International University of Sarajevo

Faculty of engineering and Natural Science

January, 2026

PROJECT

Course: Intoduction to Engineering
Theme: *Smart parking lot*

Professor:

Dr. Tarik Namas

Students:

Amina Mahmutović

SARAJEVO, January, 2026

Table of Contents

1. INTORDUCTION

Modern urban environments face increasing challenges from traffic congestion and inefficient parking management. The surge in vehicle ownership, coupled with finite parking capacity, necessitates intelligent systems that optimize access control and space utilization. This project addresses these challenges through the development of an Automated Parking Management System—a practical and scalable engineering solution.

The project's objective extends beyond a conceptual demonstration; it focuses on the design and implementation of a functional, real-world-oriented system. By integrating microcontroller-based control, sensor arrays, RFID authentication, and visual feedback mechanisms, the system automates vehicle entry/exit, real-time occupancy monitoring, and access authorization.

A critical component is the synergy between the hardware and software layers. The hardware subsystem manages physical detection, authentication, and ramp actuation, while the software provides operational logic and system monitoring via a graphical user interface (GUI). This integrated architecture enhances transparency and streamlines testing and future expansion.

Furthermore, the system is designed for modularity. The current architecture supports future enhancements, such as advanced data analytics and the mathematical modeling of parking dynamics. Specifically, it establishes a foundation for integrating simulation tools within the MATLAB/Octave environment to gain deeper insights into system performance.

From both an educational and engineering standpoint, this project serves as a practical synthesis of electronics, automation, and software development. It demonstrates the successful translation of theoretical frameworks into a functional, integrated system designed to mitigate real-world inefficiencies. Beyond its immediate application, the project functions as a versatile platform for iterative research and further technological evolution.

2. PROBLEM DESCRIPTION AND PROJECT OBJECTIVES

2.1. PROBLEM DESCRIPTION

Parking management poses a significant operational challenge in contemporary urban environments. Conventional systems frequently rely on manual intervention or fragmented technological solutions, resulting in unauthorized access, suboptimal space utilization, prolonged wait times, and a lack of real-time data for both operators and users.

A primary deficiency in traditional facilities is the absence of an integrated mechanism to simultaneously manage authentication, access control, and occupancy. Without automated monitoring, parking areas are prone to congestion and operational errors. Furthermore, the lack of immediate feedback regarding availability increases vehicle circulation time, further contributing to local traffic congestion.

Another critical limitation is the difficulty of pre-deployment optimization. Many solutions are implemented without prior simulation, making performance assessment costly and complex. This underscores the necessity for systems that bridge physical implementation with software-based simulation and visualization.

Finally, many existing solutions lack scalability. Non-modular architectures restrict adaptability, making it difficult to integrate additional sensors or advanced analytical tools, such as predictive occupancy modeling. This project addresses these deficiencies by proposing an automated, modular system that integrates hardware control with software simulation, providing a transparent and extensible solution to modern parking challenges.

2.2. PROJECT OBJECTIVES

The core mission of this project is to architect and deploy an intelligent parking management ecosystem. By synthesizing hardware-software interfaces, the system aims to provide a high-fidelity platform for the control, monitoring, and simulation of modern parking dynamics.

To realize this vision, the project is structured around the following strategic objectives:

1. Automated Kinetic Control: To design an electromechanical entry/exit system driven by a motorized ramp for precise traffic regulation.
2. Contactless Authentication: To implement a high-reliability identification layer utilizing RFID technology for secure access management.
3. Dynamic Occupancy Analytics: To develop a sensor-integrated network that provides real-time data on parking availability via visual telemetry.
4. Interactive Feedback Systems: To engineer a dual-channel (visual and acoustic) interface for immediate user notification and system status updates.
5. Advanced HMI Development: To construct an intuitive Graphical User Interface (GUI) that serves as a centralized hub for both operational monitoring and behavioral simulation.
6. Modular Scalability: To establish a flexible system architecture that facilitates future hardware expansions and software iterations.

7. Mathematical Validation: To create a rigorous simulation framework within MATLAB/Octave, enabling the analytical modeling of system performance and bottlenecks.
8. Interdisciplinary Integration: To demonstrate the practical convergence of electronics, control theory, and software development through a fully functional prototype.

By achieving these milestones, the project provides a robust and future-proof solution that bridges the gap between theoretical automation concepts and real-world industrial applications.

3. TECHNOLOGICAL STACK & FRAMEWORKS

The realization of this project is underpinned by a synergistic combination of embedded systems, high-level programming environments, and computational tools. The selected technologies ensure a robust transition from theoretical modeling to physical implementation.

3.1. The Arduino Ecosystem

The Arduino platform serves as the primary embedded controller, responsible for the low-level hardware abstraction and execution logic. It acts as the "brain" of the system, orchestrating real-time communication between RFID modules, ultrasonic sensors, and actuators. Its versatility in peripheral integration makes it an ideal choice for developing reliable automation prototypes.

3.2. Arduino Integrated Development Environment (IDE)

The Arduino IDE was utilized as the main software development kit (SDK) for firmware engineering. It facilitated the iterative process of coding, debugging, and deploying logic to the microcontroller, ensuring the firmware is optimized for real-time responsiveness.

3.3. MATLAB / GNU Octave

MATLAB and GNU Octave constitute the project's analytical and simulation framework. These tools were leveraged to:

- Develop high-fidelity mathematical models of parking dynamics.
- Process and visualize occupancy data through advanced telemetry.
- Execute hardware-independent simulations to validate system behavior before physical deployment.

3.4. Embedded C/C++ Programming

The core operational logic is engineered in Embedded C/C++. This approach was chosen to maximize hardware efficiency, providing granular control over memory management and interrupt handling, which is critical for the deterministic timing required in automated entry systems.

3.5. Human-Machine Interface (HMI) Design Tools

The Graphical User Interface (GUI) was designed using specialized toolsets within the MATLAB/Octave environment. The focus was on creating a centralized control dashboard that enhances user interaction, providing a visual bridge between the physical hardware and the operator's monitoring requirements.

4. SYSTEM FUNCTIONALITY

The operational logic of the system is based on a closed-loop interaction between hardware detection and software processing. The workflow is optimized to ensure a seamless transition from vehicle detection to data visualization.

- Automated Entry Sequence: Access is governed by an RFID-driven authentication protocol. When a tag is detected, the system validates the credentials against a pre-defined database. Upon successful authorization, the microcontroller triggers the motorized ramp actuation, synchronized with visual and acoustic feedback to guide the user.
- Dynamic Space Management: The system maintains high-fidelity awareness of parking availability through a distributed sensor array. This real-time telemetry is processed to update physical status indicators (LEDs) and the digital dashboard simultaneously, preventing entry when the capacity threshold is reached to mitigate congestion.
- Centralized Oversight (HMI): The software layer, functioning as a Human-Machine Interface (HMI), provides centralized monitoring of all system states. It enables the operator to oversee occupancy dynamics and perform hardware-independent simulations to validate system behavior under various traffic scenarios.

The architecture is inherently modular, establishing a robust foundation for future integration of advanced analytics and optimization algorithms within the MATLAB/Octave environment.

5. SYSTEM ARCHITECTURE OVERVIEW

The system is engineered using a modular, layered architecture that decouples physical hardware from operational logic. This high-level abstraction ensures system transparency, reliability, and future extensibility.

