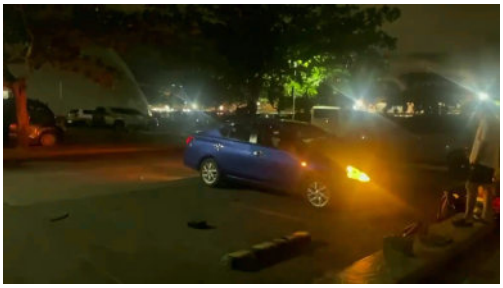
APPENDIX C: VISUAL DATA LOGS (FIELD TEST)

This appendix contains the complete photographic documentation of the experimental trials. Each scenario below presents a side-by-side comparison of the physical parking state (left) and the resulting mobile application status (right) before and after the event. The free and occupied state is represented by 0 and 1, respectively.

C.1. Scenario: Single Vehicle Arrival (Slot 2)

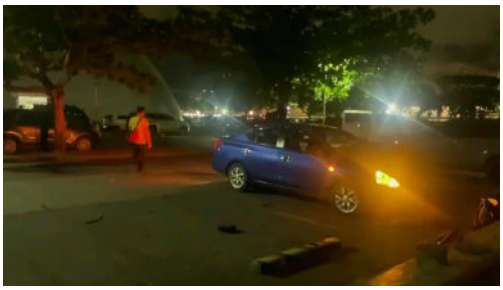
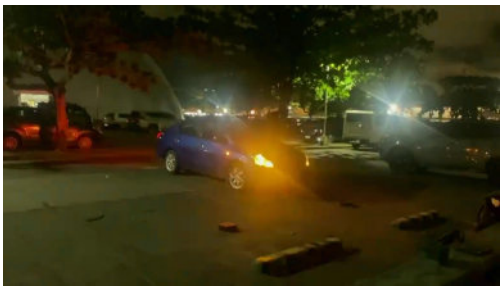
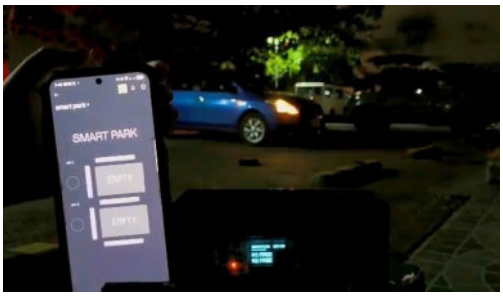
Transition tested: State 0 0 - State 1 0.

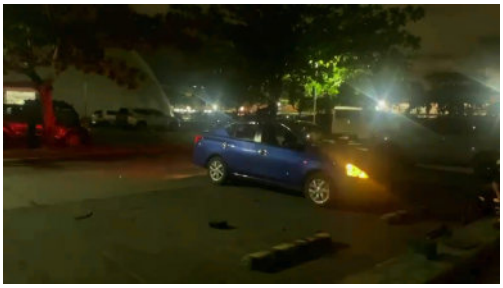
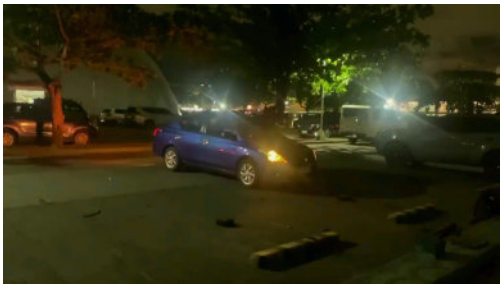
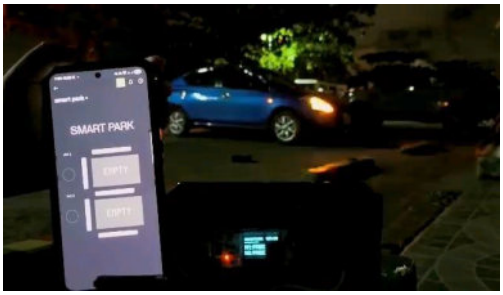
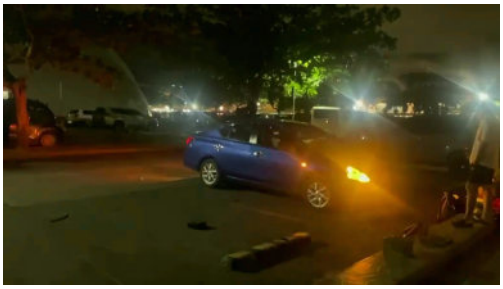
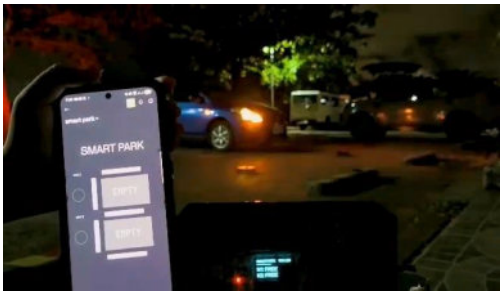
Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 3s)*		
Trial 2 *(Latency: 3s)*		
		

