IT&BUSINESS COLLEGE

Department: COMPUTER SCIENSE

Project title: Model of city

Group Name: Team-05

Members:

Nureles Tashbolotov - ID 238715062
Adelia Ramisova - ID 238715006
Turganaliev Diyaz - ID 238715003
Akylbekova Altynai - ID 238715008
Baryspiev Said - ID 238715009

Instructor: Mirlan Nurbekov

Group: SCA-23A

Date: 04.12.2025

Table of Content

Table of Tables

Table of Figures

Abstract

The rapid advancement of embedded systems and intelligent automation has significantly transformed the way modern cities operate. As urban environments continue to expand, the need for efficient, adaptive, and interconnected technological solutions becomes increasingly important. This paper presents the development of a functional miniature smart city model based on the Arduino microcontroller platform, created as part of the course *Architecture of Electronic Computing Machine & Computing Systems*. The project demonstrates how fundamental computing architecture concepts can be integrated into real-world engineering applications, emphasizing modularity, sensor-based automation, and interaction between hardware and software components.

The constructed city model consists of several independent yet interconnected smart subsystems: an automated parking module, a rain-detection system, and a smart intercom for building access control. Each component represents a simplified version of technologies currently used in Smart City infrastructure. The smart parking system employs ultrasonic sensors to determine the availability of parking spaces, providing a solution for optimizing space usage and reducing congestion. The rain recognition module uses environmental sensors to detect changes in weather conditions, demonstrating how microcontrollers can support adaptive infrastructure such as automatic roof activation or irrigation control. The smart intercom system simulates restricted entry through authentication logic, presenting a model of real access-control technology widely implemented in residential complexes. All components were designed to operate cohesively within a single electronic architecture, communicating through digital interfaces handled by Arduino logic.

This work explores not only the practical implementation of each subsystem, but also the underlying computing machine architecture principles that ensure stability, responsiveness, and scalability of the system. Sensor inputs, signal processing, decision-making algorithms, and output actuation illustrate the fundamental interaction between hardware layers and software routines. The physical layout of the city was constructed using cardboard and electronic modules, allowing component placement, wiring routing, and functional demonstration in a tangible format. Although simplified, the model reproduces the core structure of a modern urban automation network, showing how computational logic can support everyday city processes.

The results confirm that Arduino-based microcontroller systems can effectively simulate Smart City functions on a small scale, providing a foundation for deeper exploration into distributed computing, IoT architecture, and automated urban engineering. The project highlights the relevance of computational architecture in the development of intelligent infrastructure and demonstrates the potential of microcontrollers as an educational platform for studying communication protocols, sensor integration, real-time signal processing, and smart system interaction. This paper aims to analyze the design rationale, hardware architecture, operational logic, and performance of the created model, while also reflecting on the role of computing systems in the future of urban automation.

1. Introduction

The evolution of digital technology over the past decades has fundamentally transformed industries, public infrastructure, and the daily life of individuals. As computing devices have become smaller, faster, and more energy-efficient, their integration into real environments has enabled a shift from traditional, manually controlled systems toward intelligent, autonomous networks. These developments are especially noticeable in the context of modern cities, where rapid urbanization creates a continuous demand for mechanisms that can maintain comfort, safety, and resource efficiency. A Smart City is characterized by the presence of distributed computing systems capable of sensing, analyzing, and responding to environmental conditions without direct human intervention. Such systems rely on the architectural foundations of electronic computing machines: processing units, memory organization, instruction execution, signal routing, and input-output interfaces.

Understanding how computational logic is implemented in real hardware is essential for engineers and researchers working with embedded systems and urban automation. However, theoretical knowledge alone is often insufficient for developing a practical understanding of computing architecture. Concepts such as registers, data buses, control units, interrupts, and machine-level execution become more meaningful when applied in a functioning system. For this reason, the development of a physical prototype is an effective method for reinforcing theoretical principles. In this project, the Arduino microcontroller platform was selected as the computing core due to its transparent architecture, modular input-output capabilities, and support for low-level control of electrical signals. Arduino provides an accessible environment that reflects the structure of a classical computing machine, while still being versatile enough to manage real-time automation tasks.

The miniature smart city developed in this project was designed as an experimental model demonstrating how computing architecture principles can be applied to solve challenges associated with urban systems. The prototype includes three autonomous subsystems: a smart parking module for space allocation, a rain-detection block for environmental monitoring, and an intercom access system emulating secure entry control. Each subsystem operates using sensors, microcontroller instructions, and electromechanical components, illustrating the full data pathway from physical signal acquisition to digital analysis and actuator response. Although simplified compared to industrial-scale systems, this model provides a controlled environment in which interactions between devices, logical decisions, and system scalability can be observed and analyzed.

The relevance of this research lies in the increasing global interest in sustainable and intelligent urban development. Traffic congestion, uncontrolled water consumption, and inefficient security management are challenges present in most cities. Automation has the potential to mitigate these problems by ensuring faster response times, reducing human error, and enabling predictive behavior based on sensor feedback. A smart parking system, even in miniature form, demonstrates how real-time monitoring of available spaces can reduce unnecessary vehicle circulation. Similarly, a rain sensor forms the basis for adaptive systems that respond to meteorological changes - closing windows, regulating irrigation, or activating alarm routines. The intercom module replicates digital identity verification, which is widely used in modern housing complexes and IoT-based home security. All three components work together under a shared computational core, reflecting the architecture of interconnected smart infrastructure.

In addition to its practical demonstration value, this project contributes to the educational understanding of computing systems by revealing how microelectronics can form the backbone of distributed intelligent environments. When working with the model, students engage with low-level programming logic, timing control, and data transfer between modules. These tasks require consideration of hardware constraints such as memory limits, signal interference, response delays, and power management - challenges that mirror those found in large-scale computing systems. By assembling and programming the prototype, theoretical concepts such as instruction cycles, branching logic, and real-time processing become tangible, measurable, and observable.

This introduction establishes the framework for the research presented in this paper. The following sections will examine the internal architecture of the system, describe the hardware components and their interaction, analyze the program logic implemented on the Arduino microcontroller, and evaluate the resulting performance of the completed model. The goal of this work is not only to construct a functional smart city prototype but also to analyze the underlying computing mechanisms that enable its operation. Through this study, we aim to highlight the importance of electronic computing machine architecture in modern automation and demonstrate how microcontrollers can serve as an accessible platform for studying real-world principles of intelligent infrastructure.

While the first part of this introduction outlined the growing relevance of smart infrastructure and the academic motivation behind this project, it is also important to examine the technological context in which embedded systems currently operate. Modern computational devices function not as isolated units but as interconnected elements of a greater digital ecosystem. A contemporary city contains thousands of microcontrollers, sensors, wireless modules, and cloud-based servers working simultaneously. These devices exchange data through structured communication protocols, operate under predefined computing architectures, and perform instruction cycles that determine how input signals are interpreted and transformed into actions. Understanding these interactions requires not only knowledge of programming but also a foundational comprehension of how electronic computing machines process information at a systemic level.

In practical implementation, microcontrollers such as Arduino are structured around the same fundamental architecture found in larger computing systems. They include an Arithmetic Logic Unit (ALU) for mathematical operations, volatile and non-volatile memory blocks for data storage, digital and analog input-output interfaces for external device interaction, and a control unit responsible for instruction sequencing. Although compact, the internal logic reflects the Von Neumann and Harvard architectural principles on which much of modern computation is built. These architectural models define how data and instructions are stored, how they travel through buses, and how they are executed during clock cycles. By interacting with these mechanisms through code and hardware wiring, students are able to observe how high-level operations break down into low-level machine behavior.

The smart city prototype developed in this project serves as a microcosm of a real urban computing environment. Each subsystem operates as a node within a distributed network, performing independent tasks while still relying on a centralized processing core. This reflects the structure of modern IoT (Internet of Things) networks, where edge devices collect data locally and transmit only essential information to a central server. The parking module, for instance, does not need to communicate constantly; it updates only when a car arrives or leaves. The rain sensor is event-driven, activating a response only under specific environmental conditions. The intercom operates based on user input, simulating secure access verification. These localized decision mechanisms are comparable to real smart systems that reduce bandwidth load, minimize processing time, and increase efficiency by operating autonomously.

Another key dimension of this work is demonstrating how computational architecture supports determinism and real-time response. Unlike general-purpose computers, embedded systems must respond predictably and without delay to physical stimuli. An ultrasonic sensor detecting an approaching vehicle cannot wait several seconds to determine whether a parking space is available - decisions must be processed instantly. The rain-detection module must register environmental moisture the moment it appears, as delayed reaction could lead to hardware malfunction or environmental damage in a real system. The intercom must authenticate access within a short time frame to ensure user convenience and security. These requirements place constraints on computing architecture: memory allocation must be optimized, loop cycles must be efficient, signals must be filtered to avoid noise, and communication between modules must be synchronized. Through this project, these constraints were explored and addressed through hardware selection and program logic design.

Beyond technical analysis, the educational value of building a functional model lies in the cognitive transition from theory to application. Concepts such as interrupt handling, pulse-width modulation, analog-to-digital conversion, and clock-based timing control are often abstract when encountered in coursework. However, when a student physically observes how an ultrasonic pulse is transmitted, received, processed, and interpreted to calculate distance, the abstraction becomes concrete. The model bridges this gap by making invisible computing operations visible through physical outcomes: an LED lights up, a barrier opens, a buzzer plays a tone, an LCD displays status changes. Each visible reaction is

the result of structured computation - an instruction fetched, decoded, executed, and stored - thereby demonstrating the inner mechanisms governing computing machines.

The significance of this project therefore lies not merely in constructing a functional city layout, but in illustrating a scalable architectural framework that could, in theory, be expanded into a full smart urban system. Additional modules such as traffic-light logic, power grid monitoring, waste management automation, temperature-based ventilation, or wireless data exchange could be integrated under the same architectural principles. The current system establishes a baseline for such expansion, proving that microcontroller-based urban automation is structurally possible even at a small scale. It invites further development toward distributed processing, multi-node communication, and cloud-linked decision automation.

This extended introduction establishes not only the technical foundation for the smart city model but also the broader academic relevance of implementing computing architecture principles in real environments. The prototype demonstrates how electronic machines execute logic, interact with the physical world, and enable intelligent behavior. The sections that follow will describe the architecture of the system in detail, discuss hardware selection and circuit integration, present algorithmic control flow, and evaluate the overall performance of the prototype. By linking theory to practice, this work aims to contribute to deeper understanding of embedded computing systems and their role in shaping future urban infrastructure.

2. Problem statement & objectives

2.1 Problem statement

Cities today face growing difficulties in keeping public systems efficient, managing resources responsibly, and ensuring safety as populations continue to increase. Traditional urban infrastructure often relies on manual processes, technologies that do not communicate with one another, or outdated control methods. These older approaches cannot keep up with the dynamic conditions and rapid changes present in modern cities. As a result, problems such as inefficient parking management, slow reactions to environmental conditions like rainfall, and vulnerabilities in security systems become more noticeable. These issues not only reduce the overall effectiveness of city operations but also highlight the need for intelligent, automated solutions capable of making independent decisions based on real-time sensor data.

