Menoufia university
Faculty of Engineering
Electrical Engineering Department

Smart Parking System

B.Sc. Graduation Project 2025

By

Abdelrahman Ahmed Kamal Bayoumi
Ahmed Fathy Kamal Eldesokey
Kerolos Mina Mikahail Soliman
Manar Hosney Mohamed Abd-Elhafez
Maruim Farid Saeed Elkhooly
Menna Allah Mamdouh Mostafa Sakr
Mohamed Mahmoud Magdy Hashad
Mostafa Eid Mostafa Zayed
Mostafa Mohamed Ramadan Mahgoup
Nawal Wageh Abdelmoneim Helal
Saleh Atef Elsotohy Saleh

Supervisor

Prof. Tamer Fetouh

Acknowledgment

We would like to express our heartfelt gratitude to Professor Tamer Fetouh, our supervisor, whose guidance, expertise, and unwavering support have shaped our research and ensured the success of this graduation project book. His mentorship and valuable feedback have played a pivotal role in our academic growth.

Our fellow team members deserve special recognition for their dedicated collaboration, insights, and shared enthusiasm. Their contributions have been instrumental in overcoming challenges and achieving our goals.

We express our appreciation to the participants who generously shared their time and insights, providing valuable data and perspectives for our study.

To our friends and loved ones, we are grateful for their emotional support and encouragement throughout this journey.

Finally, we acknowledge and thank our families for their unwavering support, love, and understanding, which enabled us to pursue our academic goals.

In conclusion, we extend our sincere thanks to all who have contributed to the successful completion of this graduation project book. Your support, guidance, and contributions have been integral to our achievements, and we are honored to have worked with such incredible individuals.

Abstract

This project explores the design, construction, and integration of a multi-level smart parking system, a cutting-edge solution aimed at addressing the growing need for efficient and space-optimized urban parking. The study begins with an introduction to history and various types of automated parking systems, such as Puzzle, Shuttle, Silo, Crane, Vertical Rotary, and Tower Parking Systems, highlighting their evolution and specific advantages.

The mechanical design phase focuses on creating a robust prototype, discussing key components, the selection of NEMA Stepper motors, and power circuit considerations. This is followed by a comprehensive examination of electrical design, including the differences between DC and Stepper motors, motor sizing, and the integration of power circuits and PCBs to ensure reliable operation.

Embedded system development is detailed through a structured approach, from planning and requirements analysis to system design, development, testing, and deployment. The project integrates wireless charging technology and Internet of Things (IoT) solutions to enhance functionality and user experience. Detailed discussions on wireless charging history, processes, and IoT applications underscore the project's innovative and sustainable approach.

The development of a mobile application, using MAUI and Azure Database, ensures user-friendly interface and robust data management, facilitating seamless operation of the smart garage. The project demonstrates technical proficiency and innovative thinking, paving the way for future advancements in automated parking solutions and contributing to a more efficient, sustainable, and user-centric urban environment.

Introduction

The development and implementation of automated parking systems represent a crucial step in enhancing modern urban infrastructure, as cities worldwide face increasing challenges related to limited space and traffic congestion. This project aims to design and implement a multi-level smart parking system characterized by high efficiency and operational accuracy, relying on advanced technologies in control, monitoring, and communication.

The project focuses on building a fully integrated mechanical system powered by high-precision *stepper motors, which were specifically selected for their accuracy and reliable motion control. **DC motors* were intentionally avoided to eliminate issues related to unstable speed and positioning, especially in complex, multi-tasking environments such as automated parking systems.

Advanced tools and platforms such as *Node-RED* were used to implement the control logic and interconnect various components like motors and cameras using flexible and easily adjustable communication protocols. Supervisory control and monitoring were achieved using *SCADA* and *HMI* systems, providing operators with real-time feedback and control over the system’s operation.

To further enhance the system’s precision and reliability, *machine vision* technology was integrated. A *camera* was used as the primary source of feedback to monitor vehicle positions, detect presence, and ensure proper alignment within the system. The entire system depends on *image-based processing*, eliminating the need for traditional sensors and allowing for a more streamlined and intelligent decision-making process.

In the future, the system can be expanded to include new features such as *online reservation capabilities, **real-time vehicle counting, or **integration with electronic payment systems*, which would further improve user experience and operational efficiency.

In conclusion, this project demonstrates effective integration between mechanical and electronic systems and serves as a practical model for smart parking technologies capable of addressing the challenges of modern urban environments. It reflects the ability of engineering students to develop and implement advanced technical solutions using real-world tools and methods, paving the way for a more organized and efficient future in urban infrastructure.

Components

Project construction

2.1. Mechanical Design

2.1.1. Inspiration:

Through research and studies on the basic parts of the smart garage, its uses, and the codes that control its design, and with the help of smart garage design programs. This survey helps us to know and understand the main parts necessary to be available in the smart garage, which make the system achieve the specifications.

2.1.2. Main parts of the smart garage

- **The External Frame**: is fabricated from a combination of acrylic and wood materials, carefully crafted to precise dimensions. The frame's design incorporates deliberate diminutions, ensuring a sturdy and robust structure that provides a solid foundation for the entire system. The selection of acrylic and wood materials offers a durable and aesthetically pleasing solution, while the precise diminutions guarantee a snug fit for the various components, thereby enhancing the overall stability and functionality of the project.

Diminutions	Height	Width	Depth
Values(cm)	100	80	25

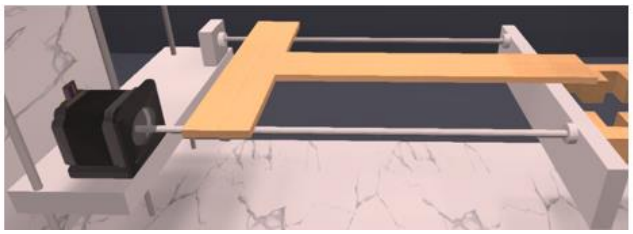
- **Project Base:** is a critical component of the prototype, serving as the foundation that supports the entire system. Fabricated from acrylic, the base plate provides a sturdy platform for the assembly of various mechanical components, ensuring stability and rigidity throughout the operation of the device. are carefully designed to maintain a compact footprint.

Diminutions	Thickness	Width	Depth
Values(cm)	2	90	40

- **Car lifter:** It is used to carry cars in front of the gate to raise them to one of the floors

when the car enters, and to carry the car from the floor to the gate when the car exits. Moving through a servo motor.

Diminutions	Width	Depth
Values(cm)	25	25



- **Floors:** The system consists of three floors. The distance between each floor is 25 cm, each floor carries only one car. When the floor is empty, it gives a signal to the system that the floor is empty, and when it is occupied, it gives a signal to the system that the floor is occupied.

Diminutions	Width	Depth	Thickness
Values(cm)	25	25	2

- **A leadscrew:** also known as translation screw, is a screw used as a linkage in a machine, to translate turning motion into linear motion. Because of the large area of sliding contact between their male and female members, screw threads have larger frictional energy losses compared to other linkages. They are not typically used to carry high power, but more for intermittent use in low power actuator and positioner mechanisms. Leadscrews are commonly used in linear actuators, machine slides (such as in machine tools), vises, presses, and jacks. Leadscrews are a common component in electric linear actuators.

Diminutions	Length	Diameter
Values(cm)	100	0.8

Figure 2.1 Lead screw

- **Linear guide:** Linear guide: a machine element that enables linear motion while bearing loads, allowing machines to operate precisely and efficiently. It consists of three main components: a mobile carriage, a rail that supports the carriage's movement, and balls that enable linear motion through a recirculating mechanism.

Diminutions	Length	Diameter
Values(cm)	30	0.8

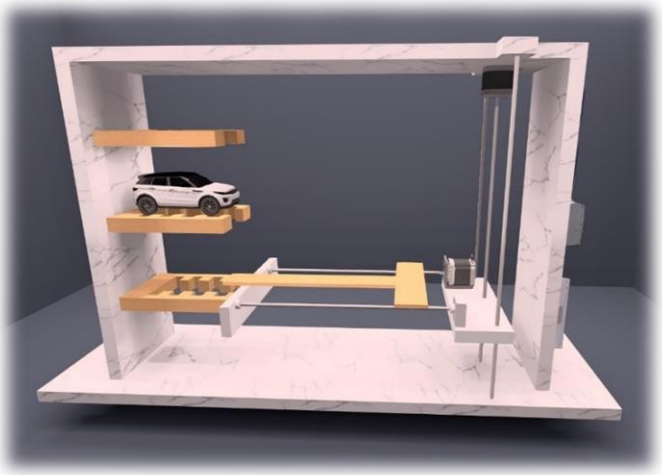
Figure 2.2 Linear guide

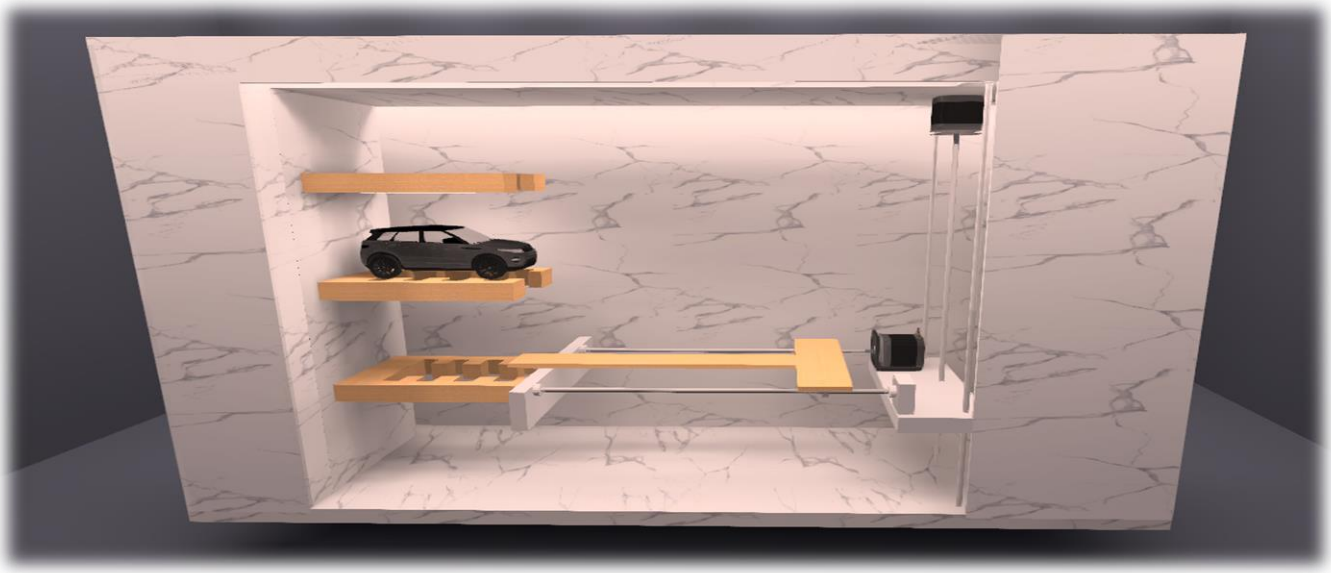
- **Bearing**: a device that bears the load and reduces the friction between two parts within a moving system, allowing for smooth motion with minimal resistance

- **Coupling**: a device that connects a motor shaft to a leadscrew, enabling the motor to rotate the leadscrew and move a load. It allows for the transmission of rotational motion from the motor to the leadscrew, and comes in different types, such as rigid and flexible couplings, to accommodate various motor shaft and leadscrew diameters.

Figure 2.3 coupling

2.2Prototype





2.2. Electrical Components

2.3.1 Programmable Logic Controller (PLC):

A Programmable Logic Controller (PLC) is an industrial-grade digital computer specifically designed for controlling manufacturing processes, electromechanical systems, or robotic devices. It is at the heart of many automation solutions due to its durability, flexibility, and ease of programming.

2.3.2 PLC Architecture:

A typical PLC consists of the following elements:

- **Central Processing Unit (CPU):** the brain of the PLC, executing the user program and processing input/output signals.
- **Memory modules:** store the user application program, configuration parameters, and system data.
- **Power supply**: provides the necessary DC voltage to the CPU and other internal circuits.
- **Input modules**: receive signals from field devices such as sensors and switches, converting them into logic levels for the CPU.
- **Output modules**: send signals to actuators, relays, or motor drivers based on the logic processed by the CPU.
- **Communication interfaces**: allow the PLC to exchange data with HMIs, SCADA systems, or other PLCs in networked environments.

Advantages of PLCs:

PLCs offer several significant advantages over traditional relay-based control systems:

- High reliability in harsh industrial environments.
- Ease of modification and reprogramming.
- Reduced wiring complexity.
- Faster troubleshooting and diagnostics.
- Flexible communication capabilities with other industrial devices.

PLC Operation:

The PLC operates in a continuous scan cycle, which typically includes these stages:

1. Reading inputs.
2. Executing the user-defined logic.
3. Updating outputs.
4. Performing internal diagnostics and communication.

This cyclic execution ensures the control system responds in near real-time to changing field conditions, maintaining safe and efficient operation.

PLC Role in the Automated Parking System:

In an automated parking project, the PLC serves as the central controller coordinating all system elements. It:

- Monitors input signals from parking sensors (e.g., vehicle presence, obstacle detection).
- Determines parking slot availability and positioning.
- Controls the electric motors that drive the movement of platforms or lifts.
- Ensures safety interlocks to prevent collisions or unsafe operations.
- Interfaces with the HMI to provide real-time feedback and user commands.

Selected PLC for the Project

For the automated parking system in this project, the chosen PLC is the **Siemens SIMATIC S7-1200 CPU 1212C DC/DC/DC**, with part number **6ES7 212-1AE40-0XB0**. This PLC model was selected due to its balance between compact size, sufficient processing power, and versatile I/O capabilities, making it highly suitable for small to medium-scale automation projects like automated parking.

Key features of the S7-1200 CPU 1212C DC/DC/DC include:

- **Integrated digital inputs/outputs:** 8 digital inputs and 6 digital outputs built in, reducing the need for separate I/O modules in small systems.
- **Flexible expansion options**: allow additional I/O modules, communication modules, and signal boards to be added to suit project needs.
- **24V DC power supply** :uses 24V DC for powering the CPU and its logic circuits, which is standard for industrial environments.
- **DC outputs:** transistor outputs capable of driving 24V DC loads, suitable for switching signals to motor drivers or relays.
- **High-speed processing:** supports fast signal acquisition and processing for real-time control of motors and sensors.
- **Integrated communication interfaces:** includes PROFINET for easy networking and HMI connectivity.

The S7-1200 family is well-supported with Siemens TIA Portal software, enabling efficient programming, simulation, and troubleshooting. Its compact design also saves panel space, an important consideration for the constrained physical layout of parking control cabinets.This choice provides a robust, scalable, industry-standard control platform that meets the functional and safety requirements of the automated parking application.

2.2 Human Machine Interface (HMI)

The Human Machine Interface (HMI) is a critical component in modern industrial automation systems. It acts as the communication bridge between human operators and automated machinery, providing a user-friendly graphical interface to monitor, control, and supervise processes in real time. HMIs display process variables, alarms, and system statuses, while also allowing operators to issue commands, adjust parameters, and respond to abnormal conditions.

An effective HMI improves operator efficiency, enhances situational awareness, and contributes to the overall safety and reliability of the automated system. Modern HMIs typically feature color displays, touchscreen capabilities, and support for advanced communication protocols, making them essential in industrial environments such as automated parking systems.

2.4.1 Selected HMI for the Project:

For this automated parking project, we have selected the SIMATIC HMI TP 177B PN/DP-6 CSTN, part number 6AV6 642-0BA01-1AX1, manufactured by Siemens. This HMI was chosen due to its proven industrial reliability, compatibility with the selected Siemens S7-1200 PLC, and its rich feature set that supports both PROFINET and PROFIBUS DP communication.

The TP 177B offers a 6-inch color CSTN display with touchscreen functionality, providing clear visualization of process data and alarms while enabling intuitive operator interaction. Its compact size is well-suited for control panels with limited space, and its robust construction makes it capable of withstanding the industrial environment of an automated parking system.

2.4.2 Operation:

The main operational features of the TP 177B HMI include:

1. **Display and Touchscreen**

- The 6-inch CSTN color display ensures high-quality, vibrant visualization, allowing operators to clearly interpret critical system information.
- The integrated touchscreen interface supports intuitive interaction using taps, gestures, and swipes, enhancing usability and minimizing training requirements for operators.

2. **Functionality**

- **Monitoring:** The HMI continuously monitors real-time data such as process variables, equipment statuses, and alarms. It presents this information in a clear and structured format, enabling operators to gain a comprehensive overview of the system.
- **Control:** Operators can interact with the touchscreen to execute control commands, such as starting or stopping processes, adjusting setpoints, or acknowledging system events. This allows active management and fine-tuning of system operations.
- **Alarm Management:** An integrated alarm management system alerts operators to critical events or abnormal conditions. The HMI enables operators to view, acknowledge, and respond to alarms quickly, promoting safe and efficient system recovery.
- **Communication:** The TP 177B supports standard industrial protocols including **PROFINET** and **PROFIBUS DP**, ensuring seamless communication with PLCs and other devices across the automation network for coordinated control and data exchange.

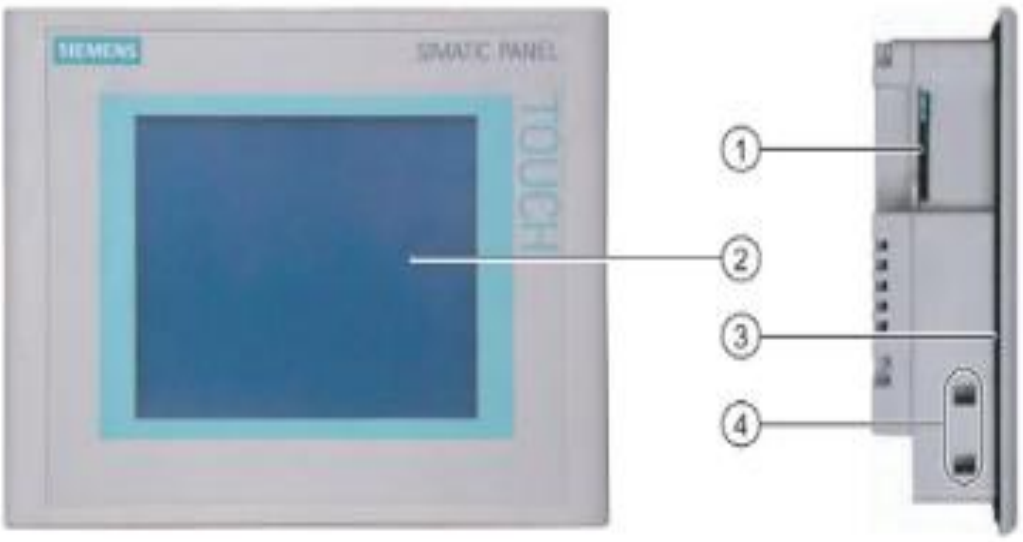
3. Configuration and Customization

- The HMI can be flexibly configured to match the requirements of the parking system. Operators or engineers can customize screen layouts, select different visualization elements, and adjust settings according to the application's demands or user preferences.

4. Diagnostics and Maintenance

- Integrated diagnostic tools help operators and maintenance personnel identify system faults, monitor performance, and troubleshoot issues efficiently. This capability reduces system downtime and improves the long-term reliability of the automated parking solution.

Front view and side view



① Slot for a MultiMedia card
② Display / touch screen
③ Mounting seal
④ Mounting clamp recess

2.5 Motors:

Introduction

Electric motors transform electrical energy into mechanical motion by having a rotor revolve around a fixed axis. These adaptable gadgets are the engine of many different applications, such as 3D printers, smart locks, and elevators. Understanding the distinctions between various motors is essential for engineers as well as enthusiasts since each motor affects not only the final application but also the choice of motor driver. Stepper motors and DC motors are the two common motor types that will be discussed in this chapter along with their corresponding motor drivers. It will also introduce stepper motor drivers and DC motor drivers that may be used for smooth control and optimization, and it will discuss the advantages and disadvantages of various motor types.

Differences between DC and Stepper motors

DC motors:

A DC motor uses electromagnetic induction to transform electrical energy into mechanical motion. The rotor of these motors is moved by the magnetic field created between its spinning and stationary parts, which in turn rotates the motor. Brushless DC (BLDC) motors and brushed DC motors are the two primary varieties of DC motors. A brushed DC motor is depicted in Figure 2.7, and the following is a description of the motor's major parts:

- **Rotor**: A wire coil wound around a core is the rotor, also known as armature. It is the revolving part of a brushed DC motor that is attached to the stator, just like the rotor of a stepper motor.
- **Stator**: The stator is made up of one or more electromagnets, or permanent magnets, and is stationary. When the rotor rotates, the stator's magnetic field interacts with the rotor's magnetic field to produce torque.
- **commutator**: A ring fixed on the rotor shaft serves as the commutator. The rotor's windings and commutator are electrically coupled. The motor may move because the commutator can change the direction of current flow in the rotor's windings.
- **Brushes**: The stationary carbon or graphite blocks that brush against the commutator are called brushes. The motor can run because of the electrical current they conduct.

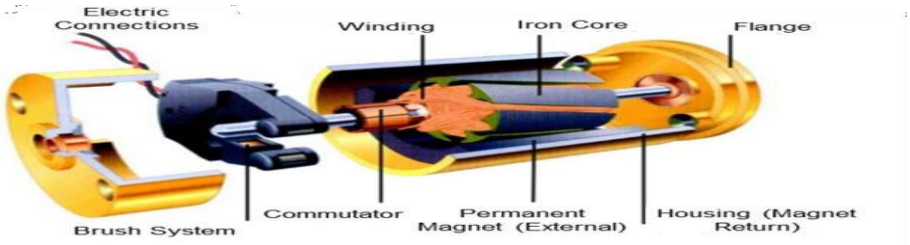


Figure2.1 brushed DC motor

There is a BLDC motor in Figure 2.8 Because BLDC motors don't have brushes, they usually have less wear and tear and are more dependable. BLDC motors feature a rotor with permanent magnets, sometimes known as electromagnets, and a stator with many coils. They use a controller to regulate the current flowing through the stator windings and electronic commutation.

Torque is produced by the applied current in DC motors. For devices like sun tracking systems, toys, and computer hard drives, DC motors are advised.

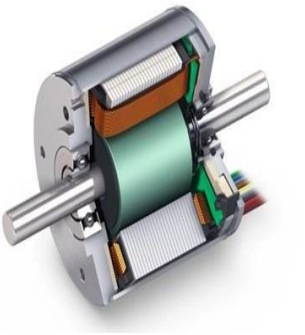


Figure2.11 BLDC DC motor

Stepper motors

Electrical pulses are converted into precise mechanical motion by a stepper motor. Stepper motors function in distinct steps, as the name suggests, with each step representing a precise angle of rotation (usually around 1.8°). Depending on the quantity of electrical pulses they have received, stepper motors rotate a predetermined number of times. Stepper motors are very controlled since every rotation adheres to the precise angle of rotation.

A stepper motor's principal parts are:

- **Rotor:** The revolving part of a stepper motor is the rotor, which is attached to the shaft. When the rotor's teeth or magnetic poles come into contact with the stator, motion is generated.
- **Stator**: The motor's stationary portion is called the stator. Wire coils in the stator that are capable of producing magnetic fields are arranged into groupings known as phases.
- **Phases of winding**: Both unipolar and bipolar stepper motors are available; unipolar stepper motors have four winding phases, while bipolar stepper motors have two. On the stator, one winding corresponds to each phase.
- **Pulses and control**: In order to rotate the stepper motor, a series of electrical pulses must be delivered to the winding phases. The timing and sequencing of these pulses determines the direction and step length of each step.

Stepper motors travel in distinct steps, which makes them extremely precise. Furthermore, they were engineered to optimize their holding torque, necessitating the maintenance of a maximum current and rendering them perfect for position-keeping applications like camera gimbals and robots.

Comparing Stepper Motor Drivers and DC Motors

Different electric motor types, such as stepper motors, brushed DC motors, and BLDC motors, each have benefits and drawbacks that make them suitable for particular uses. A handful of these most significant variations are discussed below.

1. **Function and Controllability**

Stepper motors can function in an open-loop system, in which case the specific number of steps or pulses delivered to the motor determines its exact location. Stepper motors don't require position control because they work in distinct, easily quantifiable steps. Stepper motors do, however, require an external component to change the motor's

direction and speed, such as a microcontroller (MCU). A DC power source coupled to the rotor by carbon brushes powers brushed DC motors.

More complex motors would need feedback systems, while simple brushed DC motors can be managed with an open-loop system. These motors are usually easily adjustable and don't require external controllers. BLDC motors must operate in closed-loop systems, which provide high precision but require additional control circuitry for smooth operation.

2. **Lifecycle**

Stepper motors are very dependable and have a long lifespan of up to 4 to 5 years, or 10,000 hours, thanks to their simplicity. While DC motors are also quite dependable, brushed DC motors need ongoing maintenance to avoid brush failure. Typically, brushed DC motors last for a few thousand hours before requiring maintenance. Because brushless DC motors may run for more than 10,000 hours and do not suffer from the same mechanical wear and tear caused by the brushes, they have a longer lifespan than brushed DC motors.

3. **Efficiency and Noise**

Stepper motors use a lot of energy since they always run at their full current and lose

Motor	Advantages	Disadvantages	Applications
Stepper Motor	• High accuracy • High precision • Easy to control • Long lifespan (10,000 hours)	• Less efficient • Requires external control (microcontroller) • Noisy	• 3D printers • Telescope • Disk drives • Robotics
Brushed DC Motor	• Moderate efficiency • Faster response time • Can detect overload conditions	• Shorter lifespan; require maintenance to ensure reliability • Complex control	• Electric tools/appliances • Automotive (e.g. windshield wipers) • Toys • Fans
BLDC Motor	• High efficiency • Requires little maintenance • Quiet • Very long lifespan (10,000+ hours)	• Complex control • Susceptible to extreme temperatures	• Electric vehicles • Household appliances • Medical devices (e.g. infusion pumps, imaging)

energy through heat dissipation. As a result, they are typically less efficient. Because there is less energy lost due to brush friction, DC motors are more efficient overall. Brushless DC motors are the most efficient type of DC motor. Because stepper motors have discrete steps, which make a whirring or ratcheting sound while the motor is rotating at a steady ace, they generate the greatest noise. Although brushed DC
motors make less noise, noise is still produced as the brushes brush over the commutator; BLDC motors, as predicted, produce the least noise.