Trial 3 *(Latency: 4s)*	 	 

C.2. Scenario: Single Vehicle Leaving (Slot 2)

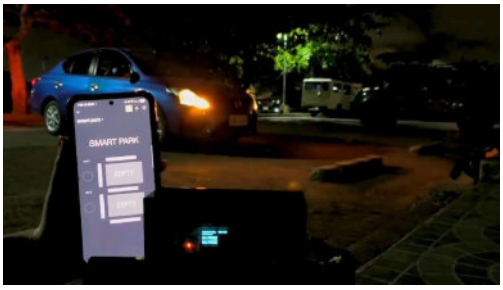
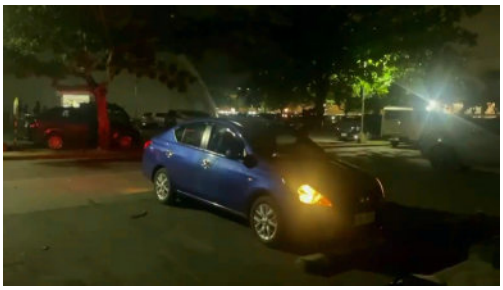
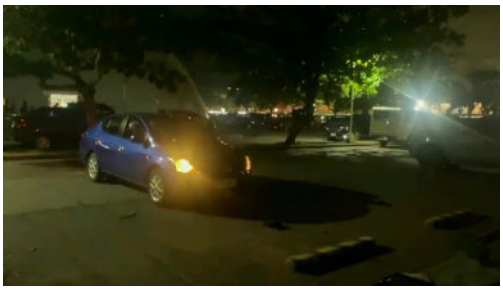
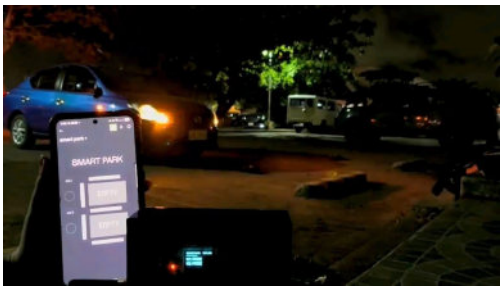
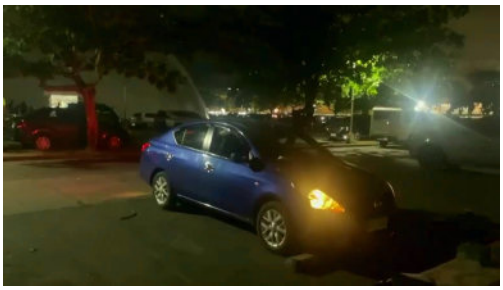
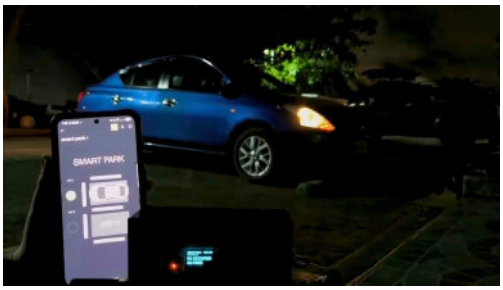
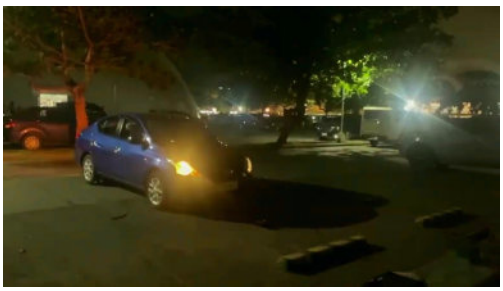
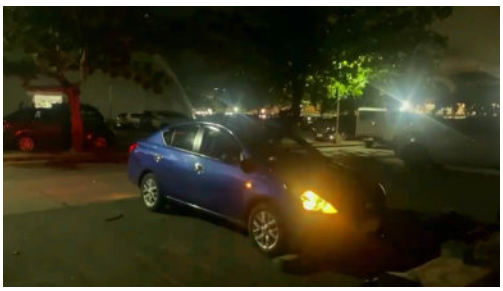
Transition tested: State 1 0 - State 0 0.

