The Magic Wand

Group #10

Jose Caraballo & Adrian Suarez

Dr. Alhalabi Bassem

Electrical engineering and computer science

12/08/2024

The Magic Wand is a prototype that uses a smartphone to unlock doors through hand gestures. It leverages accelerometer data from a smartphone to generate unique numerical values. These values correspond to a PIN known only to the user for secure access.

Contents

1 Introduction....3

1.1 Problem Description.... 3

1.2 Significance of the Problem....3

1.3 Overview of the Solution....4

1.4 Goals and objectives.... 4

1.5 Related Research....4

2 System Design....4

2.1 Project Requirements.... 4

2.2 Project Requirements Not Met....5

2.3 Project Requirements Added....5

2.4 System Diagram.... 5

2.5 State Diagram....5

3 System Implementation.... 6

3.1 Hardware.... 6

3.2 Software....6

3.3 User Interface.... 6

3.4 Data Communications.... 7

4 Testing and Performance Evaluation....7

4.1 Performance Assessment....7

5 Budget.... 7

6 Project Management and Workload Distribution....7

7 References.... 8

1 Introduction

1.1 Problem Description

The Magic Wand was designed to address common challenges with traditional key-based entry systems. It eliminates the need to fumble for keys, especially when your hands are full, such as carrying groceries, by leveraging the convenience of a smartphone that is often readily accessible. Additionally, it enhances security by requiring hand gestures unique to the user, making unauthorized access significantly more difficult.

1.2 Significance of the Problem

The problem of safe and convenient door access has become relevant to be solved, considering that traditional methods very often lead to frustration and vulnerable points in everyday scenarios. Also, fumbling for keys might get inconvenient while carrying groceries, managing children, or doing something else simultaneously, whereas physical keys can be easily stolen, lost, or copied and thus compromise security. An effective solution is that it provides an avenue where people of various groups are catered for: the house owner, tenant, disabled person, and companies in offering their access to doors a safer, convenient, and more up-to-date solution. Solutions for such include keypad entry systems, RFID technology, and face recognition systems. These alternatives suffer from significant weaknesses. Keypad systems are vulnerable to brute-force attacks, as one is sure to get the correct combination of a 4-digit PIN in many tries. Facial recognition devices, although apparently the height of security, are actually highly prone to being duplicated with a picture or digitized image of the face itself. RFID technology is essentially efficient but not totally tamper-resistant to signal interception or spoofing.

1.3 Overview of the Solution

The Magic Wand offers a revolutionary solution to the challenges posed by traditional door access methods. It makes use of unique hand gestures, recognized through a smartphone's accelerometer that enables users to unlock doors without physical keys or direct contact. By integrating gesture-based access, the Magic Wand adds an extra layer of security since the gestures are known only to the user and are difficult to replicate. This system also maximizes convenience using smartphones, devices that most already carry on and have readily available. Additionally, the Magic Wand allows for preemptive unlocking, meaning the user can open doors when still in range, therefore not having to physically touch either the door or keypad when arriving. Additionally, it has the capability to open the door from a distance remotely, thus allowing trusted people to access the home in their absence. This adds significantly to quality-of-life improvements. Combining these features, the Magic Wand offers a secure, modern, highly user-friendly alternative to traditional key-based systems, keypad entries, RFID technology, and facial recognition systems.

1.4 Goals and objectives

The goal is to develop a keyless door lock with enhanced security. The specific objectives are:

1. Develop a hardware design bluetooth compatible (Adrian Suarez)
2. Develop a hardware design for sleep mode (Adrian Suarez)
3. Develop a control system for different modes of operations (Adrian Suarez)
4. Test all Components separately to ensure proper connection (Adrian Suarez)
5. Develop sleep mode integrated within the whole system (Jose Caraballo)
6. Develop a password storage system (Jose Caraballo)
7. Develop a system code to unlock and lock the locking mechanism (Jose Caraballo)
8. Develop hand gesture interpreter for password creation and verification (Jose Caraballo)
9. Develop password handling systems (Jose Caraballo)
10. Test and debug system software(Jose Caraballo)

1.5 Related Research

The research required to develop the Magic Wand project was extensive, as it involved exploring a wide range of topics to achieve the desired functionality. Key areas of investigation included learning how to securely save passwords within the system, converting accelerometer data into numerical digits (1-6) for gesture recognition, and establishing Bluetooth connectivity with the Dabble app. Additionally, earlier stages of the project involved research into creating a custom mobile application. However, this plan was ultimately set aside in favor of the Dabble app, which conveniently provided the necessary features and integration for the project, significantly simplifying development without compromising functionality.

Links:
https://blog.arduino.cc/2016/09/05/unlock-your-door-with-a-simple-hand-gesture/

https://forum.arduino.cc/t/saving-keypad-password-to-internal-eeprom/235119

2 System Design

2.1 Project Requirements

2.1.1 The sinusoid lock shall unlock after correct password input from hand gestures.

2.1.2 The system shall sleep after predetermined time has passed with no motion detected

2.1.3 The system shall be interrupted when a button is pressed for password setup mode

2.1.4 The system shall be interrupted when a button is pressed for password verification mode

2.1.5 The relay device shall turn over when signal is sent to it

2.1.6 The system shall connect to the Dabble mobile app via Bluetooth using a mobile device

2.1.7 The system shall receive mobile device sensor data

2.1.8 The system shall detect different gestures

2.1.9 The servo motor must unlock after the correct password input from hand gestures.

2.1.10 The custom mobile app shall connect to the system.

2.2 Project Requirements Not Met

Requirement 2.1.9, which originally proposed the use of a servo motor for controlling the locking mechanism, was removed to create a more authentic and practical door lock system. The decision was made to transition to a 12V locking solenoid, aligning the project with real-world door lock standards rather than relying on the less robust servo motor.

Requirement 2.1.10, which proposed the development of a custom mobile application for the Magic Wand system, was removed due to constraints in resources and time. Instead, the project utilized the Dabble app to provide basic functionality and control, streamlining the development process.

2.3 Project Requirements Added

Requirement 2.1.2 was introduced to incorporate a low power mode into the system. This feature ensures that the Magic Wand does not run at full power when idle, thereby reducing energy consumption and extending the lifespan of battery-powered setups.

Requirements 2.1.3 and 2.1.4 were introduced to enable the system to switch between different operational modes specifically, password verification and password creation. This functionality was incorporated directly into the hardware due to compatibility limitations with the Dabble app, which could not support mode-switching via the phone interface.

Requirement 2.1.5 was added to ensure the 12V locking solenoid unlocks reliably when a signal is sent, addressing the limitation that the ESP32 microcontroller cannot directly support or supply 12V. This requirement ensures proper power management and integration of the locking mechanism into the system.

Requirement 2.1.6 was added to leverage an existing software solution capable of handling the core functionality required for the Magic Wand system. This decision ensured the project could-utilize pre-developed software for efficiency and reliability, avoiding the need to build redundant features from scratch.