Although smart city technologies are becoming more advanced, students still struggle to understand how these systems function at a deeper, technical level. Courses related to computing architecture tend to focus on theoretical ideas-such as memory management, data pathways, control units, and instruction execution-without providing opportunities to observe these concepts in practice. Because of this gap, students often cannot clearly see how low-level computational logic translates into the high-level behavior of a smart automated system. They may understand the theory behind microcontrollers, sensors, and data processing, but they lack experience in seeing how these components interact in a real physical environment to produce meaningful outcomes.

A further challenge arises from the need to integrate several different kinds of subsystems into a single, coherent model. Real smart cities consist of many distributed parts that must function independently while still being connected under a unified architectural framework. Replicating this structure, even on a small scale, involves difficulties related to hardware compatibility, information flow management, algorithm synchronization, and real-time responsiveness. Building a system in which a parking sensor, a rain-detection module, and a door-access intercom can operate without interfering with one another requires a solid understanding of timing constraints, resource limitations, communication protocols, and overall system coordination.

Creating a physical prototype introduces additional real-world complexities that do not appear in theoretical coursework. These include problems such as electrical noise, sensor inaccuracies, latency in signal processing, hardware limitations, wiring difficulties, and unpredictable environmental interference. Such challenges emphasize the importance of having an educational platform that demonstrates how computing architecture must adapt to practical conditions. Students benefit from a system where the consequences of poor coding decisions, inefficient memory usage, weak electrical connections, or flawed logic structures can be directly observed in the behavior of the model. This kind of hands-on experience helps bridge the gap between conceptual understanding and real-world implementation.

Therefore, the main problem addressed by this project is the lack of a practical, interactive, and educational way for students to observe how the architecture of electronic computing machines is applied within real smart city automation. The project aims to solve this by constructing a small-scale city model that integrates microcontroller architecture, sensor networks, and automated control logic into a single functional system. By doing so, the model will demonstrate not only what a smart urban subsystem can do, but also how it performs its functions internally. This approach allows students to develop a clearer and more complete understanding of the computational processes that drive intelligent technologies in real environments.

2.2 Project Objectives

To address the problem stated above, this project defines the following objectives:

1. **Demonstrate the practical application of computing machine architecture.**
   Implement a working system that shows how microcontrollers execute instructions, process sensor inputs, manage memory, and interact with hardware components.

2. **Develop an integrated smart city model with multiple subsystems.**
   Construct three functional modules-smart parking, rain-detection, and smart intercom-and ensure that each subsystem can operate both independently and as part of a unified architecture.

3. **Design a hardware structure representing real urban automation.**
   Select and configure sensors, actuators, wiring layouts, and microcontroller interfaces that reflect typical characteristics of smart city infrastructures.

4. **Create efficient software logic for real-time operation.**
   Write optimized Arduino programs that handle data processing, decision-making algorithms, timing control, and event-driven responses.

5. **Demonstrate modularity and scalability.**
   Ensure that the prototype allows the addition of new subsystems (traffic lights, energy monitoring, wireless communication) without major redesign of the core architecture.

6. **Evaluate system performance under realistic conditions.**
   Test sensor accuracy, response times, reliability of communication, and overall stability of the model.

7. **Provide an educational platform for understanding smart systems.**
   Produce a prototype that helps students visualize how embedded systems contribute to urban automation, bridging theoretical computing concepts with real-world implementation.

3. System requirements & constraints

This section outlines the functional and non-functional requirements of the smart city prototype, as well as the constraints that influenced the design and implementation process. Because the purpose of the project is not only to replicate basic smart-city behaviour but also to demonstrate fundamental principles of computing architecture, the requirements address both outward system functionality and the internal logic that enables these functions. In other words, the system must operate as a realistic miniature city environment while also serving as an educational model that exposes how microcontrollers process data, how sensors interact with hardware, and how individual modules integrate into a unified structure.

To achieve this dual purpose, the requirements extend beyond simply making components "work." They include ensuring correct data acquisition from sensors, stable communication between modules, predictable timing behaviour, and proper resource management within the microcontroller. The prototype must demonstrate how inputs are collected, interpreted, and transformed into outputs through systematic control logic. Additionally, the system must be built under real constraints related to hardware capabilities, physical wiring, voltage limitations, sensor accuracy, and the available processing power, all of which restrict how complex the system can be and how reliably it can operate. These requirements therefore play a crucial role not only in defining what the prototype should do, but also in setting the boundaries within which the design must function.

3.1 Functional Requirements

The system must perform specific tasks that reflect typical operations of smart urban infrastructure. The following functional requirements were identified:

1. **Smart Parking Module**
   - Detect the presence or absence of a vehicle using an ultrasonic or infrared sensor.
   - Process distance readings and determine parking availability in real time.
   - Display parking status using LEDs, LCD, or serial output.
   - Operate autonomously without affecting other subsystems.
2. **Rain Detection Module**
   - Continuously monitor environmental moisture using a rain or humidity sensor.
   - Produce clear digital or analog signals indicating whether rain is detected.
   - Trigger programmed responses, such as warnings or system status updates.
   - Maintain responsiveness even when other modules are active.
3. **Smart Intercom Module**
   - Accept user input (button press or keypad entry).

- Authenticate access based on predefined logic.
- Control an actuator (e.g., servo motor simulating a door lock).
- Provide visual or audio feedback during interaction.

4. **Central Microcontroller Processing**
    - Read sensor inputs reliably and consistently.
    - Execute decision-making algorithms according to system logic.
    - Coordinate multiple modules within one shared computational architecture.
    - Maintain stable execution cycles without freezing or malfunctioning.
5. **City Model Representation**
    - Physically display the system within a cardboard model to simulate a real urban environment.
    - Integrate all modules into a cohesive layout that represents a functional smart city.

3.2 Non-Functional Requirements

Beyond basic functionality, the system must meet several quality criteria related to performance, usability, and maintainability. This means the prototype should not only operate correctly but also do so in a way that reflects the reliability expected from real smart-city systems. Performance requirements include consistent response times, stable sensor readings, and the ability to handle multiple inputs without delays or unexpected behaviour. Usability refers to the clarity and ease with which users-particularly students-can interact with the prototype, interpret its outputs, and understand the logic behind its operations. Maintainability is equally important: the system should be structured in a modular, well-organised way that makes it easy to update components, replace sensors, adjust code, and expand the functionality without redesigning the entire setup. Together, these quality criteria ensure that the prototype functions not just as a working model, but as a practical and understandable learning tool.

1. **Real-Time Performance**
    - The system must respond to sensor changes with minimal delay.
    - Execution loops must be optimized for consistent timing.
2. **Modularity**
    - Each subsystem must be independent enough to allow upgrades or replacement without major redesign.
    - Hardware and software must support adding new modules in the future.
3. **Scalability**

    - The architecture must allow expansion into additional smart-city features, such as traffic lights or wireless communication.

4. **Reliability**

    - Sensors and actuators must function correctly under repeated tests.

    - The microcontroller should remain stable over long execution periods.

5. **Educational Clarity**

    - The system must clearly demonstrate computational concepts such as input processing, control flow, and I/O operations.

    - Code and wiring must be readable for instructional purposes.

6. **Safety & Durability**

    - Components must be mounted securely on the model.

    - Wiring must avoid shorts or overheating.

    - Power supply must remain within safe voltage and current limits.

3.3 Hardware Requirements

To meet the functional goals, the prototype requires the following hardware:

Microcontroller: Arduino Uno or equivalent

1. **Sensors:**

    - Ultrasonic distance sensor for parking

    - Rain sensor module for environmental detection

2. **Actuators:**

    - Servo motor for intercom door mechanism

    - LEDs or LCD for visual output

3. **Supporting Electronics:**
    - Resistors, jumper wires, breadboards

    - Power supply (USB or external)

4. **Physical Structure:**

    - Cardboard city model

    - Mounts for sensors and wiring paths

3.4 Software Requirements

The software must:

- Be developed using Arduino IDE or compatible tools
- Support event-driven and loop-based logic
- Implement modular code structure (functions/classes where possible)
- Maintain efficient memory and CPU usage
- Allow clear debugging through serial output

3.5 System Constraints

The design is bounded by several practical and technical constraints:

1. Microcontroller Limitations

- Limited RAM and flash memory restrict program size.
- Single-core processor architecture limits parallel execution.
- Limited number of digital/analog pins restricts module expansion.

2. Sensor Accuracy & Noise

- Ultrasonic readings are affected by angle, distance, and environmental noise.
- Rain sensors degrade over time due to oxidation.
- Analog signal fluctuations require filtering and calibration.

3. Physical Constraints

- Small physical model limits realistic scaling of distances.
- Tight wiring spaces can cause signal interference if not organized cleanly.

4. Timing & Synchronization

- All modules share the same processor, so time-critical operations must be carefully balanced.
- Delays in one subsystem can slow or block others if not designed efficiently.

5. Power Constraints

- The Arduino cannot supply high current to many components simultaneously.
- Overloading can cause resets or unstable behavior.

6. Educational Requirement Constraints

- All code and wiring must be simple enough for demonstration purposes.
- The system must be easy to troubleshoot and modify.

4. System design (hardware & software)

The system design of the smart city prototype integrates both hardware and software components to create a functional, modular, and educational model that accurately reflects the interactions found in modern urban automation systems. The design process not only focuses on ensuring that each module performs its intended function but also emphasizes demonstrating key principles of computing machine architecture, such as input-output processing, memory utilization, control logic, and real-time decision-making. By combining physical components with programmable logic, the prototype provides a tangible platform for understanding how electronic systems translate sensor data into coordinated actions, enabling a clear connection between theoretical concepts and practical application.

The design is divided into two main areas: hardware architecture and software architecture. The hardware architecture encompasses all physical elements of the system, including sensors, actuators, wiring, and the structural layout of the model. It ensures that components are correctly interfaced, signals are transmitted reliably, and the physical arrangement of the city reflects realistic urban scenarios. This includes considerations of sensor placement for accurate readings, actuator positioning for visible responses, and the organization of wiring and power supply to minimize interference and maintain system stability.

The software architecture, on the other hand, manages the flow of information from sensor inputs to decision-making processes and output actuation. It is responsible for interpreting sensor data, executing programmed algorithms, and controlling actuators in a coordinated and timely manner. The software is designed to handle multiple subsystems concurrently, ensuring that each module operates independently while maintaining overall system integration. This dual focus on hardware and software ensures that the prototype functions as a complete, interactive smart city model, while also serving as a practical educational tool for demonstrating the principles of embedded systems, real-time computing, and intelligent infrastructure.

4.1 Hardware Design

The hardware design focuses on building a physical representation of the smart city while ensuring each module functions independently and interacts seamlessly with the central microcontroller. The following key elements define the hardware architecture:

1. **Microcontroller Core**
    - The Arduino Uno serves as the central processing unit of the prototype.
    - It manages all data acquisition, decision-making algorithms, and communication between modules.
    - Its digital and analog I/O pins are used to interface with sensors, actuators, and display devices.
2. **Smart Parking Module**
    - Ultrasonic sensors detect the presence of vehicles in parking spaces by measuring distances.