NEMA Stepper motors

NEMA stepper motors are becoming increasingly used in the motion control industry because of their remarkable accuracy, dependability, and user-friendliness. Numerous industries, such as robotics, automation, 3D printing, and CNC machines, use these motors. For engineers and researchers working in these fields, it is essential to comprehend the nuances of NEMA stepper motors.

Working Principle

The fundamental idea behind how NEMA stepper motors work is the conversion of electrical pulses into mechanical motion. These motors are renowned for their capacity to precisely produce motion or angular displacement by sequentially energizing the stator's coils. A NEMA stepper motor usually has a permanent magnet for the rotor and several electromagnetic coils for the stator.

Key Characteristics

NEMA stepper motors produce discrete motion that is incremental in nature. Every step has a corresponding rotational angle, which is usually stated in steps per revolution or degrees. The step angle varies based on the particular NEMA motor type and is dictated by the motor's design and manufacture.

NEMA stepper motors are widely used in a variety of applications because a number of important features that they have:

a. High torque at low speeds: NEMA stepper motors are renowned for their capacity to produce high torque. Because of this feature, they are appropriate for uses like robotics, automation, and CNC machines that call for exact control and placement.

b. Excellent Position Control: The remarkable position control of NEMA stepper motors is one of its main benefits. Accurate positioning can be attained by carefully regulating the motor's speed and step count. This feature is especially helpful in situations when precise and consistent movements are needed.

c. Holding Torque: NEMA stepper motors have the ability to hold their position even in the absence of power. This feature, which keeps the motor in the appropriate position, is especially helpful in applications where there is a chance of power loss or intermittent power supply.

d. Step wise motion: NEMA stepper motors move in a step-wise manner, in which a discrete movement increase is represented by each step. Because of this feature, the motor may be precisely controlled in terms of its rotational or linear displacement, which makes it appropriate for applications requiring exact positioning and synchronization.

e. Open loop control: Typically, NEMA stepper motors are run in an open-loop control system, which means that the motor receives control signals without receiving feedback on its actual position. This makes the control system simpler, but it also implies that the number of steps sent determines the motor's location. However, elements like motor resonance, mechanical backlash, or outside disturbances might have an impact on positioning precision.

NEMA motor types:

NEMA stepper motors come in various types, each with unique torque and power characteristics, and are grouped according to their physical dimensions. For these motors, the National Electrical Manufacturers Association (NEMA) has set standards that guarantee component interchangeability and compatibility between different manufacturers. An overview of the NEMA 17, NEMA 23, and NEMA 34 stepper motor types that are often used is given in this section.

A. NEMA 17 stepper motor

NEMA 17 stepper motors are widely used in various industries and applications due to their compact size and versatile performance. These motors are named NEMA 17 because they have a faceplate diameter of approximately 1.7 inches (43.2 mm). Despite their relatively small size, NEMA 17 motors offer a good balance between torque, physical dimensions, and affordability, making them a popular choice for many projects and applications.

NEMA 17 stepper motors are known for their precise control and positioning capabilities. They are capable of delivering a wide range of torque outputs, typically ranging from 0.2 Nm to 0.4 Nm (or 28 oz-in to 57 oz-in), depending on the specific model and design. This torque output is suitable for applications that require moderate power and performance.

One of the key advantages of NEMA 17 motors is their compact size, which allows for easier integration into space-constrained systems. Their small form factor makes them well-suited for applications such as small-scale robotics, 3D printers, camera platforms, and automation equipment. NEMA 17 motors are often used to drive the motion of robotic arms, camera gimbals, extruder mechanisms in 3D printers, and other precision positioning tasks.

NEMA 17 stepper motors typically have a step angle of 1.8 degrees per step, resulting in 200 steps per revolution. This level of precision allows for accurate and repeatable positioning, making them suitable for applications that require precise control over motion and positioning. The step-wise motion of NEMA 17 motors enables fine adjustments and smooth movement, contributing to the overall performance of the system.

These motors are commonly used in open-loop control systems, where the control signals are sent to the motor without feedback on the actual position. While open-loop control simplifies the control system, it also means that the accuracy of positioning is dependent on the assumption that the motor moves the expected number of steps. Factors such as motor resonance, mechanical backlash, and external disturbances can affect the actual position achieved by the motor. However, with careful design considerations and appropriate control algorithms, accurate positioning can still be achieved in most applications.

NEMA 17 stepper motors are available with different winding configurations, allowing for customization based on the specific application requirements. Bipolar and unipolar windings are common options, each offering advantages in terms of torque, power consumption, and control complexity. The choice of winding configuration depends on factors such as desired torque output, speed requirements, and available power supply.

b. NEMA 23 Stepper Motor

NEMA 23 stepper motors are larger and more powerful compared to NEMA 17 motors. These motors derive their name from their faceplate diameter, which measures

approximately 2.3 inches (57.1 mm). The larger size of NEMA 23 motors allows them to deliver higher torque and handle more demanding applications that require increased power and precision control.
One of the primary advantages of NEMA 23 stepper motors is their higher torque output compared to NEMA 17 motors. The torque output of NEMA 23 motors typically ranges from 0.5 Nm to 1.5 Nm (or 71 oz-in to 213 oz-in), depending on the specific model and design. This increased torque capability makes NEMA 23 motors suitable for applications that require more power, such as larger-scale robotics, CNC machines, and industrial automation systems.
NEMA 23 motors are often used in applications where higher load capacities and more robust performance are necessary. They can drive larger loads and handle more demanding tasks, making them well-suited for applications that require precise control over motion and positioning in industrial settings. NEMA 23 motors find application in CNC routers, large-format 3D printers, automated manufacturing equipment, and other heavy-duty automation systems.

Similar to NEMA 17 motors, NEMA 23 stepper motors typically have a step angle of 1.8 degrees per step, resulting in 200 steps per revolution. This level of precision allows for accurate and repeatable positioning, ensuring precise control over the motor's rotational or linear displacement. The step-wise motion of NEMA 23 motors enables precise movements and synchronization in applications that require high accuracy.
NEMA 23 motors are commonly operated in open-loop control systems, where the control signals are sent to the motor without feedback on the actual position. While open-loop control simplifies the control system, it is important to account for factors such as motor resonance, mechanical backlash, and external disturbances that can impact the accuracy of positioning. Careful design considerations, including the use of appropriate drivers and control algorithms, help mitigate these issues and maintain accurate positioning.
NEMA 23 stepper motors are available with various winding configurations, providing flexibility for customization based on specific application requirements. Bipolar and unipolar windings are common options, each offering different trade-offs in terms of torque, power consumption, and control complexity. The selection of winding configuration depends on factors such as desired torque output, speed requirements, and available power supply.

c. NEMA 34 Stepper Motors:

NEMA 34 stepper motors are the largest among the commonly used NEMA stepper motor types. They derive their name from their faceplate diameter, which measures approximately 3.4 inches (86.4 mm). NEMA 34 motors offer even higher torque and power compared to NEMA 17 and NEMA 23 motors, making them suitable for heavy-duty applications that require robust performance and precise control. One of the primary advantages of NEMA 34 stepper motors is their significantly higher torque output. The torque output of NEMA 34 motors typically ranges from 2 Nm to 12 Nm (or 283 oz-in to 1695 oz-in), depending on the specific model and design. The increased torque

capability makes NEMA 34 motors well-suited for applications that require substantial power, such as heavy-duty industrial machinery, large-format 3D printers, and CNC routers.

NEMA 34 motors are commonly used in applications where high load capacities and rugged performance are essential. These motors can drive large loads and handle demanding tasks, making them ideal for industrial automation systems that require precise control over motion and positioning. The higher torque and power output of NEMA 34 motors enable them to deliver the necessary force for applications involving heavy materials or high inertia loads.

Similar to NEMA 17 and NEMA 23 motors, NEMA 34 stepper motors typically have a step angle of

1.8 degrees per step, resulting in 200 steps per revolution. This level of angular precision allows for accurate and repeatable positioning, ensuring precise control over the motor's rotational or linear displacement. The step-wise motion of NEMA 34 motors enables precise movements, making them suitable for applications that require high accuracy.

NEMA 34 motors are often operated in open-loop control systems, where control signals are sent to the motor without feedback on the actual position. While open-loop control simplifies the control system, it is crucial to consider factors such as motor resonance, mechanical backlash, and external disturbances that can affect positioning accuracy. Careful design considerations, including the use of appropriate drivers, motion control algorithms, and mechanical components, help mitigate these issues and maintain accurate positioning.

NEMA 34 stepper motors are available with various winding configurations, offering customization options based on specific application requirements. Bipolar and unipolar windings are common choices, each with its own advantages in terms of torque, power consumption, and control complexity. The selection of the winding configuration depends on factors such as desired torque output, speed requirements, and available power supply.

	NEMA 17	**NEMA 23**	**NEMA 34**
Faceplate Diameter	1.7 inches (43.2 mm)	2.3 inches (57.1 mm)	3.4 inches (86.4 mm)
Torque Output	0.2 Nm to 0.4 Nm	0.5 Nm to 1.5 Nm	2 Nm to 12 Nm
Step Angle	1.8 degrees per step	1.8 degrees per step	1.8 degrees per step
Steps per Revolution	200	200	200
Typical Applications	Small-scale robotics,	CNC machines,	Heavy-duty industrial
	3D printers, automation	3D printers, automation	machinery, large-format
	equipment	equipment	3D printers, CNC routers
Control System	Open-loop	Open-loop	Open-loop
Winding Options	Bipolar, unipolar	Bipolar, unipolar	Bipolar, unipolar
Power and Precision	Moderate power and	Higher power and	High power and
Requirements	precision	precision	precision
Cost	Relatively affordable	Moderate cost	Higher cost

2.6. Motor Drives

Stepper motors are widely used in various industries and applications that require precise control over motion. To harness the full potential of stepper motors, efficient and reliable stepper motor drives are essential. Stepper motor drives serve as the interface between the controller and the motor, providing the necessary signals and power to achieve accurate positioning, speed control, and smooth operation. This essay delves into the key aspects of stepper motor drives, including their types, working principles, advantages, and applications

Types of Stepper Motor Drives:

Unipolar drives:

They are a common type of stepper motor drive known for their simplicity and ease of use.

They utilize a power supply and a series of power transistors or switches to control the motor windings.

The control scheme for unipolar drives involves energizing each motor winding in a specific sequence to achieve step motion.

Working Principle:

Unipolar drives operate by using a series of switches or transistors to connect the motor windings to the power supply. The motor windings are typically center-tapped, allowing current to flow through half of the winding at a time. By selectively switching the transistors on and off in a predetermined sequence, the current flow through the windings changes, resulting in step motion.

Advantages:

1. Simplicity: Unipolar drives are relatively simple to implement and require minimal circuitry compared to bipolar drives. This simplicity makes them cost-effective and easy to integrate into various applications.

2. Lower Power Requirements: Unipolar drives typically require lower power supply voltages, reducing the complexity and cost of the power supply circuitry.

Considerations:

1. Limited Torque Output: Unipolar drives have a lower torque output compared to bipolar drives. This is due to the fact that only half of the motor winding is energized at any given time, resulting in reduced torque production.

2. Lower Efficiency: Unipolar drives are less efficient compared to bipolar drives due to the limited utilization of the motor windings.

In this project, stepper motor drivers are used to precisely control the movement of mechanical parts in the automated parking system. Each driver acts as an interface between the PLC and the stepper motors, converting logic-level pulse and direction signals into the appropriate current and voltage levels needed to drive the motor windings with high precision and smoothness.

Below are details of the stepper motor drivers used:

Advantages:

1. Simplicity: Unipolar drives are relatively simple to implement and require minimal circuitry compared to bipolar drives. This simplicity makes them cost-effective and easy to integrate into various applications.

2. Lower Power Requirements: Unipolar drives typically require lower power supply voltages, reducing the complexity and cost of the power supply circuitry.

Considerations:

1. Limited Torque Output: Unipolar drives have a lower torque output compared to bipolar drives. This is due to the fact that only half of the motor winding is energized at any given time, resulting in reduced torque production.

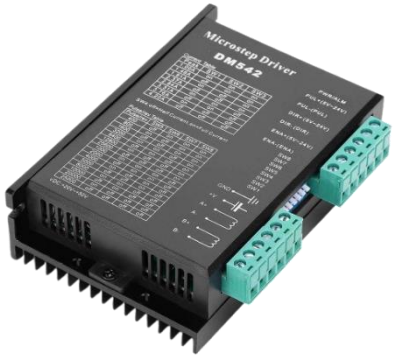
2. Lower Efficiency: Unipolar drives are less efficient compared to bipolar drives due to the limited utilization of the motor windings.

In this project, stepper motor drivers are used to precisely control the movement of mechanical parts in the automated parking system. Each driver acts as an interface between the PLC and the stepper motors, converting logic-level pulse and direction signals into the appropriate current and voltage levels needed to drive the motor windings with high precision and smoothness.

Below are details of the stepper motor drivers used:

2.7.1 Microstep Driver (Model: DM542-compatible)

This driver, commonly known as a **Microstep Driver**, supports bipolar stepper motors and provides flexible microstepping functionality. It is powered by a DC supply voltage in the range of 9–42 VDC, making it compatible with standard industrial power supplies.



Key specifications:

- **Supply voltage:** 9–42 VDC
- **Output current range:** 0.5 A to 3.5 A (peak up to 4.0 A)
- **Microstepping resolution:** 200 to 6400 pulses/rev (configurable by DIP switches)

- **Signal inputs:** opto-isolated differential signals for Enable (ENA), Direction (DIR), and Pulse (PUL)
- **Motor compatibility:** 2-phase bipolar stepper motors
- **Protection:** over-voltage, over-current, and short-circuit protection
- **Indicators:** Power/Alarm LED

Microstepping Settings

The microstepping resolution is configured using DIP switches SW1, SW2, and SW3 as shown on the driver:

Microstep Setting	Pulses/Rev	SW1	SW2	SW3
NC (Full Step)	200	ON	ON	ON
2/A	400	OFF	ON	ON
2/B	400	ON	OFF	ON
4	800	OFF	OFF	ON
8	1600	ON	ON	OFF
16	3200	OFF	ON	OFF
32	6400	OFF	OFF	OFF

3.1.2 Current Settings

The driver allows configuring the maximum phase current using DIP switches SW4, SW5, and SW6:

Rated Current (A)	Peak Current (A)	SW4	SW5	SW6
0.5	0.7	ON	ON	ON
1.0	1.2	OFF	ON	ON
1.5	1.7	ON	OFF	ON
2.0	2.2	OFF	OFF	ON
2.5	2.7	ON	ON	OFF
2.8	3.0	OFF	ON	OFF
3.0	3.2	ON	OFF	OFF
3.5	4.0	OFF	OFF	OFF

2.8 power supply unit

The power supply unit is a fundamental element in any industrial automation system. It provides the necessary and stable DC voltage to power controllers, sensors, actuators, motor drivers, and other peripheral devices. A reliable power supply guarantees safe, consistent, and uninterrupted operation of the entire control system.

Industrial power supplies are generally characterized by:

- High efficiency and stable output voltage.
- Short-circuit and overload protection.
- Wide input voltage tolerance.
- Compliance with industrial EMC standards.
- DIN-rail mounting options for ease of integration.

A well-selected power supply helps maintain the long-term reliability and safety of the system, protecting sensitive electronic components from voltage fluctuations or unexpected surges.

2.8.1 Selected 24V DC Power Supply

In this project, a **24 V DC power supply** is chosen as the primary source for supplying:

- The PLC system.
- Sensors.
- Motor drivers' control electronics.

Why 24 V DC?

- 24 V DC is the industrial standard for control systems worldwide due to its safety compared to higher AC voltages.
- It provides enough power to drive logic circuits and relays without posing significant electrical shock hazards.
- Widely supported by most industrial-grade equipment, making it highly compatible with sensors, PLCs, HMIs, and drivers.
- Reduces electromagnetic interference (EMI) and improves the overall robustness of the control system.

Typical specifications of the selected 24 V DC power supply:

- **Input voltage:** 100–240 V AC (universal).
- **Output voltage:** regulated 24 V DC.
- **Output current:** typically rated from 2 A to 10 A depending on system load.
- **Protections:** short-circuit, overvoltage, and overload.
- **Mounting:** standard DIN-rail for easy cabinet installation.
- **Certifications:** complies with international safety and EMC standards.

This 24 V DC supply guarantees consistent and safe power distribution to all critical automation components in the parking system, ensuring reliable and safe operation under industrial conditions

3

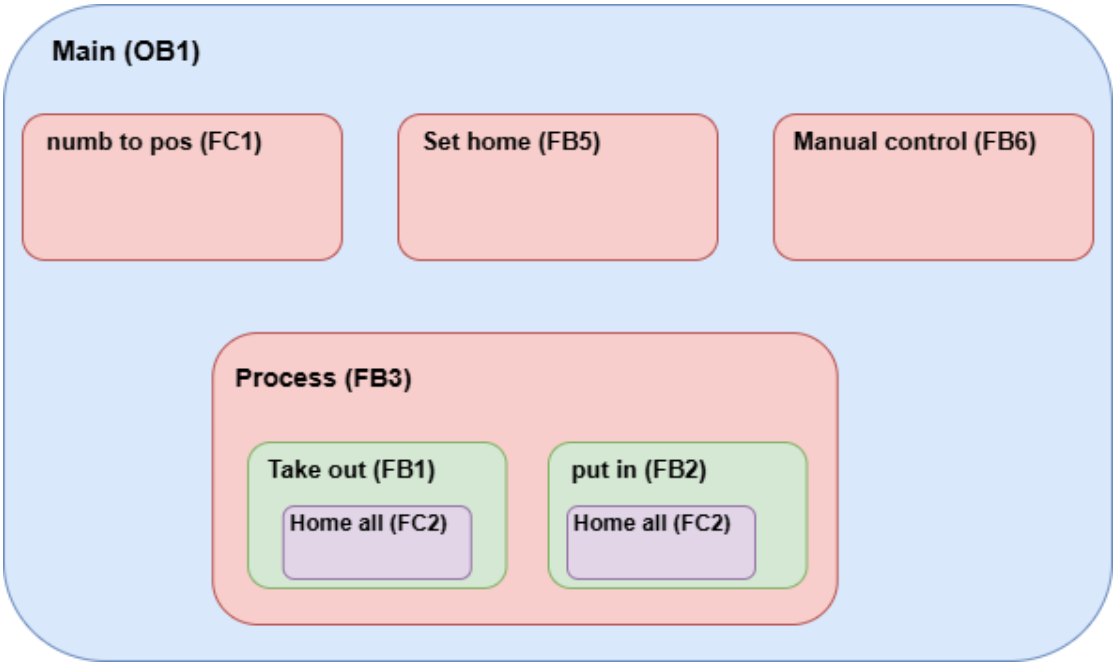
PLC

In this section, we will discuss the Programmable Logic Controller (PLC) system that plays a central role in the automation of our project. The PLC acts as the core controller that coordinates the movement of mechanical components based on real-time input from the machine vision system and commands from the user interfaces.

Importance of PLC in the System

The Programmable Logic Controller (PLC) functions as the central decision-making unit, tasked with executing logic operations, orchestrating motorized movements, and maintaining the stability and safety of the automated parking system. Its proven reliability, robust architecture, and capacity for real-time processing position as an indispensable component in industrial automation applications.

The project structure



The motion-control framework of the automated parking system is structured around four principal Function Blocks (FBs), each designated to execute a specific operational task. By coordinating these FBs through the main program (OB1), the system achieves the seamless execution of the intricate sequences required for operation. Structuring the code in this block-oriented way delivers several technical benefits, described below.

1) Readability and Clarity

Explanation

Explicitly defined Function Blocks (FBs), characterized by well-specified inputs and outputs, enhance the comprehensibility of the program, surpassing the clarity typically offered by monolithic ladder diagrams.

Benefits

- Reduced training time for new team members.
- Faster code reviews and design audits.
- Lower risk of misinterpreting critical control logic.

2) Ease of Debugging and Troubleshooting

Explanation

Each FB keeps its own instance data, allowing you to watch internal variables live during operation.

Benefits

- Faults can be traced to a single block instead of searching for the entire program.
- Maintenance teams can diagnose issues more quickly on-site.
- Clearer separation between hardware faults and logic errors.

3) Reusability

Explanation

FBs can be instantiated multiple times with different parameters.

Benefits

- Saves development time when the same motion pattern is needed for several bays or axes.
- Ensures consistent behavior across identical components.
- Reduces copy-and-paste coding errors.

4) Data Encapsulation and Memory Handling

Explanation

FBs store their variables in dedicated Data Blocks, preventing unintended interaction between unrelated parts of the program.

Benefits

• Minimises pointer or address conflicts.
• Improves program stability and predictability.
• Makes variable tracing straightforward during diagnostics.

5) Scalability

Explanation

Because each FB is self-contained, system expansion means adding new instances rather than redesigning the architecture.

Benefits

• Supports future commercial upgrades (e.g., extra parking levels).
• Keeps the original logic intact, preserving validation results.
• Facilitates stepwise deployment in large installations.

6) Separation of Concerns

Explanation

Motion control, sensor processing, user-interface handling, and safety logic are kept in distinct blocks.

Benefits

• Changes in one area rarely ripple into others.
• Simplifies compliance checks for safety-related code.
• Makes documentation more systematic.

7) Clean HMI / SCADA Integration

Explanation

FB I/O tags are exposed directly to operator panels and SCADA systems.

Benefits

•Less tag-mapping effort during UI development.
•Real-time status and alarms can be displayed consistently.
• Operator actions remain clearly linked to the underlying logic.

8) Documentation and Maintainability

Explanation

Each FB can carry its own comments, version number, and author information.

Benefits

- Simplifies hand-over to maintenance teams or future developers.
- Improves traceability of changes over the project life-cycle.
- Elevates overall project professionalism and audit readiness.

Overview of the function blocks:

The PLC program in this automated parking system is designed around two main components:

1. Motion Control and Sequence Execution.

2. Coordination with External Technologies.

This separation ensures that the logic governing mechanical movement is kept distinct from that which interfaces with sensors, user interfaces, and external systems. Such a design improves reliability, simplifies debugging, and aligns with best practices for industrial control systems.

1. Motion Control and Sequence Execution:

This part of the project handles the physical movement of the parking mechanism, including vertical and horizontal positioning, homing, and manual overrides. The motion logic is divided into modular Function Blocks (FBs) and Function Calls (FCs) to allow for scalability and ease of testing.

a. Numb to pos (FC1):

This Function Call converts a user-selected slot number into a corresponding physical position. It assigns the calculated target distance to a shared variable used by the motion control blocks. This separation between input interpretation and movement execution improves modularity and flexibility.

b. Set Home (FB5):

This Function Block is used during system startup or after a reset. It sets the reference "home" position at the entrance point of the system. It also ensures that the vertical and horizontal Technology Objects (used for axis control) are powered and ready.

c. Home all (FC2):

This function homes both horizontal and vertical axes. It moves the horizontal axis first, followed by the vertical axis, to ensure safe and collision-free referencing. This is typically used at the start of the system or between motion sequences to guarantee consistent positioning.

d. Process (FB3):

This is the main controller block for movement operations. It manages the sequence of actions needed to retrieve or store a vehicle plate. It includes conditional logic to determine whether the operation is a retrieval or insertion and handles the motion calls accordingly. It is composed of two major subcomponents:

Take out (FB1):

This block performs the car retrieval sequence. It moves the platform (or fork) to the correct floor to pick up the car plate and then returns to the home position. It calls the Home all function (FC2) to reset the axes afterward.

Put in (FB2):

This block executes the insertion sequence. It moves the platform to the designated floor to return the plate and then returns to home. Like the retrieval process, it uses Home all (FC2) at the end of the operation.

e. Manual control (FB6):

This Function Block provides manual control of the motors for authorized users, such as system administrators or maintenance personnel. It allows direct movement commands for both vertical and horizontal axes in absolute or relative mode. This feature is essential for testing, calibration, and emergency situations.

2. Coordinating with other technologies:

Beyond controlling mechanical movements, the PLC program must also interact with other systems, including the machine vision unit (Python-based) and three user interfaces (HMI, SCADA, and Node-RED). Dedicated function blocks handle these responsibilities to isolate external dependencies from the core logic.

a. Sensor Check (FC4):

This block processes input from the machine vision system. To reduce noise and false triggers, it applies a filtering mechanism that only accepts a signal if it remains stable for a specified number of scan cycles. This improves system accuracy and reliability when reacting to visual detection.

b. Stop other GUI (FC5):

To avoid command conflicts when multiple user interfaces are available, this block ensures that only one UI is active at a time. When a user initiates an operation from one interface (e.g., HMI), the system blocks input from the others (e.g., Node-RED or SCADA). This logic preserves synchronization and operational integrit.

Overview of Data Blocks Used in the PLC Project

The PLC program uses several Data Blocks (DBs) to store system states, motion parameters, UI data, and persistent configuration. Each block serves a specific purpose and helps maintain separation of logic, improve modularity, and simplify debugging.

1. DB1 – Communication:

This data block acts as the main interface between the PLC logic and external systems, including the user interfaces (HMI, Node-RED, and SCADA) and the machine vision system. It holds all shared variables required for monitoring and control-such as operation commands, parking slot selections, status feedback, and communication flags. Keeping UI-related variables centralized in this block allows for clean integration and simplified troubleshooting.