2.4 System Diagram

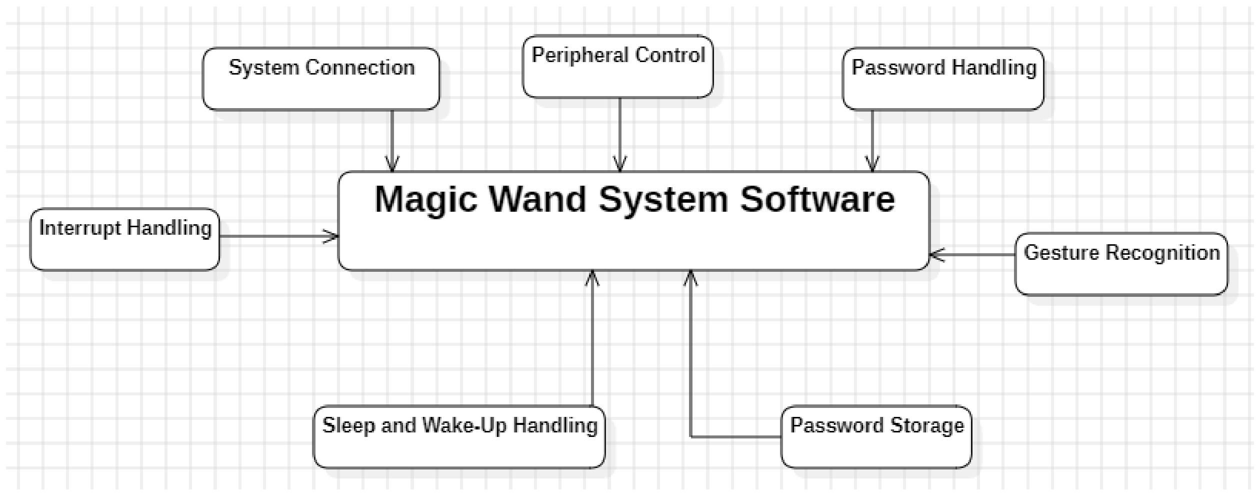


Figure 1. System Block Diagram

The block diagram illustrated the modular design of the Magic Wand System, which consisted of seven interconnected subsystems: System Connection, Gesture Recognition, Interrupt Handling, Password Handling, Password Storage, Peripheral Control, and Sleep and Wake-Up Handling. Each subsystem operated independently, contributing to the overall functionality and efficiency of the system.

At the core of the system was the Magic Wand System Software, coordinating interactions among subsystems. The System Connection Subsystem established Bluetooth communication with the Dabble app, enabling the system to receive accelerometer data from the mobile device for gesture-based control. The Gesture Recognition Subsystem processed this data to identify user gestures for password creation or verification.

User inputs, including button presses and motion detection, were managed by the Interrupt Handling Subsystem, which triggered system state changes such as password setup, verification, or wake-up from sleep. The Password Handling Subsystem oversaw password creation and validation, working closely with the Password Storage Subsystem to securely store and retrieve passwords. Together, they ensured efficient and secure password management.

The Peripheral Control Subsystem managed physical components, including buttons, a motion sensor, and a 12V locking solenoid. Once a valid password was verified, it activated the relay to unlock the solenoid, providing a secure and reliable locking mechanism. The Sleep and Wake-Up Handling Subsystem conserved energy by placing the system into deep sleep after periods of inactivity, with the motion sensor serving as a wake-up trigger to maintain responsiveness.

The design ensured that the 12V solenoid lock unlocked following correct password input by having the Password Handling Subsystem send a signal to the relay, which then activated the solenoid. This practical approach adhered to real-world security standards and offered a dependable unlocking process.

To achieve energy efficiency, the system implemented a low-power sleep mode via the Sleep and Wake-Up Handling Subsystem. After a set period of inactivity, the system entered deep sleep, conserving power. The Interrupt Handling Subsystem allowed the system to awaken when motion was detected, ensuring it remained both energy-efficient and responsive.

User interaction relied on hardware-based interrupts for prompt and reliable responses. Pressing the reset button initiated password setup mode, while pressing the main button enabled password verification mode. This straightforward approach ensured intuitive transitions between modes.

The relay device activated when the Peripheral Control Subsystem received a signal from the Password Handling Subsystem, enabling the ESP32 to control the 12V locking solenoid effectively. This arrangement addressed the ESP32's voltage limitations while ensuring proper power management.

The system established Bluetooth communication with the Dabble app through the System Connection Subsystem, enabling the Gesture Recognition Subsystem to process accelerometer data from the mobile device. This integration ensured accurate inputs for password creation and validation, allowing users to interact with intuitive hand movements.

Originally, the design intended to use a servo motor for locking, but the decision to use a 12V solenoid offered greater security and practicality. Similarly, the project chose the Dabble app instead of a custom mobile application, streamlining the implementation and retaining essential functionality.

The mechanical design focused on a robust and secure locking mechanism with a 12V solenoid, along with securely mounted push buttons and a PIR motion sensor. All hardware components were housed in a protective enclosure, ensuring durability and long-term reliability.

The electrical design met power requirements for all components. A shared power supply delivered 3.3V to the ESP32 and other low-power elements, while a separate 12V supply powered the solenoid. The relay allowed the ESP32 to control the 12V solenoid, and voltage regulators ensured stable power delivery and protected components from damage.

By integrating the ESP32 for subsystem coordination and data processing, employing the PIR motion sensor for sleep and wake-up transitions, using push buttons for password operations, and leveraging Bluetooth communication for gesture-based control, the system maintained a modular, scalable, and reliable architecture. This design met all key requirements by combining energy efficiency, user-friendly interaction, robust security, reliability, and scalability into a cohesive and adaptable security solution.

Analysis of Subsystems:

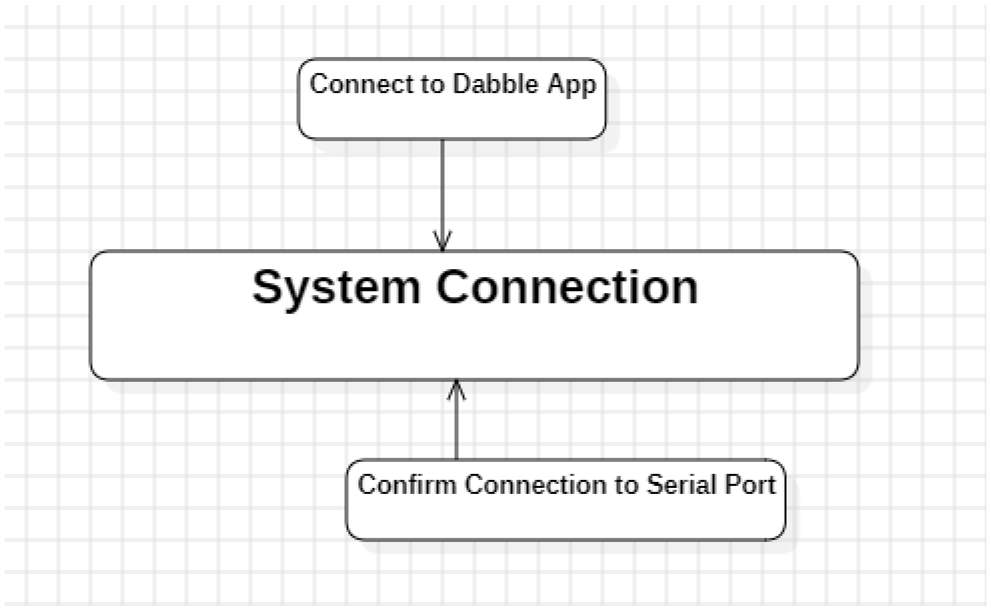


Figure 2. System Connection Block Diagram

- The System Connection Subsystem is responsible for establishing communication between the ESP32, the serial monitor, and the Dabble app. It ensures Bluetooth connectivity to transmit accelerometer data to the Gesture Recognition Subsystem, as well as serial communication for system feedback. This subsystem serves as the foundation for gesture input, enabling interaction between the user and the system.