5.1. The Centralized Control Core

At the heart of the architecture is the microcontroller-based control unit, acting as the primary processing hub. This unit is responsible for orchestrating the entire system cycle: from aggregating sensor telemetry and executing RFID authentication to triggering actuation protocols. By centralizing decision-making, the architecture ensures deterministic performance in real-time parking operations.

5.2. Hardware Abstraction Layer(HAL)

The physical layer consists of a coordinated sensor array and actuator network designed for real-time environment interaction:

- Perception: Sensors provide continuous data streams on occupancy and vehicle proximity.
- Authentication: The RFID module acts as a secure gateway for identity verification.
- Actuation & Feedback: Servo-driven mechanisms manage physical throughput, while synchronized visual and acoustic indicators (LEDs/Buzzer) provide instantaneous system status updates.

5.3. Logic & Operational Framework

The control logic layer serves as the decision-making engine, governing the interaction between hardware components. This layer implements rigorous operational constraints, such as:

- Access Validation: Synchronizing RFID input with database credentials.
- Conflict Resolution: Managing edge cases, such as preventing ramp actuation when occupancy thresholds are reached or when unauthorized access is detected.
- System Synchronization: Ensuring zero-latency communication between the sensing, processing, and feedback layers.

5.4. Software Supervision & Virtualization Layer

The software layer, developed within the MATLAB/Octave environment, functions as a high-fidelity monitoring and simulation hub. This architecture allows for Hardware-in-the-Loop (HIL) testing, where system dynamics—such as response latency and occupancy throughput—can be analyzed in a virtualized setting. This ensures that operational efficiency can be validated independently of the physical prototype.

5.5. Inter-Layer Communication & Data Flow

Communication within the system is governed by a structured, bi-directional data flow:

- Upward Path: Raw telemetry from the sensor array is transmitted to the control logic for real-time processing.
- Downward Path: Command signals are dispatched from the microcontroller to the actuators and feedback mechanisms.
- Parallel Processing: The simulation environment operates concurrently, enabling the analytical evaluation of both real-world and synthetic data sets.

5.6. Scalability and Modular Extensibility

The decoupled nature of this architecture ensures a robust foundation for iterative development. The system is designed for extensibility, allowing for the integration of:

1. Advanced AI-driven occupancy optimization.
2. High-security biometric or cloud-based authorization.
3. Sophisticated data analytics for long-term parking dynamics.

This separation between physical control and software-based simulation facilitates a pre-deployment optimization phase, making the platform both an educational tool and a scalable engineering solution.

6. HARDWARE ARCHITECTURE & COMPONENTS

The physical implementation of the system is built upon a resilient hardware foundation, engineered for real-time responsiveness and modular integration. Each component is selected to ensure deterministic interaction between the environment and the control logic.

6.1. Central Processing Unit (Microcontroller)

The microcontroller acts as the centralized processing hub, orchestrating the synchronization of all peripheral devices. Its primary function is to execute the firmware logic that governs:

- Interrupt-driven sensor processing for immediate vehicle detection.
- Real-time signal generation for precision servo-motor actuation.
- Peripheral communication across the RFID and feedback modules.

By centralizing these tasks, the unit ensures low-latency execution of parking protocols, providing a stable bridge between raw data inputs and physical outputs.

6.2. RFID Authentication Module

The RFID reader serves as the system's security gateway, enabling high-speed, contactless identity verification.

- Operational Sequence: The module captures the Unique Identifier (UID) from passive tags and transmits the data to the controller for database cross-referencing.
- System Integrity: This component eliminates the need for manual intervention, providing a secure and scalable authentication layer that mimics industrial-grade access control systems.

6.3. Sensing Network (Occupancy Detection)

The system utilizes a distributed sensor array to facilitate real-time environment perception. These sensors function as the primary data source for:

- Occupancy Telemetry: Continuous monitoring of parking modules to distinguish between occupied and vacant states.
- Access Validation: Providing the critical data required to enforce capacity thresholds and prevent overflow. This high-frequency data stream allows the controller to maintain a dynamic occupancy map, ensuring the system's operational logic remains accurate and responsive.

6.4. Electromechanical Actuation (Servo Mechanism)

A precision servo motor serves as the primary actuator for the physical ramp interface. Controlled via Pulse-Width Modulation (PWM) signals from the microcontroller, the servo ensures:

- Kinematic Precision: Smooth and controlled ramp positioning for vehicle throughput.
- Logic-Driven Actuation: Automatic movement triggered exclusively by successful authentication or exit protocols. The implementation of a servo-based mechanism provides the mechanical reliability and deterministic control necessary for automated entry systems.

6.5. Visual Telemetry (LED Indicators)

The system incorporates a series of LED indicators to serve as a Human-Machine Interface (HMI) at the physical level. These indicators reduce operational ambiguity by:

- Status Encoding: Utilizing color-coded signals to represent space availability (e.g., Green/Red) and authorization results.
- System Transparency: Providing instantaneous visual feedback to both users and operators, thereby streamlining the traffic flow and enhancing overall usability.

6.6. Acoustic Feedback (Piezoelectric Buzzer)

An audible signaling unit is integrated to provide secondary operational feedback. By generating distinct acoustic alerts during authorization cycles, the buzzer ensures:

- Multi-modal Notification: Enhancing user awareness by complementing visual signals during entry/exit events.
- Error Detection: Providing immediate audible warnings in the event of unauthorized access attempts or system constraints. This redundancy in feedback mechanisms minimizes operational errors and improves the overall human-factor engineering of the system.

6.7. Power Management & Distribution

The system is supported by a centralized power distribution network designed for low-voltage stability. Key focus areas include:

- Voltage Regulation: Ensuring a constant potential difference to protect sensitive CMOS-based components from transients.
- Operational Reliability: Implementation of common grounding and filtered power lines to mitigate electromagnetic interference (EMI) and ensure deterministic hardware performance. The power architecture is optimized for high efficiency, making it suitable for both prototyping and scalable field applications.

7. SOFTWARE DESIGN ARCHITECTURE

The software framework defines the operational intelligence of the system, orchestrating the interaction between hardware perception and physical actuation. To ensure reliability, the design follows a modular approach, bifurcating the logic into two distinct functional domains: embedded control and supervisory simulation.