2. DB2 – Position:

DB2 is used to store motion-related parameters, including the target coordinates, speed settings, and direction values for the vertical and horizontal axes. These values are dynamically updated by the logic blocks (numb to pos, put in, take out, etc.) and then used by the motion control instructions (e.g., MC_MoveAbsolute). Organizing all motion settings in one block improves clarity and consistency across function blocks.

3. DB13 – Manual:

This block is used exclusively for manual operation control. It stores user-defined values for movement, including jogging direction, target distance, and control flags for each axis. DB13 is accessed by the Manual Control (FB6) block and enables administrators to test or calibrate the system without relying on the automated logic. Separating manual controls into a dedicated DB avoids accidental interference with automatic operation.

4. DB26 – Permanent Data:

DB26 holds persistent system values, such as the calibrated home position of each axis. These values are typically written during homing procedures (Set Home) and referenced during automated movements. Storing these constants in a separate block ensures they are preserved even during logic updates or system resets.

5.DB44 – Car Shown:

This data block provides the SCADA system with real-time car presence indicators used in animation and visual representation. It includes three key variables that allow SCADA to render car images accurately in accordance with the system's current state and timing of parking or retrieval operations. This improves operator visibility and supports user-friendly visual feedback.

6.DB49 – Login:

DB49 is used to coordinate user authentication data across multiple user interfaces. It collects login credentials and session details and enables the transfer of this data to a centralized SQL Server via the Python backend. This integration supports centralized access management, session logging, and enhanced system security.

7.DB16 – Sensors:

This data block interfaces with the machine vision system to determine the status of parking slots and platform occupancy. It records detection results indicating whether each slot is empty or occupied, and whether a vehicle is present on the platform. These inputs are essential for ensuring safe and accurate execution of parking and retrieval operations.

The technology object

In Siemens TIA Portal, Technology Objects (TOs) are predefined, hardware-linked software objects used to simplify the control of advanced functions such as motion, positioning, PID control, and more. They encapsulate complex automation behaviors into configurable components that can be directly integrated with PLC logic.

Technology Objects help bridge the gap between the control software and physical devices like drives and motors. Instead of writing custom low-level control code, developers can instantiate TOs that manage tasks such as motion commands, axis homing, speed control, and safety interlocks with built-in diagnostics and standardized interfaces.

Positioning Axis Technology Object (TO_PositioningAxis):

The TO_PositioningAxis is a specific type of Technology Object designed for controlling a single-axis motion system. It is commonly used for linear or rotary axes driven by stepper motors, servo motors, or frequency-controlled motors.

In this project, TO_PositioningAxis was used to control the horizontal and vertical movement of the automated parking system's mechanical components. Notably, it was

implemented without using encoders, relying on open-loop control of stepper motors. This approach leverages the precise step-per-pulse characteristics of stepper drives, which, when configured correctly, provide reliable motion without the need for feedback devices.

Benefits of Using TO_PositioningAxis:

- Simplified Configuration: All parameters such as speed, acceleration, jerk, and limits are defined in the object configuration rather than being coded manually.
- Integrated Motion Instructions: Standard motion instructions like MC_Power, MC_MoveAbsolute, MC_Home, and MC_Reset can be directly used in Ladder Logic to control the axis.
- Built-in Status and Diagnostics: The object provides access to structured status flags (e.g., AxisReady, Error, Busy), reducing the complexity of writing custom diagnostic logic.
- Safe Homing and Travel Limits: Even in open-loop systems, safety-related features like soft limits and controlled homing procedures are supported.
- Parameter Reusability: Values such as position targets and speeds can be assigned dynamically at runtime, improving flexibility.

Application in the Project: Stepper Motor Control without Encoder:

In this system, the vertical and horizontal axes were each controlled by a TO_PositioningAxis instance connected to a stepper motor. Instead of relying on feedback from encoders, the system uses the following setup:

- Open-loop Motion: Stepper motors are driven via step/direction signals, assuming that motion completes reliably based on pre-calibrated distances.
- Homing via Sensors: The MC_Home instruction uses digital input signals from mechanical or optical sensors to define a zero position, allowing motion control to be referenced.
- Position Tracking: Once homed, all movements are executed in absolute or relative mode, and the current position is tracked internally by the TO based on the issued pulses.
- Safety Management: Motion limits and emergency stop conditions are managed through logical interlocks and limit switch inputs.

Key Motion Control Functions:

The following function blocks are used to interface with the Technology Object and issue motion commands. These blocks are executed within the PLC logic using Ladder Diagram (LAD) programming.

1. MC_Power:

Purpose:
MC_Power is used to enable or disable power to the axis. This instruction establishes the connection between the PLC and the drive system and prepares the motor for receiving motion commands.

Typical use case:
Executed during system startup or reinitialization and must remain enabled throughout the motion cycle for the axis to remain active.
Key Inputs:

- Axis: Reference to the axis (TO_Positioning Axis).
- Enable: Boolean trigger to turn on the axis.

Key Outputs:

- Status: Indicates whether the power is successfully enabled.
- Busy: Indicates execution is ongoing.
- Error / ErrorID: Reports any faults or problems during the operation.

2. MC_Home:

Purpose:
The MC Home block initiates the homing procedure to define the zero-reference point for an axis. It is particularly essential in open-loop stepper systems where position tracking is reset at startup.

Typical Use Case:
Executed after power-on to move the axis until it triggers a homing sensor (e.g., limit switch), establishing a known mechanical position.
Key Inputs:

- Execute: Starts the homing sequence.
- Position: Optional — specifies where the zero point should be set after homing.
- HomingMode: Determines the behavior of the home operation.

Key Outputs:

- Done: Indicates homes completed successfully.
- Busy: Operation is in progress.
- Error / ErrorID: Reports on any issue encountered.

3. MC_MoveRelative:

Purpose:
MC_MoveRelative commands the axis to move a specified distance from its current position, using motion parameters defined either in the TO or overridden by the block.

Typical Use Case:
Used when relative positioning is desired, such as jogging the motor by a set distance during manual operation or calibration.
Key Inputs:

- Distance: The relative travel length from the current position.
- Velocity, Acceleration, Deceleration: Optional overrides for motion profile.
- Execute: Starts the motion.

Key Outputs:

- Done: Motion completed.
- Busy: Indicates movement is ongoing.
- Error / ErrorID: Diagnostic feedback.

4. MC_MoveAbsolute:

Purpose:
MC_MoveAbsolute moves the axis to a specific absolute position relative to its zero-reference point established during homing.

Typical Use Case:
Frequently used in automated sequences where the axis must move to predefined coordinates, such as aligning with a parking level or returning to home.
Key Inputs:

- Position: Target absolute position.
- Velocity, Acceleration, Deceleration: Motion profile parameters.
- Execute: Starts the movement.

Key Outputs:

- Done: Movement successfully completed.
- Busy: Instruction is executing.
- Error / ErrorID: Indicates if the motion failed.

5. MC_Halt:

Purpose:
MC_Halt stops the axis immediately by decelerating the motor to a standstill, using a

controlled stop ramp. It does not reset the position counter, unlike emergency stop procedures.

Typical Use Case:
Used to interrupt a movement due to user input, interlock condition, or sensor feedback while retaining current axis state.
Key Inputs:

- Execute: Trigger to initiate halting.
- Deceleration: Optional input for how quickly the axis should decelerate.

Key Outputs:

- Done: Axis has come to a stop.
- Busy: Instruction is active.
- Error / ErrorID: Indicates halting failure if any.

Detailed Analysis of Core Function Blocks and Control Logic:

1. Startup (OB100):

In the Siemens TIA Portal programming environment, Organization Blocks (OBs) define the execution structure of a PLC program. OB100 is executed once during the startup of the CPU, specifically when the CPU transitions from STOP to RUN mode.

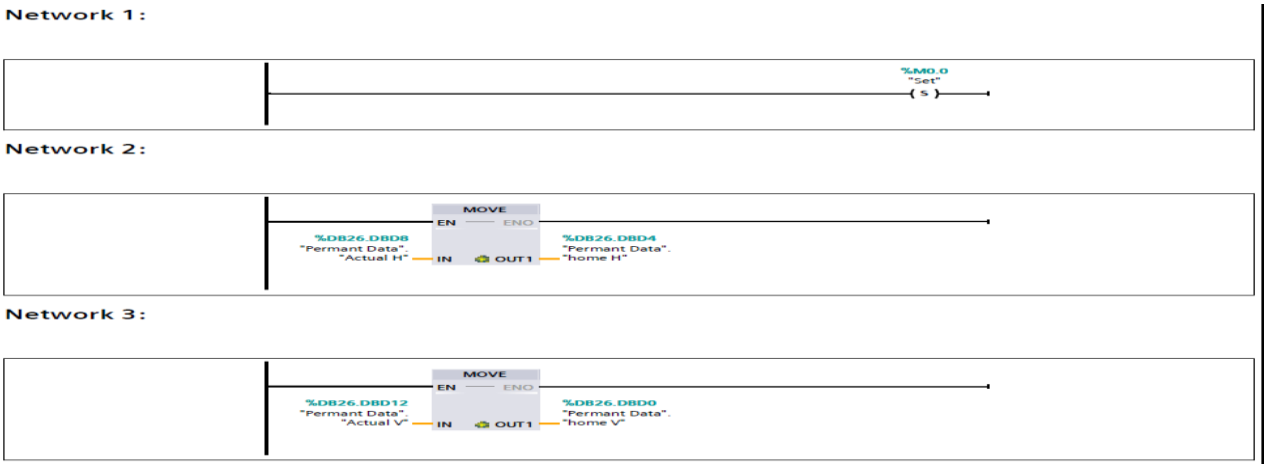
In this project, Organization Block OB100 is utilized to handle system initialization tasks that must occur once when the PLC transitions from STOP to RUN mode. Specifically, OB100 is programmed to:

Activate marker M0.0, which serves as an enabling condition for the Set Home function block (FB5).

Transfer the most recent displacement values for both the horizontal and vertical axes from the permanent data block to the Set Home block. These values represent the last known home reference points prior to shutdown.

This mechanism ensures that the system preserves its reference positioning in cases where an unexpected power loss occurs. During normal operation, the displacement values stored in the permant data block are expected to be zero, reflecting a fully reset or freshly homed state. However, in the event of a restart without a proper shutdown or re-homing sequence, these stored values can be used to reestablish axis references, thereby enhancing the robustness and continuity of the motion control logic. By integrating this

logic into OB100, the system is able to resume operation in a consistent and safe state, even after unexpected interruptions.



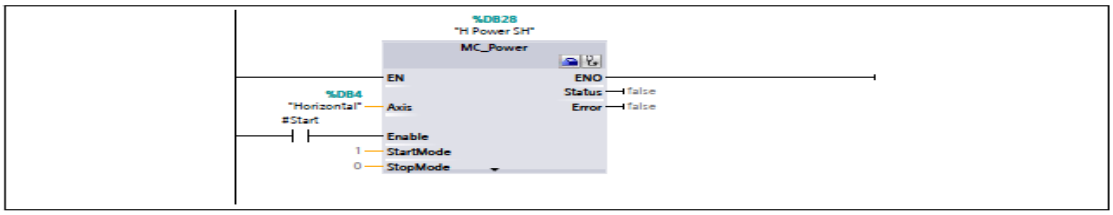
2. Set Home (FB5)

Function Blocks (FBs) are reusable program modules that maintain their own instance-specific memory. This means that variables declared inside an FB retain their values between successive calls, making FBs ideal for operations that require state tracking or sequential logic.

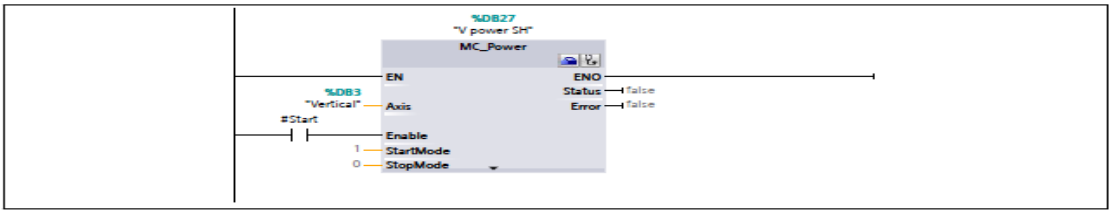
The Set Home function block (FB5) is responsible for executing the homing procedure for both the horizontal and vertical motion axes using Siemens Technology Objects. This block ensures that each axis is properly referenced before any operational movement begins.

The structure of FB5 consists of several sequential logic networks:

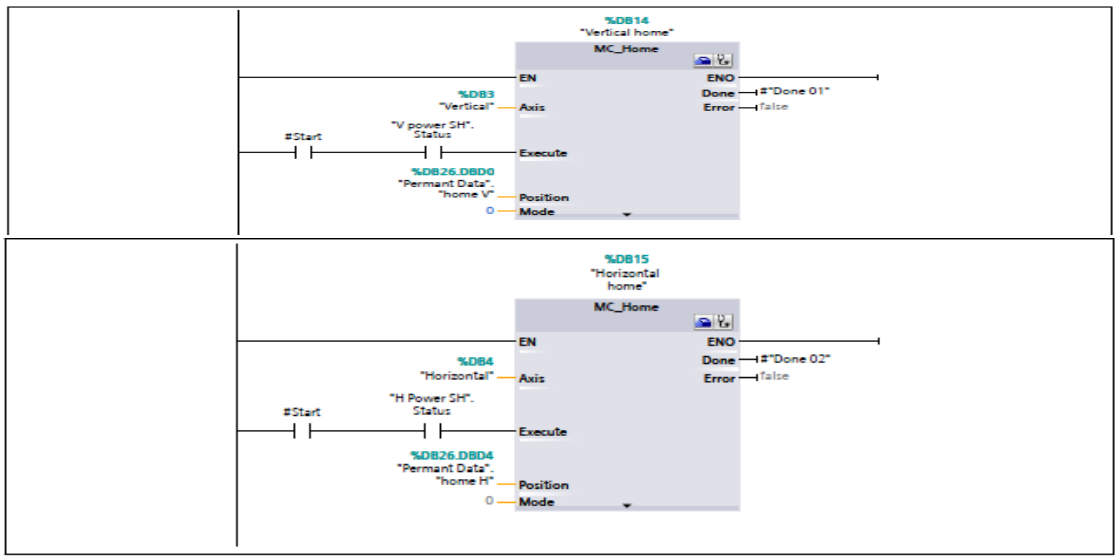
- Networks 1 and 2 activate MC_Power instruction for the horizontal and vertical axes, respectively. This step ensures that both Technology Objects are enabled and ready to receive motion commands.
- Networks 3 and 4 execute the MC_Home function for each axis. These instructions initiate the homing cycle by moving each axis toward its configured home sensor, thereby establishing a reliable reference position for motion control.
- The final network generates a "Done" signal once both home operations are completed successfully. This signal serves as an indicator that the homing sequence has been carried out without errors, allowing the system to proceed to subsequent operational logic.



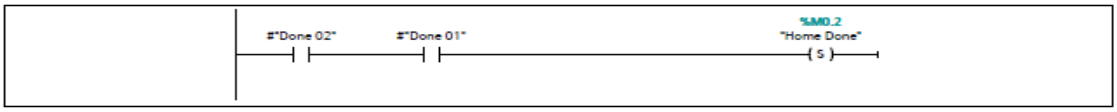
Network 2:



Network 3:



Network 5:



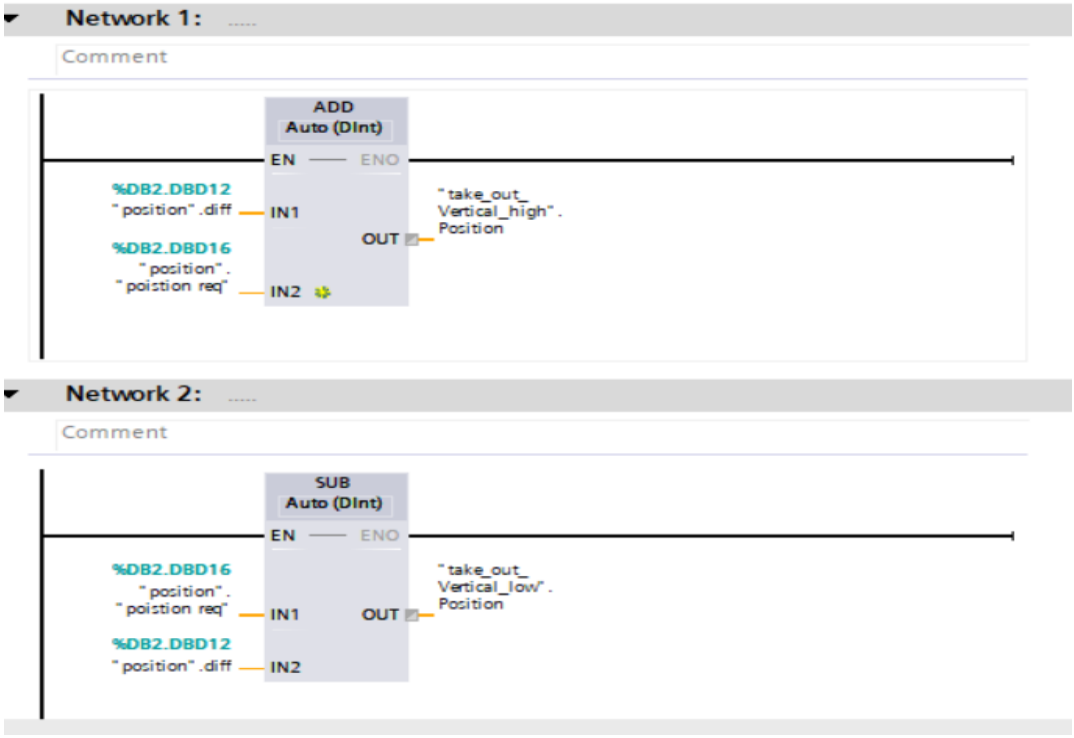
3. Take out (FB1):

The Take out function block (FB1) is designed to execute the car retrieval sequence by controlling the motion of the mechanical fork or platform used to lift the vehicle plate from a designated parking slot.

The initial two networks within this block perform preparatory calculations required for the movement path. Specifically, these networks:

- Subtract and add a predefined offset (referred to as "diff") from the target position (position.req) obtained from the position data block (DB2).
- These calculated values represent two key motion positions:

- A position slightly above the designated slot.
- A position slightly below the slot level, where the plate is engaged.

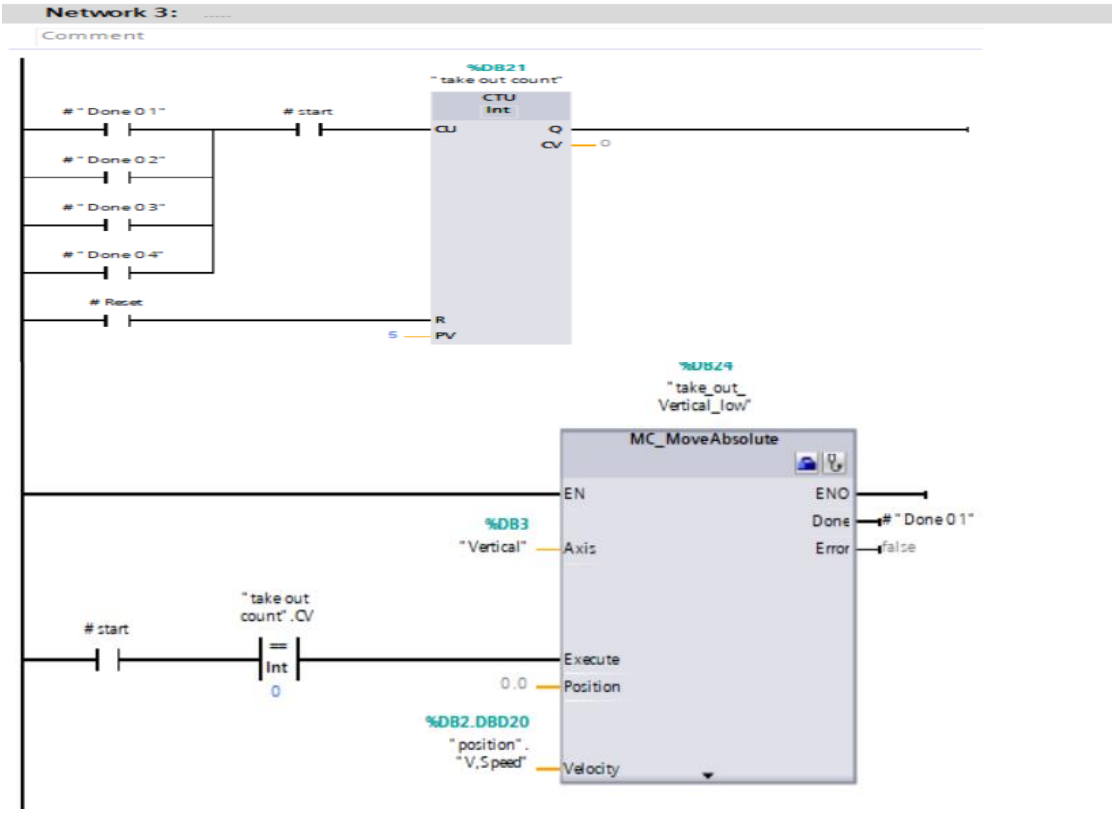


Following the initial position calculations, the subsequent network implements a counter to manage and track the step-by-step progression of the etrieval movement. This counter functions as a sequential control mechanism, determining the current stage within the overall take-out process.

The associated data block for this counter is configured as retained, meaning its value is preserved even in the event of a power loss or system interruption. As a result, the system can resume operation from the last completed step rather than restarting the entire sequence, thereby enhancing operational resilience and continuity.

The counter increments upon the successful completion of each motion step (e.g., vertical approach, horizontal travel, platform engagement), ensuring that the system advances in a controlled and logical order. This design improves both motion accuracy and fault recovery, aligning with best practices in sequential automation.

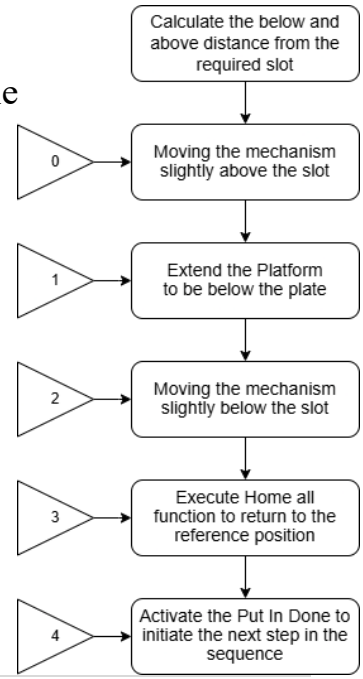
Counter	Operation	output
0	Moving the mechanism slightly below the slot	Done 01
1	Extend the Platform to be below the plate	Done 02
2	Moving the mechanism slightly above the slot	Done 03
3	Execute Home all function to return to the reference position	Done 04
4	Activate the Take out Done to initiate the next step in the sequence	[NONE]



4. Put in (FB2):

The Put In function block follows the same sequential structure and control logic as the Take out block, utilizing the same stepwise execution framework governed by a counter. However, it includes a minor but functionally significant modification: the operation begins by moving the mechanism slightly above the target slot, rather than below it as in the retrieval process.

This adjustment ensures a top-down approach when placing the vehicle plate back into position, optimizing alignment accuracy and reducing mechanical interference during insertion.



Main (OB1):

The Main Organization Block (OB1) in Siemens TIA Portal serves as the central cyclic execution unit of the PLC program. It is executed continuously while the CPU is in RUN mode and orchestrates all operational logic by sequentially calling Function Blocks (FBs) and Functions (FCs) that manage input validation, motion control, interface coordination, and system safety.

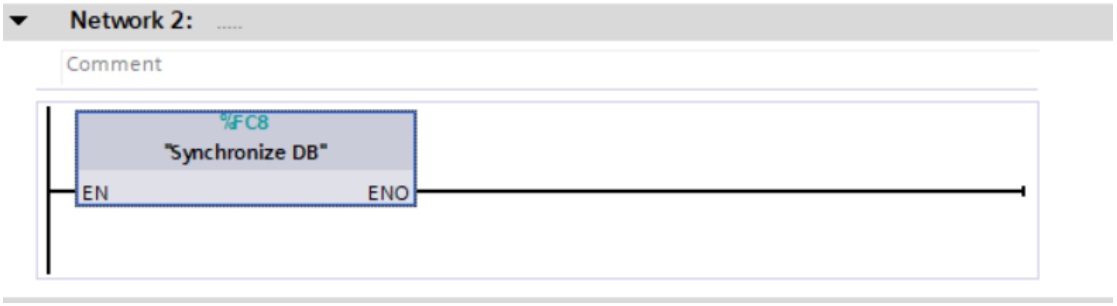
The following is a detailed breakdown of OB1 logic network by network, as implemented in this automated parking system.

Network 1 – Only one (FC3):

This network ensures that only the most recent command is executed—whether it is an input (car insertion) or output (car retrieval) request. It filters out any overlapping or simultaneous commands and enables the logic to focus exclusively on the last selected position and operation type. This prioritization is crucial for avoiding motion conflicts and ensures the system responds in a controlled and deterministic manner.