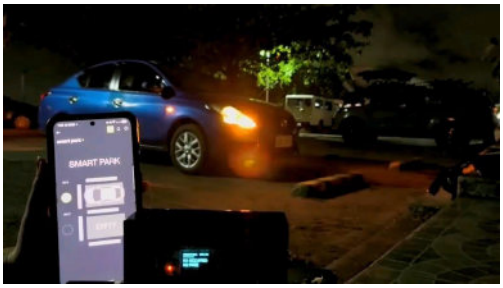
Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 4s)*	 	 

Trial 2 *(Latency: 3s)*	 	 
Trial 3 *(Latency: 4s)*	 	 

C.3. Scenario: Single Vehicle Arrival (Slot 1)

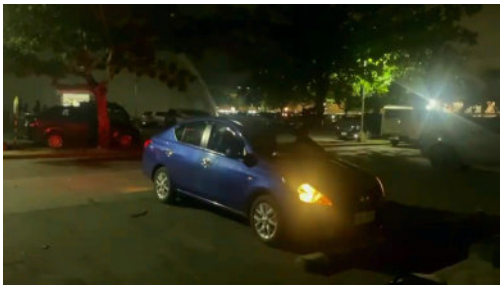
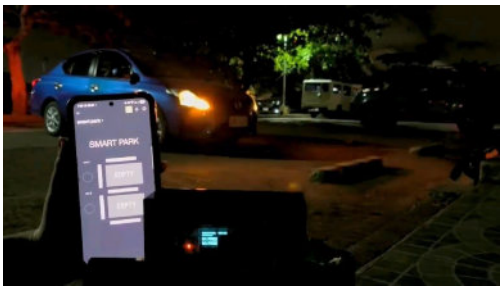
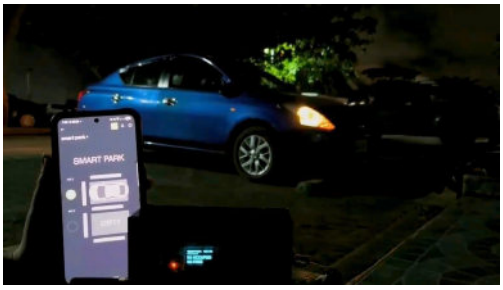
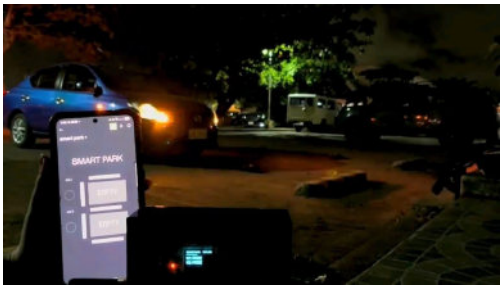
Transition tested: State 0 0 - State 0 1.