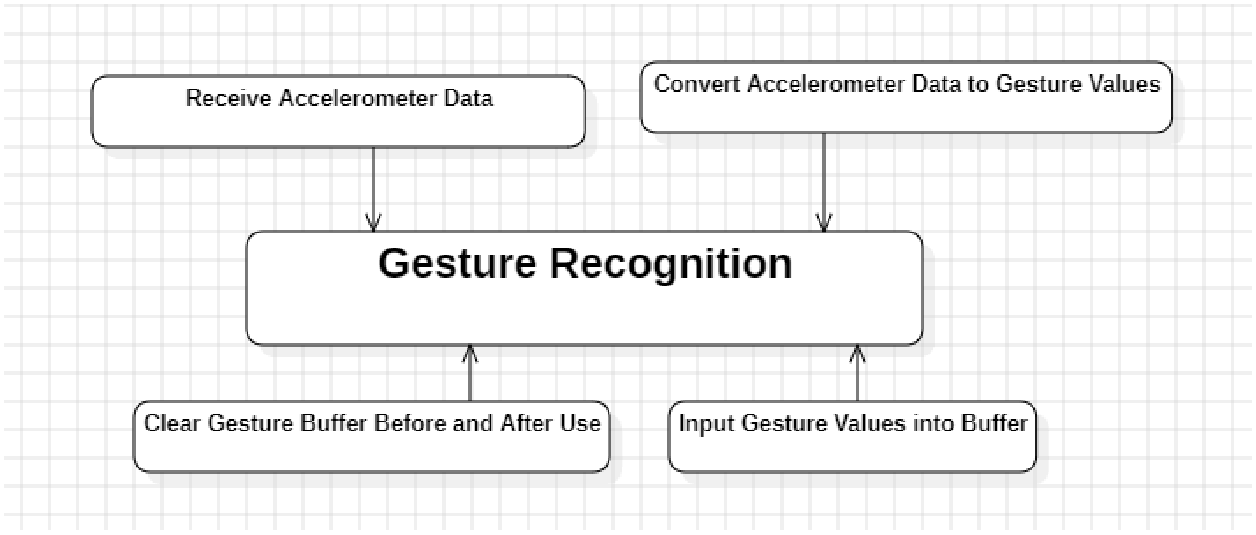

Receive Accelerometer Data
Convert Accelerometer Data to Gesture Values
Gesture Recognition
Clear Gesture Buffer Before and After Use
Input Gesture Values into Buffer

Figure 3. Gesture Recognition Block Diagram

- The Gesture Recognition Subsystem processes accelerometer data received from the Dabble app, filtering raw data and interpreting it to detect user gestures. These gestures are stored in a buffer and passed to the Password Handling Subsystem for password creation or validation. This subsystem ensures accurate gesture recognition, forming the basis of the system's user input mechanism.

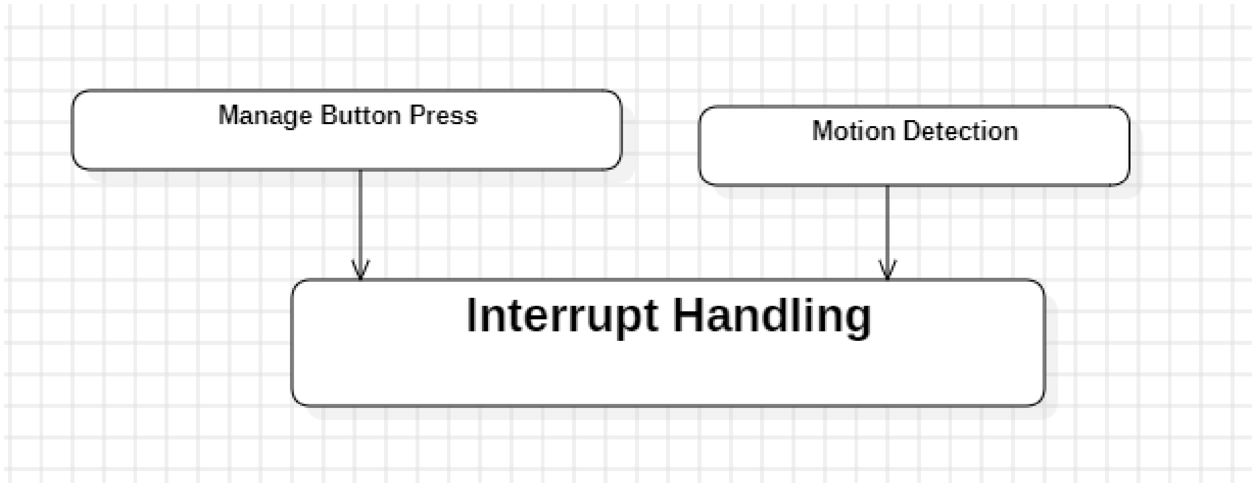


Figure 4. Interrupt Handling Block Diagram

- The Interrupt Handling Subsystem monitors and responds to button presses and motion detection events. It manages triggers for system state transitions, such as initiating password setup or verification, or waking the system from sleep mode. This subsystem ensures that the system reacts promptly to user actions and environmental changes.

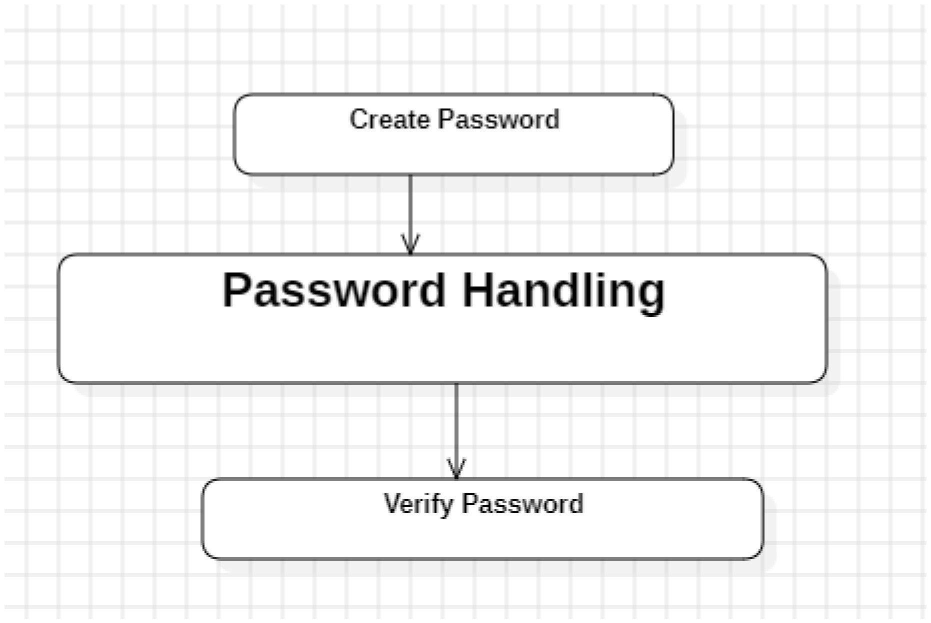


Figure 5. Password Handling Block Diagram

- The Password Handling Subsystem oversees password creation, validation, and user interaction. It processes gesture data from the Gesture Recognition Subsystem and interacts with the Password Storage Subsystem to securely store or retrieve passwords. This subsystem ensures that passwords are managed efficiently and securely, maintaining the system's overall security.