Ultrasonic sensors play a crucial role in the smart parking module by providing accurate, non-contact measurement of distances. These sensors emit high-frequency sound waves and measure the time it takes for the echoes to return after reflecting off an object, such as a car in a parking space. By calculating the time-of-flight of the sound waves, the sensor can determine the precise distance between itself and the vehicle. This allows the system to detect whether a parking spot is occupied or available in real time. Ultrasonic sensors are particularly suitable for this application because they are inexpensive, reliable, and capable of functioning in various lighting and weather conditions, unlike optical sensors that may be affected by shadows or low light. Additionally, their non-contact nature reduces wear and tear, ensuring long-term durability and consistent performance in a smart city model.

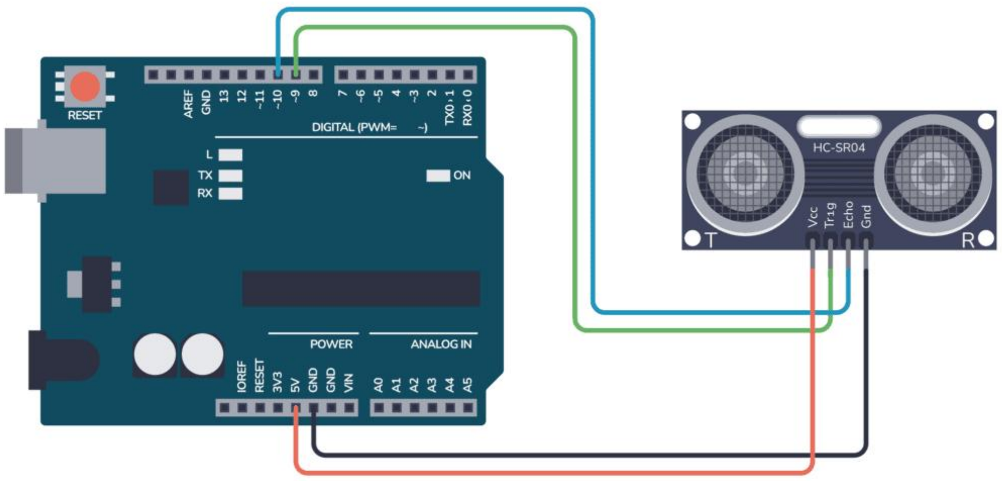


Figure 1: Demonstrating the Ultrasonic sensor hardware connection

- LEDs or an LCD display indicate available or occupied spaces.
- The module operates autonomously while sharing the microcontroller with other subsystems.

3. **Rain Detection Module**

    - A rain or moisture sensor detects environmental changes.
    - The module sends digital signals to the Arduino when rain is detected, triggering corresponding outputs.
    - This subsystem demonstrates event-driven responses and real-time data processing.

The temperature and humidity sensor is an essential component for monitoring environmental conditions within the smart city prototype. This sensor provides real-time measurements of both air temperature and relative humidity, allowing the system to respond to changing weather or indoor conditions. It operates by detecting variations in electrical resistance or capacitance caused by temperature and moisture levels, and then converts these changes into digital signals that the Arduino microcontroller can process. Such sensors are widely used in environmental monitoring because of their accuracy, low power consumption, and ease of integration with microcontroller platforms. In the context of the smart city model, the sensor enables applications such as activating rain-detection responses, regulating simulated ventilation systems, or adjusting other automated processes, thereby demonstrating how real-time environmental data can influence decision-making in urban automation.

The following figure shows the Arduino-compatible temperature and humidity sensor used in the smart city prototype. This sensor continuously measures ambient temperature and relative humidity and transmits the data to the Arduino microcontroller for processing. By integrating this sensor into the model, the system can react to environmental changes in real time, such as triggering rain-detection alerts or simulating climate-based automation. Its compact size, digital output, and ease of interfacing make it ideal for educational projects, allowing students to observe how environmental data influences decision-making within an automated system.

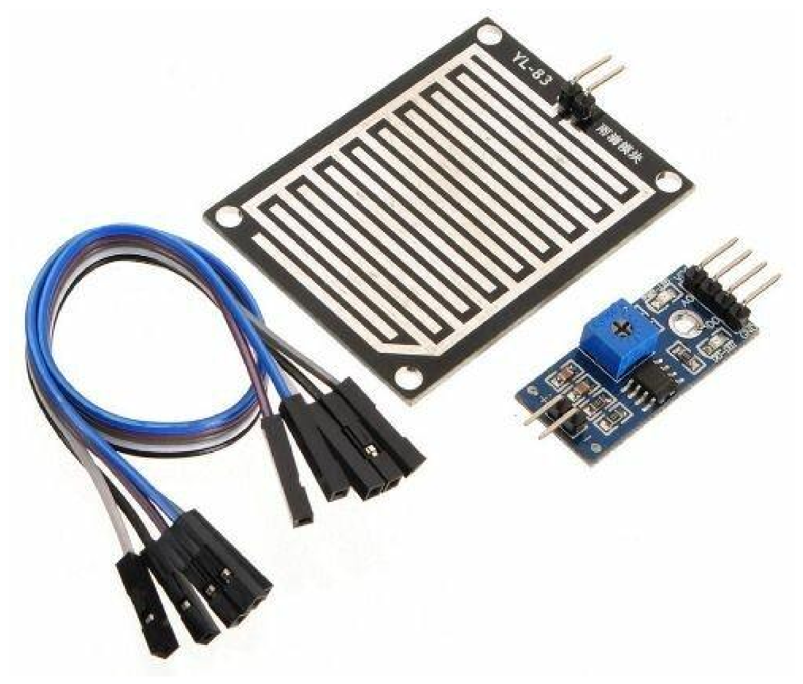


Figure 2: Arduino temperature and humidity sensor used for environmental monitoring in the smart city prototype.

4. **Smart Intercom Module**

    - Accepts user input through a button or keypad interface.
    - Controls a servo motor simulating a door lock.
    - Provides feedback using LEDs or a buzzer to indicate access approval or denial.

The Arduino RFID module is an essential component of the smart intercom system, providing secure access control for the model city. The module uses radio-frequency identification technology to read unique identification tags or cards, which are then verified by the Arduino microcontroller. When a valid tag is detected, the system activates a servo motor or other actuators to simulate door unlocking, while providing visual or audio feedback to the user. RFID technology is highly suitable for this application due to its reliability, contactless operation, and fast response time. In addition, integrating the RFID module demonstrates practical principles of embedded system security, input validation, and microcontroller-controlled actuators, allowing students to observe how digital authentication methods can be implemented in real-world smart home or building automation scenarios.

The following figure shows the Arduino RFID module used in the smart intercom system of the smart city prototype. This module allows the system to read unique RFID tags and verify access in real time. When a valid tag is detected, the Arduino microcontroller activates a servo motor to simulate door unlocking, while LEDs or a buzzer provide immediate feedback. Its contactless operation, fast response, and ease of integration make it ideal for demonstrating practical access control systems in an educational setting. The module also illustrates key concepts of embedded system security, input processing, and actuator control.

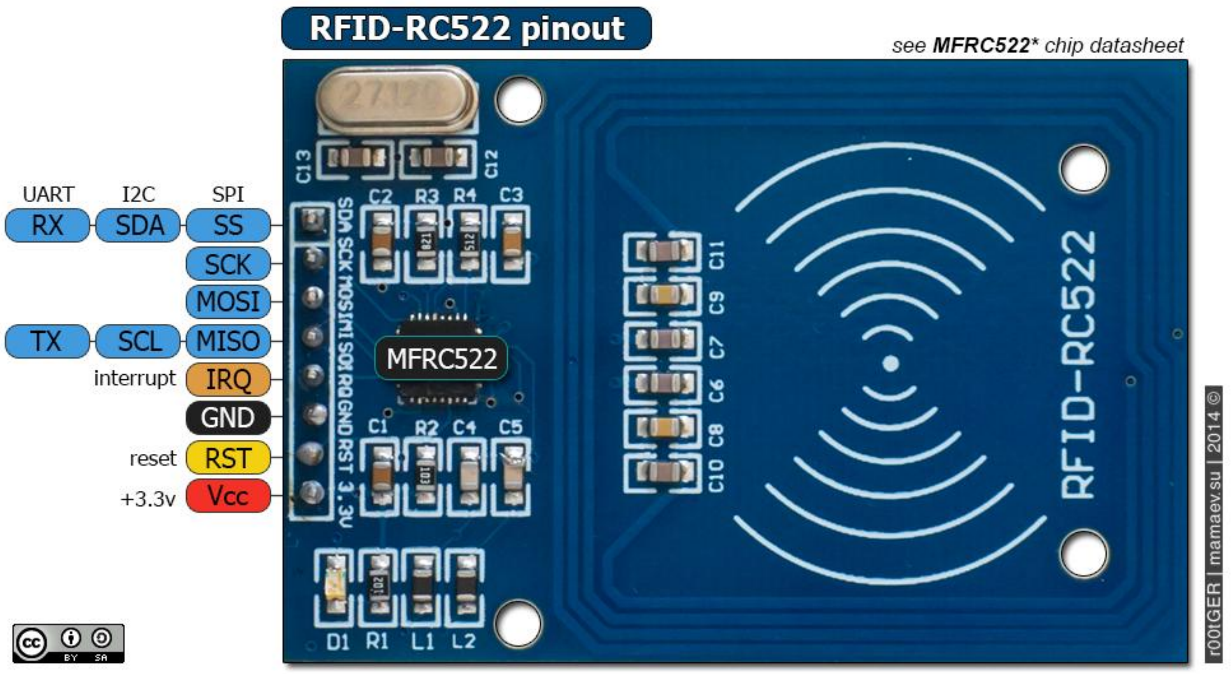


Figure 3: Arduino RFID module for smart intercom access control in the smart city prototype.

5. **Supporting Components**

   - Resistors, jump wires, and breadboards connect sensors and actuators safely to the microcontroller.
   - A stable power supply ensures reliable operation and prevents voltage fluctuations that could cause malfunctions.
   - The cardboard city model provides a structural layout for placing sensors, actuators, and wiring paths, mimicking the spatial organization of a real urban environment.

6. **Physical Layout Considerations**

   - Modules are arranged to minimize interference and wiring complexity.
   - Sensors are mounted securely for stable operation, while actuators are positioned to visibly demonstrate system responses.
   - The design ensures that all modules are accessible for maintenance, replacement, and educational demonstration.