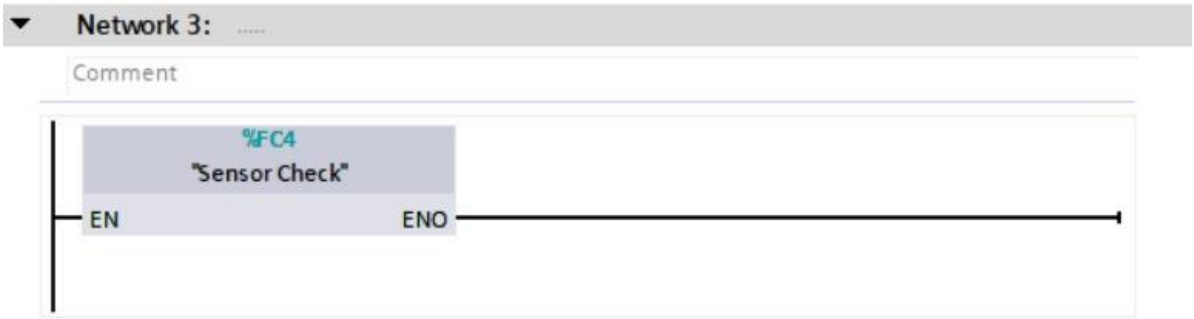
Network 2 – Data Synchronization (FC8):

This network calls the Synchronize DB function (FC8), which is responsible for aligning variable states across multiple data blocks. In modular systems where different function blocks operate independently, variable duplication across DBs can introduce inconsistency. FC8 resolves this by enforcing value consistency, making inter-block communication more reliable and reducing the risk of logic faults due to mismatched state variables.



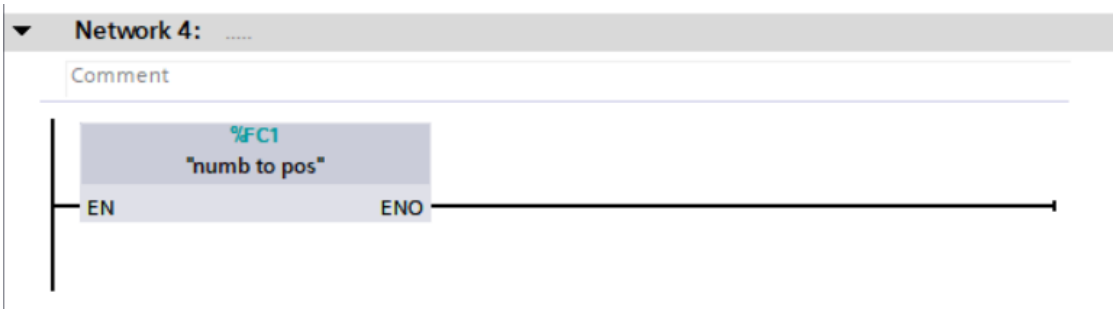
Network 3 – Sensor check (FC4):

In this network, the Sensor Check function (FC4) is executed to validate and stabilize sensor readings—particularly those originating from the machine vision system. The function applies signal filtering by accepting only those sensor states that remain valid over a defined time threshold, thus preventing false triggers due to transient fluctuations or noise. This ensures that sensor-based decisions are based on stable and verified inputs.



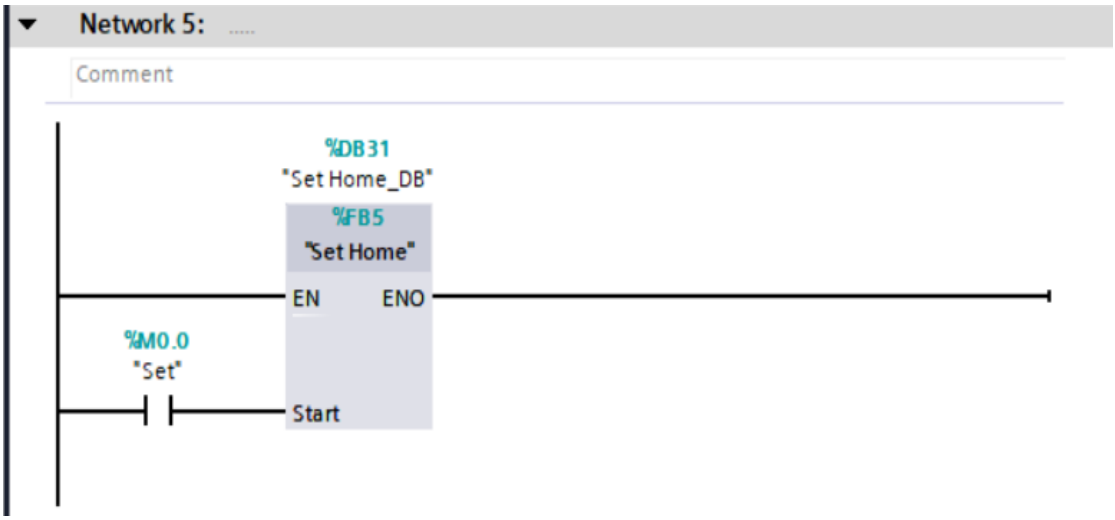
Network 4 – Position Conversion (FC1):

This network invokes the numb to pos function (FC1), which translates the selected slot number into a set of absolute position values. These coordinates—used for motion control—are stored in the designated position data block (DB2). The output from this network serves as the primary reference for both Put In and Take Out motion routines.



Network 5 – Homing Routine (FB5):

This network handles the homing process by calling the Set Home function block (FB5). The block enables the motion axes, executes the homing procedure using MC_Home, and sets a status flag indicating successful referencing. This step is mandatory before any automated movement, as it ensures that both vertical and horizontal axes begin from a known and safe mechanical reference point.

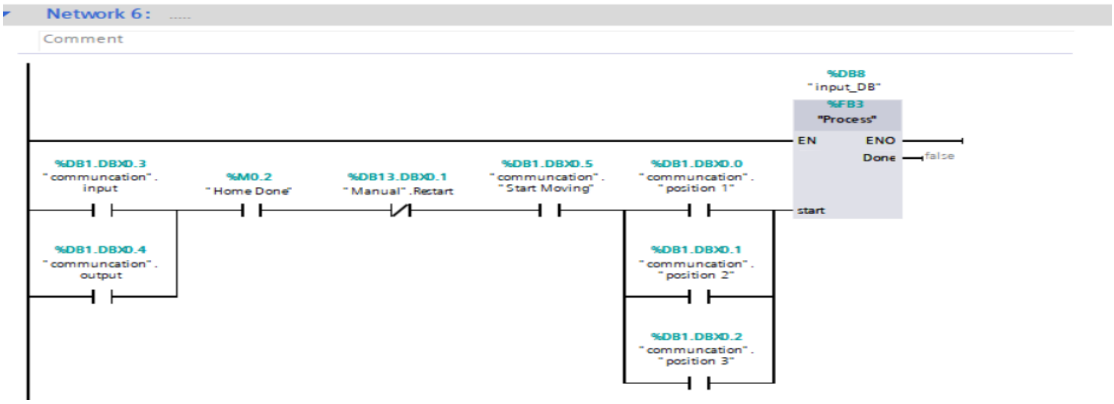


Network 6 – Automated Motion Execution (FB3):

This network controls the automated execution of motion sequences, managed through the Process block (FB3). It coordinates both the Put In (FB2) and Take Out (FB1) routines, executing the appropriate one based on the selected operation type. The logic is conditioned by three critical requirements:

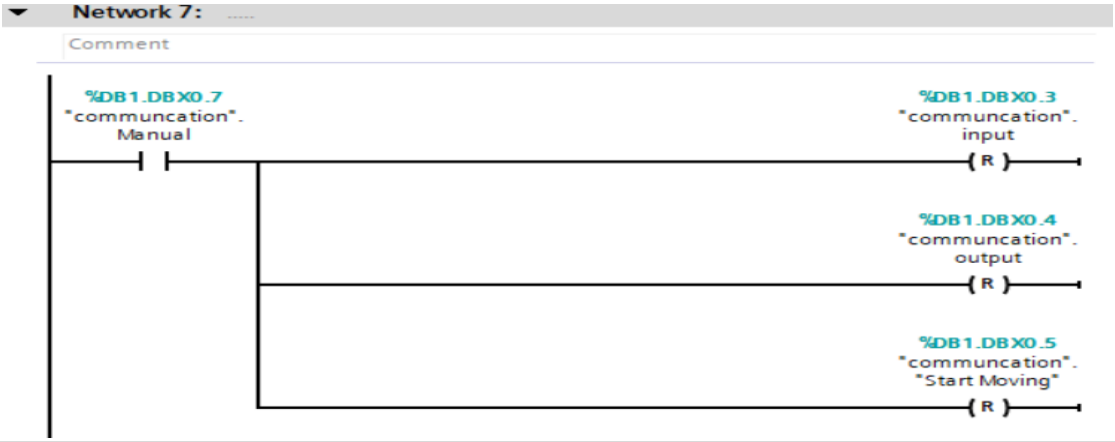
1) A valid operation type must be selected (input or output).
2) A position to move to.
3) The homing procedure must be completed successfully (Home Done), and.
4) Manual control must be deactivated.

Only when these conditions are met does the system proceed to execute the full motion sequence. This gating mechanism ensures operational safety, precision, and mutual exclusivity between automated and manual mod.



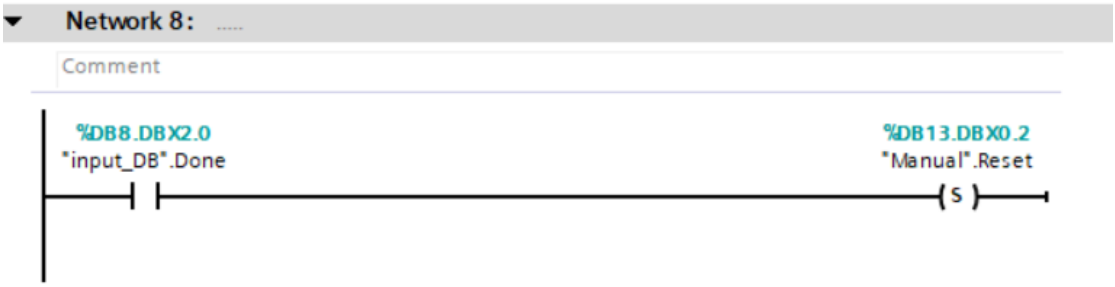
Network 7 – Manual Reset Activation:

This network is responsible for handling manual reset conditions. When the manual reset is triggered by the operator, this logic forcibly resets automated-relevant states, including the input command, output command, and start signal. This allows the operator to terminate or cancel an in-progress operation safely and return the system to an idle state before initiating a new task.



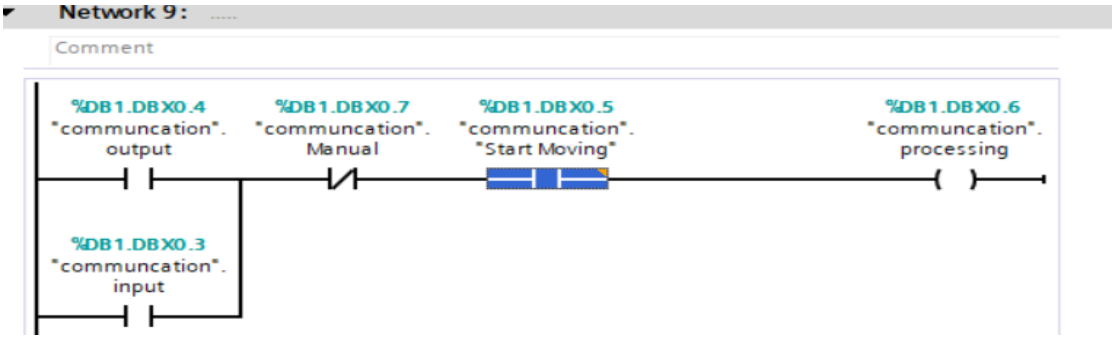
Network 8 – Automatic Reset After Process Completion:

Once an automated process (either insertion or retrieval) is completed successfully, this network sets a reset flag that instructs the system to reinitialize all relevant tags and data blocks by activating Reset (FB7). This ensures that no residual data remains from the previous cycle and prepares the system for a clean transition into the next operation.



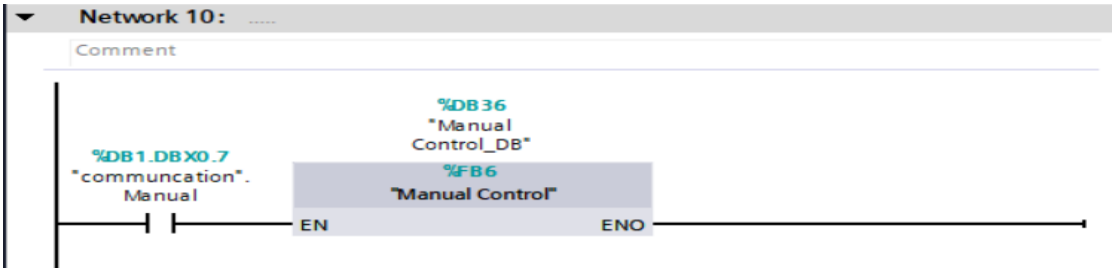
Network 9 – Process Timing Tag Activation:

This network activates a dedicated process timing tag that remains true for the entire duration of the active motion sequence. This tag can be used for tracking operation time, enabling visual indicators, or recording timestamps for logging or diagnostic purposes. It provides an effective means of monitoring process activity and verifying system responsiveness.



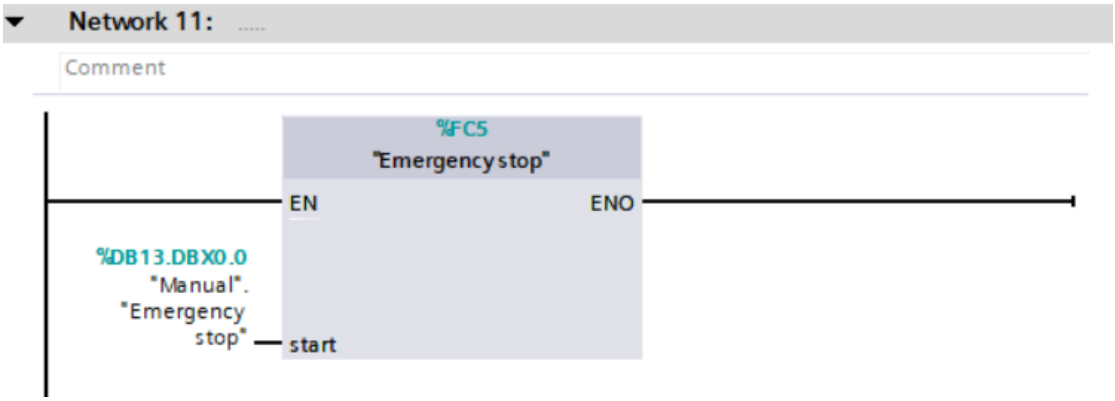
Network 10 – Manual Control (FB6):

In this network, the Manual Control function block (FB6) is invoked. This block provides authorized personnel or administrators with the ability to control the platform manually, outside of the automated cycle. The function allows for jog operations and absolute/relative positioning of both axes, making it particularly useful for system testing, calibration, or fault recovery.



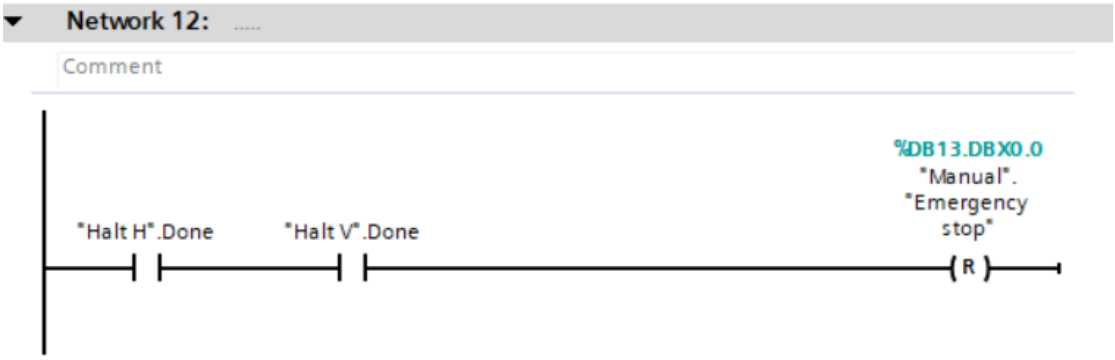
Network 11 – Emergency Stop Activation:

This network implements the emergency stop (E-Stop) logic, which immediately halts all motion-related operations. It does so by executing the MC_Halt function for all active axes, ensuring that movement ceases in a controlled and safe manner. This network is essential for addressing emergency situations such as hardware malfunctions, obstructions, or safety breaches.



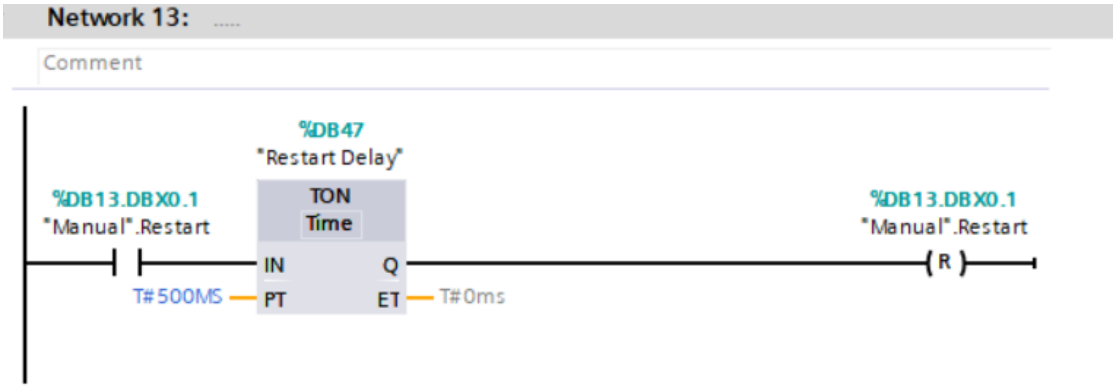
Network 12 – Resetting the Emergency Stop Flag:

Once the emergency condition has been resolved and motion has come to a complete stop, this network is used to reset the emergency stop flag. This prepares the system for normal operation and enables the subsequent logic for motion restart. Ensuring a controlled reset of the emergency state is critical for system integrity and user safety.



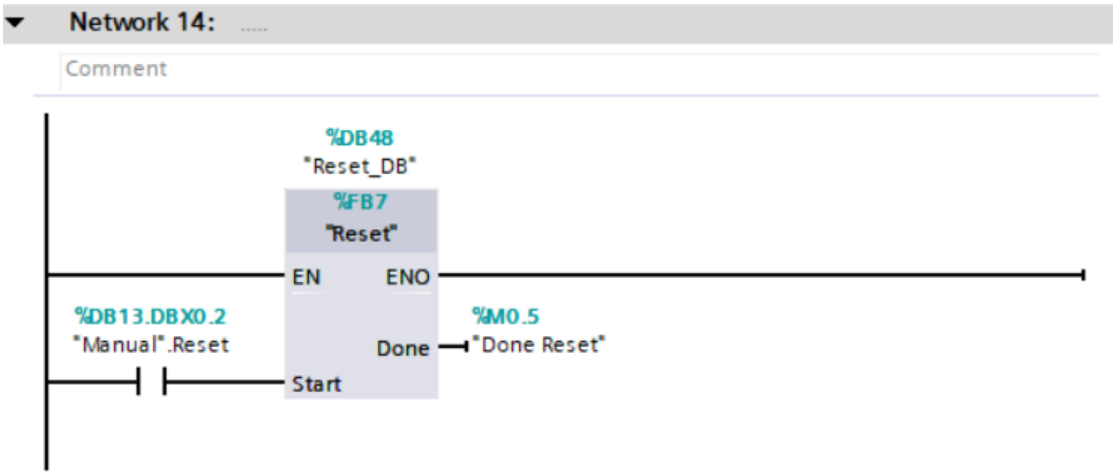
Network 13 – Restarting After Emergency Stop:

After the emergency stop state is cleared, this network handles the controlled restart of system operations. It re-enables necessary logic components and re-engages previously halted motion sequences, allowing the system to return to full functionality without manual reprogramming or CPU cycling.



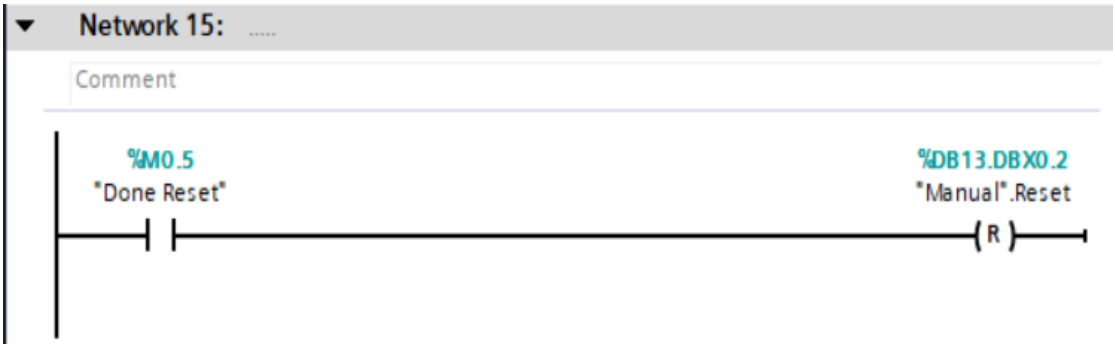
Network 14 – Reinitialization of System Tags and Data Blocks:

This network performs a comprehensive reinitialization of all system tags and memory elements required for operation. It is typically used after a reset or emergency stop recovery to ensure the system starts from a known, validated state. This process mitigates the risk of undefined behaviors due to partial resets or incomplete data clearance.



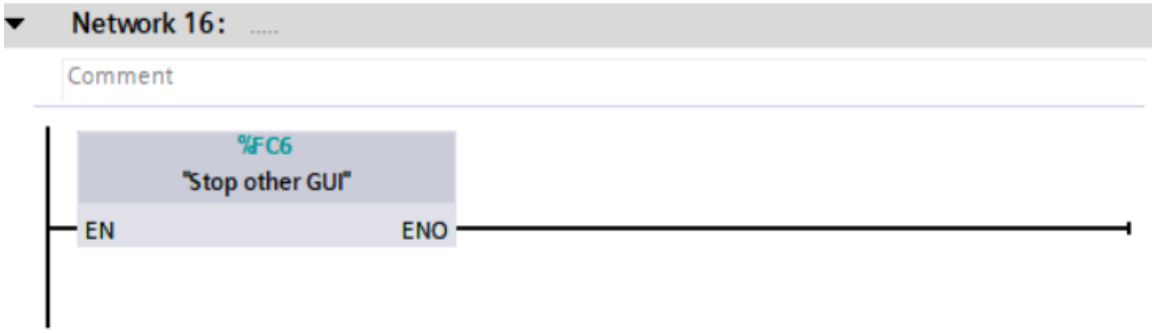
Network 15 – Resetting the Reinitialization Trigger:

Following the reinitialization process, this network resets the trigger bit that initiated the data and tag reset in Network 14. This ensures that the reset procedure is not repeatedly executed in subsequent cycles and maintains the logical integrity of the startup sequence.



Network 16 – User Interface Lockout (FC5):

This final network enforces exclusive UI control logic. It ensures that when one graphical user interface (GUI)—such as HMI, SCADA, or Node-RED—is actively issuing commands, all other UIs are temporarily disabled. This prevents command collisions and maintains safe, orderly execution by enforcing a single source of control at any given time.



HMI

Introduction:

A human-machine interface or man-machine interface is a combination of software and hardware which will help the operator to communicate between systems and Machines. The HMI system is inevitable in most industries. The automation system is not completed without an HMI. With the help of an HMI, the operator can check the Plant operation in real-time. The HMI can alarm the operator if there is any fault. So Basically, the HMI can be considered as the user interface in manufacturing or process Control system. The HMI will give a graphic based visualization of industrial control and monitoring system. With the help of HMI, we can increase productivity in a plant with the help of a centralized control center. While the term can technically be applied to any screen that allows a user to interact with a device, HMI is most used in the context of an industrial process.

Needing of Human Machine Interface:

- It can do the visual display of the data and it can find out the production faults too.
- It can determine the production time, trends, and tags.
- Improved efficiency and production.
- It would sound the alarm if there were any faults.
- By using the HMI, the operator can give instructions to the machine.
- By using HMI, we can control, manage, and visualize the device process in an Industry.
- This will allow the user to determine and respond to abnormal conditions.

Human Machine Interface Working Principal:

The HMI consists of a monitoring screen, operating panel, and communication ports. The HMI can be integrated into our process with the help of a PC, and it is done by using certain software and then the program will be transferred to the HMI and the HMI will run the program. The devices in the factory can be connected to the HMI like the

machinery in a production line and the input-output sensors and these sensors will give the data to the PLC. The monitoring screen in HMI will allow the user to interact with the HMI. The HMI has an operating panel and by this, the operator can do the required functions. The communication ports in the HMI are to program the HMI to communicate with other devices so the devices in a factory can communicate with HMI.

Types of HMI:

There are several different types of Human-Machine Interfaces (HMI), which can take a variety of forms depending on the needs of the application.

Some common types of HMI include:

1) Graphical User Interfaces (GUIs):

These are computer-based HMI that use a graphical interface to present process data and allow the operator to input commands. GUIs can be created using software such as Microsoft Windo ws or a specialized HMI software package.

2) Touchscreens:

These are HMI that use a touchscreen display to present process data and allow the operator to input commands. They are often used in applications where a physical control panel is not practical, such as in environments where space is limited

3) Physical Control Panels:

These are physical HMI that use buttons, switches, and displays to present process data and allow the operator to input commands. They are often used in industrial Settings where a GUI or touchscreen may not be practical, such as in environments with high levels of vibration or dust.

4) Web-based HMI:

These are HMI that use a web browser to present process data and allow the operator to input commands. They can be accessed from any device with a web browser, allowing operators to monitor and control processes remotely.

Advantages of Human-Machine Interface:

Human-Machine Interfaces (HMI) have several advantages that make them useful

For controlling and monitoring industrial processes:

1) Improved Efficiency:

HMI allows operators to easily access process data and input commands, which can improve the efficiency of the process being controlled. By presenting data in a clear and concise manner, operators can make more informed decisions and take timely action when necessary.

2) Increased Safety:

HMI can improve safety by presenting and recording alarms and other important information to the operator. This allows operators to take timely action to address potential problems before they become serious.