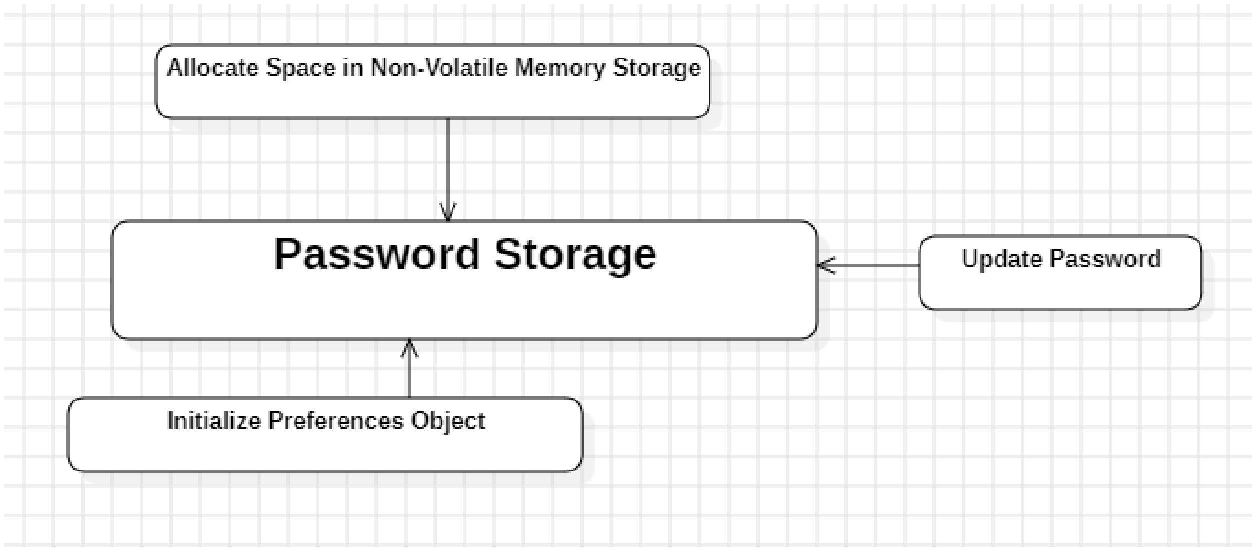

Allocate Space in Non-Volatile Memory Storage
Password Storage
Update Password
Initialize Preferences Object

Figure 6. Password Storage Block Diagram

- The Password Storage Subsystem provides non-volatile storage for user-defined passwords using the ESP32's Preferences Library. It securely stores new passwords and retrieves existing ones for validation by the Password Handling Subsystem. This subsystem ensures that passwords are preserved even during power cycles, supporting the system's reliability.

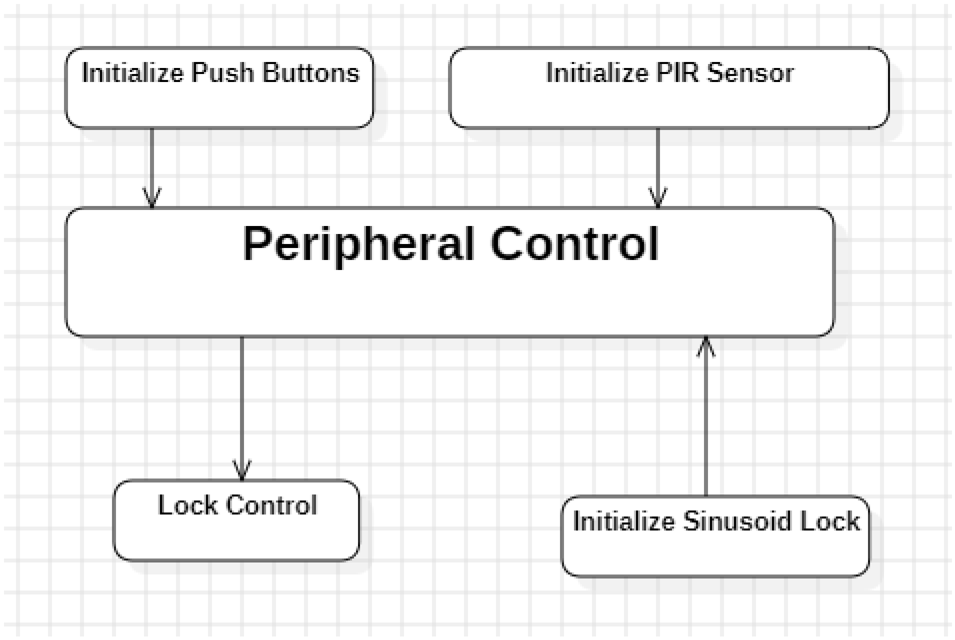


Figure 7. Peripheral Control

- The Peripheral Control Subsystem handles the initialization and management of hardware peripherals such as buttons, the PIR motion sensor, and the lock mechanism. It processes commands from the Password Handling Subsystem to control the lock mechanism, ensuring that the physical components of the system function seamlessly.

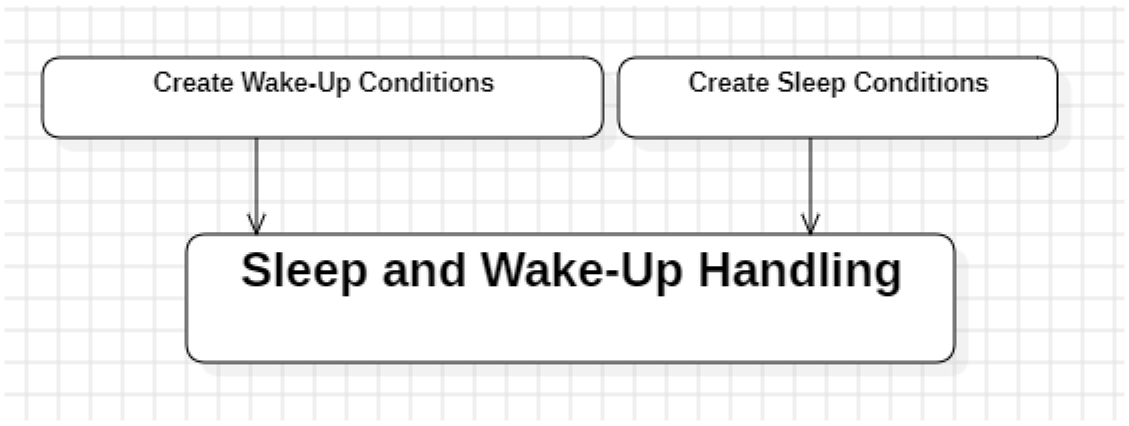


Figure 8. Sleep and Wake-Up Handling Block Diagram

- The Sleep and Wake-Up Handling Subsystem manages the system's power states to enhance energy efficiency. It transitions the system into deep sleep during inactivity and configures wake-up triggers, such as motion detection events handled by the Interrupt Handling Subsystem. This subsystem ensures that the system conserves energy without compromising responsiveness.

The System Connection subsystem interacted with the Gesture Recognition subsystem by facilitating Bluetooth communication between the ESP32 and the Dabble app. This connection allowed accelerometer data to transfer seamlessly, enabling the system to process real-time motion inputs from a mobile device for gesture recognition.

The Gesture Recognition subsystem worked closely with the Password Handling subsystem by passing detected gestures for further processing. These gestures were used to create a new password during the setup phase or validate an existing password during the unlocking process. This interaction ensured smooth and secure password management.

The Password Handling subsystem interacted with the Password Storage subsystem to manage persistent password data. When a new password was created, it was stored in non-volatile memory using the Preferences Library. During password validation, the stored password was retrieved and compared against the input gestures for verification.

In coordination with the Peripheral Control subsystem, the Password Handling subsystem issued commands to engage or disengage the lock mechanism based on the results of password validation. This ensured that only authenticated users could unlock the system, maintaining the mechanism's security.