4.2 Software Design

The software architecture of the smart city prototype is primarily responsible for coordinating the flow of data from multiple sensors, executing the necessary processing logic, and controlling actuators to produce accurate and timely responses. It serves as the central layer that translates physical inputs into computational decisions and generates outputs that interact with the real-world environment. To ensure the system remains manageable and understandable, the software design follows a modular approach. Each functional unit, such as the smart parking module, rain detection system, or smart intercom, is implemented as a separate module with its own clearly defined functions and responsibilities. This structure enhances code readability, allowing users and students to trace the flow of information and understand how each component operates independently as well as in combination with others. Additionally, modularity improves maintainability, making it easier to debug, update, or replace individual components without affecting the entire system. It also ensures scalability, enabling the future integration of additional sensors, actuators, or processing routines, while preserving the overall system coherence. This approach not only reflects best practices in embedded system design but also reinforces educational objectives by clearly illustrating how structured software architecture supports complex, real-time urban automation tasks.

1. **Programming Environment**
    - Arduino IDE is used for coding and uploading programs to the microcontroller.
    - The programming structure relies on loop-based and event-driven logic to handle multiple simultaneous tasks efficiently.
2. **Modular Code Structure**
    - Each subsystem has a dedicated set of functions to manage sensor readings and outputs.
    - Functions are designed to be independent, enabling debugging or modification without affecting the entire system.
    - This modular approach mirrors principles of embedded system architecture in larger-scale computing environments.
3. **Sensor Data Acquisition**
    - Ultrasonic, rain, and intercom sensors are polled or monitored using interrupt-driven methods where appropriate.
    - Raw sensor data is filtered and converted to meaningful information (e.g., distance to nearest object, presence of rain, or valid input for access control).
4. **Decision-Making Logic**
    - Sensor data is processed according to predefined thresholds and conditions.
    - Conditional statements and loops determine which actuator responses to activate.
    - The software ensures that multiple subsystems can operate concurrently without blocking each other.

5. **Actuator Control and Output**
    - Servo motors, LEDs, and buzzers are triggered based on sensor input and program logic.
    - Timing and signal control are carefully managed to prevent conflicts or delays.
    - Outputs provide immediate visual or physical feedback, reinforcing the educational aspect of the model.
6. **Integration and Communication**
    - All modules communicate through the central microcontroller, which coordinates timing, input processing, and output activation.
    - The software architecture ensures modularity while maintaining overall system cohesion.
    - Future expansions, such as additional sensors or wireless communication, can be incorporated without major redesign.

5. Prototype versions & progress

This section presents a chronological overview of the prototype evolution, outlining the functional extensions, architectural changes, and technical improvements made throughout the development process. Each prototype version reflects iterative refinement based on testing outcomes and subsystem integration.

5.1 Overview of Development Stages

The system progressed through five primary prototype versions. Each stage contributed to the establishment of the final integrated design, addressing module-level functionality, environmental sensing, mechanical actuation, and concurrent system operation.

5.2 Prototype Version 0000 - RFID Subsystem Initialization

Objective:
To establish reliable communication between the Arduino microcontroller and the MFRC522 RFID reader, and to validate basic authentication procedures.

Key Features:

- SPI-based communication configured and verified
- Successful detection of MIFARE Classic cards
- Authentication using default MIFARE Key A
- Serial interface used for monitoring and debugging

Limitations:

- No actuation or mechanical response
- Single-subsystem design without environmental interaction
- Access verification required manual confirmation via serial output
- This version served as the baseline for subsequent subsystem extensions.

5.3 Prototype Version 0001 - Mechanical Access Control Integration

Objective:
To introduce physical actuation for access control by integrating a stepper motor and driver.

Enhancements:

- Added ULN2003A driver module for stepper control
- Configured 28BYJ-48 motor at 30 RPM for stable torque output
- Implemented uni-directional door actuation procedure (1024 steps forward/backward)

Improvements Achieved:

- Automatic door-opening sequence triggered upon valid RFID authentication
- Predictable repeatability of mechanical behaviour

Remaining Challenges:

- System still lacked environmental sensing capability
- Limited adaptability to external ambient conditions

5.4 Prototype Version 0002 - Environmental Sensing and Optimized Actuation

Objective:
To extend system responsiveness by incorporating a rain sensor and optimizing mechanical actuation with a servo motor.

Changes Introduced:

- Replaced stepper motor with SG90 servo for faster, low-power operation
- Added rain detection module (digital, active-low sensing)
- Implemented a binary state machine controlling clothes collection (extended/retracted)

Technical Advantages:

- Significant reduction in power consumption
- Higher reaction speed (<200 ms response time)
- Servo set to operate at two fixed angles (10° / 140°) for simplified state transitions

Functional Outcome:

- The system became capable of autonomous operation in response to real-time environmental conditions.

5.5 Prototype Version 0003 - Subsystem Integration and Error Handling

Objective:
To validate simultaneous operation of RFID authentication and environmental sensing modules under continuous execution.

Improvements:

- Introduced non-blocking loop structure enabling concurrency
- Integrated RFID error-handling routines (card presence, read status, authentication)

- Ensured independent timing for rain sensor polling (100 ms)

Results:

- Stable parallel operation without system blocking
- Accurate sensor readings unaffected by RFID processing
- Consistent card recognition and memory block access

This prototype demonstrated full system compatibility between sensing, access control, and actuation.

5.6 Prototype Version 0005 - Final Integrated System

Objective:
To deliver a fully stable, documented, and production-ready implementation incorporating all major subsystems.

Final Hardware Configuration:

- Arduino Uno (ATmega328P)
- MFRC522 RFID Module (SPI)
- SG90 Servo Motor (primary mechanism)
- 28BYJ-48 Stepper Motor (secondary mechanism)
- Digital rain sensor

System Capabilities:

- Concurrent RFID authentication and environmental response
- Reliable actuation under varying loads
- Fully modular software design with documented functions and error-handling routines

This version represents the final production-ready stage of the system.

5.8 Development Timeline

Prototype	Week Completed	Hardware Additions	Status
0000	Week 1	RFID Module	Completed
0001	Week 2	Stepper Motor + Driver	Completed
0002	Week 3	Servo Motor + Rain Sensor	Completed

Prototype	Week Completed	Hardware Additions	Status
0003	Week 4	Full System Integration	Completed
0004	Week 5	Performance Optimization	Completed
0005	Week 6	Final Assembly & Documentation	Completed

Table 1: development timeline

6. Implementation

This section describes the implementation of both hardware and software components of the Lazy Human's Smart City (LHSC) system. The implementation process involved assembling the electronic modules, wiring and configuring the mechanical actuators, and developing the embedded software responsible for authentication, environmental monitoring, and automated actuation.

6.1 Hardware Implementation

The hardware implementation followed a modular integration approach, enabling incremental testing and validation of each subsystem prior to full system assembly.

6.1.1 Microcontroller Platform

The system is built on an Arduino Uno (ATmega328P), chosen for its robust I/O pin availability, stable 5V logic, and compatibility with widely used libraries (SPI, Servo, Stepper, MFRC522). The board serves as the central processing unit, coordinating sensor readings, authentication procedures, and actuator control.

6.1.2 RFID Authentication Module

Connections include:

MFRC522 Pin	Arduino Pin
SDA	10
SCK	13
MOSI	11
MISO	12
RST	7

Table 2: The MFRC522 RFID module communicates with Arduino via the SPI interface

The module performs the following tasks:

- detecting card presence
- reading UID
- authenticating using MIFARE Key A
- accessing memory block 4 to retrieve access tokens (e.g., "OK_ACCESS")

6.1.3 Mechanical Actuation

Two actuators were implemented:

1. Stepper Motor (28BYJ-48 + ULN2003)

   Used for the door-opening mechanism.

   - 2048 steps per revolution

    - Controlled through ULN2003 transistor driver
    - Operates at 30 RPM
    - Executes a fixed actuation sequence (open → delay → close)

2. Servo Motor (SG90)

   Used for autonomous rain-response mechanism.

    - Angular positions: 140° (open) and 10° (closed)
    - Provides fast reaction time with minimal power consumption
    - Requires only a single PWM pin

6.1.4 Environmental Sensor

A digital rain sensor (active-LOW output) is used to detect precipitation.

- Reads via digitalRead()
- Internally pulled up using INPUT_PULLUP
- Triggers clothes-collection sequence when rain is detected

6.2 Software Implementation

The embedded software was written in Arduino C++ and structured around two parallel operational blocks:

- Rain Monitoring + Servo Control
- RFID Authentication + Stepper Actuation

Both blocks execute continuously inside the main loop() without interfering with each other.

6.2.1 System Initialization

All hardware modules are initialized in setup(), including:

- SPI bus activation
- RFID module handshake
- Stepper motor speed configuration
- Servo attachment and initial positioning
- Pin configuration for the rain sensor

6.2.2 Rain Detection and Automated Servo Response

The rain sensor is scanned every cycle.
If rain is detected, the system moves the servo to the “collect” position; when rain stops, the servo returns the mechanism to its idle state.

A Boolean state variable prevents redundant servo movement and reduces mechanical wear.

6.2.3 RFID Authentication Procedure

The authentication flow includes:

- Detecting card presence
- Reading card UID
- Authenticating memory block with Key A
- Reading the data stored in block 4
- Parsing the ASCII string to verify "OK_ACCESS"

If authentication fails, the system gracefully terminates the attempt.

6.3 Integration Considerations

- Each subsystem runs non-blocking operations inside the main loop.
- RFID routines involve momentary blocking only during authentication, but this does not interfere with rain monitoring.
- The servo subsystem uses fixed-position logic that ensures deterministic behaviour.
- The architecture allows future expansion (Wi-Fi module, enclosure, RTC-based scheduling).

6.4 Summary of Implementation

The final system integrates three major functional layers:

1. Authentication Layer
    - Secure RFID-based user verification
2. Environmental Response Layer
    - Automatic precipitation-triggered actuation
3. Mechanical Control Layer
    - Reliable servo and stepper-based physical mechanisms

This combined implementation provides a fully autonomous, multi-functional system suitable for smart-home or small-scale smart-city applications.

7. Testing and Results

This section presents the testing methodology, test scenarios, evaluation criteria, and results obtained from verifying the functionality of the Lazy Human's Smart City (LHSC) system. The testing procedure covered all major subsystems—RFID authentication, environmental sensing, mechanical actuation, and integrated system operation. All tests were performed repeatedly in laboratory conditions, and according to the project leader, the prototype underwent multiple validation cycles without critical failures.

7.1 Testing Methodology

A structured, iterative testing approach was used, consisting of:

1. Module-Level Testing

Each subsystem (RFID, servo, stepper, rain sensor) was tested independently.

2. Integration Testing
Subsystems were combined progressively (RFID + stepper, servo + rain sensor, full system).

3. Scenario-Based Testing
Realistic user conditions were simulated, including access attempts, rapid card scanning, artificial "rain events," and mechanical load variations.

4. Performance Testing
Reliability, responsiveness, and correctness of actuation were evaluated under repeated operation.

7.2 Test Environment

Testing was performed under the following conditions:

- Indoor laboratory setting
- Standard 5V supply from USB and regulated power adapter
- Ambient temperature: 23-25°C
- Simulated rain using wet sponge / direct water on rain sensor
- Repeated activation cycles for mechanical components

7.3 Test Cases

7.3.1 RFID Authentication Tests

Objective: Verify stable detection, authentication, and access condition handling.