3) Remote Monitoring and Control:

Many HMI can be accessed remotely, allowing operators to monitor and control processes from a distance. This can be particularly useful in applications where the process is located in a remote or hazardous location.

4) Easy to Use:

HMI is designed to be easy to use, even for operators who may not have technical expertise. This makes them an accessible tool for controlling and monitoring industrial processes.

5) Saving time and effort:

HMI can identify the type of fault and its location, which facilitates and speeds up the repair process and reduces the time of the fault, as well as the time of stopping operation or production.

Disadvantages of Human-Machine Interface:

Human-Machine Interfaces (HMI) have some potential disadvantages that should be considered when determining whether to use one:

1) Dependency on the HMI:

If the HMI fails or becomes unavailable, the operator may not be able to access process data or input commands, which can hinder the operation of the process being controlled.

2) Potential for Human Error:

Operators can make mistakes when using HMI, such as inputting incorrect commands or failing to notice an alarm. This can lead to problems with the process being controlled.

3) Additional Cost:

Implementing an HMI can be an additional cost, as it requires the purchase and installation of the HMI itself as well as any associated hardware and software.

4) Complexity:

Some HMI can be complex, which can make them difficult to use or require additional training for operators. This can be a disadvantage in applications where the operator needs to be able to quickly and easily access process data and input commands.

The list of top 10 manufactures of HMI:

- Honeywell International Inc.
- Red lion.
- Rockwell Automation Inc.
- Schneider Electric SE
- ABB Ltd.
- General Electric SE.
- Emerson Electric Co.
- Siemens
- itsubishi Electric Corporation.

How to choose an HMI?

When choosing a Human Machine Interface (HMI) for an industrial process or system, there are several factors to consider to ensure that the HMI is the best fit for the specific application.

1) Compatibility: It is important to choose an HMI that is compatible with the equipment and systems that it will be controlling. This includes compatibility with the control system, communication protocols, and data formats used by the equipment.

2) Hardware: The HMI should be designed to be rugged and durable, suitable for the environment where it will be installed. It should be able to withstand the temperature, humidity, and vibration that it will be exposed to.

3) Display: The HMI should have a display that is clear, easy to read, and easy to navigate. The screen should be of a good resolution, with a high-quality display and backlight.

4) Usability: The HMI should be easy to operate and understand, with intuitive navigation and clear, concise displays. It should be easy to use and understand by operators with different levels of expertise

5) Data visualization : The HMI should be able to display data in a clear and meaningful way, using graphics, charts, and other types of visual representations to help operators understand the process and make informed decisions.
6) Customizability: The HMI should be customizable, with the ability to adapt to the specific needs of the process or equipment, as well as to the preferences of the operator.
7) Remote access: The HMI should be designed to allow remote access so that operators can monitor and control the process from a remote location.
8) Scalability: The HMI should be designed to be scalable, with the ability to handle more complexity and data as the process or equipment becomes more advanced.
9) Support: The HMI vendor should provide good technical support and documentation, as well as software updates and upgrades.
10) Cost: The HMI should be cost-effective and provide a good return on investment.

5

Node-RED Implementation

Introduction to Node-RED:

Node-RED is a flow-based development tool for visual programming, developed by IBM. It is especially useful for wiring together hardware devices, APIs, and online services in new and interesting ways. In this project, Node-RED plays a central role in integrating different components of the smart parking system — such as sensors, logic control, and cloud connectivity — using a low-code interface.

Why Node-RED Was Used:

Node-RED was chosen for this project because of its:

- Ease of integration with IOT devices and protocols (e.g., MQTT, HTTP, Modbus, etc.).
- Graphical interface that simplifies the development and debugging of automation logic.
- Compatibility with cloud services and databases.
- Real-time monitoring and dashboard capabilities.

System Architecture Using Node-RED:

In the Smart Parking System, Node-RED is responsible for:

- Reading sensor data (e.g., ultrasonic sensors to detect car presence).
- Controlling the parking logic (e.g., managing lift movement, parking slot assignment).
- Visualizing system status through a live dashboard.
- Sending/receiving data from the PLC (using Modbus or HTTP depending on your setup).
- Connecting to a cloud platform for data logging or remote monitoring.

Flow Description:

The Node-RED flow is divided into several stages:

a) Input Nodes:

Nodes such as Modbus Read or MQTT In are used to gather data from the PLC or sensors.

Input includes sensor status (car detected, slot available), lift position, etc.

b) Processing / Logic Layer:

Function nodes are used to process the data, apply logic (e.g., if slot is free and car is detected, then activate lift).

Conditional statements are written in JavaScript inside function nodes to manage the parking rules.

c) Output Nodes:

Actuator signals are sent back to the PLC (via Modbus Write or digital output).

The live system state is displayed on a custom Dashboard.

Data may also be stored in a cloud database or exported to CSV.

Dashboard Interface:

A user-friendly dashboard was built using Node-RED's UI nodes:

- Real-time slot availability display.
- Lift movement status.
- Entry/Exit gate control.
- System alerts or logs.

Why Node-RED Was Chosen Over SCADE

In the context of this Smart Parking System, Node-RED was selected instead of SCADE for several key reasons related to flexibility, usability, and integration capabilities.

1. Ease of Use and Rapid Development:

Node-RED offers a drag-and-drop interface that allows quick prototyping and development of logic flows without the need for complex coding.

SCADE, while powerful for safety-critical systems (like aerospace or automotive), requires more time and expertise due to its model-based design approach and formal methods.

In a student project or prototyping environment, Node-RED enables faster iteration and testing.

2. IOT and Integration Web:

Node-RED is designed for IOT applications, making it easy to:

- Connect to sensors via protocols like MQTT, Modbus, HTTP.
- Send data to cloud platforms.
- Create web-based dashboards with minimal effort.
- SCADE is not built for IoT or web-based visualization, and lacks native support for dashboards or remote monitoring tools.

Node-RED provides a better environment for integrating real-time data visualization and cloud communication.

3. Open Source and Community Support:

Node-RED is open-source, with a large community and thousands of freely available nodes/plugins.

SCADE is a commercial tool, often requiring expensive licenses and specific training.

For educational and budget-limited environments, Node-RED is far more accessible and scalable.

4. Flexibility in Control Logic:

Node-RED supports JavaScript-based function nodes, allowing custom control logic, data processing, and advanced conditions easily.

SCADE, while strong in deterministic behavior and code generation, is less flexible for experimenting or changing logic on-the-fly during prototyping.

5. Real-time Monitoring and Debugging:

Node-RED allows live editing, debugging, and visualization of data in real-time.

SCADE is more oriented toward offline design, simulation, and code generation, with a focus on formal verification rather than live control.

Node-RED was more practical for monitoring parking slot status, lift movement, and sensor feedback live.

Conclusion:

While SCADE excels in certified, high-integrity systems like aviation or railway control, it would have added unnecessary complexity and cost to this smart parking project. Node-RED, on the other hand, provided:

- Faster development.
- Rich IoT integrations.
- Visual dashboards.
- Real-time control.

This made it the ideal choice for a flexible, scalable, and visually interactive smart parking system.

Simplicity of Programming with Node-RED:

One of the main reasons Node-RED was selected for this project is its exceptional simplicity in programming. Unlike traditional development environments that require writing long and complex code, Node-RED offers a drag-and-drop interface that allows users to build logic using visual blocks called nodes.

This simplicity has several key benefits:

- No deep coding knowledge required: Even team members with minimal programming experience could contribute by connecting logic visually, reducing the learning curve significantly.
- Faster development cycles: Complex logic such as sensor reading, decision-making, and output control can be developed and tested in minutes.
- Visual Debugging: Node-RED allows real-time visualization of data as it flows through the nodes, making it extremely easy to trace errors and fix issues quickly.
- Reusability and Modularity: Once a flow is created, it can be exported and reused in other parts of the project or shared across teams.
- Custom scripting when needed: For advanced users, Node-RED supports JavaScript within Function nodes and also allows the execution of external scripts (e.g., Python) when necessary.

This ease of programming made Node-RED the ideal platform for developing, testing, and deploying the automation logic in the Smart Parking System, especially in a university or prototyping context where quick iterations and experimentation are key.

User Authentication and Car Matching Logic

A key feature of our Smart Parking System is the user authentication mechanism, which ensures that only authorized users can access their vehicles and interact with the system during both entry and exit phases.

Entry Phase:

When a user arrives and wants to park their vehicle, they interact with the Node-RED dashboard through a user interface. The following steps occur:

1. The user enters their username and password via a secure dashboard input.

2. This data is then transmitted to the PLC, which temporarily holds it in specific memory registers.

3. A Python script, running within the Node-RED flow, reads this data from the PLC and checks its validity against a predefined database (local or cloud-based).

4. If the credentials are valid:

- The system checks for available parking slots.
- A command is sent to the parking lift and gate to allow the user entry.
- The user's identity is linked with the assigned parking slot and car number for future reference.

Exit Phase:

When the user returns to retrieve their car:

1. They are prompted again to enter their username and password.

2. The credentials are sent to the PLC, then read by Python, which verifies the information.

3. The system checks whether the car associated with this user is currently parked.

4. If a match is found:

- The system automatically identifies the parking slot and floor where the car is located.
- A command is issued to bring the car down using the lift.
- The exit gate is opened, and the car is released.

Benefits of This Approach:

- Ensures secure access to vehicles.
- Prevents unauthorized retrieval of parked cars.
- Facilitates personalized automation, where the system knows exactly which car belongs to which user.
- Integrates PLC memory usage with Python and Node-RED, combining industrial reliability with modern automation flexibility.

This seamless collaboration between PLC, Python, and Node-RED dashboards creates a secure, efficient, and user-friendly smart parking experience.

The Role of Node-RED in Industry 4.0:

Node-RED plays a pivotal role in Industry 4.0 by serving as a flexible, low-code integration platform that connects devices, sensors, software systems, and cloud services across industrial environments. It acts as a middleware layer that empowers engineers and developers to design, prototype, and deploy smart manufacturing solutions quickly and efficiently.

1. Data Acquisition & Real-Time Monitoring:

Node-RED excels at collecting data from a wide range of sources:

Industrial Protocols: Supports Modbus, OPC-UA, MQTT, and more for PLC communication.

IOT Devices: Reads sensor data via HTTP, MQTT, BLE, LoRaWAN, etc.

Legacy Systems: Bridges older equipment with modern infrastructure.

This makes it ideal for creating real-time dashboards and monitoring solutions without needing complex coding.

2. Edge Computing & Decision Making:

Node-RED enables edge intelligence by running on low-power edge devices (like Raspberry Pi, industrial PCs, or gateways). This allows localized decision-making close to the machines, reducing latency and bandwidth use.

Example: If a temperature sensor detects overheating, Node-RED can autonomously trigger a cooling system without needing cloud approval.

3. System Integration & Interoperability:

One of Industry 4.0's biggest challenges is system fragmentation. Node-RED acts as a universal connector:

- Integrates ERP systems (e.g., SAP).
- Connects MES, SCADA, PLCs, and IOT platforms.
- Enables cloud connectivity with AWS, Azure, or IBM Watson.

Its drag-and-drop logic simplifies creating digital twins, predictive maintenance workflows, and production analytics pipelines.

4. Cloud and AI Integration:

Node-RED can send data to and receive insights from AI/ML models hosted in the cloud. It bridges industrial equipment with:

- Machine learning inference APIs.
- Cloud databases (Firebase, InfluxDB, MongoDB).
- AI-based anomaly detection systems.

This supports predictive maintenance, quality control, and energy optimization.

5. Security & Scalability:

While lightweight, Node-RED supports:

- Token-based authentication.
- TLS/SSL encryption.
- Custom user access control.

It can be scaled using Docker, Kubernetes, or cloud functions, making it ready for industrial-grade deployment.

Summary:

In essence, Node-RED is a glue technology in Industry 4.0 — it doesn't replace control systems but connects and enhances them. It enables:

- Faster integration.
- Rapid prototyping.
- Smart automation.
- Data-driven decisions.
- Its simplicity, flexibility, and extensibility make it a core enabler of smart factories, digital transformation, and cyber-physical systems in the Industry 4.0 ecosystem.

Advantages of Web-Based Node-RED in Industrial Applications:

- Node-RED's web-based development environment is one of its most powerful and practical features. It transforms the way engineers, developers, and operators design, monitor, and manage automation logic-directly from a browser, without the need for heavy IDEs or proprietary software.
- Accessibility & Remote Management.
- Being browser-based, Node-RED can be accessed from any device on the same network (or securely via VPN/cloud).
- Enables remote monitoring and control of factory systems, which is crucial for distributed operations and Industry 4.0.
- Supports remote diagnostics and real-time troubleshooting-minimizing downtime.
- No Installation Required on Client Devices.

  Since Node-RED runs on a central server (e.g., industrial PC, Raspberry Pi,or edge gateway), users don't need to install any application locally.

- Ideal for multi-user environments where multiple engineers or technicians may need to access the system from different workstations.
- Real-Time Dashboard Creation.
- The web interface allows rapid development of real-time dashboards to visualize data, control devices, and interact with processes.
- Users can create interactive UI components (buttons, charts, gauges) with zero front-end coding.
- Rapid Prototyping and Deployment.
- Drag-and-drop flow creation in a browser accelerates development cycles.
- Changes can be deployed instantly and observed in real time-perfect for iterative design and testing in R&D or production settings.
- Multi-Platform and Cross-Browser Support.
- Compatible with all major browsers (Chrome, Firefox, Edge).
- Works across platforms (Windows, macOS, Linux, even mobile/tablets), giving engineers the freedom to work from their preferred device.
- Collaboration and Transparency.
- Engineers can share flows or collaborate on logic design by simply exporting JSON flows.
- The visual nature of the web interface helps both technical and non-technical stakeholders understand the automation logic clearly.
- Integrated Debugging Tools

- The debug pane in the web UI provides live feedback, making it easier to trace data flow, identify errors, and validate logic.
- Users can inspect payloads, headers, and node status live as the system runs.

In Industrial Contexts:

- Web-based Node-RED offers centralized control with decentralized access-critical for smart factories and IIoT environments.
- It supports the plug-and-play philosophy of Industry 4.0: scalable, modular, and cloud-ready.

Node-RED Nodes Used in the Smart Parking System:

Our Smart Parking System integrates multiple technologies including PLC (for automation), Python (for logic and identity verification), MQTT (for lightweight messaging), and databases (for storage and validation). Below is an organized overview of the Node-RED nodes used to make this system function seamlessly:

1. PLC Communication Nodes:

S7-comm-read & S7-comm write.

- Purpose: Exchange data with the PLC (e.g., sensor input, parking slot status, lift commands).
- Library: node-red-contrib-s7-comm.
- Functionality:
  - Read data from PLC coils/registers (e.g., car detected, floor sensors).
  - Write control signals (e.g., move lift, open gate).

2. Python Integration Nodes :(exec Node)

- Purpose: Run external Python scripts for complex logic or database handling.
- Use Case in Project:

  Read credentials from PLC → send to Python → validate against database.

  Match car ID to user using Python-based logic.

- Example:

  Python 3 verify-user by username password.

3. MQTT Communication Nodes:

mqtt in / mqtt out

- Purpose: Publish and subscribe to topics for lightweight messaging between components.
- Functionality:
  - MQTT topics used for sending real-time system updates to dashboards or mobile apps.
  - Can be used to sync data between multiple Node-RED instances or microcontrollers.

4. Database Interaction Nodes

mysql / mongodb / sqlite Node (depending on your DB).

- Purpose: Store and retrieve data like:
  - User credentials.
  - Entry/exit logs.
  - Parking slot assignments.
- Functionality:
  - Query database when verifying user identity during car exit.
  - Insert new records when cars enter/exit.
- Function Node:

Prepares SQL queries or processes results from the DB before outputting to dashboards or PLC.

5. Dashboard Nodes

UI-text, UI-input, UI-button, UI-chart, etc.

- Purpose: Create user interfaces for:
  - Login forms (username/password input).
  - Real-time car movement visualization.
  - Displaying slot availability or system state.

6. Utility Nodes

inject, debug, function, switch, template.

These are general-purpose nodes used to:

- Inject test data.
- Debug and trace flow execution.
- Create custom logic.
- Route messages conditionally.

Conclusion of this chapter:

Node-RED significantly simplified the development process for the Smart Parking System. It served as a powerful integration tool that connected sensors, logic controllers, user interfaces, and cloud platforms seamlessly. The use of Node-RED also enhanced the system's flexibility and scalability, making future expansion or cloud integration much easier.

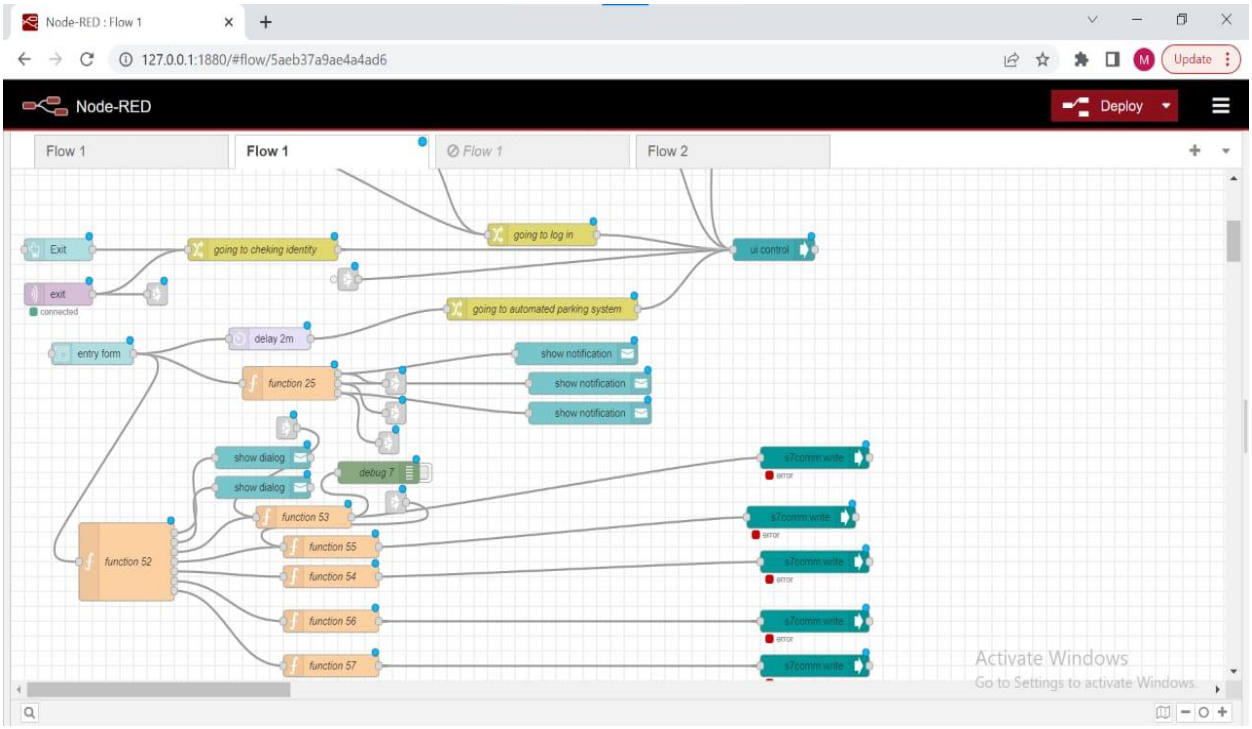


Figure 1

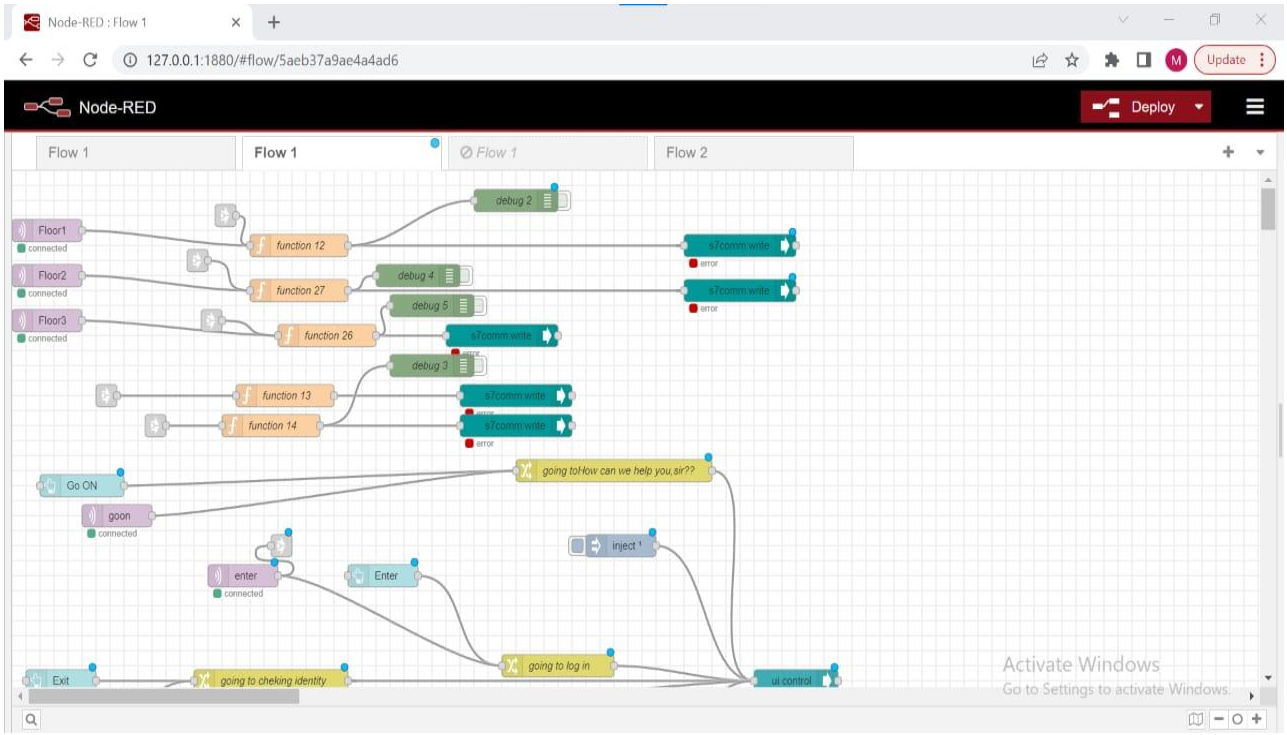


Figure 2

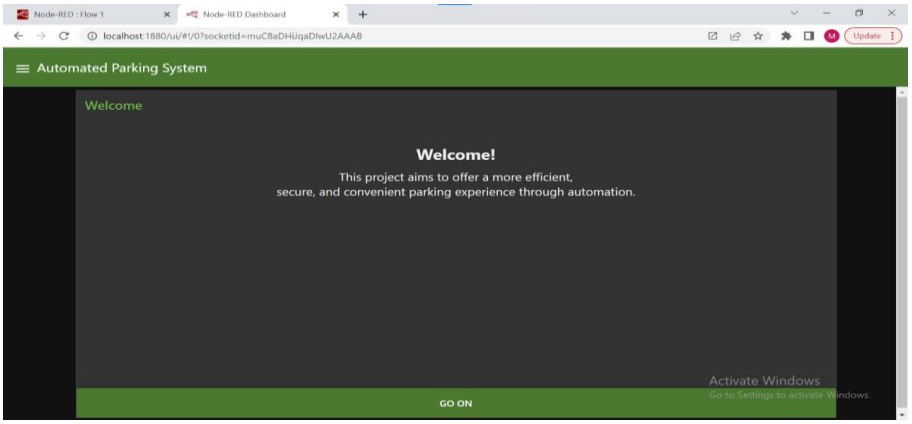


Figure 3

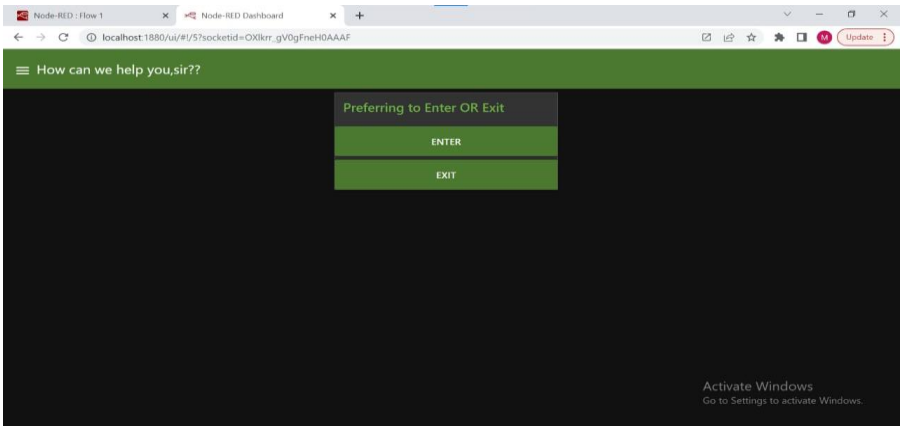


Figure 4

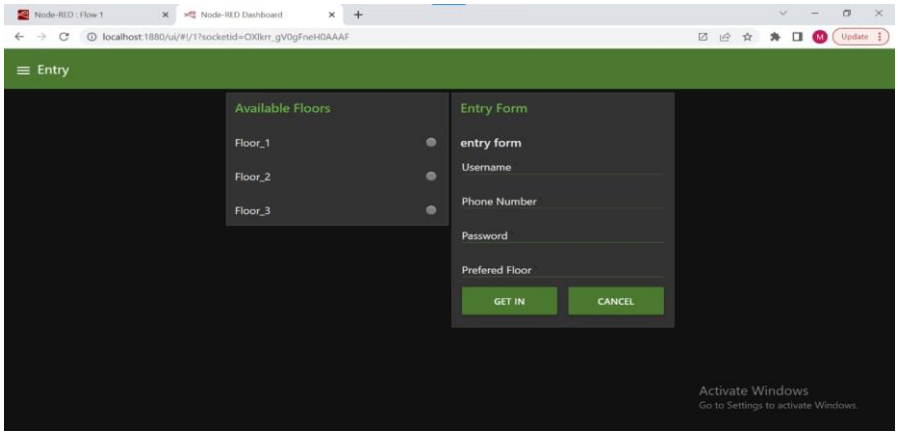


Figure 5

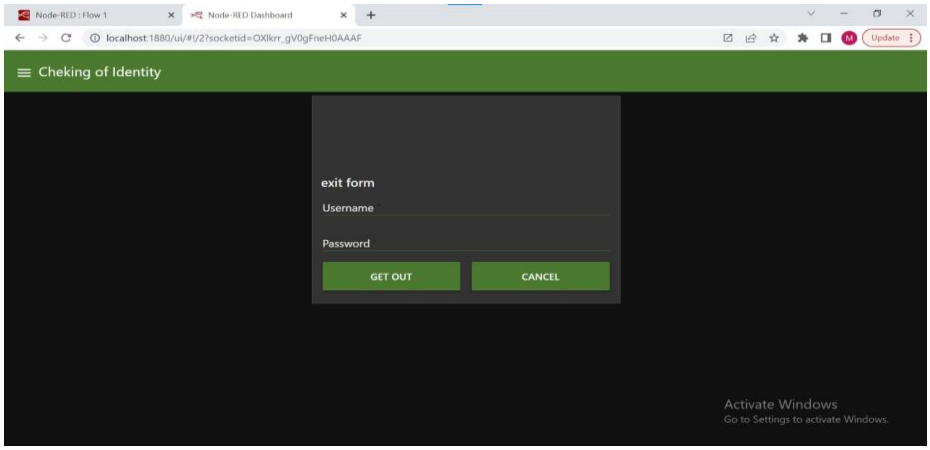


Figure 6

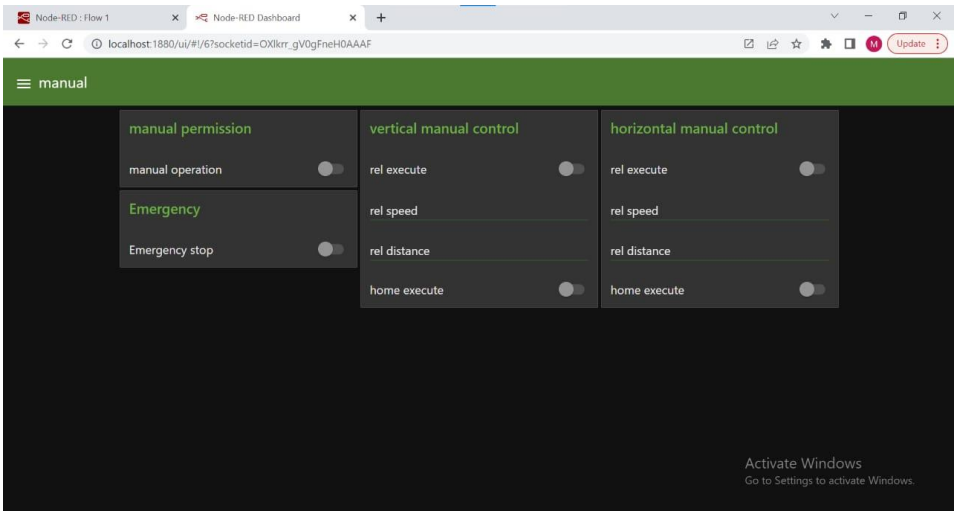


Figure 7

6

SCADA System

The SCADA system (Supervisory Control and Data Acquisition) is one of the essential systems in the field of industrial automation and intelligent control. It is widely used in large facilities such as factories, infrastructure networks, power stations, transportation systems, and construction sites. SCADA enables operators to monitor and control industrial operations remotely, contributing to improved efficiency, reduced downtime, and enhanced safety and stability.