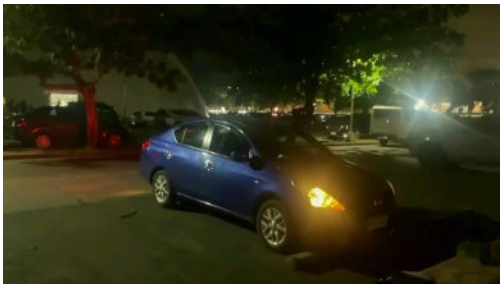
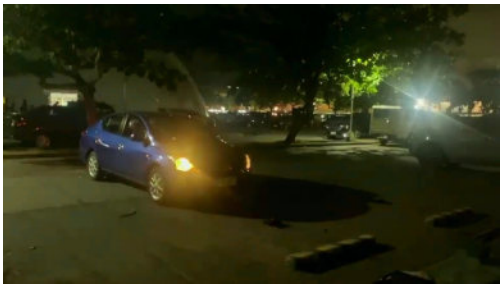
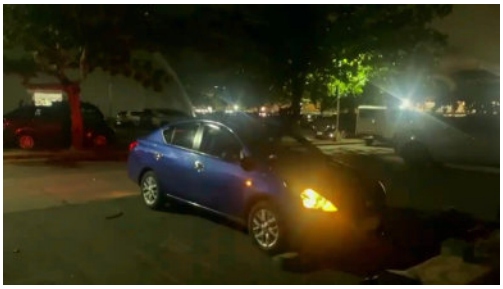
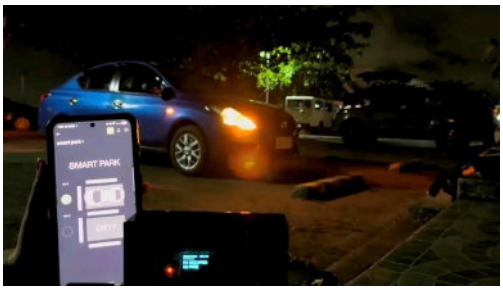
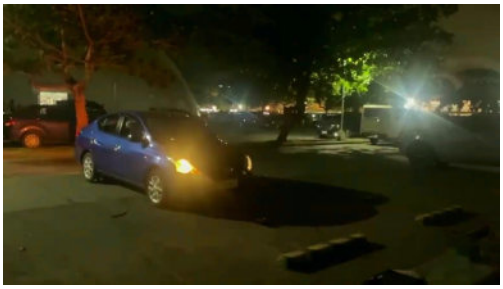
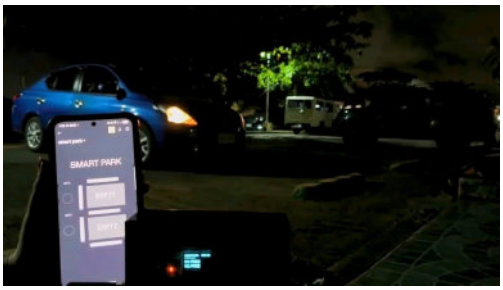
Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 3s)*	 	 
Trial 2 *(Latency: 4s)*	 	 
Trial 3 *(Latency: 4s)*		

C.4. Scenario: Single Vehicle Leaving (Slot 1)

Transition tested: State 0 1 - State 0 0.

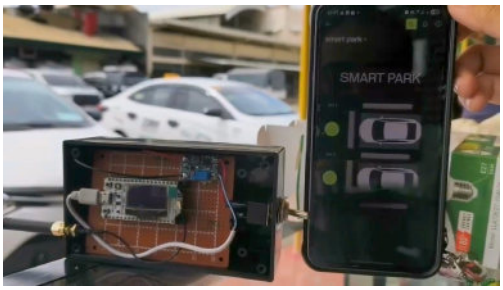
Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 5s)*	 	 
Trial 2 *(Latency: 4s)*		

		
Trial 3 *(Latency: 4s)*	 	 

C.5. Scenario: Simultaneous Vehicle Arrival (Slot 1 and Slot 2)

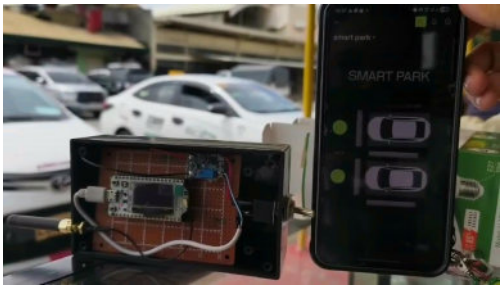
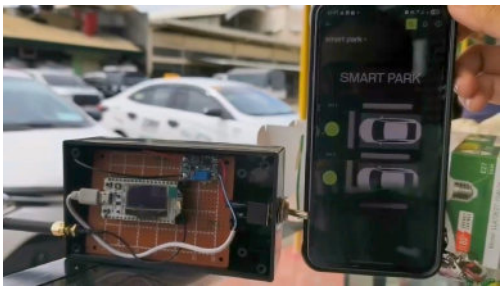
Transition tested: State 0 0 - State 1 1.

Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 4s)*		

		
Trial 2 *(Latency: 3s)*	 	 
Trial 3 *(Latency: 3s)*	 	 

C.6. Scenario: Simultaneous Vehicle Departure (Slot 1 and Slot 2)

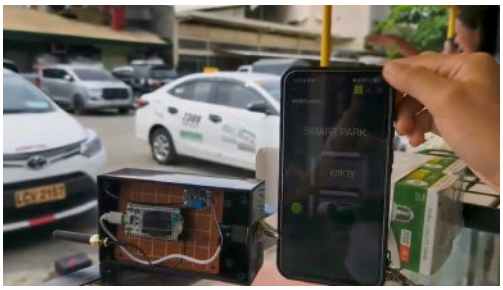
Transition tested: State 1 1 - State 0 0.

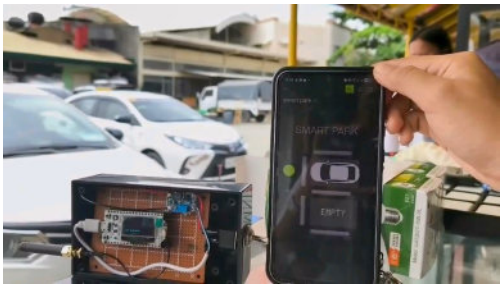
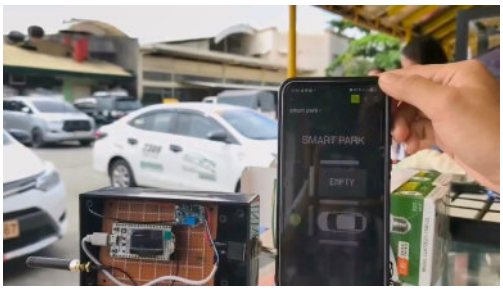
Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 5s)*	 	 
Trial 2 *(Latency: 6s)*	 	 
Trial 3 *(Latency: 5s)*		

C.7. Scenario: Swap (Slot 1 out, Slot 2 in)

Transition tested: State 0 1 - State 1 0.

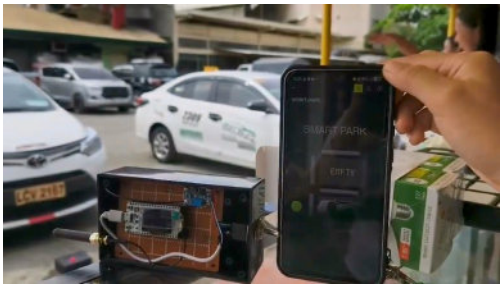
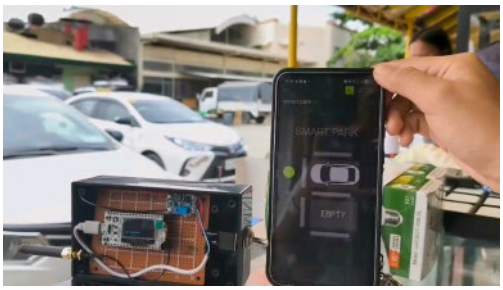
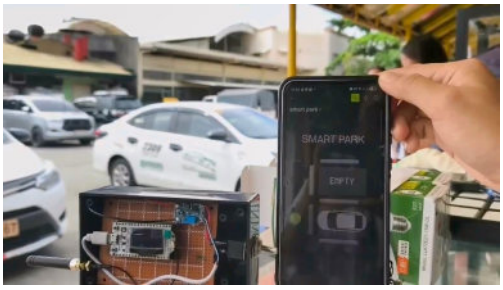
Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 3s)*	 	 
Trial 2 *(Latency: 4s)*		

		
Trial 3 *(Latency: 3s)*	 	 

C.8. Scenario: Swap (Slot 2 out, Slot 1 in)

Transition tested: State 1 0 - State 0 1.

Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 3s)*		

		
Trial 2 *(Latency: 4s)*	 	 
Trial 3 *(Latency: 3s)*	 	 

C.9. Scenario: Slot 2 Arrives (Slot 1 full)

Transition tested: State 0 1 - State 1 1.

Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 2s)*	 	 
Trial 2 *(Latency: 2s)*	 	 
Trial 3 *(Latency: 3s)*		

C.10. Scenario: Slot 2 Departed (Slot 1 full)

Transition tested: State 1 1 - State 0 1.

Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 4s)*	 	 
Trial 2 *(Latency: 4s)*		

		
Trial 3 *(Latency: 5s)*	 	 

C.11. Scenario: Slot 1 Arrived (Slot 2 full)

Transition tested: State 1 0 - State 1 1.

Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 4s)*		

		
Trial 2 *(Latency: 3s)*	 	 
Trial 3 *(Latency: 3s)*	 	 

C.12. Scenario: Slot 1 Departed (Slot 2 full)

Transition tested: State 1 1 - State 1 0.

Trial & Response Time	***INITIAL STATE***	***FINAL STATE***
Trial 1 *(Latency: 3s)*	 	 
Trial 2 *(Latency: 4s)*	 	 
Trial 3 *(Latency: 3s)*		

	 SMART PARK	 SMART PARK

APPENDIX D: BILL OF MATERIALS AND COST ANALYSIS

To validate the "low cost" claim, a comprehensive financial breakdown of the prototype's components was conducted. The detailed costs below represent the expenditure required to manufacture one single sensor node (i.e., one parking slot).

Component	***Specification***	***Unit Cost (PHP)***	***Total (PHP)***
Microcontroller	*Heltec WiFi LoRa 32 V2*	*₱806.00*	*₱806.00*
Ultrasonic Sensor	*JSN-SR04T (Waterproof)*	*₱154.00*	*₱154.00*
Magnetometer	*GY-271 (HMC5883L)*	*₱116.00*	*₱116.00*
Battery	*2x 18650 Li-Ion (Rechargeable)*	*₱35.00*	*₱70.00*
BMS	*1S/2S Battery Management System*	*₱46.00*	*₱46.00*
Enclosure	*Plastic Case*	*₱220.00*	*₱220.00*
MPPT	*Lithium Battery Charger Module SD05CRMA*	*₱39.50*	*₱39.50*
Regulator	*SX1808 Step-up Boost Converter*	*₱180.00*	*₱180.00*
Misc.	*Wires, PCB, Connectors, Hinges, Switch*	*~₱400.00*	*₱400.00*
COST PER SLOT	–	–	***₱1,989.50 (~$36)***

Table III: Material Cost Per Sensor Node (SMART PARK Prototype)

Solution	Model	Connectivity	Cost Per Slot (Est.)	Cost Difference
SMART PARK	**Prototype Node**	**LoRa**	***₱1,989.50 (~$36)***	**Baseline**
Bosch	Parking Lot Sensor PLS	LoRaWAN	₱13,000 (~$230)	+453% (81.91% Reduction)
NKE Watteco	Smart Parking Sensor	LoRaWAN	₱10,000 (~$180)	+453% (81.91% Reduction)
MOKOSmart	LW009-IG	LoRaWAN	₱11,000 (~$195)	+453% (81.91% Reduction)

Table IV: Cost Comparison vs. Commercial Alternatives (2024)

The Receiver Gateway is a one-time fixed cost that is amortized over the total number of parking spaces served by the system. It contains the power supply unit and the main LoRa/Wi-Fi communication bridge.

Component	***Specification***	***Unit Cost (PHP)***	***Total (PHP)***
Microcontroller	*Heltec WiFi LoRa 32 V2*	*₱806.00*	*₱806.00*
Battery	*2x 18650 Li-Ion (Rechargeable)*	*₱35.00*	*₱70.00*
Enclosure	*Plastic Case*	*₱65.00*	*₱65.00*
Regulator	*SX1808 Step-up Boost Converter*	*₱180.00*	*₱180.00*
Misc.	*Wires, PCB, Connectors, Hinges, Switch*	*~₱200.00*	*₱200.00*
COST PER SLOT	–	–	***₱1,321.00 (~$23)***

Table IV: Material Cost for Receiver Gateway(SMART PARK Prototype)