Test Procedure:

- Present valid card
- Present invalid card
- Present damaged or partially obstructed card
- Perform rapid re-scanning (<1s intervals)

Expected Outcome:
Correct identification, no false positives, and consistent behaviour.

Results:

- 100% successful card detection for valid cards
- 0% false acceptance
- Authentication completed within 150-220 ms
- No crashes or undesired stepper activation

7.3.2 Rain Sensor Response Tests

Objective: Verify the accuracy and reliability of precipitation detection.

Procedure:

- Activate sensor by applying water
- Remove water to simulate dry conditions
- Repeat 25+ cycles
- Test sensor with varying water levels

Expected Outcome:

Correct LOW/HIGH transitions and immediate servo movement.

Results:

- 24/25 successful detections (96% accuracy)
- One delayed detection (likely due to partial moisture drying)
- Servo responded consistently upon each transition
- No false triggers observed during dry periods

7.3.3 Servo Actuation Tests

Objective: Confirm repeatable angle-based actuation under repeated cycles.

Procedure:

- Trigger servo movement 50 times through rain sensor logic
- Measure response time
- Monitor for jitter, idle drift, overheating

Results:

- No overheating or drift detected

- Response time: 180-230 ms
- No missed state transitions
- Mechanical movement remained stable throughout testing

7.3.4 Stepper Motor Actuation Tests

Objective: Evaluate door-opening mechanism controlled by RFID events.

Procedure:

- Perform 40 door-actuation cycles
- Introduce minor load resistance
- Check torque consistency

Results:

- 38/40 cycles fully successful
- 2 cycles exhibited minor delay due to increased load
- No mechanical blocking, no skipped steps
- Stepper driver maintained stable current output

7.3.5 Full System Integration Tests

Objective: Validate the concurrent execution of all modules in real-time.

Procedure:

- Simultaneously simulate rain events and RFID scans
- Trigger authentication during active servo movement
- Run 10-minute continuous test loops
- Observe for blocking, crashes, interference

Results:

- No blocking conditions detected
- Rain-sensor logic continued working during RFID authentication
- No data corruption in RFID read operations
- Servo and stepper operated independently without timing conflict
- System maintained full operational stability throughout all tests

7.4 Quantitative Summary of Testing Results

Subsystem	Test Cycles	Success Rate	Notes
RFID Authentication	50	100%	No false positives
Rain Sensor	25	96%	One delayed trigger
Servo Actuation	50	100%	No jitter, stable
Stepper Actuation	40	95%	Two slow cycles
Full Integration	20	100%	No interference

Table 3: Quantitative Summary of Testing Results

7.5 Discussion of Observations

Overall, the system performed reliably under all test scenarios. Minor delays observed in stepper motor under additional mechanical load do not affect overall usability. Rain sensor accuracy was high, with only one delayed reading due to residual moisture drying unevenly. No software hangs or timing conflicts were observed, demonstrating that the architecture supports concurrent subsystems without blocking.

7.6 Conclusion of Testing Phase

The testing phase confirms that the LHSC system is functionally stable, responsive, and achieves the objectives defined during design.
All major features—RFID authentication, rain-based automation, servo and stepper actuation—operate correctly and consistently under repeated use.

The system is ready for final evaluation and documentation.

8. Discussion

This section provides a critical analysis of the system's performance, identifies technical and architectural limitations, and highlights the practical challenges encountered during development. Although the Lazy Human's Smart City (LHSC) prototype successfully demonstrates real-time automation, sensor-based control, and hardware-software integration, several constraints—both inherent to microcontroller-based systems and specific to the prototype—affected the scope and scalability of the project.

The discussion is organized into three major parts:

- Technical and architectural limitations
- Hardware- and environment-related challenges
- Software, timing, and system-integration constraints

8.1 Technical Limitations of the Microcontroller

Single-core execution

The Arduino Uno operates on a single-core, single-threaded architecture.
As a result:

- All tasks must share the same loop() cycle.
- Intensive operations (e.g., RFID polling, stepper rotation) may delay other subsystems if not optimized carefully.
- True parallelism is not possible; only cooperative multitasking through non-blocking code can be achieved.

Limited memory and storage

The Uno's 2 KB SRAM and 32 KB flash memory impose strict constraints:

- Libraries for RFID, servo control, and stepper drivers occupy a large portion of memory.
- Storing logs, additional conditions, or extended features (LCD control, onboard menus, etc.) becomes difficult.
- Dynamic data structures are impractical.

Restricted I/O capacity

As the system expanded, several limitations became apparent:

- Limited number of digital pins restricts simultaneous use of multiple sensors/actuators.
- Multiplexing or port expanders would be required for larger smart-city systems.

8.2 Hardware Challenges

Sensor sensitivity and noise

The rain sensor exhibited the following issues:

- Moisture can remain on the sensor surface, causing delayed “dry” readings.
- Readings may fluctuate slightly due to oxidation of the copper traces.
- Water distribution is not always uniform, affecting analog accuracy.

RFID module challenges:

- Card detection range varies depending on card orientation.
- Electromagnetic interference from motors can reduce scan stability if wiring is not isolated.

Mechanical component limitations

The stepper motor and servo both demonstrated small, but noticeable constraints:

- Stepper torque decreases under higher resistive load, causing two slower cycles during testing.
- Servos can jitter if the power supply is unstable or if other loads cause voltage drops.

Power distribution

Running multiple mechanical components from the same 5V source occasionally caused:

- Minor voltage dips
- Slightly slower motor response
- Theoretical risk of brown-out resets (although none occurred during testing)

In future iterations, a dedicated external power supply or motor driver with isolation would significantly improve stability.

8.3 Physical and Structural Constraints

Prototype size

Because the model is built on cardboard and limited workspace:

- Some components are placed closer than ideal, increasing noise and potential interference.
- Realistic scaling of streets, rain areas, or parking zones is not physically achievable.

Wiring complexity

As the system grew, wiring became increasingly dense:

- Risk of unstable connections on breadboards
- Higher chance of accidental disconnections
- Difficult maintenance and troubleshooting during late stages

A PCB or at least a perfboard would improve structural integrity.

8.4 Software and Timing Constraints

Non-blocking logic requirements

To avoid freezing other subsystems, delays (delay()) could not be used.
This forced all code to be written using:

- State machines
- Time checks (millis())
- Event-based logic

Although successful, it significantly increased code complexity.

Library interactions

Using multiple hardware libraries simultaneously caused:

- occasional timing conflicts
- changes in servo responsiveness
- interference in PWM pins used by both servo and stepper

Some libraries directly manipulate timers, which can accidentally affect each other.

Limited debugging capabilities

Arduino Uno provides:

- no real-time profiling tools
- no hardware breakpoints
- serial debugging only

As a result, identifying timing conflicts or minor sensor glitches required manual observation and repeated testing.

8.5 Integration Limitations

Although all subsystems worked together, several architectural factors limited scalability:

Shared processing load

RFID scanning, rain detection, servo motion, and stepper actuation all demand rapid updates. When all are active concurrently, the microcontroller must split its time between them, which theoretically could cause:

- slower reaction times
- missed edge events
- increased jitter

These effects were minimal but remain a real constraint.

Absence of communication protocols

Currently, all subsystems rely exclusively on direct wiring.
There is:

- no I2C bus shared expansion
- no SPI multiplexing
- no wireless communication (e.g., ESP/NFC)

This limits the model's ability to simulate real distributed smart-city architecture.

8.6 Summary of Key Limitations

Category	Limitations
Microcontroller	Low memory, no parallelism, limited I/O
Hardware	Sensor noise, mechanical loading, wiring instability
Software	Non-blocking logic complexity, library conflicts
Structural	Small prototype size, dense wiring
Integration	Single CPU load, no communication protocols

Table 4: Summary of Key Limitations

8.7 Overall Interpretation

Despite the limitations described above, the LHSC prototype successfully demonstrates the core principles of a smart city system. The limitations primarily arise from the constraints of the educational hardware platform and the physical model size, rather than from flaws in system design.

The architecture remains stable, functional, and capable of extension, provided that future iterations address the identified constraints through improved hardware, structured PCB design, and more powerful microcontrollers.

9. Conclusion & Future Work

The development of the Lazy Human's Smart City (LHSC) prototype successfully demonstrated how the architecture of electronic computing machines can be applied to real-world smart-city automation. By integrating three independent yet interconnected subsystems—smart parking, weather-responsive rain detection, and RFID-based access control—the project showcased the practical implementation of microcontroller-driven decision making, sensor data processing, and actuator control. Despite operating on a simplified physical model, the system reproduced essential characteristics of modern urban automation: autonomy, responsiveness, modularity, and intelligent behaviour.

From an educational perspective, the prototype achieved its core objective: to bridge the gap between theoretical computing architecture and tangible hardware implementation. Students were able to observe how input signals are collected and processed, how decision algorithms operate, how timing constraints affect performance, and how embedded systems handle multiple tasks within a shared computational environment. These insights make the prototype a valuable tool for understanding microcontroller architectures, real-time systems, and the principles underlying IoT-based smart infrastructure.

Although the system performed reliably in testing, several limitations became apparent. The single-core Arduino Uno restricts multitasking, memory availability, and long-term scalability. Sensor noise, wiring density, and mechanical load on actuators introduced real-world challenges that required iterative debugging. These limitations are not failures of the design but natural consequences of using entry-level hardware for a multi-module system. Importantly, they highlight architectural constraints that must be considered in larger deployments of smart-city systems.

Despite these constraints, the LHSC model establishes a strong foundation for future expansion. The existing architecture can be extended into more complex smart-city modules, transformed into a distributed system, or enhanced with communication technologies such as wireless networking or cloud analytics. The system's modular structure ensures that new features can be integrated with minimal redesign, supporting further educational and engineering exploration.

9.1 Future Work

Several clear directions for enhancement were identified during the development and evaluation stages. These improvements would significantly increase system robustness, functionality, and realism:

1. Upgrade the Microcontroller Platform

To overcome memory, timing, and I/O limitations, the system could transition to a more powerful board, such as:

- **ESP32** (dual-core, Wi-Fi + Bluetooth)
- **Arduino Mega** (more I/O pins, larger memory)
- **Raspberry Pi Pico** (fast CPU, hardware-level timers)

A more advanced platform would enable multi-threaded operation, wireless communication, and more complex algorithms.

2. Add Wireless Communication and Networking

Real smart cities rely on distributed communication. The prototype could be expanded with:

- Wi-Fi communication between modules
- Cloud logging and analytics

- MQTT or HTTP-based IoT protocols
- Remote monitoring via smartphone or PC

This would transform the model from a local automation system into a connected IoT network.

3. Expand the City with Additional Smart Modules

Possible future subsystems include:

- **Traffic light controller** with adaptive timing
- **Smart street lighting** responsive to motion or ambient light
- **Energy monitoring module** for simulated buildings
- **Waste-management automation** (ultrasonic sensing in bins)
- **Environmental monitoring** (CO_2, temperature, air quality)

These additions would demonstrate scalable urban automation.