Finally, the Interrupt Handling subsystem is integrated with the Sleep and Wake-Up Handling subsystem to manage power states effectively. Motion sensor triggers were processed as wake-up events, transitioning the system from deep sleep to an active state. Similarly, during inactivity, the system transitioned to sleep mode to conserve power, ensuring energy efficiency without compromising responsiveness.

Module	**Team Member**
System Connection	Jose Caraballo
Peripheral Control	Jose Caraballo
Gesture Recognition	Jose Caraballo
Password Handling	Jose Caraballo
Interrupt Handling	Jose Caraballo
Sleep and Wake-Up Handling	Jose Caraballo
Password Storage	Jose Caraballo

Table 1. Module Development Responsibilities

2.5 State Diagram

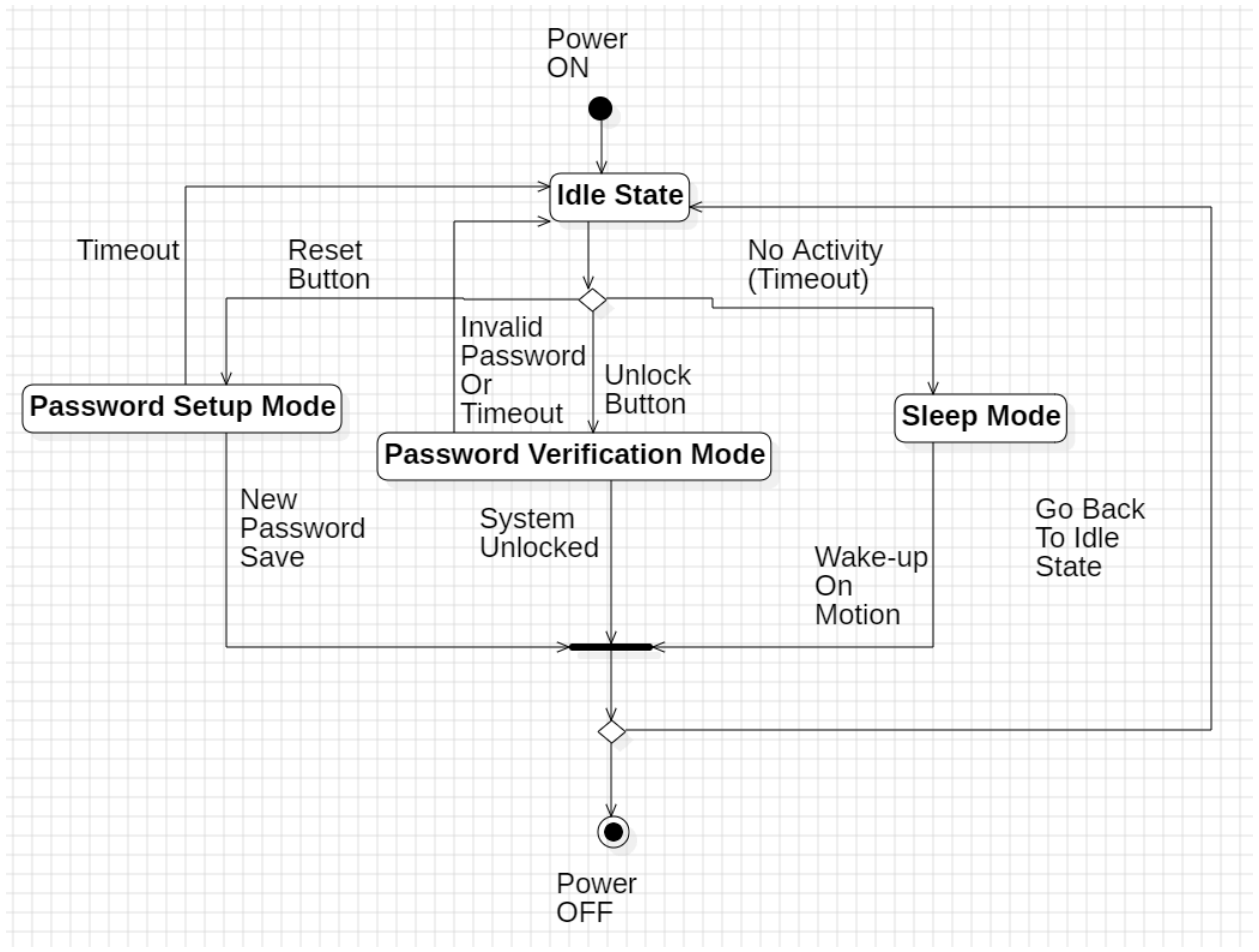


Figure 9. System State Diagram

- Power ON
  - Initial state of the system when there is power.
  - Transition:
    - Connect to Dabble state to establish Bluetooth communication.
- Connect to Dabble
  - The system attempts to connect to the Dabble app.
  - Transitions:
    - Connection Successful: Moves to Idle State.

        - Connection Unsuccessful: Moves to Waiting for Connection to Dabble State.

- Waiting for Connection to Dabble
    - The system waits and retries to connect to the Dabble app.
    - Transitions:
        - Connection Successful: Moves to Idle State.
        - Timeout/No Activity: Moves to Sleep Mode if no activity is detected.
- Idle State
    - The system is awaiting user commands.
    - Transitions:
        - Reset Button Pressed: Moves to Password Setup Mode.
        - Unlock Button Pressed: Moves to Password Verification Mode.
        - Timeout/No Activity: Moves to Sleep Mode.
- Password Setup Mode
    - Enable users to create a password or renew a password.
    - Transitions:
        - New Password Saved: Returns to Idle State.
        - Timeout: Returns to Idle State.
- Password Verification Mode
    - Enable users to verify password
    - Transitions:
        - System Unlocked: Returns to Idle State after unlocking the system.
        - Invalid Password or Timeout: Returns to Idle State.
- Sleep Mode
    - The system enters a low-power state to conserve energy
    - Transitions:
        - Wake-up on Motion: Returns to Connect to Dabble
- Power OFF
    - System is completely powered down
    - Transitions:
        - Power Supplied: Moves back to the Power ON state when power is restored.

Current State	Event	Next State	Description

Power ON	Power Supplied	Connect to Dabble	Connect to Dabble state to establish Bluetooth communication.
Connect to Dabble	Connection Successful	Idle State	Successfully connects to the Dabble app.
Connect to Dabble	Connection Unsuccessful	Waiting for Connection to Dabble	The system retries the connection to Dabble.
Waiting for Connection to Dabble	Connection Successful	Idle State	Reconnects successfully to Dabble and moves to idle.
Waiting for Connection to Dabble	No Activity (Timeout)	Sleep Mode	No activity detected; enters sleep mode.
Idle	Reset Button Pressed	Password Setup Mode	Enable users to create a password or renew a password.
Idle	Unlock Button Pressed	Password Verification Mode	User presses the unlock button to verify and unlock.
Idle	No Activity (Timeout)	Sleep Mode	No activity detected; enters sleep mode.
Password Setup Mode	New Password Saved	Idle State	User completes the password setup, saving it in the system.
Password Setup Mode	No Activity (Timeout)	Idle State	Setup times out, returning to idle.
Password Verification Mode	Invalid Password or Timeout	Idle State	Password entry fails or verification times out, returning to idle.
Password Verification Mode	Valid Password (System Unlocked)	Idle State	Password entry is successful; the system unlocks and returns to idle.
Sleep Mode	Wake-up on Motion Detected	Connect to Dabble	Returns to Connect to Dabble

Power OFF	Power Supplied	Power ON	The system powers on when power is restored.