7.1. Embedded Firmware & Real-Time Logic

The embedded software, developed in Embedded C/C++, constitutes the system's real-time engine. It is designed for deterministic execution, ensuring that all hardware interrupts and data processing cycles occur with minimal latency.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>
#include <SPI.h>
#include <MFRC522.h>

#define I2C_ADDR 0x27
LiquidCrystal_I2C lcd(I2C_ADDR, 16, 2);
Servo rampServo;

// ----- RFID RC522 -----
#define SS_PIN   53
#define RST_PIN  49
MFRC522 rfid(SS_PIN, RST_PIN);
```

Figure 7.1. – Included Libraries and Hardware Interface Initialization

This segment imports the required libraries and initializes the main hardware interfaces: I2C for the LCD, SPI for the RFID reader, and the servo interface for ramp actuation. It establishes the software-to-hardware bridge for all core modules.

7.1.1. Execution Workflow

The firmware follows a rigorous operational sequence:

- System Initialization: Upon power-up, the controller executes a configuration routine, defining I/O architectures, establishing communication protocols, and calibrating sensor thresholds to a known baseline.

- Continuous Control Loop: Post-initialization, the system enters a high-frequency execution cycle, maintaining constant vigilance over the sensor array and authentication interface.

7.1.2. Core Functional Modules

The firmware architecture is subdivided into specialized tasks:

1. Authentication Protocol: Capturing and validating RFID UIDs against embedded logic to manage secure access.
2. Actuation Control: Executing precision PWM-based signaling for smooth servo-ramp transitions.
3. Telemetry Acquisition: Constant monitoring of the sensor array to update the real-time occupancy map.
4. Feedback Coordination: Synchronizing visual and acoustic indicators to reflect real-time system states.

By enforcing these logic-driven constraints, the software prevents concurrency issues, such as unauthorized entry or overflow scenarios, ensuring the system remains within safe operational boundaries.

7.2. Program Structure & Modular Design

The software is engineered with a modular topology to ensure high maintainability and encapsulation of functions. By decoupling specific hardware tasks from the main execution cycle, the system allows for independent debugging and seamless extensibility. The architecture is organized into the following specialized modules:

- Initialization & Configuration: Establishes communication protocols and pin mapping.
- Security & Authentication: Manages RFID data acquisition and validation.
- Perception & Telemetry: Processes raw sensor inputs into occupancy status.
- Actuation & Feedback: Controls mechanical movement and HMI (LED/Buzzer) signaling.

This modularity facilitates iterative technological advancement, enabling the integration of more complex sensors or algorithms without altering the core system framework.

```
const byte WHITELIST[][4] = {
  {0xFD, 0x1E, 0x37, 0x06},  // Kartica 1
  {0x46, 0x53, 0x19, 0x06}   // Kartica 2
};
const int WHITELIST_COUNT = 2;
```

Figure 7.2 – RFID Authorized User Definition (Whitelist)

The system uses a whitelist-based access control approach. Only RFID tags whose UID matches one of the predefined entries are considered authorized for entry. This provides a simple and reliable authentication mechanism.

7.3. Decision-Making Logic & System Intelligence

The decision-making engine represents the logical core of the parking system, utilizing a rule-based framework to evaluate multi-source inputs. The system operates on a series of conditional constraints to ensure operational integrity:

1. Logic Example: For a vehicle to gain entry, the system must satisfy a logical "AND" condition: (1) Valid RFID Authentication AND (2) Occupancy < Capacity Threshold.

2. Conflict Resolution: If any condition fails—such as an unauthorized tag or a full parking lot—the logic triggers a rejection protocol, activating visual/acoustic error feedback while maintaining the ramp in a locked state.

This deterministic logic ensures predictable behavior across all operational scenarios, mitigating the risk of manual error or system overflow.

```
// ----- PINOVI -----
const int redPins[3]   = {13, 10, 7};
const int greenPins[3] = {12, 9, 6};
const int btnPins[3]   = {11, 8, 5};

const int entryPin = 22;
const int exitPin  = 23;

// ===== PAYMENT + BUZZER + PAID LED =====
const int payPin     = 24;  // PAY button
const int buzzerPin  = 27;  // active buzzer
const int paidLedPin = 28;  // optional PAID LED

const int servoPin = 46;
```

Figure 7.3 – Pin Configuration and I/O Mapping

This part maps the microcontroller pins to system peripherals: parking status LEDs, parking-spot buttons, entry/exit sensors, payment button, buzzer, paid-status LED, and the servo pin. It defines the physical wiring logic used by the firmware.

7.4. Temporal synchronization & Communication

To maintain real-time responsiveness, the software implements precise timing control mechanisms. This ensures the synchronized operation of asynchronous hardware components, such as sensors and actuators.

- Conflict Mitigation: By managing execution cycles, the system prevents signal interference and ensures that actuator movements (ramps) do not conflict with sensor data acquisition.
- Latency Optimization: Efficient communication protocols minimize the delay between user authentication and physical feedback, providing a seamless user experience.

```
bool checkRFID() {
  // Check if new card is present
  if (!rfid.PICC_IsNewCardPresent()) {
    return false;
  }

  // Verify if the card can be read
  if (!rfid.PICC_ReadCardSerial()) {
    return false;
  }

  // Display UID on Serial Monitor for debugging
  Serial.print("UID tag: ");
```

```
  String content = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(rfid.uid.uidByte[i], HEX);
    content.concat(String(rfid.uid.uidByte[i] < 0x10 ? " 0" : " "));
    content.concat(String(rfid.uid.uidByte[i], HEX));
  }
  Serial.println();

  // Check against whitelist
  bool authorized = false;
  for (int i = 0; i < WHITELIST_COUNT; i++) {
    bool match = true;
    for (byte j = 0; j < 4; j++) {
      if (rfid.uid.uidByte[j] != WHITELIST[i][j]) {
        match = false;
        break;
      }
    }
    if (match) {
      authorized = true;
      break;
    }
  }

  // Halt PICC
  rfid.PICC_HaltA();

  return authorized;
}
```

Figure 7.4 – RFID Authentication and Access Control Function

The function detects whether a new RFID card is present, reads the card UID, and compares it against the whitelist. It returns true only if the scanned card is authorized, enabling secure entry control.

7.5. Architectural scalability & Future Extensibility

The software is engineered with a high degree of extensibility, serving as a robust foundation for future research and development. The modular design allows for the seamless integration of:

- Advanced Data Logging: Capturing long-term parking dynamics for statistical analysis.
- Network Integration: Facilitating IoT-based remote monitoring and cloud-based authentication.
- Intelligent Algorithms: Implementing AI-driven predictive occupancy models.

Furthermore, this architecture establishes a synergistic link with simulation environments, enabling the transition from physical prototypes to complex, software-based simulations for deeper performance insights.

```
void openRamp() {
  if (rampIsLowered) {
    rampServo.write(90);
    rampIsLowered = false;
    rampActive = true;
    rampLowerTime = millis() + 5000;
    updateLcd();
  }
}