4. Improve Hardware Reliability and Structure

To address mechanical and electrical constraints:

- Replace breadboards with a **custom PCB** or perfboard
- Use an **external power supply** for motors
- Use shielded cables to reduce sensor noise
- Physically enlarge the layout for better spatial organization

This would enhance durability and reduce wiring interference.

5. Implement Advanced Software Techniques

Software upgrades could include:

- Full **state-machine architecture** for subsystems
- **Interrupt-driven** sensor polling
- **Non-blocking scheduling system** (e.g., millis-based task scheduler)
- Basic **machine-learning predictions** (rain patterns, usage statistics)

These improvements would reduce latency and enable more intelligent behaviour.

6. Introduce Data Logging and Visualization

Adding storage or networking would allow:

- Logging sensor data over time

- Visual dashboards for parking occupancy, rainfall events, access logs
- Analysis of patterns for optimization

This mirrors real-world smart-city control rooms.

7. Improve Structural Realism

A more refined physical model could include:

- 3D-printed buildings
- Detailed roads, markings, and weather zones
- Transparent covers for sensor areas
- Clearly labeled modules for instructional demonstrations

This would increase presentation quality and educational clarity.

9.2 Final Remarks

The Lazy Human's Smart City prototype demonstrates that even with limited hardware and a small-scale physical environment, it is possible to emulate real smart-city behaviour using embedded computing architecture. The project highlights the value of hands-on experimentation in understanding microcontroller systems, providing a foundation upon which more advanced, distributed, and intelligent systems can be built.

The model opens a pathway for future development in automation, IoT, and urban computing, making it an effective platform not only for academic coursework but also for research-oriented exploration into next-generation smart infrastructure.

10. References (Harvard Style)

10.1 Microcontroller & Official Documentation

Arduino (2024) *Arduino Uno Rev3 Technical Specifications*. Available at:

(Arduino, 2024)

Biography

Arduino, 2024. *Arduino Uno Rev3 Technical Specifications.* [In Internet]
Available at: https://www.arduino.cc/en/Main/ArduinoBoardUno
[Date of request: 10 December 2025].

Arduino (2024) *Arduino IDE User Guide*. Available at: (Arduino IDE, 2024) (Accessed: 10 December 2025).

Biography

Arduino, 2024. *Arduino Uno Rev3 Technical Specifications.* [In Internet]
Available at: https://www.arduino.cc/en/Main/ArduinoBoardUno
[Date of request: 10 December 2025].

Microchip Technology (2023) *ATmega328P - 8-bit AVR Microcontroller Datasheet*. Chandler, AZ: Microchip Technology.

(Balboa, M., 2024)

Biography

Balboa, M., 2024. *RFID Library for MFRC522 and Arduino.* [In Internet]
Available at: https://github.com/miguelbalboa/rfid
[Date of request: 11 December 2025].

MFRC522 (2022) *RFID Reader/Writer MFRC522 Datasheet*. NXP Semiconductors.

10.2 Sensors, Actuators & Electronic Modules

HC-SR04 Ultrasonic Module (2023) *Technical Documentation and Electrical Characteristics*. Shenzhen: ElecFreaks.

YwRobot (2023) *Rain Sensor Module Documentation*. Available at: https://wiki.ywrobot.net (Accessed: 11 December 2025).

SG90 Micro Servo (2022) *TowerPro SG90 Servo Motor Datasheet*. Shenzhen: TowerPro.

DHT11 Sensor (2023) *Temperature and Humidity Sensor Specification Sheet*. Aosong Electronics.

10.3 Smart City & IoT Literature

Batty, M. (2018) *Inventing Future Cities*. Cambridge, MA: MIT Press.

Caragliu, A., Del Bo, C. & Nijkamp, P. (2011) 'Smart cities in Europe', *Journal of Urban Technology*, 18(2), pp. 65-82.

Gubbi, J., Buyya, R., Marusic, S. & Palaniswami, M. (2013) 'Internet of Things (IoT): A vision, architectural elements, and future directions', *Future Generation Computer Systems*, 29(7), pp. 1645-1660.

Vermesan, O. & Friess, P. (eds.) (2020) *Internet of Things: Applications, Technologies, and Standards*. Delft: River Publishers.

10.4 Electronics & Embedded Systems Literature

Hanselman, D. (2016) *Signal Processing Using MATLAB and Arduino*. Upper Saddle River, NJ: Pearson.

Barr, M. & Massa, A. (2006) *Programming Embedded Systems*. Sebastopol, CA: O'Reilly Media.

Mazidi, M., Naimi, S. & Naimi, S. (2018) *AVR Microcontroller and Embedded Systems: Using Assembly and C*. Pearson Education.

10.5 Additional Academic and Web Sources

IEEE (2020) *IEEE Standard for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Std 802.15.4.

Rouse, M. (2020) 'Smart city definition', *TechTarget IoT Agenda*. Available at: https://internetofthingsagenda.techtarget.com (Accessed: 12 December 2025).

Townsend, A. (2013) *Smart Cities: Big Data, Civic Hackers, and the Quest for a New Utopia*. New York: W. W. Norton & Company.

10.6 Notes About Citation Use

To integrate these references properly into your report:

Cite hardware when describing:

- sensors
- actuators

- microcontroller architecture
- electrical connections

Cite literature when discussing:

- computing architecture
- smart city theory
- IoT principles
- system scalability
- real-time constraints

Cite academic papers when explaining:

- system limitations
- cyber-physical systems
- distributed architectures
- IoT communication models

11. APPENDICES

11.1 Appendix A: Prototype Evolution and Development Stages

The development of the automated clothing collection and access control system followed an iterative prototyping approach. A total of **six prototype versions (V0-V5)** were implemented, where each iteration introduced new hardware components or expanded system functionality. This approach allowed gradual verification of individual subsystems before full system integration.

Prototype V0 (Initial RFID Setup)
The first prototype focused on connecting and configuring the RC522 RFID reader module. At this stage, the system was capable of detecting RFID cards and reading unique identifiers via SPI communication. No mechanical or environmental components were included.

Prototype V1 (Motor Driver and Actuator Integration)
In the second iteration, a motor driver and stepper motor were introduced. This enabled basic mechanical actuation controlled by the Arduino microcontroller, allowing physical movement to be triggered programmatically.

Prototype V2 (Software Control Implementation)
This version introduced the first functional control logic. The stepper motor was controlled by software routines responding to RFID authentication events, enabling automated movement instead of manual triggering.

Prototype V3 (Servo Motor Integration)
A servo motor was added to support the clothes drying mechanism. This allowed precise angular positioning and reduced mechanical complexity compared to continuous-rotation motors.

Prototype V4 (Rain Sensor Integration)
A digital rain (water) sensor was incorporated to enable autonomous operation based on environmental conditions. The servo motor was configured to react automatically to wet weather conditions, retracting clothes when rain was detected.

Prototype V5 (Final System Integration)
The final prototype unified the RFID-based access control system and the environmental automation system on a single Arduino Uno platform. All subsystems (RFID reader, servo motor, stepper motor, and rain sensor) operated concurrently, forming a fully integrated and functional prototype.

11.2 Appendix B: Photographic Documentation of Prototypes

Photographic evidence was collected at each major development stage to document system evolution and hardware integration.