Table 2. State Transition Table

3 System Implementation

3.1 Hardware

- **Hardware Components**
  - **Toggle Push Button x2:** These 2 buttons are used for password input and password verification. Pressing it allows the user to set a password which is then stored in the ESP32, and the second button allows the user to input a password to unlock the door. Tested
  - **12V Locking mechanism:** It will be unlocked when the password is input correct, wait 5s then lock again by itself.
  - **Relay:** this is necessary to use because the lock is 12V and the esp32 does not supply the 12V so we need to send a signal to the relay to flip the switch inside the relay to provide the 12V to lock the solenoid.
  - **Motion sensor:**This motion sensor is used to wake up the whole lock device so power consumption is minimum and adds security benefits too so the door wont unlock at a random time.
  - **12V power supply:** this is used to supply 12V to the locking solenoid.
  - **Esp32:** Is the brain of the operation which also ties all the components together to make one embedded system.

- **Circuit Design and Function**
  - **Toggle Push Button x2:** on both push buttons One side of the switch is connected to Ground, while the other side is connected to pin 32 & 33 of the ESP32. This allowed us to use pull down resistors internally.
  - **12V Locking mechanism:** The 12V Locking mechanism is connected to the relay and the ground of the 12V power supply.

- **Relay:** The relay is connected to 5V and ground of the esp as well as pin 15 of the esp32 while the other end of the relay is connected to the 12V positive terminal and the locking solenoid. This is necessary to use because the lock is 12V and the esp32 does not supply the 12V so we need to send a signal to the relay to flip the switch inside the relay to provide the 12V to lock.
- **Motion sensor:**This motion sensor is connected to 5V and ground as well as pin 2 of the esp32 to receive signal that there is motion being detected using infrared.
- **Esp32:** Ties all the components together to make one embedded system. As well is where you load the code to the system.

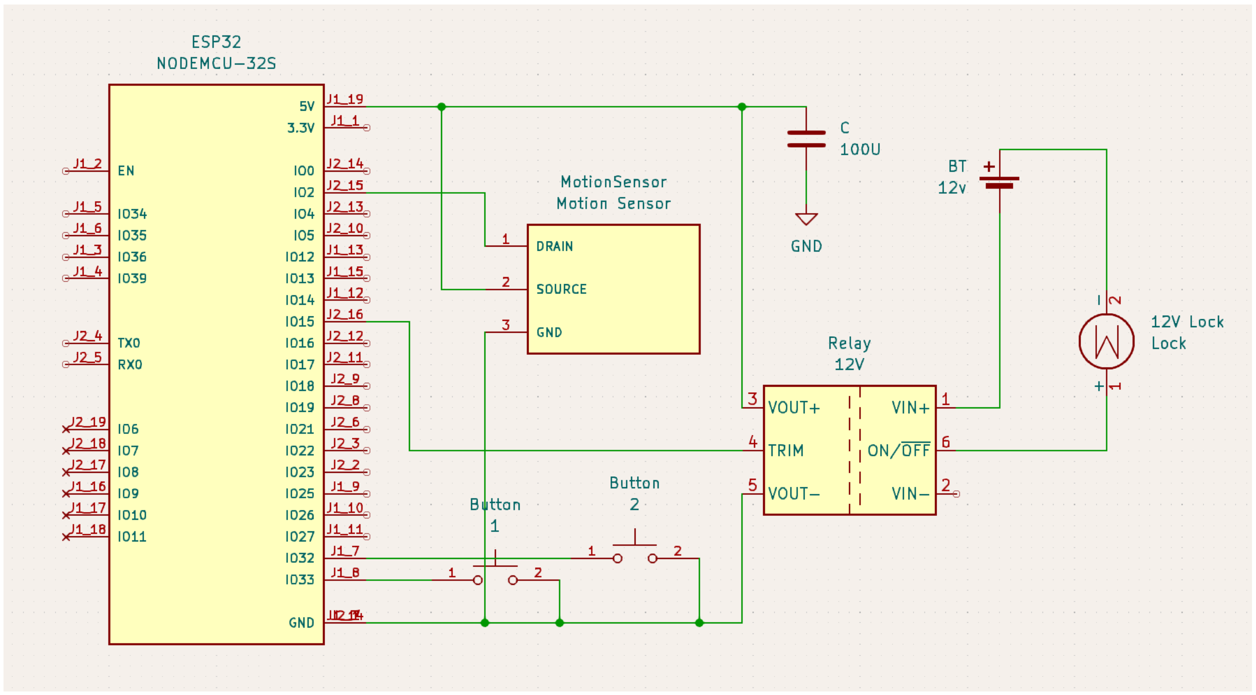


Figure 10. Describing how the components were integrated into the system.

Toggle Push Buttons - testing and calibrations where done by Adrian Suarez

- **Procedure:**
  - Connected each button to a GPIO pin on the ESP32.
  - Configured the ESP32 to display "1" on the serial monitor when a button was pressed.
  - Pressed each button multiple times to verify proper functionality.

- **Data:**
    - Out of 10 button presses, all were registered accurately on the serial monitor, indicating a 100% success rate.
    - Some debounce issues were observed during individual testing but were later fixed in main code.

Motion Sensor - testing and calibrations where done by Adrian Suarez

- **Procedure:**
    - Connected the motion sensor to the ESP32.
    - Programmed the ESP32 to display "1" on the serial monitor when motion was detected.
    - Simulated motion at varying distances and angles to check sensor responsiveness.
- **Data:**
    - The motion sensor consistently detected motion up to 2.5 meters.
    - It triggered correctly in 9 out of 10 trials (90% success rate), with 1 false negatives at the maximum range.

Relay and 12V Lock - testing and calibrations where done by Adrian Suarez

- **Procedure:**
    - Connected the relay and locking mechanism to the ESP32 and a 12V power source.
    - Programmed the ESP32 to send a signal to the relay every 10 seconds, which in turn activated the locking mechanism to unlock the door.
    - Observed and tested the lock's response.
- **Data:**
    - The relay successfully activated the lock in 10 out of 10 trials.
    - The lock unlocked within <1 second of the signal being sent and relocked automatically after 5 seconds, demonstrating consistent performance.

After Each component was tested separately I created a small system to ensure all components would work with each other as follows:

Testing Workflow - testing and calibrations where done by Adrian Suarez

- Connected all components to form an integrated system with the following sequence:
    - Pressing a button triggered the ESP32 to enable motion detection.
    - When motion was detected, the ESP32 sent a signal to the relay to unlock the lock.
    - The lock unlocked for 5 seconds and then relocked automatically.

3.2 Software

The software of the system was designed with a modular architecture model, thereby ensuring separation of responsibilities between functionalities, while allowing for free communication between them. This model was used for its advantages in maintainability and testability. The modules manage the main functionalities of the system, such as system connection, gesture recognition, password handling, interrupt handling, sleep and wake-up handling, peripheral control, and password storage. Additionally, the system relies on the ESP32 microcontroller to operate, and the Dabble mobile application for the accelerometer data of mobile devices to record hand gestures..