SCADA works by collecting real-time data from various parts of the system and displaying it through a graphical supervisory interface. This allows users to track the system's performance and make immediate decisions in case of any malfunctions or alarms. The system also logs and analyzes data over time, helping improve performance and plan for future maintenance.

Key features of SCADA include:

- Real-time monitoring of all system components.
- Remote control of operations without the need for on-site presence.
- Visual data representation through graphs and dashboards for easier understanding and decision-making.
- Automatic alerts in case of system faults or abnormal readings.
- Data logging and reporting for analysis documentation and continuous improvement.

In this project, SCADA was used to display the overall status of the automated parking system and monitor the movement of vehicles across different levels.

It also provided alerts in case of errors or abnormal stops. This integration helped enhance system performance and reliability, and simplified the supervision and operation process for users and maintenance teams.

Often, SCADA will also have a separate component called HMI (human-machine

interface). Altogether, SCADA is a system that has software and hardware elements.

Programmable Logic Controller (PLC):

A Programmable Logic Controller (PLC) is considered an industrial "mini-computer" capable of communicating with various devices, systems, and machines within an operational environment. It is specifically designed to perform control tasks in industrial processes, as well as to detect faults and assist in real-time troubleshooting.

PLCs are widely used in industrial automation systems and are considered essential components of SCADA systems. They are responsible for executing commands and directly controlling field devices based on data received from the system or the operator. Known for their speed and reliability, PLCs are ideal for managing continuous and precise operations.

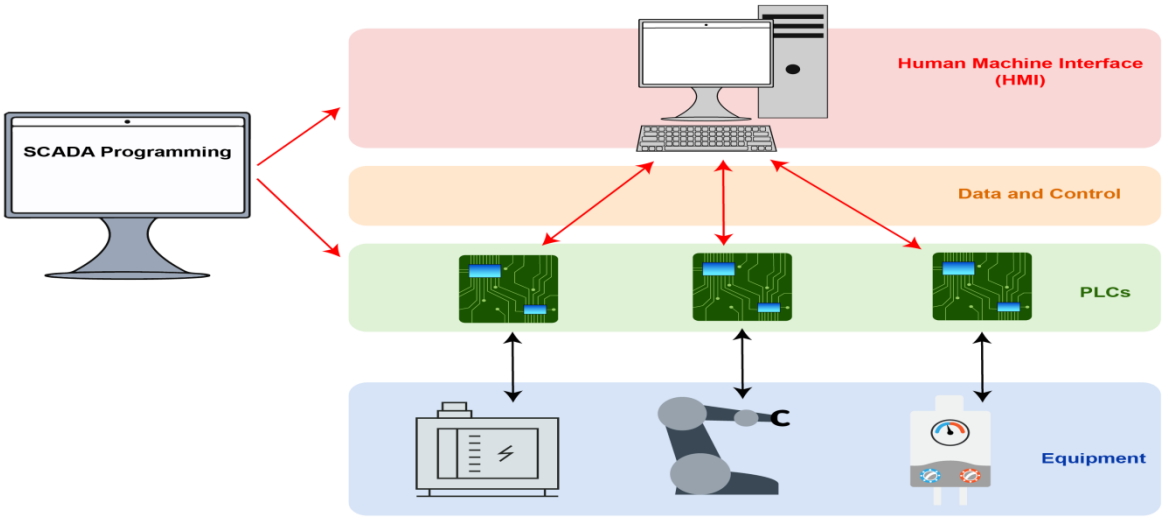


Figure 1-1 SCADA System

This system simplifies machine usage while also providing necessary functions and are used for any situation that requires high-reliability control by a human operator. Plus, they are an intricate part of industrial automation, and SCADA systems cannot function without them.

Some Primary Functions of PLCs within SCADA Systems Include:

- Controlling industrial robots.
- Managing assembly and production lines.
- Operating and managing counters and timers.
- Reading and processing various measurements such as pressure and temperature.
- Performing logical and mathematical operations related to system tasks.

- Interfacing with sensors and interpreting their signals.
- Controlling motors with specific speeds and directions.
- Switching lights on and off as required.
- Operating fans or ventilation systems.
- Monitoring and controlling circuit breakers to ensure safety.

These diverse and integrated functions are essential in many industrial applications. In our automated parking system project, several of these functions were applied such as platform movement control, motor operation, and system status monitoring contributing to the creation of a smart, efficient and reliable system.

How do SCADA and PLC work together?

SCADA is most easily understood as the central database that stores and controls your operations at large. This communicates with the PLC, which is how the system functions and communicates within the operational environment.

Figure 1-2 Control room building

The Role of the SCADA System in the Automated Parking Project:

In this project the SCADA system was utilized as the central platform for control and monitoring within the automated parking system. It played a key role in coordinating various system operations, such as managing vehicle movement, organizing levels, and tracking the real-time operational status.

One of the primary applications implemented through SCADA was the vehicle entry and exit process where the system automatically detects car movement and updates the availability of parking slots in real time. This process was visualized through carefully designed graphical interfaces that clearly displayed the structure of the parking system and platform status.

In addition an interactive user login interface was developed within the SCADA system. Each user could enter their username and password to access the system securely. This

feature improved system security and enabled the assignment of specific access privileges, allowing control and monitoring to be limited based on user roles.

The system also included animations that visually represented the movement of vehicles during the parking and retrieval processes. These animations enhanced the user experience and provided a more intuitive understanding of system operation, even for non-technical users.

The interface was further supported with clearly labeled graphical elements such as digital buttons, counters, and indicators showing vehicle positions, making it easy for operators to interact with the system without the need for deep technical knowledge.

The integration of SCADA with user login functionality and animated visualizations of vehicle movement reflects the project's advanced level of development. It demonstrates the system's readiness for real-world application, where precision, security and flexible control are essential.