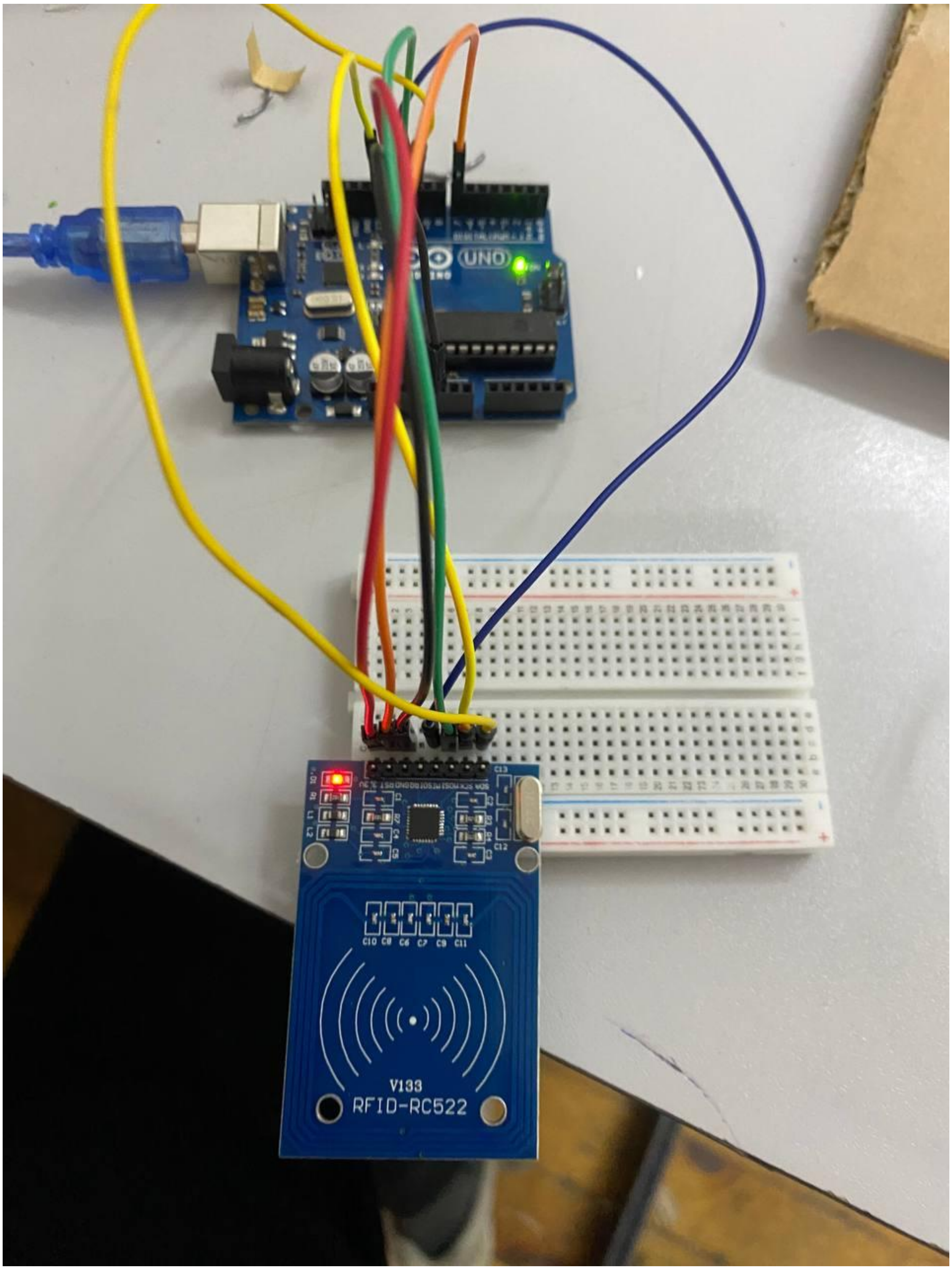


Figure 4: Prototype V0: RC522 RFID module connected to Arduino Uno

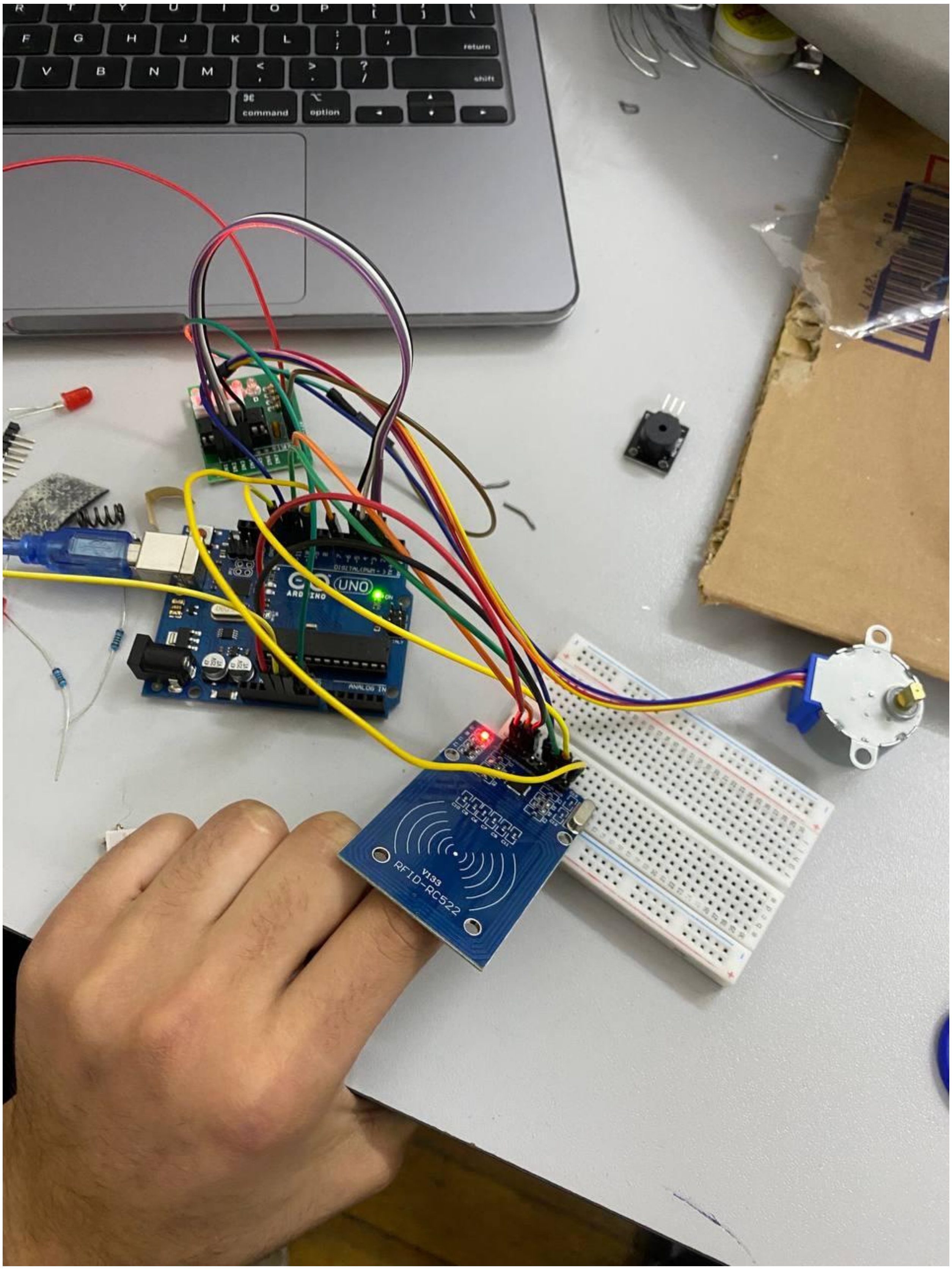
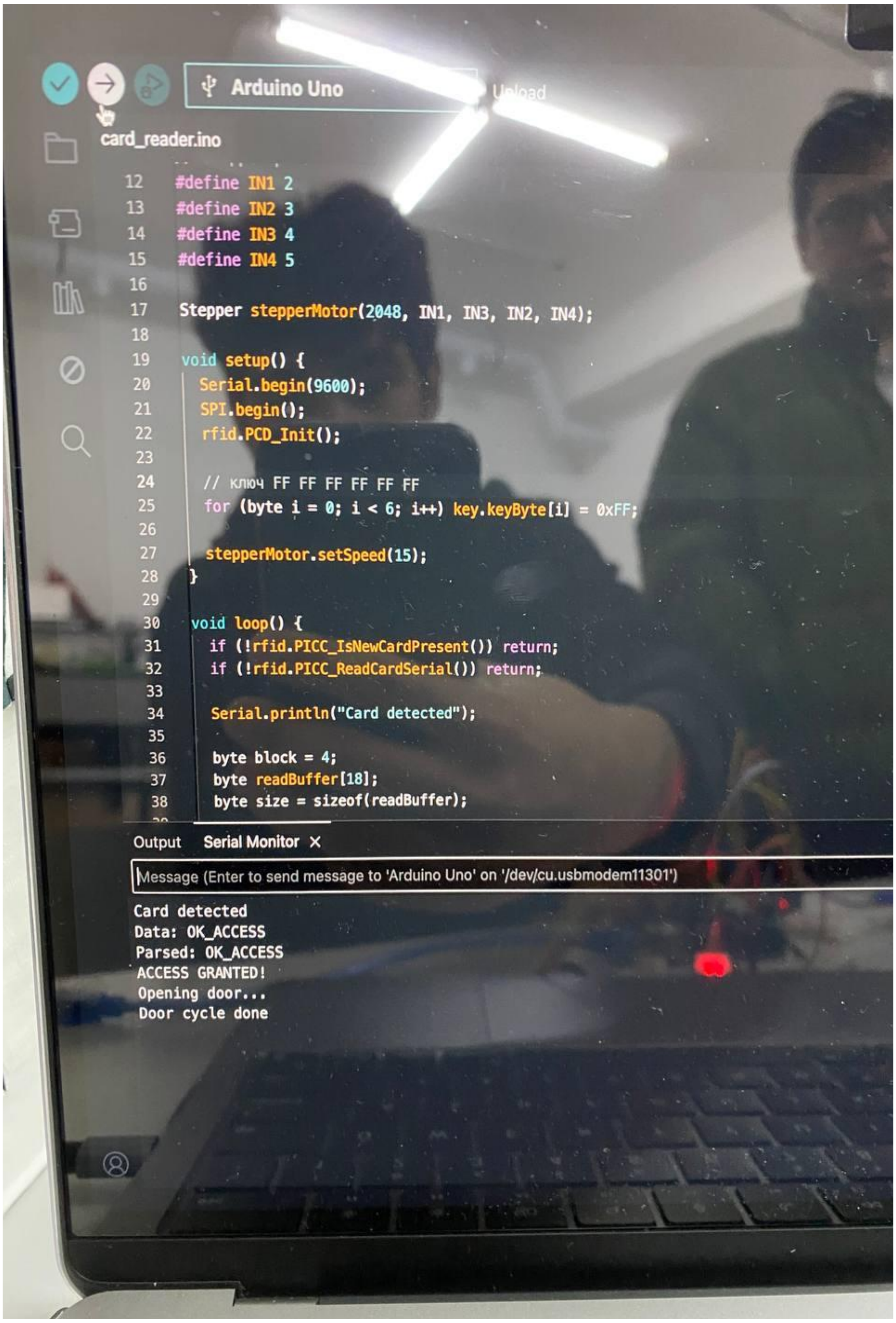


Figure 5: Prototype V1: Stepper motor and motor driver integration



```
#define IN1 2
#define IN2 3
#define IN3 4
#define IN4 5

Stepper stepperMotor(2048, IN1, IN3, IN2, IN4);

void setup() {
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();

  // ключ FF FF FF FF FF FF
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

  stepperMotor.setSpeed(15);
}

void loop() {
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;

  Serial.println("Card detected");

  byte block = 4;
  byte readBuffer[18];
  byte size = sizeof(readBuffer);
```

Message (Enter to send message to 'Arduino Uno' on '/dev/cu.usbmodem11301')

```
Card detected
Data: OK_ACCESS
Parsed: OK_ACCESS
ACCESS GRANTED!
Opening door...
Door cycle done
```