System Module Responsibilities:

- System Connection
    - Purpose: The System Connection module was developed to initialize and ensure successful connections between the ESP32, the serial monitor, and Dabble app.
    - Key Functions: confirmPortConnection(), confirmDabbleConnection()
- Peripheral Control
    - Purpose: The Peripheral Control module was responsible for managing the initialization and control of system peripherals such as the buttons, PIR motion sensor, and sinusoid lock.
    - Key Functions: setupPeripherals(), openLockMechanism()
- Gesture Recognition
    - Purpose: The Gesture Recognition module was responsible for the processing of accelerometer data from mobile devices and interpreting user gestures through the data record.
    - Key Functions: processAccelData(), filterAccelXAxisData(), filterAccelYAxisData(), filterAccelZAxisData(), detectAccelGesture(), clearGestureBuffer()

- Password Handling
    - Purpose: The Password Handling module manages password creation, validation, and secure storage.
    - Key Functions: setupPassword(), updatePinCode(), verifyPassword()
- Interrupt Handling
    - Purpose: The Password Handling module manages button presses and sensor-based interrupts.
    - Key Functions: setupInterrupt(), handleButtonPress(), handleResetButtonPress(), handleMotionInterrupt()
- Sleep and Wake-up Handling
    - Purpose: The Sleep and Wake-up Handling module manages system sleep mode.
    - Key Functions: sleepMode(), initializeSecuritySystem()
- Password Storage
    - Purpose: The Password Storage module manages persistent storage for the user-defined password.
    - Key Functions: createSecurityConnection(), updatePinCode()

Software Interfaces:

- Dabble Mobile Application
    - Purpose: Transmit accelerometer data from a mobile device to the ESP32.
    - Interface: External mobile application
- ESP32 Deep Sleep API
    - Purpose: Manage low-power modes for the system.
    - Interface: Uses esp_sleep_enable_ext0_wakeup() and esp_deep_sleep_start() for sleep management.

Hardware Interfaces:

- Sinusoid Lock
    - GPIO Pins: Pin 15
    - Purpose: Engage and disengage lock.
    - Interface: Digital output that toggles active high and active low as a result of user actions.

- Passive Infrared (PIR) Motion Sensor
    - GPIO Pins: Pin 2
    - Purpose: Wake up the system from deep sleep
    - Interface: Digital input with an interrupt trigger on a rising edge.
- Push Buttons
    - GPIO Pins: Pin 32 and Pin 33
    - Purpose: Activate password verification and password setup modes.
    - Interface: Digital input using the internal pull-up configuration.

System Libraries:

- ESP32 Servo Library (Version 3.0.5):
    - Enables the use of servo motors with the ESP32 microcontroller. This library was utilized to operate the sinusoid lock.
- DabbleESP32 Library (Version 1.5.1):
    - Handles Bluetooth communication with the Dabble mobile application, effectively establishing a connection with the ESP32 and the sensors of a mobile device.
- Preferences Library:
    - Provides a portion of the on-board non-volatile memory of the ESP32 microcontroller to store the password data of users.

3.3 User Interface

The user interface of the system was very basic and consisted of the serial monitor of the Arduino IDE which required the use of a computer, the Dabble mobile application which required a mobile device, and two push buttons on a breadboard.

With the Arduino IDE, information was provided to the user on the peripheral components of the device were initializations, as well as functionalities of the system such as: when the user was required to connect the Dabble mobile app, when the connection to Dabble was established, when sleep mode was activated, and the gestures that were recorded when the password creation and password verification modes were utilized.

For the Dabble app, the user was required to use the sensor reading functions, establish the bluetooth connection, and move the mobile device in linear motions, such as up, down, left, right, forward, and backward. As for the physical push buttons, the user had to click either one if they desired to use the password creation mode or the password verification mode.

Additionally, the user was given the ability to configure the system through the use of the password creation function of the system. The function allowed the user to create their own unique six pin password based on a unique sequence of hand gestures. Alongside the password creation, the user is allowed to reset the current pin, and replace it with another unique pin of their choice.

After a user has created their unique password, they may utilize the password verification button to enter in their six digit pin number to unlock the system. Once the system is unlocked it will remain in that state for a predetermined period of time, before re-locking. Any additional configurations to the system will require access to the systems software to make adjustments to functions that have been hardcoded.

3.4 Data Communications

The system of the project has different forms of communication with the main form of communication being facilitated by the Dabble mobile app, without this application no accelerometer data can be collected and processed, thereby acting as the critical communication infrastructure of the system. Additional methods of communication with subsystems occur with the push buttons to communicate the user's necessity to create or use a password, or the motion detector to alert the microcontroller not to go into a sleep mode.

Data that is processed throughout the times of operation for the system are recorded and stored within a portion of the on-board non-volatile memory of the ESP32 microcontroller using the Preferences library. With this library data can be stored permanently, and accessed by users whenever needed.

Additionally, the system does not rely on internet connectivity, but does rely on bluetooth communication to gather accelerometer data from the Dabble app.

4 Testing and Performance Evaluation

For the project, there were three steps utilized to examine the system. With the first examination step, the master pin was utilized to ensure that the password verification mode, or system unlock, worked at all times. This method of testing provided insight to determine if the accelerometer data was recorded, gesture data was converted into strings, observe how the threshold was affecting the gestures, if there was overflow in the gestures recorded, if the system stopped prematurely, the functionality of the lock, and it allowed us to confirm if any observable bugs, or errors, were in the system.

The second procedure was to examine the functionality of the password creation mode, this examination was fairly simple as the test objective was to ensure that the system can create a password, if no password was stored, and if one was stored, can the system reset the password by removing the current pin and replacing it with a new pin. Additionally, like the first testing procedure, it also allowed us to observe a majority of the system's functionality with the expectation of the locking system.

With the final test procedure, the object was to examine if the sleep mode of the system worked as intended, to do this the system was required to run without any activity for about a minute. Once the system went into sleep mode, the interrupt service routine added to the PIR sensor was utilized to wake the system up again.

Of the three methods provided, the most important examination procedures utilized to ensure that the system met all requirements needed were both the first testing procedure and the second test procedure, as these procedures examined the essence of what the system was designed to be.

The performance of the Magic Wand System was evaluated using three key testing procedures that directly assessed the system's core functionality. Each procedure focused on specific subsystems, allowing for a comprehensive evaluation of the system's reliability, accuracy, and responsiveness. The primary performance metrics used to assess the system were gesture recognition accuracy, password creation and verification success rate, system sleep mode functionality, and wake-up responsiveness.

Performance Examination Metrics:

- Gesture Recognition Accuracy
    - Objective: Evaluate the system's ability to accurately recognize user gestures for password creation and verification.
    - Evaluation Method:
        - Number of correctly identified gestures relative to total gesture attempts.
- Password Verification Functionality
    - Objective: Ensure the system can successfully verify user passwords and trigger the unlock mechanism.
    - Evaluation Method:
        - Percentage of successful password verifications relative to total attempts.
- Password Creation Functionality
    - Objective: Ensure the system can create and store a new password when no password exists or update an existing password when a reset occurs.
    - Evaluation Method:
        - Percentage of successful password creations relative to total attempts.
- Sleep Mode Functionality
    - Objective: Ensure the system enters sleep mode after a period of inactivity to conserve energy.
    - Evaluation Method:
        - Percentage of successful transitions to sleep mode relative to total attempts
- Wake-Up Functionality
    - Objective: Determine how accurately the system wakes up from sleep mode in response to motion detected by the PIR sensor.
    - Evaluation Method:
        - Percentage of successful wake-ups relative to total attempts.

4.1 Performance Assessment

Overall, the system's performance was considered mostly successful, as it achieved the goal of creating and verifying passwords to unlock the system. However, there were several issues affecting its performance, including sensitivity problems with hand gesture reading, difficulties returning to idle mode after using password setup or verification modes, and errors in clearing recorded gesture data after timeouts.

One observable issue involved hand gesture reading. Gestures often recorded two inputs each time a single gesture was made or an incorrect gesture was recorded. A threshold and debounce solution was implemented, but these measures did not work as intended. To manage this issue, the team discovered a workaround by having users rotate the phone rather than moving it in the intended directions. When the workaround was utilized, the gesture readings were extremely accurate and reliable, unlike the previously mentioned issues with the intended gesture method.