void closeRamp() {
  if (!rampIsLowered) {
    rampServo.write(0);
    rampIsLowered = true;
    rampActive = false;
    updateLcd();
  }
}
```

Figure 7.5 – Ramp Actuation Control (Open/Close)

These functions control the ramp using a servo motor. When access is granted, the ramp is opened and a timer is set to automatically close it after a defined time window, ensuring safe and consistent operation.

8. SYSTEM OPERATIONAL WORKFLOW

The operational logic is governed by a sequential execution protocol that synchronizes physical detection with logical validation. This workflow is designed to optimize vehicle throughput while maintaining strict access control and financial integrity.

8.1. Automated Entry Protocol

The entry sequence is triggered by a passive RFID scan. The system executes a dual-validation cycle:

- Authorization Check: Cross-referencing the tag UID with the secure database.
- Capacity Check: Verifying that the occupancy count is below the critical threshold. If both conditions are met, the controller initiates ramp actuation, timestamps the entry, and provides synchronized visual/acoustic confirmation.

8.2. Transactional Logic (Payment Proccesing)

The system employs a time-based billing algorithm. The duration of stay is calculated as the delta between entry and payment timestamps.

- Validation: Access for exit is only granted once the payment status is flagged as "Successful" in the system logic.
- Conflict Handling: Any discrepancy in the transaction result keeps the exit barrier in a locked state to prevent unauthorized departures.

8.3. Exit Authorization & Session Reset

The exit procedure is initiated by an exit-side sensor. The system performs a final verification of the payment flag. Upon confirmation:

- The ramp is actuated for the exit duration.
- The occupancy counter is decremented in real-time.
- The parking session is archived, clearing the buffer for the next user.

8.4. Deterministic Ramp Timing

To ensure safety and mechanical longevity, ramp movement follows a deterministic timing model. The barrier remains in the "Open" state for a calibrated interval before executing an automated closure routine. This prevents potential collisions and ensures the physical security of the facility.

8.5. Algorithmic Visualisation & Virtulization

The entire workflow is mapped via logic flowcharts, serving as the blueprint for the MATLAB/Octave simulation. This allows for the virtualization of the parking dynamics, enabling stress-testing of the workflow under high-traffic scenarios.

9. GUI DESIGN & MATLAB/OCTAVE SIMULATION

The MATLAB/Octave environment serves as a high-fidelity supervisory simulation layer for the Smart Parking Lot System. It features an interactive Graphical User Interface (GUI) that encapsulates the operational logic of the embedded implementation. This software-based simulation facilitates pre-deployment optimization, enabling controlled testing and visualization of system states within a deterministic computational framework.

The GUI visualizes a 24-slot modular parking architecture through a deterministic timing model (timer-based refresh mechanism). Each node within the sensor array is represented by color-coded status indicators, distinguishing between real-time availability and user classification (Regular vs. Subscriber). Beyond spatial monitoring, the system integrates bi-directional data flow to provide dynamic pricing analytics and financial telemetry, including daily and cumulative revenue streams.