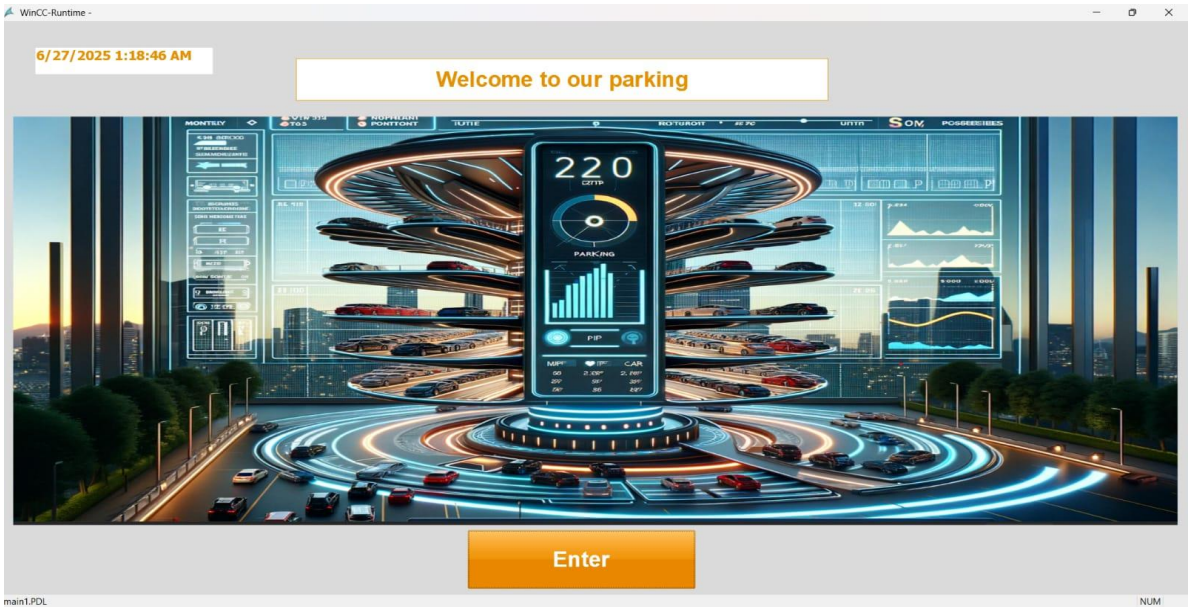


Figure 1

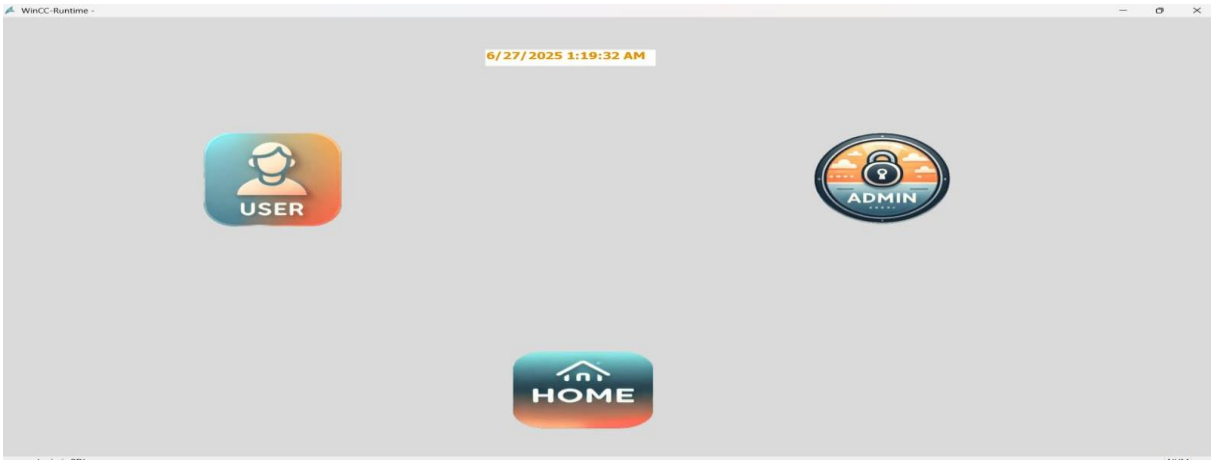


Figure 2

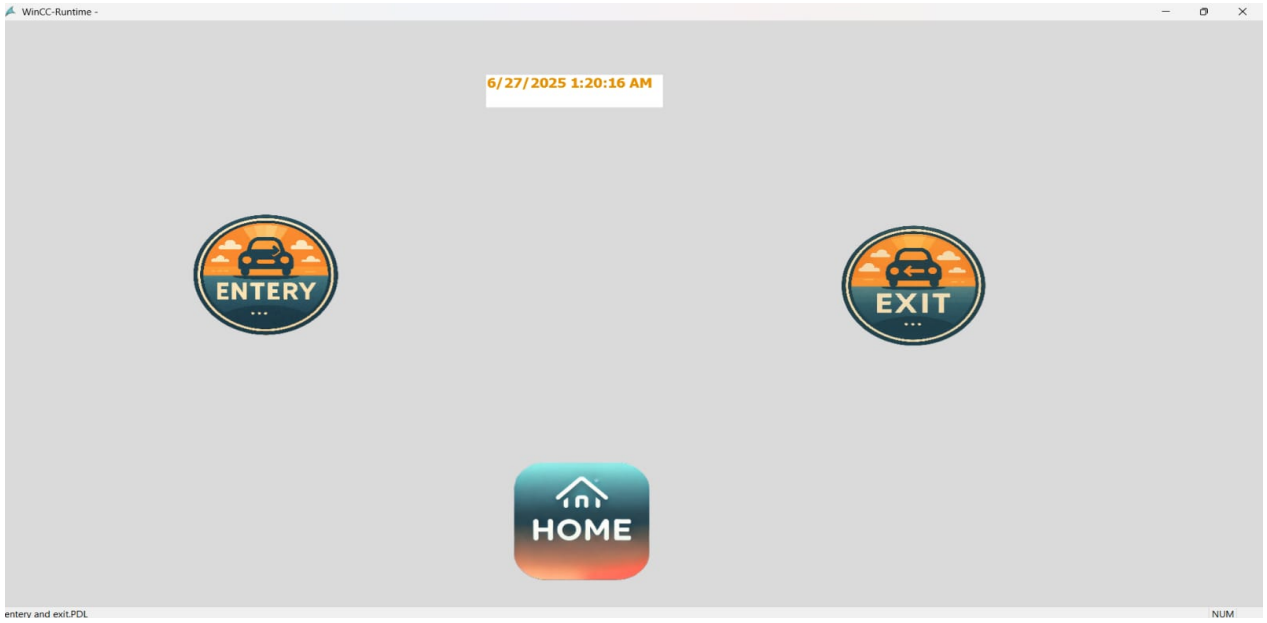


Figure 3

Figure 4

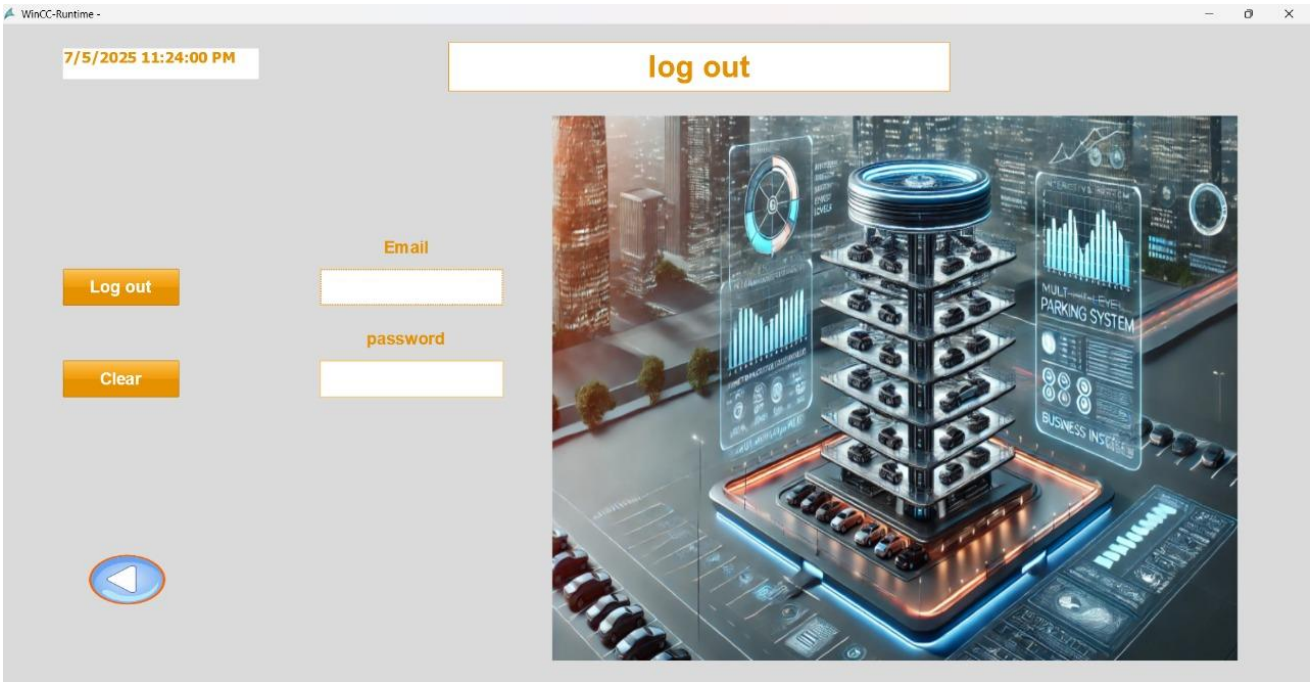


Figure 5

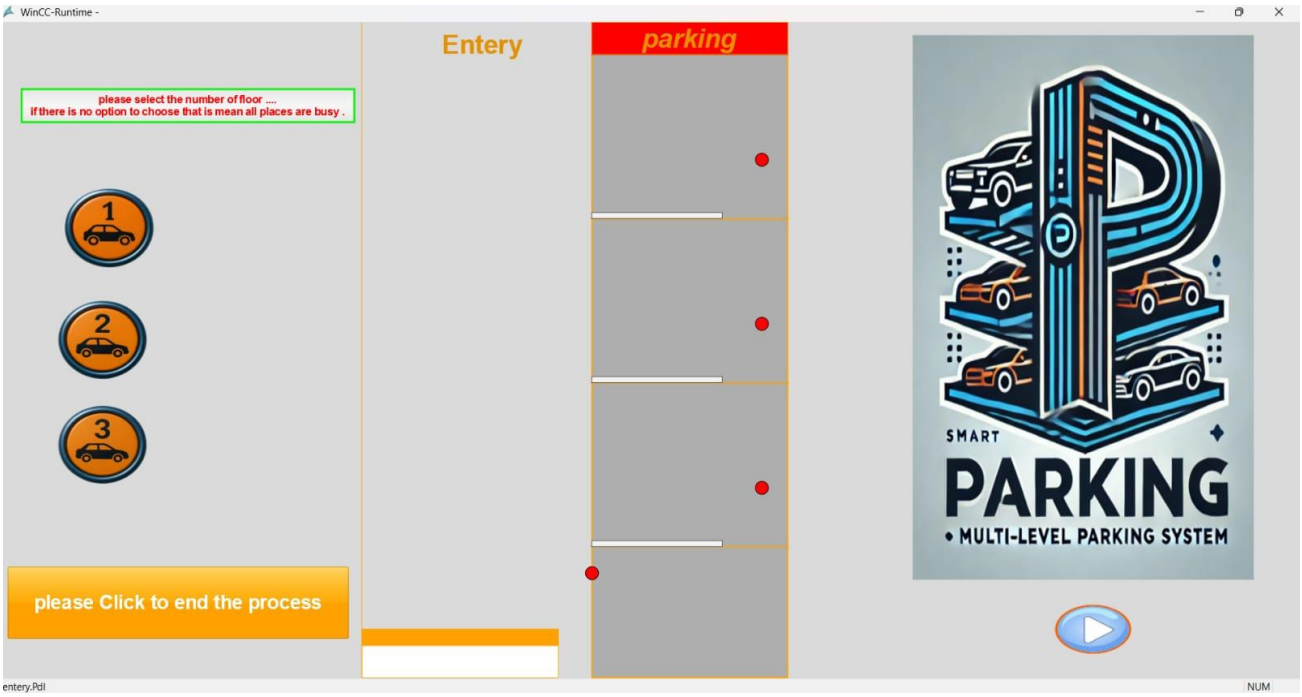


Figure 6

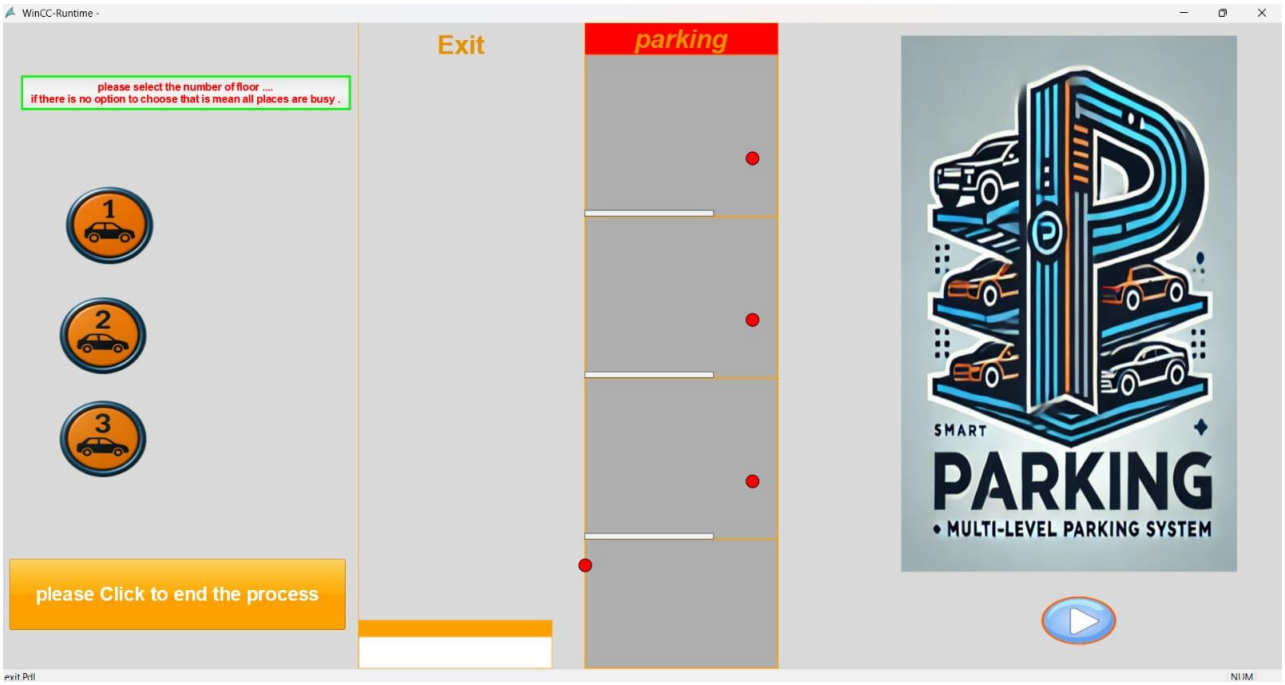


Figure 7

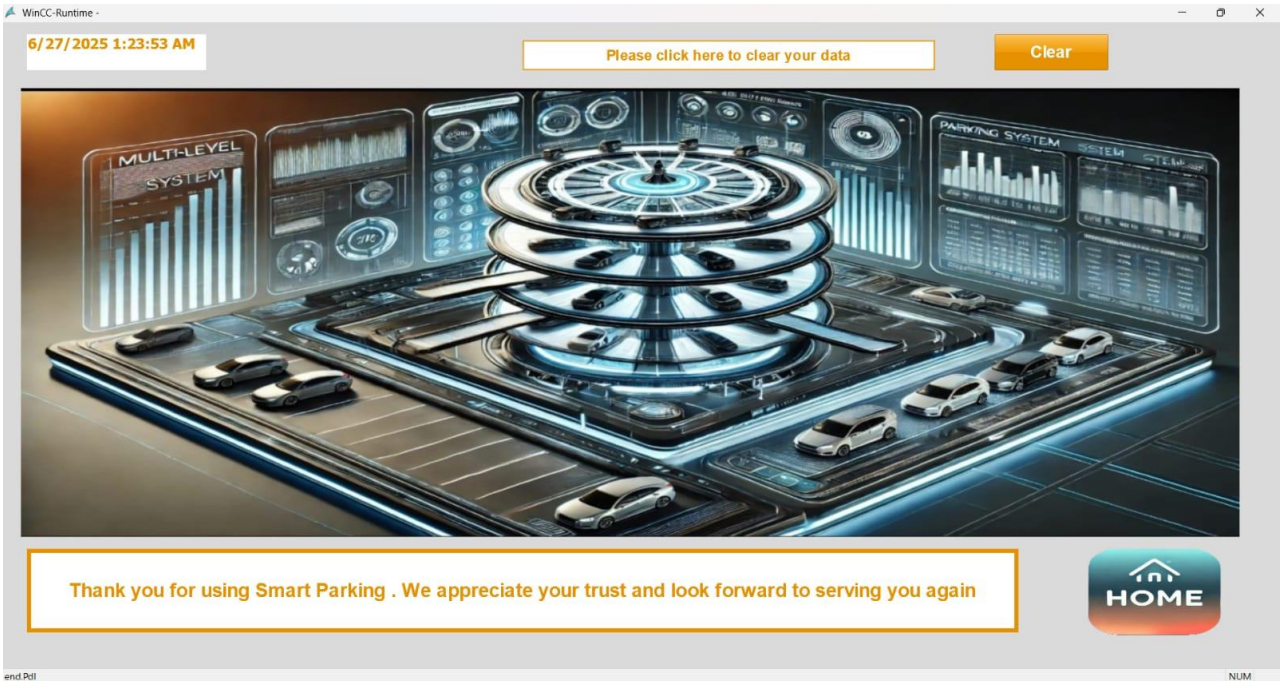


Figure 8

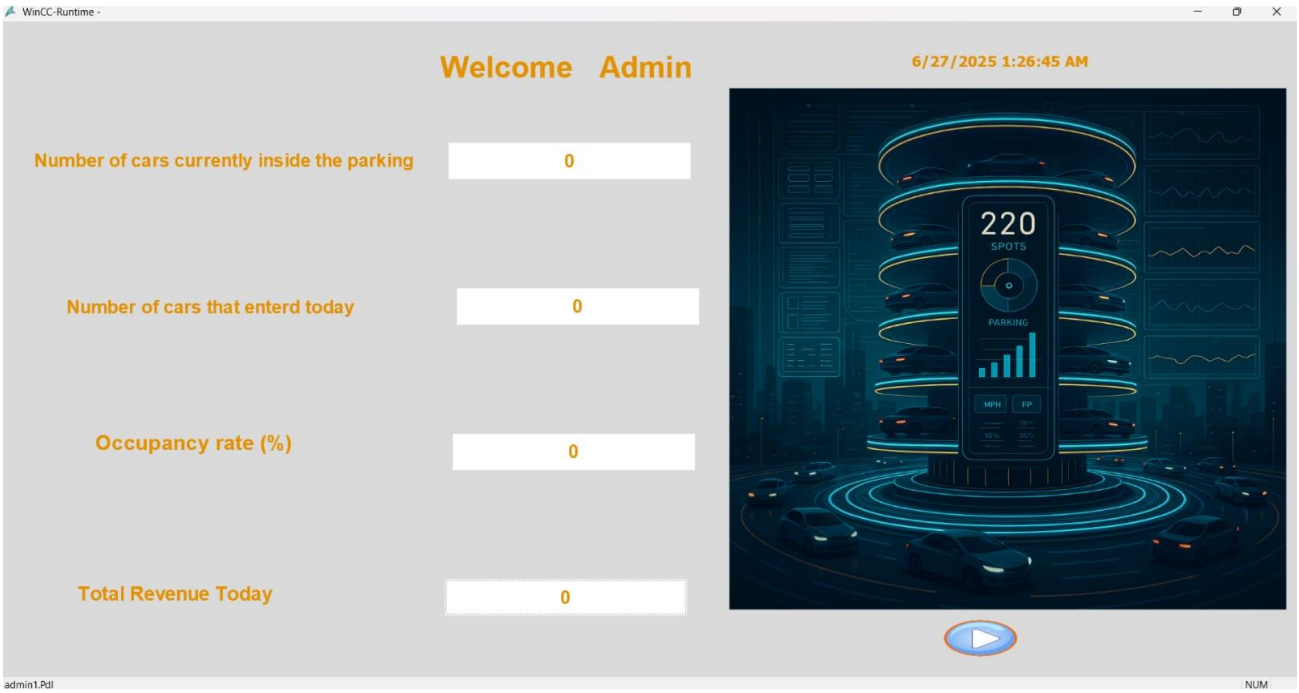


Figure 9

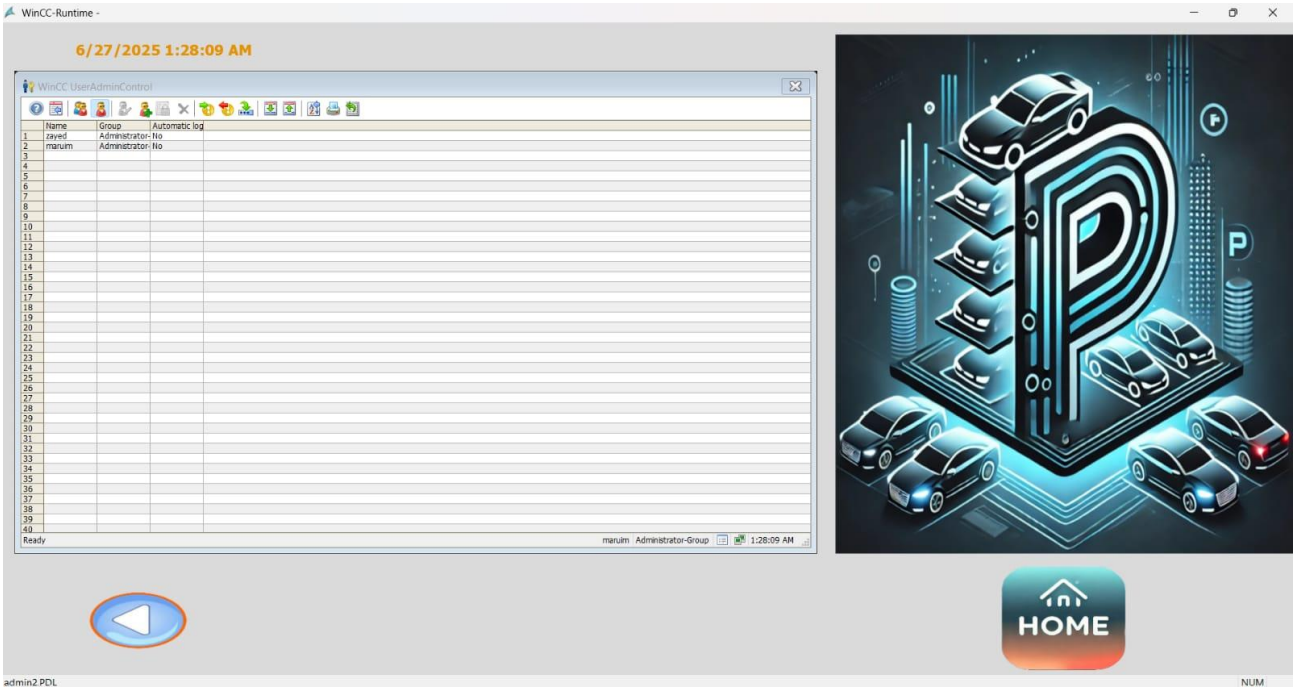


Figure 10

7

Database

Introduction:

In the era of smart infrastructure, automation is no longer a luxury-it is a necessity. As cities grow denser and vehicle ownership increases, conventional parking methods have proven inefficient and prone to human error. The Smart Garage system seeks to address these inefficiencies by automating the process of vehicle entry, exit and parking allocation. Central to the operation of any intelligent system is the ability to manage, store, and retrieve data reliably and efficiently. This is where the role of a structured relational database becomes indispensable.

In any smart system, especially in one as dynamic and real-time dependent as a Smart Garage, data integrity, consistency, and speed are not just technical goals but functional necessities. The ability to track users, register vehicles, allocate parking spaces, monitor active sessions, and control access through automated gates depends heavily on a solid database architecture.

This chapter presents a detailed academic examination of the relational database designed for the Smart Garage system. It explains not only the technical structure of the database but also its significance, the rationale behind its architecture, and the problems it prevents. Through this exploration, the chapter aims to demonstrate the pivotal role that structured data plays in the operation, reliability, and scalability of smart systems.

Challenges Faced Without a Database System:

Before we explore the architecture and function of the database itself, it is important to understand the implications of designing a Smart Garage system without a dedicated

database. If the system were to rely solely on real-time input/output operations or temporary memory (e.g., using RAM or session-based storage), it would suffer from several serious limitations.

1) **Data Volatility:**

Without persistent storage, all information regarding users, vehicles, and sessions would be lost the moment the system shuts down or reboots. This would require re-registration of users and vehicles every time, making the system practically unusable.

2) **Inability to Handle Multiple Users Simultaneously:**

In the absence of structured data access, it becomes exceedingly difficult to handle concurrent operations by multiple users. Race conditions could arise when multiple users attempt to enter or exit simultaneously, leading to potential system crashes or erroneous parking assignments.

3) **No Record of Parking History:**

A smart garage without a database cannot store historical session data. This limits the ability to analyze usage patterns, verify billing logs, enforce security, or resolve disputes related to parking durations or access history.

4) **Inconsistency and Redundancy:**

Hardcoding or file-based alternatives often lead to data redundancy and inconsistency, especially when users register multiple times or when vehicles are not correctly linked to their respective owners. Databases solve this problem through normalization and referential integrity.

5) **Limited Scalability:**

Any system not backed by a robust database design will face difficulty in scaling. Adding new features-such as tiered access control, real-time analytics, or cloud integration-would become complicated and fragile without a reliable data backend.

In essence, the absence of a well-designed relational database compromises not only the stability and functionality of the system but also its ability to grow and integrate with other technologies.

The Role and Importance of the Database in Smart Garage Architecture:

The Smart Garage system integrates hardware components such as sensors, gates, and microcontrollers with software elements like user interfaces, registration portals, and

control logic. At the heart of this interaction lies the SQL Server-based relational database, which acts as the central nervous system of the application.

1) **Centralized Data Management:**

The database centralizes information about all entities: users, vehicles, parking slots, and session logs. This ensures that all decisions—whether to open the gate, allow a car to park, or bill a user—are made based on real-time, verified data.

2) **Relationship Enforcement Through Foreign Keys:**

Each vehicle is linked to a user, and each session is linked to a vehicle. These relationships are explicitly enforced using foreign key constraints, which ensure that invalid or orphaned records cannot exist in the system. For example, a vehicle entry cannot exist without an associated user.

3) **Integrity and Security:**

The database ensures that user credentials, vehicle plate numbers, and parking assignments are stored securely and retrieved efficiently. Fields like user-password are constrained by data type, and plate numbers are marked as unique to avoid duplicate entries.

4) **Real-Time Decision Support:**

When a vehicle attempts to enter the garage, the system checks whether:

- The user is registered.
- The vehicle is associated with the user.
- The parking slot is available.
- The vehicle has already entered and not exited.

All these checks are possible only because the database provides fast and structured access to current and historical data.

5) **Fault Tolerance and Recovery:**

In case of power loss or system crash, the database ensures that all records remain intact. Once the system reboots, the previous state can be reconstructed accurately from the database tables, allowing uninterrupted.

Conclusion:

With the rapid expansion of urban areas and the continuous increase in the number of vehicles, modern cities are facing growing challenges in managing traffic congestion and

providing adequate, safe parking spaces. This project presents an innovative and efficient solution through the design and implementation of a *multi-level automated parking system*, aimed at improving space utilization and facilitating the parking and retrieval process in a structured and intelligent manner.

The system is based on the concept of vertical space optimization, where vehicles are arranged across multiple levels and moved automatically without the need for direct human intervention. Cars are transported vertically or horizontally to assigned parking slots, ensuring organized storage and easy access when needed. The system relies on accurate mechanisms to monitor the movement and position of vehicles within the structure, contributing to high performance and minimal errors.

The project was executed through several phases, including mechanical design, system control, monitoring, and final prototype testing. Key factors such as safety, stability, speed, and operational accuracy were carefully considered to ensure the system performs reliably under real-world conditions. The final model is user-friendly and designed to be scalable, allowing for future enhancements such as vehicle counting, reservation features, or integration with other infrastructure systems.

This project serves as a practical model that can be implemented in high-traffic areas such as shopping malls, airports, hospitals, and administrative buildings, especially where ground space is limited. It reflects the ability of engineering students to address real-life problems by developing smart, technically sound solutions that contribute to building a more efficient, sustainable, and intelligent urban infrastructure.

8

Python & Machine Vision

PLC– Database Communication and Automation Logic

● Overview:

In this part of the smart parking system, we developed a Python-based middleware that facilitates real-time bi-directional communication between a Siemens PLC and a Microsoft SQL Server database. This system manages car entry and exit operations, error handling, user authentication and QR code generation.

The integration is achieved using:

- **Snap7** for communicating with the Siemens PLC over TCP/IP.
- **pyodbc** for executing SQL commands and stored procedures.
- **PIL** and **qrcode** for creating custom QR codes.
- Internal logic to set/reset PLC flags, update database records, and manage entry/exit triggers.

● System Functions:

1. Database and PLC Initialization:

- **SQL Server Connection** is established to a database called parking using trusted Windows authentication.
- **PLC Connection** uses Snap7 to connect to the IP address 192.168.0.1, accessing DB49 for data exchange and other DBs for control flags.

2. Entry Process Logic:

The entry process function is triggered when (DB1.DBX1.7) is set (Entry trigger).

The logic includes:

- Reading the following from **DB49**:
  - username: stored at DBX0.0
  - password: DBX256.0
  - position: DBW512
  - user phone: DBX520.0
- Input validation is applied (e.g., non-empty credentials, valid position).
- Data is inserted into the **USERS** table, and a stored procedure vehicle_entry is executed to register the session.
- The **QR code** is generated and saved using PIL and qrcode.
- Error codes and flags are written back to the PLC:
  - DBW516: Status code (0 = OK, 4 = Invalid data).
  - DBX518.0: Set True on success.
- **QR Code Sample Content:**

```
Username: manar
Password: hosney
Position: 1
```

- **The code is saved as an image with the filename pattern:**

qrcodes/manar_38_pos1.png

PLC input trigger bit is reset after processing.

3. Exit Process Logic

The exit process function is triggered when (**DB1.DBX2.0)** is set.

The logic includes:

- Reading username and password from DB49.
- Verifying credentials against the database (USERS table).

- Fetching the latest active parking session from Parking_Session.
- Calling the vehicle_exit stored procedure to finalize the session.
- Updating **DB16** to reflect the new parking spot status.
- Writing back:

➢ Final position (DBW512)

➢ Error codes (DBW516) based on mismatch or missing session:

- 0: Success.
- 1: Wrong password.
- 2: Wrong username.
- 3: No session or user not found.

- Resetting the **output trigger bit** (DB1.DBX2.0).
- If another car is waiting (from DB44.DBX10.1), a **start pulse** is sent to DB16.DBX0.0 to activate automatic intake.

● **Error Handling & Status Feedback**:

The system includes robust error detection and reporting at both the entry and exit levels.

Code	Meaning	DB Location
0	Success	DB49.DBW516
1	Password mismatch	DB49.DBW516
2	Username mismatch	DB49.DBW516
3	No active session or user not found	DB49.DBW516
4	Invalid or empty input	DB49.DBW516

This allows the HMI, nodered and scada to give visual alerts or messages to the user/operator.

● **PLC Flags and Tag Management:**

➢ **PLC → Python:**

Purpose	Address	Type
Entry Trigger	DB1.DBX1.7	Bool
Exit Trigger	DB1.DBX2.0	Bool

➢ **Python → PLC:**

Purpose	Address	Type
Start Moving Flag	DB49.DBX518.0	Bool
Error Codes	DB49.DBW516	Int
Position Returned	DB49.DBW512	Int
Occupancy Tags	DB16.DBX0.1–3	Bool
Pulse to Position 0	DB16.DBX0.0	Bool

- Each tag is set using set_bool () and written with plc.db_write ()

● **Real-Time Trigger Monitoring**

A while loop constantly checks for rising edges on trigger bits:

```
input_trigger = bool (input_byte [0] & (1 << 7))
output_trigger = bool (output_byte [0] & (1 << 0))
```

If any change is detected, the corresponding process (entry_process () or exit_process ()) is called. Delay of 0.1 seconds is added to prevent rapid re-triggering.

Python code:

Database communication:

```python
import snap7
from snap7.util import get_string, get_int, set_int, set_bool
import pyodbc
import time
import qrcode
import os
from PIL import Image, ImageDraw, ImageFont
# ---------------- Database Connection ----------------
conn = pyodbc.connect(
    'DRIVER={SQL Server};'
    'SERVER=DESKTOP-ET7IGUU\WINCC;'
    'DATABASE=PARKER;'
    'Trusted_Connection=yes;'
)
cursor = conn.cursor()
# ---------------- PLC Connection ----------------
plc_ip = '192.168.0.1'
plc = snap7.client.Client()
plc.connect(plc_ip, rack=0, slot=1)
db_number = 49
# ---------------- Set Start Moving Flag ----------------
def set_start_moving():
    try:
        data = bytearray(plc.db_read(db_number, 518, 1))
        set_bool(data, 0, 0, True)
        plc.db_write(db_number, 518, data)

        print("[PLC] Start Moving flag (DBX518.0) set to True.")
    except Exception as e:
        print(f"[PLC] Error setting start moving: {e}")
# ---------------- Generate QR Code ----------------
def generate_qr_code(data, filename):
    from PIL import Image, ImageDraw, ImageFont
    qr = qrcode.QRCode(version=1, box_size=10, border=5)
    qr.add_data(data)
    qr.make(fit=True)
    img = qr.make_image(fill_color="black", back_color="white").convert("RGB")
    lines = data.split('|')
    try:
        font = ImageFont.truetype("arial.ttf", 20)
    except:
        font = ImageFont.load_default()
    temp_draw = ImageDraw.Draw(img)
    line_heights = []
    max_width = 0
    for line in lines:
        bbox = temp_draw.textbbox((0, 0), line, font=font)
        width = bbox[2] - bbox[0]
        height = bbox[3] - bbox[1]
        line_heights.append(height)
        max_width = max(max_width, width)
    total_text_height = sum(line_heights) + 10
    new_height = img.height + total_text_height
    new_img = Image.new("RGB", (img.width, new_height), "white")
    new_img.paste(img, (0, 0))
    draw = ImageDraw.Draw(new_img)
    y_text = img.height + 5
    for i, line in enumerate(lines):
        bbox = draw.textbbox((0, 0), line, font=font)
        text_width = bbox[2] - bbox[0]
        x_text = (img.width - text_width) // 2
        draw.text((x_text, y_text), line, font=font, fill="black")
        y_text += line_heights[i]
    new_img.save(filename)
# ---------------- ENTRY Process ----------------
def entry_process():
    try:
        data = bytearray(plc.db_read(49, 0, 528))
        username = get_string(data, 0).strip()
        password = get_string(data, 256).strip()
        position = get_int(data, 512)
        usernumber = get_string(data, 520).strip()

        if not username.strip() or not password.strip() or len(password.strip())< 6 or position == 0:
            print("[ENTRY] Invalid input.")
            set_int(data, 516, 4)
            set_bool(data, 518, 0, False)
        else:
            cursor.execute("SELECT user_id FROM USERS WHERE user_name = ?", username)
            existing_user = cursor.fetchone()
            if existing_user:
                print(f"[ENTRY] Username '{username}' already exists. Registration denied.")
                set_int(data, 516, 5)
                set_bool(data, 518, 0, False)
            else:
                print("[ENTRY] Registering user...")
                cursor.execute("""
                    INSERT INTO USERS (user_name, user_phone, user_password)
                    OUTPUT INSERTED.user_id
```

Figure 1 & 2

```
                    VALUES (?, ?, ?)
                """, username, usernumber, password)
                user_id = cursor.fetchone()[0]
                print(f"[ENTRY] user_id = {user_id}")
                print(f"[ENTRY] Executing vehicle_entry for place {position}")
                cursor.execute("EXEC vehicle_entry @chosen_place_id = ?, @user_id = ?", position, user_id)
                conn.commit()
                set_bool(data, 518, 0, True)
                set_int(data, 516, 0)  # no error
                print(f"[ENTRY] Success: user {username} at position {position}")
                set_start_moving()
                # ---------------- Generate QR Code ----------------
                if not os.path.exists("qrcodes"):
                    os.makedirs("qrcodes")

                qr_data = f"Username: {username}|Password: {password}|Position: {position}"
                qr_filename = f"qrcodes/{username}_{user_id}_pos{position}.png"
                generate_qr_code(qr_data, qr_filename)
                print(f"[ENTRY] QR Code generated and saved as {qr_filename}")
        # Clear fields after use
        for i in range(0, 256): data[i] = 0         # username
        for i in range(256, 512): data[i] = 0       # password
        set_int(data, 512, 0)                       # position
        for i in range(520, 527): data[i] = 0       # phone
        plc.db_write(49, 0, data)
        # Reset trigger DB1.DBX1.7
        input_reset = bytearray(plc.db_read(1, 1, 1))
        input_reset[0] &= ~(1 << 7)
        plc.db_write(1, 1, input_reset)
        print("[ENTRY] Input trigger DB1.DBX1.7 reset.")
    except Exception as e:
        print(f"[ENTRY] Error: {e}")
# ---------------- EXIT Process ----------------
def exit_process():
    try:
        data = bytearray(plc.db_read(49, 0, 528))  # DB49 full read
        username = get_string(data, 0).strip()         # DB49.DBX0.0
        password = get_string(data, 256).strip()       # DB49.DBX256.0
        cursor.execute("SELECT user_id FROM USERS WHERE user_name = ?", username)
        user_result = cursor.fetchone()
        cursor.execute("SELECT user_id FROM USERS WHERE user_password = ?", password)
        pass_result = cursor.fetchone()
        error_code = 0
        final_position = 0
```

```
        if user_result and pass_result and user_result[0] == pass_result[0]:
            cursor.execute("""
                SELECT TOP 1 session_place_id
                FROM Parking_Session
                WHERE session_user_id = ? AND exittime IS NULL
                ORDER BY session_id DESC
            """, user_result[0])
            place = cursor.fetchone()
            if place:
                final_position = place[0]
                cursor.execute("EXEC vehicle_exit @place_id = ?", final_position)
                conn.commit()
                set_bool(data, 518, 0, True)
                print(f"[EXIT] Success: User '{username}' exited from position {final_position}")
                set_start_moving()
            else:
                error_code = 3
                print("[EXIT] No active session found.")
        elif user_result:
            error_code = 1
            print(f"[EXIT] Password incorrect for user '{username}'")
        elif pass_result:
            error_code = 2
            print(f"[EXIT] Username incorrect for password provided.")
        else:
            error_code = 3
            print(f"[EXIT] Invalid credentials: username='{username}', password='{password}'")
        # Write back final position and error code
        set_int(data, 512, final_position)  # DB49.DBW512
        set_int(data, 516, error_code)      # DB49.DBW516
        # Clear username & password
        for i in range(0, 256): data[i] = 0
        for i in range(256, 512): data[i] = 0
        plc.db_write(49, 0, data)
        # Set bit for output position
        tag_data = bytearray(plc.db_read(1, 0, 1))
        if final_position == 1:
            set_bool(tag_data, 0, 0, True)
        elif final_position == 2:
            set_bool(tag_data, 0, 1, True)
        elif final_position == 3:
            set_bool(tag_data, 0, 2, True)
        plc.db_write(1, 0, tag_data)
        # Reset trigger DB1.DBX2.0
        tag_reset = bytearray(plc.db_read(1, 2, 1))
```

Figure 3 & 4

```
            tag_reset[0] &= ~(1 << 0)
            plc.db_write(1, 2, tag_reset)
            print("[EXIT] Output trigger DB1.DBX2.0 reset.")
        except Exception as e:
            print(f"[EXIT] Error: {e}")
# ---------------- Main Loop ----------------
previous_input = False
previous_output = False
try:
    while True:
        input_byte = plc.db_read(1, 1, 1)   # DB1.DBB1
        output_byte = plc.db_read(1, 2, 1)  # DB1.DBB2
        input_trigger = bool(input_byte[0] & (1 << 7))     # DB1.DBX1.7
        output_trigger = bool(output_byte[0] & (1 << 0))   # DB1.DBX2.0
        if input_trigger and not previous_input:
            print("[MAIN] Entry Trigger Detected")
            entry_process()
        if output_trigger and not previous_output:
            print("[MAIN] Exit Trigger Detected")
            exit_process()
        previous_input = input_trigger
        previous_output = output_trigger
        time.sleep(0.1)
except KeyboardInterrupt:
    print("[MAIN] Interrupted by user.")
finally:
    plc.disconnect()
    print("[MAIN] PLC disconnected.")
```