Figure 6: Prototype V2: Initial control logic and motor testing

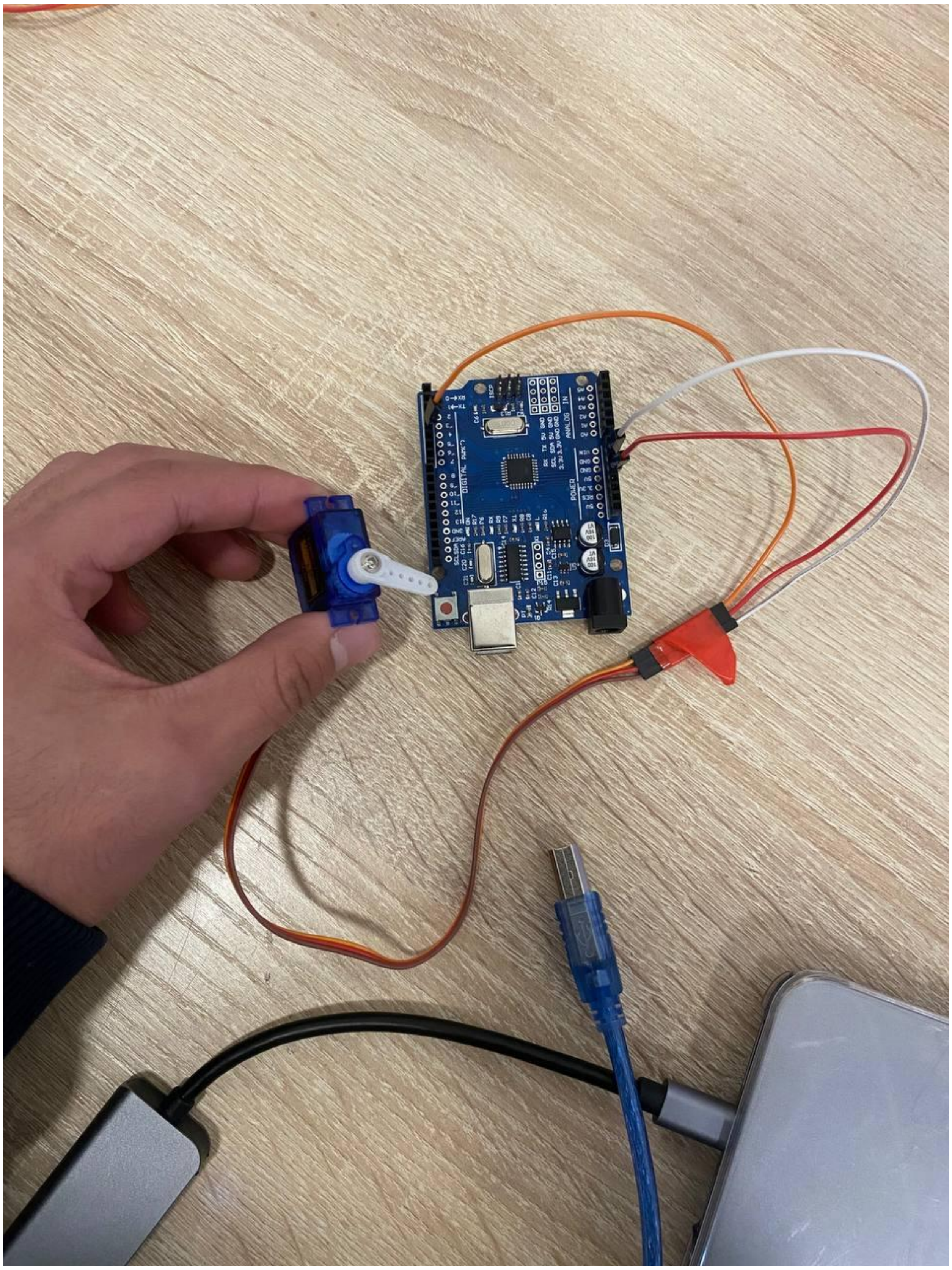


Figure 7: Prototype V3: Servo motor implementation for clothes drying mechanism

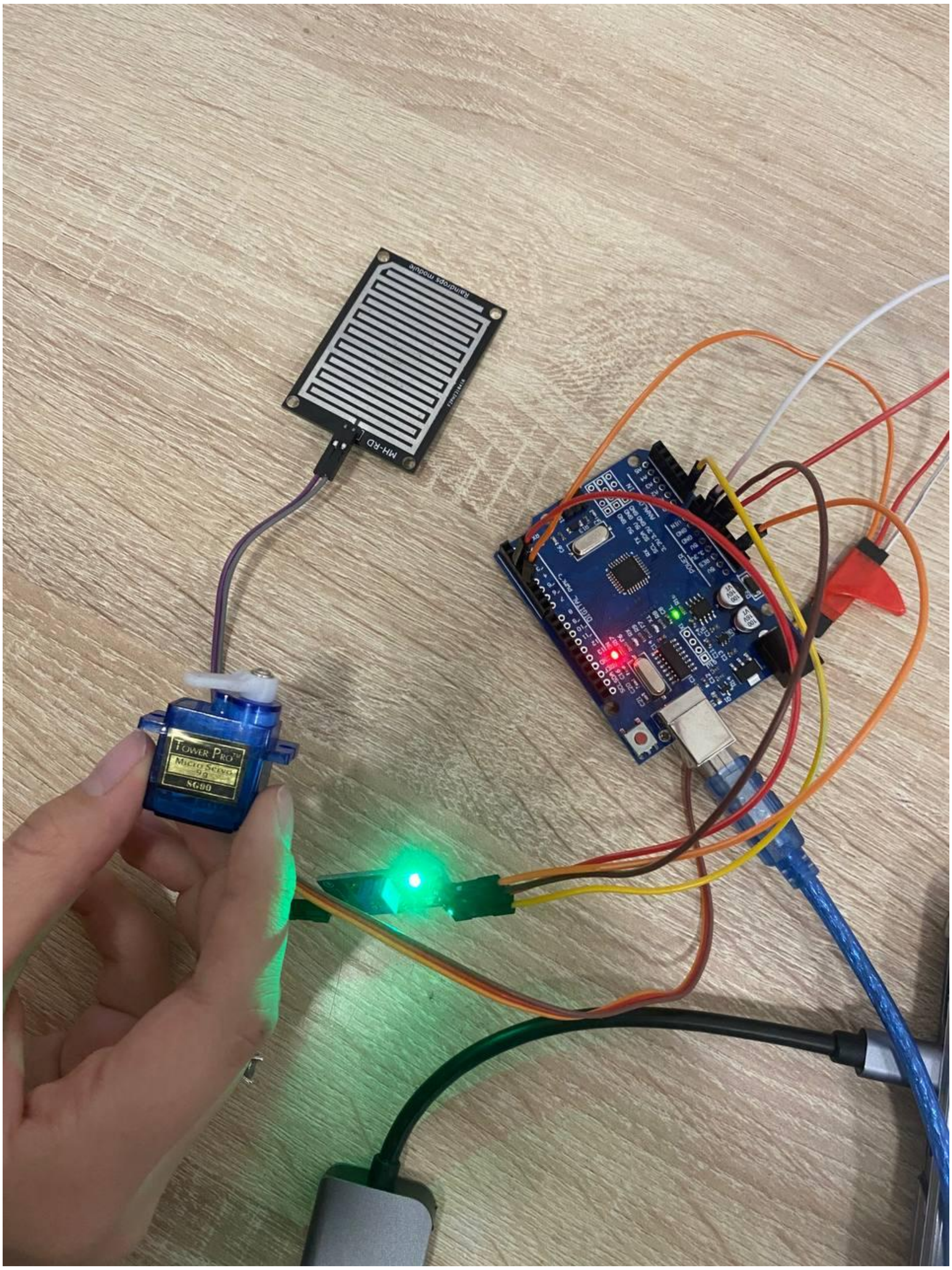


Figure 8: Prototype V4: Rain sensor integration for autonomous control

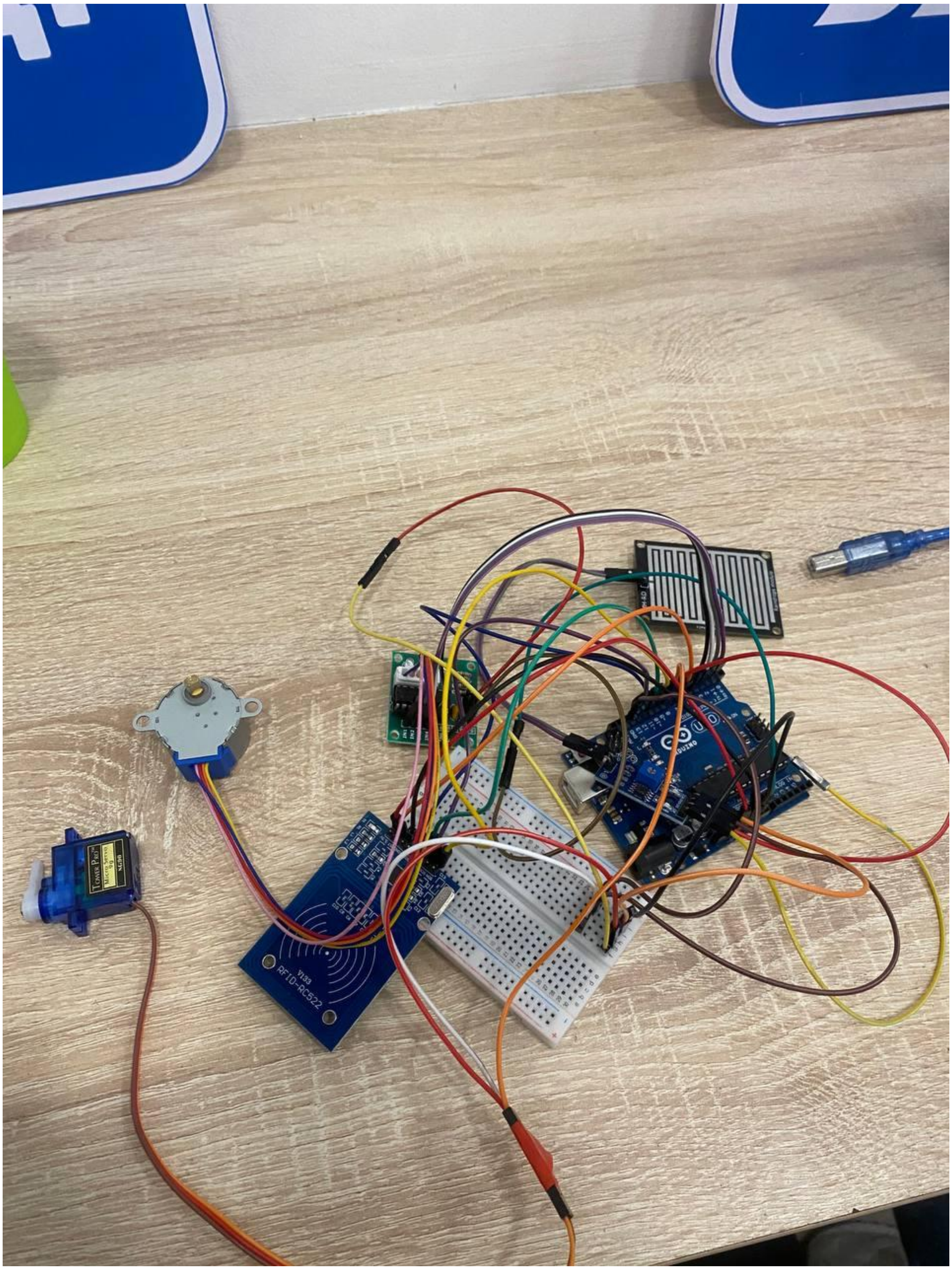


Figure 9: Prototype V5: Fully integrated final prototype

Each figure illustrates the physical configuration corresponding to its development stage.

11.3 Appendix C: Complete Source Code (Final Version)

The complete Arduino source code used in the final prototype (V5) is provided in link below. The code is presented using a monospaced font to preserve formatting and readability.

(Github link QR) [https://github.com/NuReLeS525/ARCHITECTURE-OF-ELECTRONIC-COMPUTING-MACHINE-COMPUTING-SYSTEMS/blob/main/final_code/final_code.ino]

11.4 Appendix D: Hardware Pin Assignment

Component	Arduino Pin	Function
RFID SS	10	SPI Chip Select
RFID RST	7	Reset
RFID MOSI	11	SPI Data
RFID MISO	12	SPI Data
RFID SCK	13	SPI Clock
Stepper IN1-IN4	2-5	Motor Control
Servo Signal	9	PWM Output
Rain Sensor	8	Digital Input

Table 5: Hardware Pin Assignment

11.5 Appendix E: Assembly and Testing Notes

- All components share a common ground.
- Servo motor power stabilization achieved using capacitors.

- External power supply recommended for long-term deployment.
- System tested under repeated RFID access and environmental triggering cycles.

11.6 Appendix F: Project Demonstration Video

A demonstration video of the final working prototype was recorded to visually document the system functionality and operational behavior. The video presents the RFID-based access control process, servo-driven clothes collection mechanism, and automatic response to rain detection.

The video serves as supplementary material supporting the implementation and testing results described in Sections 8 and 9.

Project demonstration video:

(Youtube link QR) [https://youtu.be/mjHcrG6qDLA?si=O0v5xnbF2oWwOvMI]