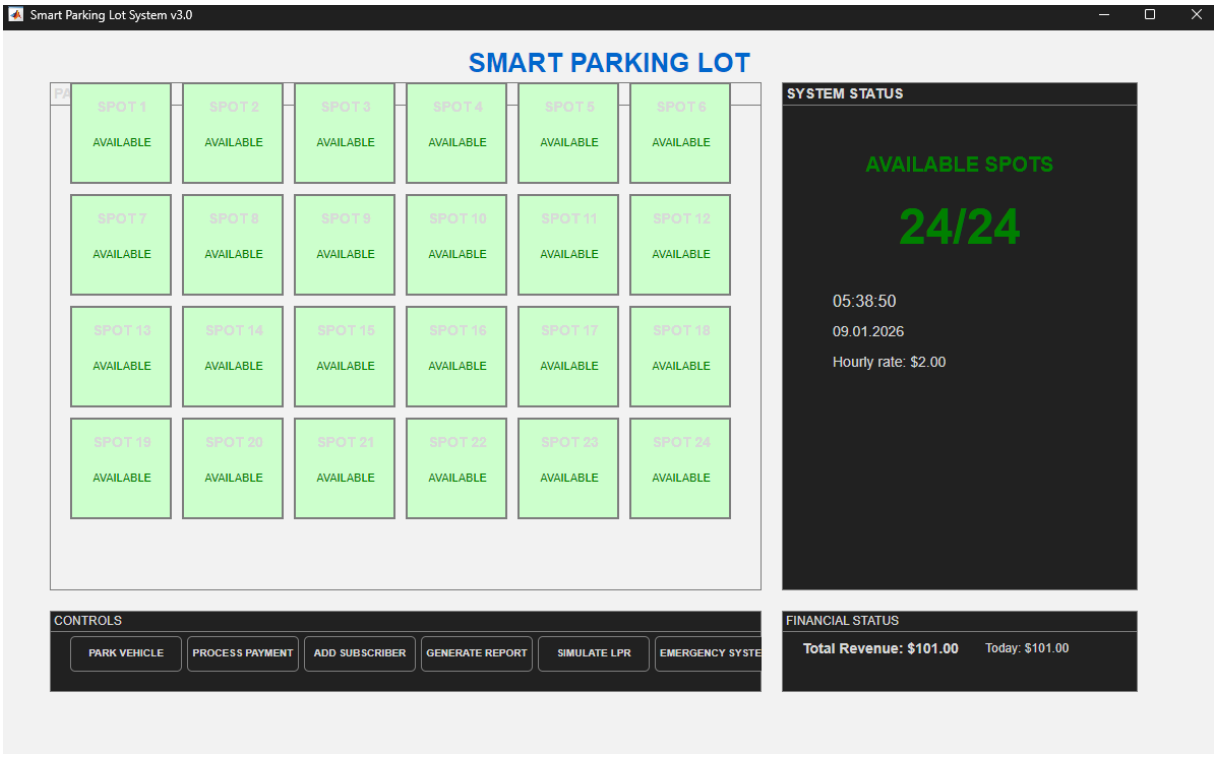

Figure 9.1 – Main MATLAB/Octave GUI overview

9.1. Core Functional Implementation

The simulation architecture is designed for modularity, comprising the following critical components:

- Dynamic Parking Allocation: Facilitates vehicle placement via an interactive dialog or direct spatial interaction, ensuring system transparency during state transitions.
- Automated Payment Processing: An integrated algorithm calculates duration-based costs and executes transactions across multiple payment protocols.
- Subscriber Management Framework: Manages lifecycle data for monthly subscribers, integrating fixed-fee structures into the broader financial model.
- Analytical Reporting: Generates structured datasets (daily/weekly/monthly) and supports persistent storage, establishing a robust foundation for advanced data analytics.
- LPR Synthesis: Simulates License Plate Recognition (LPR) to automate user classification, bridging the gap between physical detection and logical authorization.
- Emergency Evacuation Protocol: A safety-critical mechanism that overrides standard logic to reset occupancy and trigger global gate actuation, simulating a real-world safety reset.

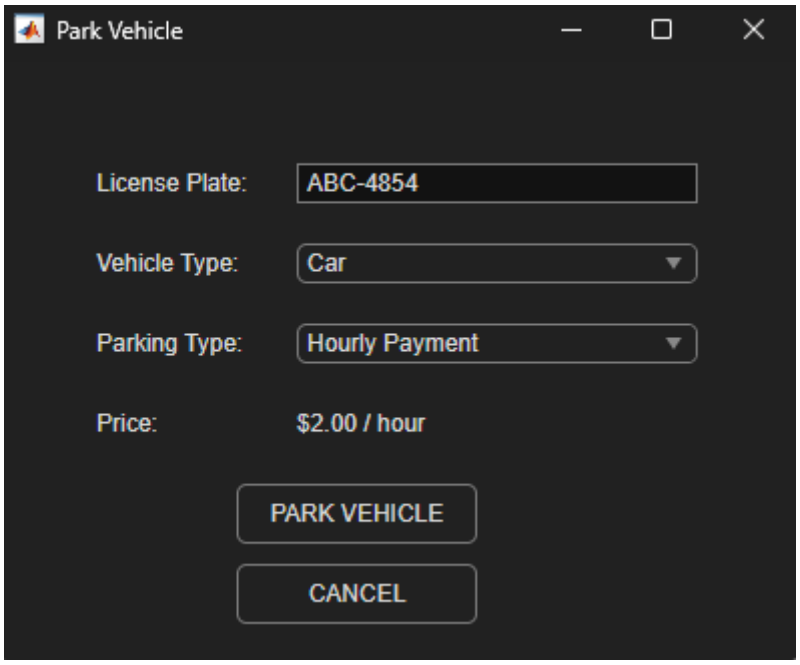

Figure 9.1.1. – Vehicle parking dialog (data entry interface)

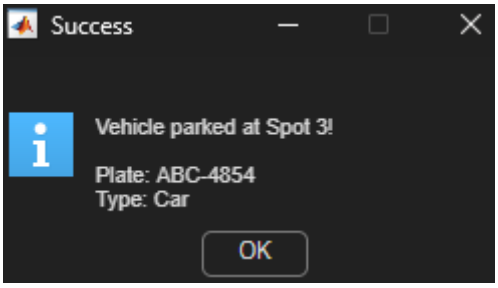

Figure 9.1.2. – Parking allocation confirmation and state update

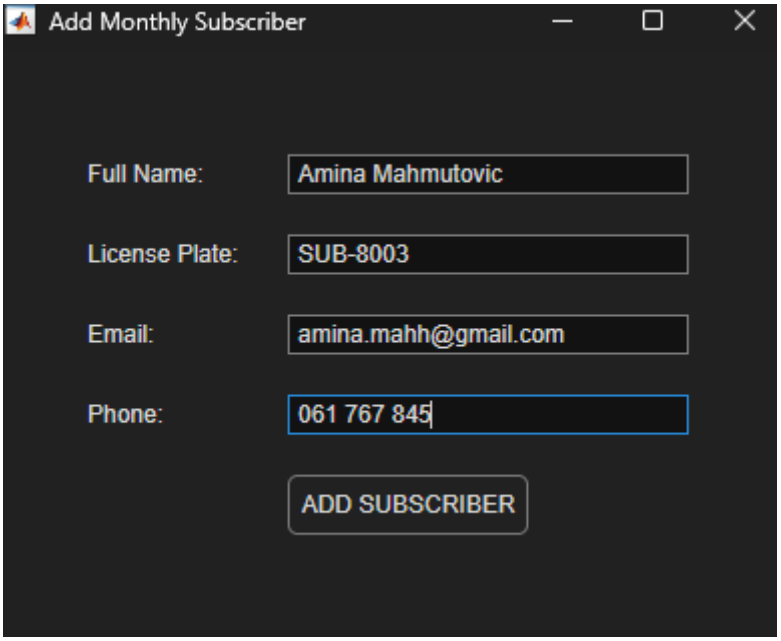

Figure 9.1.3.– Monthly subscriber registration form

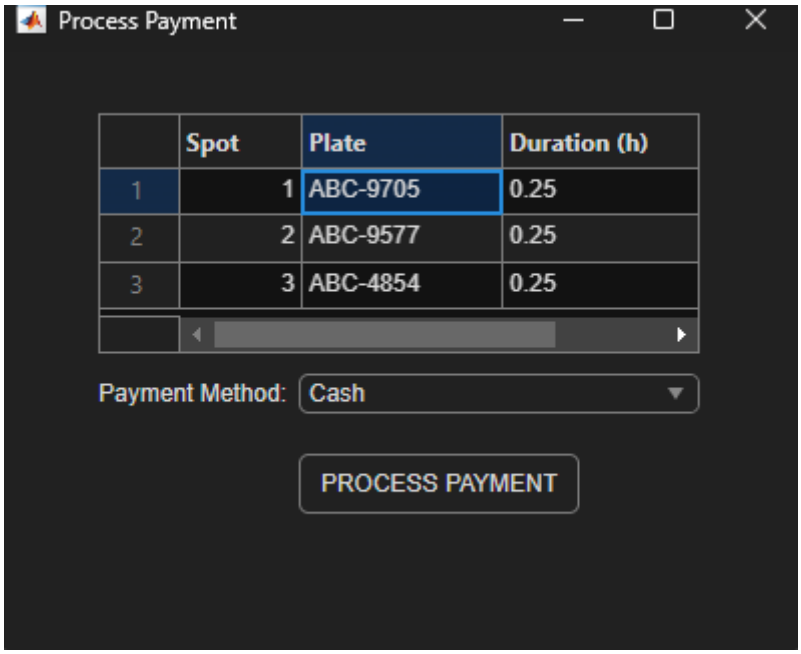

Figure 9.1.4. – Payment processing interface

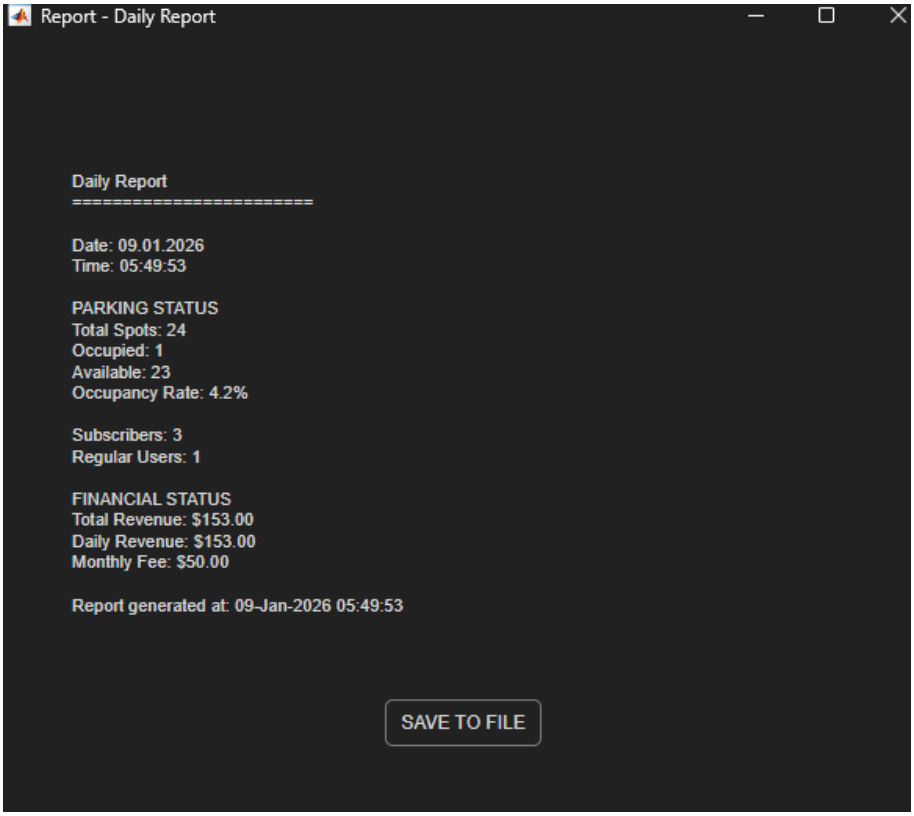

Figure 9.1.5. – Automated report generation output

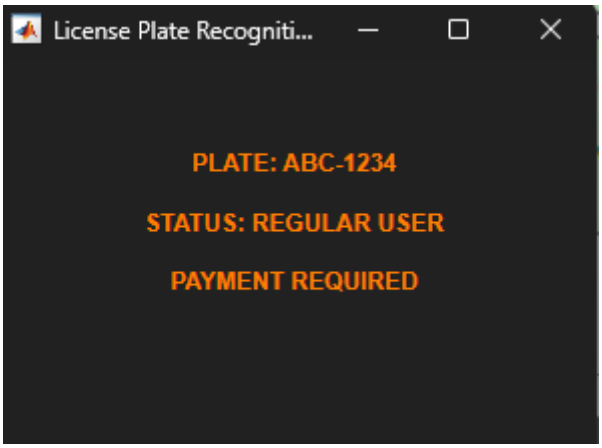

Figure 9.1.6 – License Plate Recognition simulation result

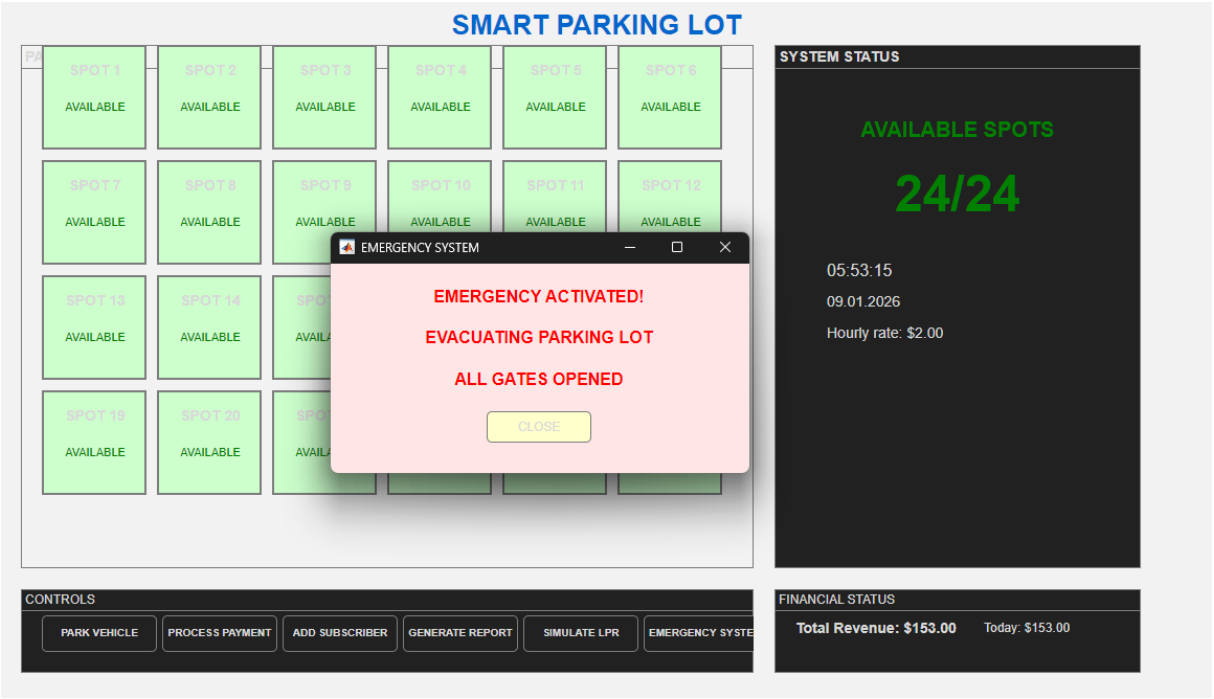

Figure 9.1.7. – Emergency mode interface

9.2. Data Architecture and State Management

The application implements a centralized data model leveraging encapsulation via the appData structure, bound to the primary UI framework. This integrated architecture ensures that critical system variables—including spatial occupancy matrices, temporal entry logs, financial parameters, and subscriber metadata—are maintained with high integrity. By utilizing a unified update routine, the system guarantees synchronous state transitions across all GUI components, mitigating data fragmentation and ensuring system transparency.

9.3. Deterministic Execution and Dynamic Pricing Logic

To ensure real-time occupancy monitoring, the system utilizes a deterministic timing model (periodic update timer) with a 1-second resolution. This mechanism synchronizes the temporal display with live telemetry of parking availability and revenue streams.

The rule-based framework for pricing supports dynamic rate adjustment predicated on multi-variable inputs, such as temporal peaks, weekend cycles, and real-time occupancy density. This practical synthesis of mathematical modeling and automation allows for the pre-deployment optimization of smart pricing strategies, effectively simulating how the system would mitigate real-world inefficiencies in a physical implementation.

9.4. Key MATLAB Implementation Snippets

Code Snippet 9.4.1. — Main Application Initialization (GUI Setup)