Figure 5

Machine Vision:

```python
import cv2
import pickle
import cvzone
import numpy as np
import time
import snap7
from snap7.util import set_bool
# === PLC Settings ===
plc_ip = '192.168.0.1'     # PLC IP Address
db_number = 16              # Data Block number
num_parking_spots = 4      # Number of parking spots
# Connect to PLC
plc = snap7.client.Client()
plc.connect(plc_ip, rack=0, slot=1)
# === Load Video and Parking Positions ===
cap = cv2.VideoCapture('carPark1.mp4')
with open('CarParkPos', 'rb') as f:
    posList = pickle.load(f)
width, height = 100, 90
screen_res = (1366, 768)
# === Thresholds and Timings ===
thresholds = [2000, 3000, 1000, 2800]  # You can adjust based on your video
freeze_delay = 3        # seconds to freeze after spot 0 changes
stability_delay = 2     # seconds a new state must be stable to be accepted
# === State Management ===
spot_states = {
    i: {"stable": None, "temp": None, "start_time": None}
    for i in range(len(posList))
}
freeze_start_time = None
freeze_active = False
# === Write parking status to PLC at DB16.DBX0.0 to DBX0.3 ===
def write_parking_status_to_plc(parking_status):
    try:
        byte_offset = 1  # Starting from byte 1 (DBX0.0)
        num_bytes = 1    # One byte is enough for 4 bits
        data = bytearray(plc.db_read(db_number, byte_offset, num_bytes))
        for i in range(num_parking_spots):
            set_bool(data, 0, i, parking_status[i])
        plc.db_write(db_number, byte_offset, data)
    except Exception as e:
        print(f"PLC Communication Error: {e}")
# === Check Parking Space ===
def checkParkingSpace(imgPro):
    global freeze_start_time, freeze_active
```

Figure 1

```
    current_time = time.time()
    spaceCounter = 0
    for i, pos in enumerate(posList):
        x, y = pos
        imgCrop = imgPro[y:y + height, x:x + width]
        count = cv2.countNonZero(imgCrop)
        threshold = thresholds[i] if i < len(thresholds) else 2700
        busy_now = 0 if count < threshold else 1
        state = spot_states[i]
        if state["stable"] is None:
            state["stable"] = busy_now
        if i == 0:
            # Freeze logic for spot 0
            if busy_now != state["stable"]:
                if state["temp"] != busy_now:
                    state["temp"] = busy_now
                    state["start_time"] = current_time
                elif current_time - state["start_time"] >= stability_delay:
                    state["stable"] = state["temp"]
                    state["temp"] = None
                    state["start_time"] = None
                    freeze_start_time = current_time
                    freeze_active = True
            else:
                state["temp"] = None
                state["start_time"] = None
            stable_busy = state["stable"]
        else:
            # Other spots freeze logic
            if freeze_active:
                if current_time - freeze_start_time < freeze_delay:
                    continue
                else:
                    freeze_active = False
            if busy_now != state["stable"]:
                if state["temp"] != busy_now:
                    state["temp"] = busy_now
                    state["start_time"] = current_time
                elif current_time - state["start_time"] >= stability_delay:
                    state["stable"] = state["temp"]
                    state["temp"] = None
                    state["start_time"] = None
```

Figure 2

```
            else:
                state["temp"] = None
                state["start_time"] = None
            stable_busy = state["stable"]
        # Drawing on video
        color = (0, 255, 0) if stable_busy == 0 else (0, 0, 255)
        thickness = 5 if stable_busy == 0 else 2
        if stable_busy == 0:
            spaceCounter += 1
        cv2.rectangle(img, pos, (x + width, y + height), color, thickness)
        cvzone.putTextRect(img, str(count), (x + width // 2, y + height // 2), scale=1,
                           thickness=2, offset=0, colorR=color)
    # Display total free spaces
    cvzone.putTextRect(img, f'Free: {spaceCounter}/{len(posList)}', (100, 50), scale=3,
                       thickness=5, offset=20, colorR=(0, 200, 0))
    # Show freeze warning
    if freeze_active and (current_time - freeze_start_time < freeze_delay):
        remaining = int(freeze_delay - (current_time - freeze_start_time))
        msg = f"🔴 Waiting for spot 1 to stabilize... ({remaining})"
        cvzone.putTextRect(img, msg, (100, 120), scale=2,
                           thickness=3, offset=10, colorR=(0, 0, 255), colorT=(255, 255, 255))
# === Main Loop ===
while True:
    success, img = cap.read()
    if not success:
        break
    # Resize image to fit screen
    scale_w = screen_res[0] / img.shape[1]
    scale_h = screen_res[1] / img.shape[0]
    scale = min(scale_w, scale_h)
    new_w, new_h = int(img.shape[1] * scale), int(img.shape[0] * scale)
    img = cv2.resize(img, (new_w, new_h))
    # Preprocessing
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
    imgThresh = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                      cv2.THRESH_BINARY_INV, 25, 16)
    imgMedian = cv2.medianBlur(imgThresh, 5)
    kernel = np.ones((3, 3), np.uint8)
    imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)
    # Analyze spots
    checkParkingSpace(imgDilate)
    # Build status list & write to PLC
    parking_status = [1 if spot_states[i]["stable"] == 1 else 0 for i in range(num_parking_spots)]
    write_parking_status_to_plc(parking_status)
    # Show video
    cv2.imshow('Parking Monitor', img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# === Cleanup ===
cap.release()
cv2.destroyAllWindows()
plc.disconnect()
```

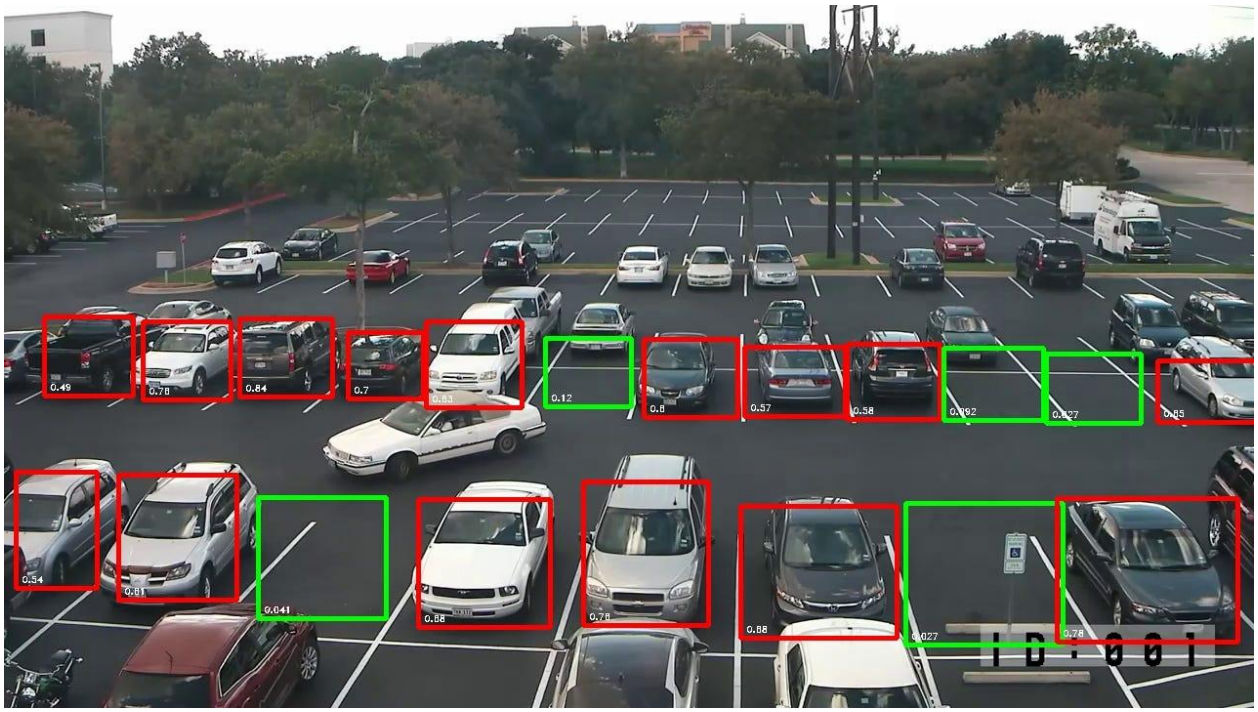
Figure 3 &4

Machine Vision for Parking Space Occupancy Detection

- **Overview:**

In our project, we implemented a Machine Vision system to automatically detect the occupancy status of four parking spaces using camera footage. Based on the visual output, the system determines whether each space is occupied or free, then sends corresponding status signals to a PLC (Programmable Logic Controller) via the Snap7 industrial communication library.

This approach replaces physical sensors by using image processing techniques to analyze video frames in real time. Each spot is monitored for pixel intensity patterns that indicate whether a car is present or not.



System Components:

1/Parking Spot Picker Script:

A helper script is used to manually define the positions of parking spots on a reference image. This setup uses OpenCV's mouse events to allow the user to click and define rectangular areas for each parking space.

- Left-click adds a spot.
- Right-click removes a spot if clicked inside its rectangle.

- These positions are saved using the pickle library in a file (CarParkPos) to be used during real time detection.

2/Main Machine Vision Detection Logic Script:

A separate script processes the video stream to monitor each parking space in real time.

1. Initialization and PLC Connection:

- Connects to the Siemens PLC at IP 192.168.0.1, using rack 0 and slot1.
- Sets up the PLC to receive status updates on Data Block 16.
- Loads parking region positions from a pickle file created earlier.

2. Image Processing Steps:

- Convert each video frame to grayscale.
- Apply Gaussian Blur to reduce noise.
- Use Adaptive Thresholding to highlight significant areas.
- Apply Median Filtering to enhance quality.
- Perform Dilation to thicken white areas for better detection.

These steps help isolate bright regions (typically the car body) and improve consistency across lighting conditions.

3/Dynamic Status Analysis:

Each of the four defined zones is processed individually:

- A cropped region of interest (ROI) is extracted.
- The number of white pixels in that ROI is counted using cv2.countNonZero().
- This value is compared to a custom threshold for that specific spot.
- Based on this comparison, the spot is considered occupied (1) or free (0).

➢ **Spot 0** is dedicated to detecting whether a car is currently on the forklift— indicating the car is in transition before being parked.
➢ **Spots 1, 2, and 3** represent the actual parking locations.

4/Freeze and Stability Logic:

To avoid flickering and false detection due to temporary movement or lighting changes:

- A temporary state is tracked separately from the stable state.
- If a new state is observed, a timer starts.
- If that new state persists for at least 2 seconds, it replaces the stable state.

Special Logic for Spot 0 (Forklift Zone):

- Spot 0 has a priority freeze mechanism.
- After a valid state change in spot 0, the system waits 150 seconds (freeze_delay) before evaluating the remaining spots.
- This ensures that cars are fully picked up or dropped off before checking the rest of the parking area.

5/Communication with PLC:

After all spots are evaluated and stabilized:

- A 4-bit binary status array is built ([1, 0, 1, 1], etc.).
- Using the snap7.util.set_bool () function, the system writes the occupancy status to DB16[DBX0.0 → DBX0.3] (1 bit per spot).
- A single byte is updated in the PLC's Data Block.

6/Visualization and Monitoring

During runtime, the interface displays:

- Each parking spot is drawn as a rectangle:
  - **Green** for free.
  - **Red** for occupied.
- Pixel count is shown at the center of each spot for reference.
- A freeze timer is shown when spot 0 is in transition.
- The total number of free spaces is displayed at the top

Hardware Components:

⇒Lenovo HD Integrated Webcam.

<u>Purpose in the System:</u>

The **Lenovo 300 FHD Webcam, 1080p** is used as the primary vision sensor in the project. It captures real-time video of the parking area, which is then processed using computer vision techniques to determine the availability of each parking space.

Display Size	3.5 inches
Digital Zoom	2 multiplier x
Optical Zoom	4 multiplier x
Video Resolution	1080p
Connectivity Tech	USB
Shooting Modes	Low Light
Lens Type	Wide Angle

9

Communication Between System Components

1. Introduction:

This chapter outlines the communication infrastructure used to enable interoperability between different system components in a control environment. It specifically focuses on using S7 communication over TCP/IP as the communication protocol over a local area network (LAN) built using Ethernet cables and a traditional router. The setup enables effective real-time data exchange between a Programmable Logic Controller (PLC), Human-Machine Interface (HMI), SCADA software, Node-RED and a Python-based camera vision system.

2. Network Topology Overview:

The system components are connected in a star topology using a traditional router that functions as a Layer 2 Ethernet switch when all devices are in the same IP subnet. The router's LAN ports are used to interconnect the following devices:

a) Siemens S7-1200 DC\DC\DC PLC.

b) Laptop 1 running Python-based application, SCADA (WinCC), and SQL database communication.

c) Laptop 2 running Node-RED

d) A real Siemens HMI screen (separately connected and configured to communicate with the PLC)

Network Details:

- IP Subnet: 192.168.0.1/24
- Router IP: 192.168.0.11
- PLC IP: 192.168.0.1
- Laptop 1 (Python + SCADA + SQL): 192.168.0.10
- Laptop 2 (Node-RED): 192.168.0.10
- HMI Panel: 192.168.0.13

The Siemens S7-1200 PLC communicates natively with devices such as Human-

Machine Interfaces (HMIs), SCADA systems, Node-RED platforms, and Python scripts using the S7comm protocol.
All devices communicate directly over the LAN with no routing required since they share the same subnet.

3. Communication Protocol: S7 Communication over TCP/IP

S7 communication over TCP/IP is used as the primary communication protocol between clients (e.g., SCADA or Node-RED) and the PLC. It operates over TCP port 102 and adheres to the client-server model:
- Clients (Node-RED, SCADA, Python Snap7) initiate connections.
- Server (PLC) responds to requests by reading/writing data.

S7 Communication (S7Comm) over TCP/IP:

S7Comm (Siemens S7 Communication) is the native protocol used by Siemens SIMATIC S7-series PLCs and C7 controllers. It has been integrated into all S7-300/400/1200/1500 CPUs since the 1990s and supports PLC programming, data exchange, and diagnostics in Siemens automation systems. In practice,this protocol can be used both through Industrial Ethernet and through other physical or network layers such as over RS-485 for MPI (Multi-Point-Interface) or Profibus and by default it uses TCP port 102 with an ISO-on-TCP (RFC1006) transport (TCP + TPKT + COTP) to communicate between a master (PC/HMI/SCADA) and a server (the PLC). In a typical Siemens TIA Portal environment, engineering tools and HMI/SCADA systems use S7Comm to download programs to the PLC, read/write data blocks, monitor memory bits and registers, and perform diagnostic queries (e.g. retrieving CPU status or alarm lists). Because S7Comm is optimized for Siemens hardware, it can efficiently pack multiple read/write requests into one PDU and maintain cyclic polling connections for real-time data exchange.

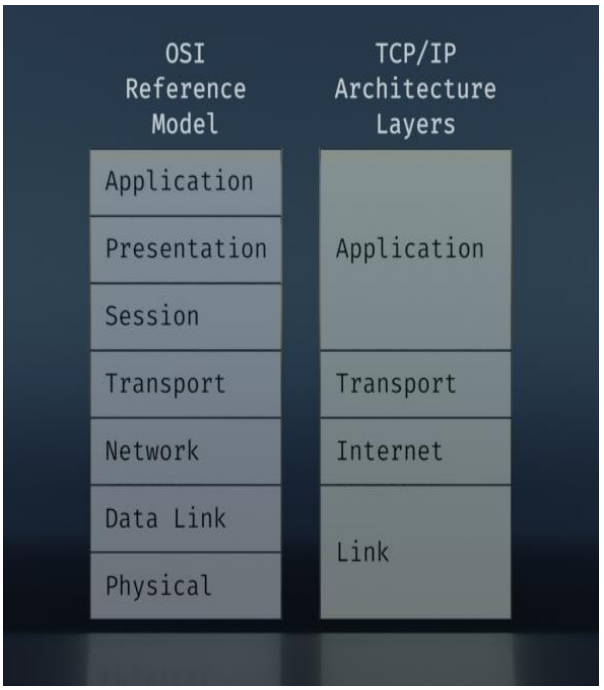


Figure: TCP/IP stack layers (OSI vs. Internet model).

S7Comm operates at the Application/Session layer above TCP (Port 102) and IP, riding on Ethernet at the Physical/Data-Link layers.

In the OSI/TCP-layered view, S7Comm spans the top layers (Presentation/Session/Application) while leveraging TCP/IP underneath. A client (e.g. HMI or SCADA) opens a TCP socket to the PLC's S7 server interface on port 102; the PLC's built-in

communication processor (CP) then handles the ISO-on-TCP handshake. Once connected, S7Comm frames (with protocol ID 0x32) carry Siemens-defined messages encapsulated over the TCP stream.

Technical description

Generally, the communication designed by Siemens follows the traditional master-slave (or client-server) model where a master (or client) PC sends S7 requests to the slave (or server) PLC device).

The implementation of the S7 protocol over TCP/IP is based on the blockoriented ISO transport service. This protocol is encapsulated in the TPKT and ISO-COTP protocols, allowing the transfer of the PDU (Protocol Data Unit) via TCP, using by default TCP/102 port for communications.

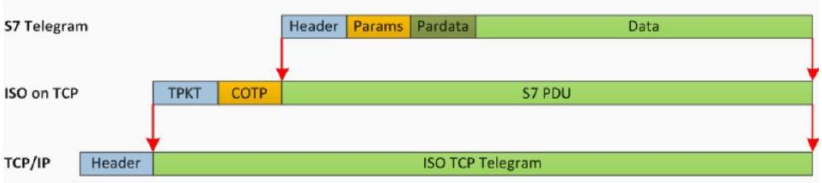
Each of the data blocks that are transmitted are so-called PDUs, the length of which is negotiated during connection setup.

The protocol is function and/or command oriented, which means that the transmission generally consists of an S7 request and a response to that request. Each of these commands consists of the following commands fields, the last two being optional:

- Header.
- Parameter set (and parameter data).
- Data.

For a better understanding of S7 encapsulation, the following picture shows how S7 is first encapsulated in ISO over TCP and then in TCP/IP:

- Application Layer: S7 Protocol
- Transport Layer: ISO Transport Class 0 (COTP)
  Encapsulation ISO on TCP: TPKT (RFC 1006)
- Network Layer: TCP/IP
- Data Link Layer: Ethernet MAC
- Physical Layer: Ethernet Cable



S7 Telegram:

It is the fundamental message format used in Siemens S7 protocol communication. It contains all the information needed to perform operations such as reading from or writing to the PLC's memory (Data Blocks, Inputs, Outputs, etc.).

It is structured as a Protocol Data Unit (PDU) and is part of the application layer.

Structure of the S7 Telegram

Segment	Description
Header	Identifies the type of operation (e.g. read, write, setup communication).
Params	Contains details like the number of items, memory area, and address.
ParData	Optional section: additional parameters or data associated with the request.
Data	The actual payload — for example, values to write or buffer to fill with read data.

ISO on TCP:

ISO on TCP (also known as RFC 1006) is a transport protocol that allows ISO Transport Protocol (Class 0) Connection-Oriented Transport Protocol (COTP) to operate over standard TCP/IP networks. It's the middle layer in the communication stack between the TCP/IP transport layer and the application layer (like S7comm) in Siemens PLC systems.

Segment	Description
TPKT	RFC 1006 Transport Packet Header is a small but critical protocol used to wrap ISO protocols (like COTP and S7comm) so they can be transmitted over standard TCP/IP networks.
COTP	Connection-Oriented Transport Protocol is part of the ISO/OSI model's Transport Layer (Layer 4), specifically Transport Class 0, which is the most basic and lightweight transport protocol in the ISO/OSI model.

TCP/IP :

The TCP/IP layer is at the heart of modern digital communication, including industrial protocols like S7comm used by Siemens PLCs. It provides the transport and network functionality that makes Ethernet-based automation possible.

TCP (Transmission Control Protocol) – Layer 4 (Transport):

- Ensures reliable, ordered, error-checked delivery of data.
- Establishes a connection-oriented session between two devices.
- Uses Port 102 for S7comm.
- Handles retransmissions, acknowledgements, and flow control.

IP (Internet Protocol) – Layer 3 (Network):

- Routes packets across the network using IP addresses
- Defines source and destination IP (e.g., 192.168.1.101 → 192.168.1.100)
- Responsible for fragmentation, routing, and addressing

Why TCP/IP Is Critical:

• Feature	• Importance in S7comm
• Reliability	• TCP guarantees delivery, which is crucial in automation
• Standardization	• TCP/IP runs on all modern networking hardware
• Flexibility	• Works across switches, routers, Wi-Fi, VPNs, etc.
• Multi-client	• Multiple clients can connect to a PLC simultaneously

S7Comm's efficiency and tight integration make it ideal for Siemens control networks. It is the default choice in TIA/STEP7-based setups for communications between PLCs, HMIs, SCADA, engineering stations, and other clients.

S7Comm is also widely supported (often via licensed or reverse-engineered drivers) in third-party systems. Commercial OPC servers and data-collection tools typically include an "S7" driver. the S7Comm can read/write data blocks, flags, timers, and counters by interfacing with the S7 CPU's data areas, implying full support of the native S7Comm functions. Even third-party HMIs or SCADA packages that emulate Siemens S7 links rely on S7Comm: in practice, a HMI that "supports S7-300" can often talk to an S7-1200

as long as Put/Get communication is enabled in the PLC and data blocks are set to non-optimized access.

The use of TCP ensures reliable delivery, while the S7 protocol defines the structure for data requests and responses.

4. Snap7 library in Python:

In addition to SCADA and HMI communication with the Siemens S7-1200 PLC, the system leverages a Python-based application to achieve direct, real-time data exchange with the PLC. This is implemented using the Snap7 library, an open-source, high-performance toolkit that provides full client-side access to Siemens S7 controllers over Ethernet.

Snap7 is an open-source, cross-platform communication library that allows software applications to communicate directly with Siemens S7 series PLCs over Ethernet using the S7Comm protocol. Snap7 is designed to provide developers with programmatic access to read, write, and interact with PLC memory areas, making it ideal for custom applications like Python-based control systems, SCADA extensions, and data acquisition.

The Snap7 library natively supports the S7Comm protocol, enabling the Python application to connect to the PLC over TCP port 102, perform read/write operations, and access memory areas such as:

- Data Blocks (DB).
- Inputs (I).
- Outputs (Q).
- Merker Memory (M).

5.Communication Between PLC, Python Code and SQL Database:

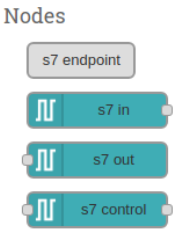
In addition to real-time communication between the Python vision system and the Siemens S7-1200 PLC via the Snap7 library, the system also includes a mechanism for storing and retrieving data from a SQL database.

In the implemented system architecture, the Python application plays a critical intermediate role, bridging real-time industrial control data from the Siemens S7-1200 PLC to the SQL database. The processed information is then written to an SQL database using the pyodbc library. This enables long-term storage of critical system data. This is essential for tasks such as:
- Logging object detection results from the vision system.
- Storing PLC input/output states over time.

6. Communication Between PLC and Node-red:

Node-RED, a visual programming tool for wiring together hardware devices and APIs, can be integrated with Siemens PLCs using the S7comm protocol. The communication is made possible by installing the node-red-contrib-s7 package, which provides specialized nodes for reading from and writing to S7 PLCs.

As shown in the image below, three nodes are provided under the PLC category:



- S7 in – Reads data from the S7 PLC.
- S7 out – Writes data to the S7 PLC.
- S7 control – Controls the connection to the PLC (start/stop or reconnect).

7. Role of Traditional Router:

In this setup, the traditional router serves primarily as a Layer 2 switch, forwarding Ethernet frames between devices in the same subnet. Key functions include :
- MAC-based forwarding: Routes Ethernet frames using learned MAC addresses.
- DHCP server (optional): Assigns IP addresses dynamically unless static IPs are configured.
- Switching, not routing: Since all IPs are in the same subnet, no routing occurs; ARP resolves IPs to MAC addresses.

This configuration simplifies deployment and avoids the need for more complex industrial switches when only basic LAN communication is needed.

8. ARP and Address Resolution:

To transmit an IP-based message (e.g., S7 request) over Ethernet, the sender must first resolve the destination device's MAC address using the Address Resolution Protocol (ARP):
1. Device A (e.g., Node-RED or SCADA) wants to send to PLC at 192.168.0.1
2. It sends an ARP broadcast: "Who has 192.168.0.1?"
3. The PLC replies: "192.168.1.100 is at AA:BB:CC:DD:EE:01"
4. Device A stores this in its ARP cache and sends the Ethernet frame.

This resolution step ensures the frame has the correct destination MAC, enabling the switch/router to forward it properly.

9. Simultaneous Communication:

The PLC can handle multiple concurrent TCP connections, such as simultaneous access by the SCADA system, Node-RED, and the Python-based system using Snap7. Each client communicates over a separate TCP socket, allowing parallel read/write operations as long as the PLC's connection limit is not exceeded (usually 3–8 connections on S7-1200).

10. Conclusion:

The use of S7 communication over TCP/IP with a traditional router enables reliable and scalable communication between control system components. By leveraging ARP, Ethernet switching, and standardized TCP/IP networking, the system supports efficient real-time monitoring, control, and programming with minimal hardware complexity. The addition of a Python-based camera vision system using Snap7 further extends functionality by allowing intelligent detection or tracking to interact directly with the PLC.

Summary

With the rapid expansion of urban areas and the continuous increase in the number of vehicles, modern cities are facing growing challenges in managing traffic congestion and providing adequate, safe parking spaces. This project presents an innovative and efficient solution through the design and implementation of a *multi-level automated parking system*, aimed at improving space utilization and facilitating the parking and retrieval process in a structured and intelligent manner.

The system is based on the concept of vertical space optimization, where vehicles are arranged across multiple levels and moved automatically without the need for direct human intervention. Cars are transported vertically or horizontally to assigned parking slots, ensuring organized storage and easy access when needed. The system relies on accurate mechanisms to monitor the movement and position of vehicles within the structure, contributing to high performance and minimal errors.

The project was executed through several phases, including mechanical design, system control, monitoring, and final prototype testing. Key factors such as safety, stability, speed, and operational accuracy were carefully considered to ensure the system performs reliably under real-world conditions. The final model is user-friendly and designed to be scalable, allowing for future enhancements such as vehicle counting, reservation features, or integration with other infrastructure systems.

This project serves as a practical model that can be implemented in high-traffic areas such as shopping malls, airports, hospitals, and administrative buildings, especially where ground space is limited. It reflects the ability of engineering students to address real-life

problems by developing smart, technically sound solutions that contribute to building a more efficient, sustainable, and intelligent urban infrastructure.