Another issue impacted the system's ability to return to idle mode after deactivating either password setup mode or password verification mode. Due to this, the system became stuck, repeatedly reverting to the password setup mode, forcing users to either shut down the system or press the password verification button to temporarily exit and then return later.

The final major issue concerned the password buffer. When gestures were recorded during password setup or verification, interruptions such as button presses or timeouts caused the system to reuse previously recorded data, merging it with newly recorded data. Although a clear solution existed, time constraints and limited experience prevented the issue from being resolved.

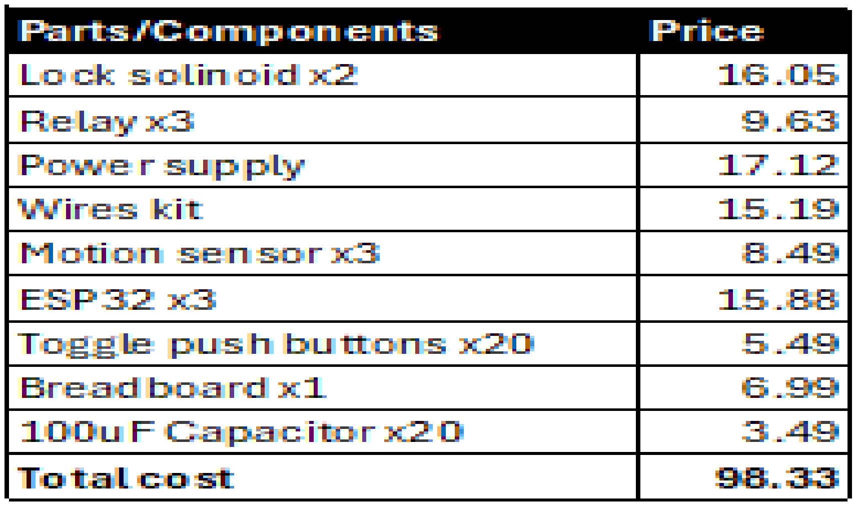
In summary, while the system demonstrated successful password creation and verification, these technical difficulties related to gesture input accuracy, mode transitions, and data management significantly affected its overall reliability and user experience.

Performance Examination	Attempts	Successes	Success Rate Percentages (%)	Target Success Rate Percentage (%)	Status
Gesture Recognition Accuracy	60	47	78.3	80	Failure
Password Verification Functionality	30	26	86.6	80	Success

Password Creation Functionality	30	21	70	80	Failure
Sleep Mode Functionality	10	10	100	90	Success
Wake-Up Functionality	10	10	100	90	Success

Table 3. Performance Metrics

5 Budget



Parts/Components	Price
Lock solinoid x2	16.05
Relay x3	9.63
Power supply	17.12
Wires kit	15.19
Motion sensor x3	8.49
ESP32 x3	15.88
Toggle push buttons x20	5.49
Breadboard x1	6.99
100uF Capacitor x20	3.49
Total cost	**98.33**

Table 4. Project cost of parts.

6 Project Management and Workload Distribution

The team was well-organized to ensure clarity and efficiency in completing the project, with each member assigned specific roles and responsibilities to avoid confusion. **Jose Caraballo** took charge of all software-related tasks, focusing primarily on the code development for the gesture-based system. His work involved figuring out how to convert hand movements detected by the accelerometer into numerical values ranging from 1 to 6 for password input. He was also responsible for implementing the sleep mode functionality, ensuring the system entered a low-power state when no motion was detected, enhancing energy efficiency. In essence, Jose handled the entirety of the software development, including logic, algorithms, and system responsiveness.

On the other hand, **Adrian Suarez** managed all hardware-related components. His responsibilities included sourcing the necessary parts, such as the locking mechanism, relay, motion sensor, and ESP32 microcontroller, and ensuring they were correctly integrated into the system. Adrian tested each component individually to verify functionality, writing specific test code for each part to confirm proper connections and performance. He also worked on combining all components into a cohesive system, troubleshooting hardware-related issues, and ensuring that the entire setup worked seamlessly with the software.

This clear division of responsibilities allowed the team to collaborate effectively, leveraging each member's strengths to achieve a fully functional gesture-based locking system.

Team Member	Primary Job Task
Adrian Suarez	System Hardware & Testing
Jose Caraballo	System Software

Table 5. Job Responsibilities

Name	Effort
Jose Caraballo	60%
Adrian Suarez	40%
Total	100%

Table 6. Job Workload

7 References

[1]Subhajit, “ESP32 Save Data Permanently using Preferences Library,” IotCircuitHub, Jan. 26, 2024. https://iotcircuithub.com/esp32-preferences-library-tutorial/ (accessed Dec. 08, 2024).

[2] STEMpedia, “GitHub - STEMpedia/DabbleESP32: DabbleESP32 is library for using Dabble app with ESP32. It uses BLE of ESP32 for communicating with the app. The app consists virtual modules that help in accessing certain features of smartphone via hardware and helps in providing hardware control via smartphone.,” GitHub, Sep. 21, 2019. https://github.com/STEMpedia/DabbleESP32/tree/master (accessed Dec. 08, 2024).

[3] “Sleep Modes — ESP-IDF Programming Guide v4.1 documentation,” Readthedocs-hosted.com, 2016. https://espressif-docs.readthedocs-hosted.com/projects/esp-idf/en/stable/api-reference/system/sleep_modes.html (accessed Dec. 08, 2024).

[4] Arduino.cc, 2024. https://docs.arduino.cc/built-in-examples/digital/Debounce/

[5] Arduino.cc, 2024.
https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/

[6] “How to Use Arduino Interrupts The Easy Way,” www.youtube.com.
https://www.youtube.com/watch?v=SXZkX3cJqDs (accessed Jul. 10, 2023).

[7]“c_str() - Arduino Reference,” Arduino.cc, 2024.
https://www.arduino.cc/reference/tr/language/variables/data-types/string/functions/c_str/
(accessed Dec. 08, 2024).
[8]Arduino.cc, 2024.
https://docs.arduino.cc/language-reference/en/variables/constants/inputOutputPullup/

[9]Arduino.cc, 2024. https://docs.arduino.cc/language-reference/#variables

[10]Arduino.cc, 2024.
https://docs.arduino.cc/language-reference/en/variables/variable-scope-qualifiers/volatile/
(accessed Dec. 08, 2024). https://forum.arduino.cc/t/using-floats/185897

[11]“Tutorial on the basics of using the ‘buffer’ instruction?,” Arduino Forum, Mar. 31, 2011.
https://forum.arduino.cc/t/tutorial-on-the-basics-of-using-the-buffer-instruction/57068/8
(accessed Dec. 08, 2024).

[12]Programming Electronics Academy, “The ESP32 Wake-Up Call: Exploring External Triggers in Deep Sleep Mode (ext1),” YouTube, Sep. 15, 2023.
https://www.youtube.com/watch?v=0KxI2aArGlo (accessed Dec. 08, 2024).

[13]“ChatGPT,” *Chatgpt.com*, 2024. https://chatgpt.com/?model=gpt-4

[13]A. Team, “Unlock your door with a simple hand gesture | Arduino Blog,” Arduino Blog, Sep. 05, 2016. https://blog.arduino.cc/2016/09/05/unlock-your-door-with-a-simple-hand-gesture/
(accessed Dec. 08, 2024).

[14]Arduino Forum, “saving keypad password to internal EEPROM,” Arduino Forum, May 21, 2014. https://forum.arduino.cc/t/saving-keypad-password-to-internal-eeprom/235119 (accessed Dec. 08, 2024).