```
function SmartParkingLot()
    clear all; close all; clc;

    fig = uifigure('Position', [100 100 1200 700], ...
                'Name', 'Smart Parking Lot System v3.0', ...
                'Color', [0.95 0.95 0.95]);

    titleLabel = uilabel(fig);
    titleLabel.Text = 'SMART PARKING LOT';
    titleLabel.FontSize = 26;
    titleLabel.FontWeight = 'bold';
    titleLabel.FontColor = [0 0.4 0.8];
    titleLabel.Position = [400 650 400 40];
    titleLabel.HorizontalAlignment = 'center';

    appData = initializeParkingData();
    setappdata(fig, 'ParkingData', appData);

    createParkingGrid(fig);
    createStatusPanel(fig);
    createRevenuePanel(fig);
    createControlButtons(fig);

    updateDisplay(fig);
    startUpdateTimer(fig);
end
```

This snippet shows the application entry point: it initializes the GUI window, loads the system state (appData), builds all interface modules (grid, status, revenue, controls), and starts real-time updates via a timer.

Code Snippet 9.4.2. — Central Data Model + Real-Time Display Update

```
function appData = initializeParkingData()
    appData = struct();
    appData.rows = 4;
    appData.columns = 6;
    appData.totalSpots = appData.rows * appData.columns;

    appData.spotStatus = zeros(appData.rows, appData.columns);
    appData.spotType = ones(appData.rows, appData.columns);
    appData.entryTimes = NaT(appData.rows, appData.columns);
    appData.spotPrices = ones(appData.rows, appData.columns) * 2.5;

    appData.totalRevenue = 0;
```

```
    appData.dailyRevenue = 0;

    appData.baseHourlyRate = 2.5;
    appData.monthlyFee = 50;
end

function updateDisplay(fig)
    appData = getappdata(fig, 'ParkingData');
    available = sum(appData.spotStatus(:) == 0);

    appData.countLabel.Text = sprintf('%d/%d', available, appData.totalSpots);

    currentPrice = calculateCurrentPrice(appData);
    appData.priceLabel.Text = sprintf('Hourly rate: $%.2f', currentPrice);

    appData.totalRevLabel.Text = sprintf('Total Revenue: $%.2f', appData.totalRevenue);
    appData.dailyRevLabel.Text = sprintf('Today: $%.2f', appData.dailyRevenue);

    setappdata(fig, 'ParkingData', appData);
end
```

This snippet presents the system's internal data model (spot states, pricing, time tracking, and revenue) and demonstrates the real-time synchronization mechanism that updates the GUI (availability counter, dynamic price, and financial statistics).

Code Snippet 9.4.3. — Payment Processing Logic (Business Rule Enforcement)

```
function processPayment(fig)
    appData = getappdata(fig, 'ParkingData');
    occupiedSpots = {};

    for i = 1:appData.rows
        for j = 1:appData.columns
            if appData.spotStatus(i,j) == 1 && appData.spotType(i,j) == 1
                entryTime = appData.entryTimes(i,j);
                duration = hours(datetime('now') - entryTime);
                if duration < 0.25, duration = 0.25; end

                cost = duration * appData.spotPrices(i,j);
                occupiedSpots{end+1} = struct('Row', i, 'Col', j, 'Cost', cost);
            end
        end
    end

    spotInfo = occupiedSpots{1};
    appData.totalRevenue = appData.totalRevenue + spotInfo.Cost;
    appData.dailyRevenue = appData.dailyRevenue + spotInfo.Cost;

    appData.spotStatus(spotInfo.Row, spotInfo.Col) = 0;
    appData.entryTimes(spotInfo.Row, spotInfo.Col) = NaT;

    setappdata(fig, 'ParkingData', appData);
    updateDisplay(fig);
end
```

This snippet illustrates the core transactional workflow: the system calculates parking duration and payment cost, updates total/daily revenue, releases the parking spot, and immediately refreshes the user interface to reflect the new state.

10. PHYSICAL MODEL (PARKING MAQUETTE)

The physical implementation of the system is realized through a scaled, functional prototype designed to demonstrate the practical synthesis of the project's core principles. This maquette serves as a tangible platform to validate the integration of hardware components and to demonstrate the operational fidelity of the control logic in a real-world-oriented environment. The model features a modular spatial arrangement that replicates a contemporary parking facility, including dedicated parking modules, centralized entry/exit gateways, and a high-precision servo-driven ramp mechanism.

Each parking module is equipped with a distributed sensor array for real-time occupancy monitoring, complemented by integrated LED indicators and acoustic signaling units that provide immediate, intuitive feedback. The RFID interface is strategically positioned at the access control point to simulate secure identity verification. When an authorized tag is detected, the system executes the embedded firmware logic, triggering the ramp actuation and providing synchronized visual and audible confirmation. Entry and exit sensors, along with a dedicated payment trigger, allow for the full simulation of the parking lifecycle—from initial authentication to final transaction and session reset.

Beyond a conceptual demonstration, the physical prototype bridges the gap between theoretical frameworks and functional engineering. By directly interfacing with the microcontroller, the maquette ensures that all physical interactions are governed by deterministic execution, allowing for the direct observation of the rule-based logic in action. This implementation establishes a robust foundation for system transparency and pre-deployment testing, providing a controlled environment to identify and mitigate potential operational inefficiencies before further technological advancement or specialized engineering development.

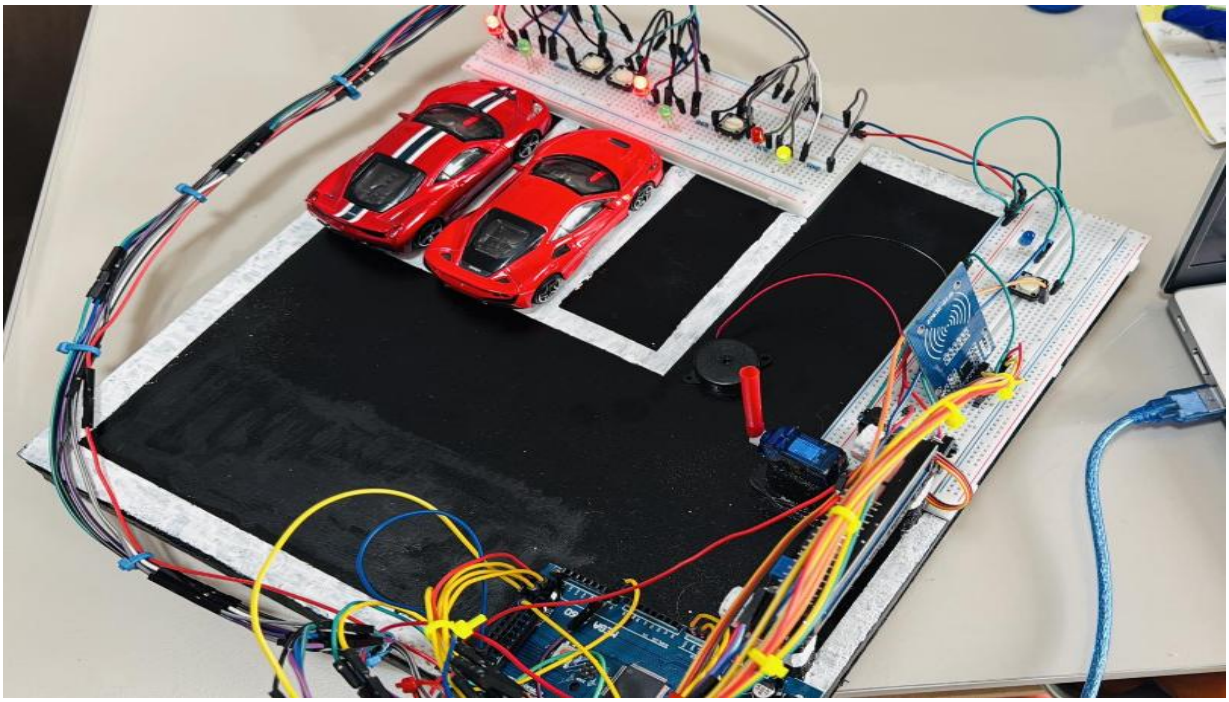

Figure 10.1. – Physical model of the automated parking syste I

This figure illustrates the complete physical parking maquette, including the entry ramp mechanism, parking layout, and integrated electronic components. The model demonstrates the real-world placement of sensors, actuators, and control hardware used to simulate vehicle entry and parking operations.

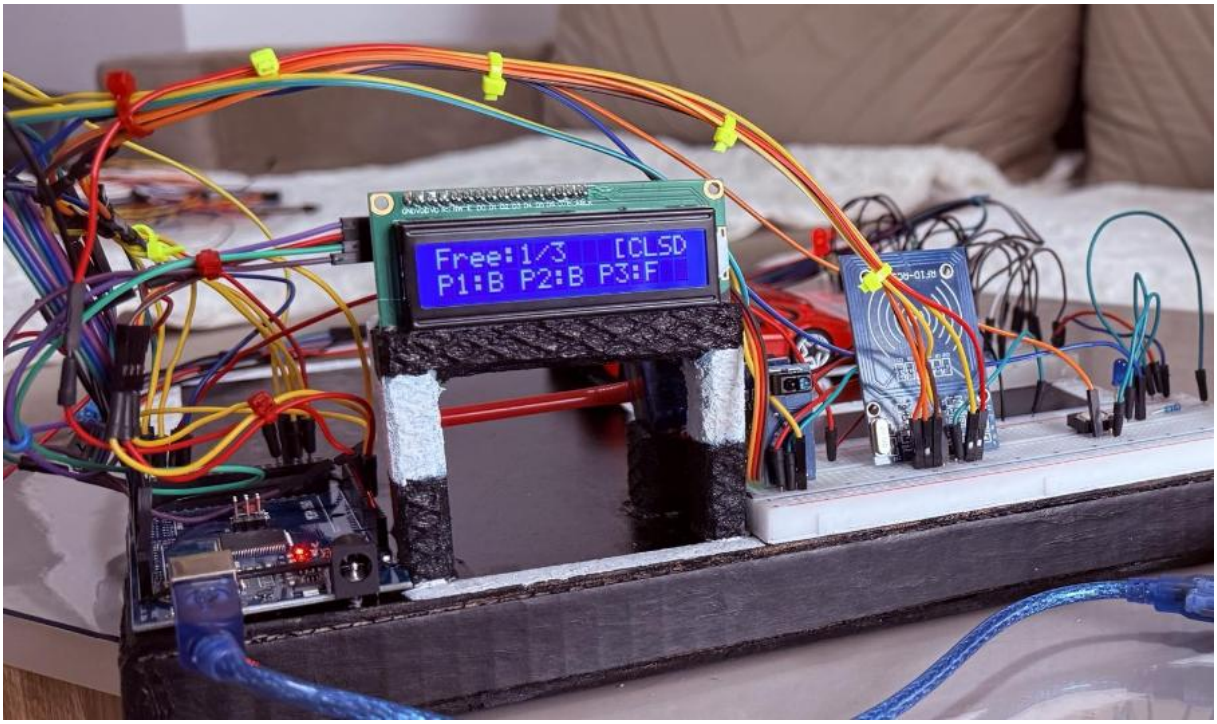

Figure 10.2. – Hardware implementation and system status display

This figure presents a detailed view of the hardware implementation, highlighting the LCD status display, RFID reader, and wiring architecture. The display provides real-time information on parking availability and ramp status, confirming correct system operation during testing.

11. TESTING AND SYSTEM VALIDATION

This section outlines the rigorous verification protocols applied to evaluate the functionality, reliability, and logical consistency of the automated parking management system. Testing was conducted through a dual-track approach, assessing both the physical prototype and the computational workflow to ensure full compliance with the engineering requirements.

11.1. Functional Verification

Functional testing was executed to validate the operational integrity of individual hardware modules and their integrated performance. The testing phase focused on the following critical parameters:

- Security & Authentication: Validating the RFID processing logic and the precision of the access authorization database.
- Electromechanical Response: Ensuring the deterministic actuation of the servo-driven ramp and its automated closure routines.
- Sensing Accuracy: Calibrating the distributed sensor array for real-time occupancy monitoring and visual status synchronization.
- HMI Feedback: Verifying the consistency of the acoustic alerts and visual indicators (LCD/LED) across various system states. All core components demonstrated operational stability, maintaining expected performance levels under standard environmental conditions.

11.2. Workflow Validation & Logic Stress-Testing

The comprehensive system lifecycle was validated by simulating high-fidelity operational scenarios, including peak-traffic entry, variable parking durations, and transactional processing. The validation focused on the enforcement of operational constraints and edge cases, such as:

- Congestion Mitigation: Confirming the automatic rejection protocol when the parking capacity threshold is reached.
- Financial Integrity: Verifying that the exit barrier remains in a locked state if the payment status is not flagged as "Successful." The results confirmed that the system maintains a coherent state-machine architecture, consistently applying the defined ruleset and ensuring a reliable, autonomous parking environment.

11.3. Edge Case Assessment & Fault Tolerance

To evaluate the system’s robustness, edge case testing was conducted under critical and non-standard operational conditions. These scenarios were designed to challenge the rule-based framework and ensure system stability during:

- Security Breaches: Attempted access using unauthorized or corrupted RFID identifiers.
- Capacity Overflows: Vehicle entry requests initiated while the system is at maximum occupancy.
- Protocol Violations: Premature exit attempts bypassing the transactional verification stage.

- Rapid Interaction Cycles: Successive, high-frequency user inputs intended to trigger potential latency issues. In every scenario, the system demonstrated high fault tolerance, responding with immediate rejection protocols and clear multi-modal feedback, thereby maintaining operational integrity and preventing unauthorized state transitions.

11.4. Summary of Validation Result

The comprehensive testing phase confirms that the automated parking management system fulfills its functional and engineering objectives with a high degree of reliability. The results validate the synergy between the hardware and software layers, ensuring a predictable and secure operational environment. No critical failures or logical discrepancies were observed, establishing the prototype as a functional and scalable platform capable of addressing tangible real-world parking inefficiencies.

12. LIMITATIONS AND FUTURE IMPROVEMENTS

While the automated parking management system successfully achieves its primary functional objectives, acknowledging its current constraints is essential for establishing a robust foundation for future research and specialized engineering development.

12.1. System Constrains

The current implementation faces certain limitations necessitated by the prototype-scale development. The physical model is restricted to a finite number of parking modules, which may not fully encapsulate the complexities and traffic dynamics of large-scale commercial facilities. From a hardware standpoint, the system operates as a standalone unit without network connectivity, thereby lacking support for real-time remote monitoring or centralized data management. Furthermore, while the payment logic is functionally sound, it remains a simplified simulation rather than a fully integrated financial transaction gateway.

12.2. Future Directions and Scalability

To transition this prototype into a scalable, real-world-oriented system, several iterative advancements are proposed:

- IoT & Network Integration: Incorporating wireless communication modules to facilitate cloud-based monitoring and bi-directional data flow for remote system supervision.
- Advanced Perception & Automation: Replacing manual triggers with sophisticated sensor arrays and automated vehicle detection technologies to enhance operational realism.
- Intelligent Algorithms: Moving beyond basic rule-based logic to include advanced data analytics and AI-driven predictive modeling for parking occupancy dynamics.
- Extended Simulation Framework: Deeper integration with the MATLAB/Octave environment to conduct high-fidelity performance analysis and pre-deployment optimization.

By addressing these limitations, the system can evolve from a conceptual prototype into a versatile platform capable of mitigating contemporary urban parking inefficiencies through iterative technological advancement.

13. CONCLUSION

This project successfully achieved the design and functional implementation of an automated parking management system, representing a synergistic combination of embedded hardware control and software-based simulation. The developed solution serves as a practical synthesis of core principles from electronics, automation, and software engineering, translated into a coherent, working prototype.

By integrating RFID-based authentication, sensor-driven telemetry, and deterministic ramp actuation, the system establishes a secure and efficient framework for parking occupancy management. The physical maquette further validates the project's feasibility, bridging the gap between theoretical design and tangible real-world application. This dual approach—combining a functional prototype with a MATLAB/Octave simulation framework—enhances system transparency and provides a versatile platform for both physical testing and analytical performance evaluation.

The adopted modular architecture ensures that the system remains scalable and adaptable, supporting future iterative advancements such as IoT integration and advanced data analytics. In summary, the project successfully fulfills its intended objectives, demonstrating how theoretical knowledge can be synthesized into a scalable engineering solution. This automated system not only addresses contemporary parking inefficiencies but also establishes a robust foundation for further research into intelligent infrastructure and smart city development.

14. REFERENCES

Bowman, S., Park, S., Martin, F., & Ohland, M. (2020). *Thinking Like an Engineer: An Active Learning Approach* (5th ed.). Pearson.

https://www.researchgate.net/profile/Laurita-Santos/publication/383466567_Programming_logic_through_robotics_Use_of_Scratch_and_Arduino_for_robot_creation_and_interactive_projects/links/68c6cebdf3032e2b4be11c83/Programming-logic-through-robotics-Use-of-Scratch-and-Arduino-for-robot-creation-and-interactive-projects.pdf

https://link.springer.com/article/10.1007/s10291-014-0370-z

AVR Atmel - Steven F **Barrett**, Daniel **Pack**, Mitchell Thornton, 2008 – Springer

https://ieeexplore.ieee.org/abstract/document/9345329

https://ieeexplore.ieee.org/abstract/document/9319817