

MailTonBox

Alessandro Ferrante

*Una Soluzione IoT Intelligente
per l'Integrazione Digitale
delle Cassette Postali*

Marzo 2025

Sommario

Questo documento presenta la progettazione e l'implementazione di un sistema IoT innovativo, denominato MailTonBox, sviluppato per monitorare in tempo reale l'arrivo della posta all'interno di una cassetta postale. Il sistema è composto da due tipologie di unità basate su microcontrollori ESP32: il MailTon, responsabile dell'elaborazione dei dati, delle comunicazioni Internet e della gestione centralizzata, e una o più unità CtrlMailBox, ciascuna incaricata del monitoraggio fisico di una singola cassetta postale. Il sistema è stato progettato per supportare la gestione multi-dispositivo, consentendo l'associazione e la comunicazione simultanea con più unità CtrlMailBox. La trasmissione dei dati tra dispositivi avviene tramite il protocollo LoRa, scelto per le sue caratteristiche di affidabilità, basso consumo e lunga portata. L'architettura include inoltre un'integrazione con la piattaforma Telegram per la notifica istantanea all'utente degli eventi rilevati, nonché un modello di intelligenza artificiale ottimizzato per dispositivi embedded, finalizzato al monitoraggio ambientale interno. I risultati ottenuti dimostrano l'efficacia dell'approccio adottato, offrendo una soluzione pratica, scalabile e replicabile per migliorare l'interazione con infrastrutture di uso quotidiano come le cassette postali.



Figura 1: CtrlMailBox, MailTon, TonyBot



Figura 2: MailTon



Figura 3: CtrlMailBox

Indice

1 Introduzione	4
1.1 Contesto e motivazione	4
1.2 Obiettivi del progetto	4
2 Architettura del Sistema	4
2.1 Panoramica generale	4
2.2 Componenti Hardware	5
2.2.1 MailTon	5
2.2.2 CtrlMailBox	5
2.3 Protocolli di comunicazione	6
3 Logica di Funzionamento	6
3.1 Rilevamento della posta	6
3.2 Monitoraggio ambientale e predizioni AI	6
3.3 Gestione degli errori	7
4 Algoritmo di Rilevamento Posta	8
4.1 Calibrazione Iniziale	8
4.2 Rilevamento degli Eventi	8
4.3 Miglioramento dell’Affidabilità	8
5 Gestione degli Stati del Sistema	9
5.1 Stati Principali	9
5.2 Transizioni di Stato	9
5.3 Gestione delle Notifiche	9
6 Implementazioni Software	10
6.1 MailTon	10
6.1.1 Configurazione	10
6.1.2 Gestione multi-dispositivo	12
6.1.3 Comunicazione LoRa	14
6.1.4 Integrazione con Telegram	17
6.1.5 Implementazione del modello AI	18
6.2 CtrlMailBox	18
6.2.1 Configurazione	19
6.2.2 Comunicazione LoRa	20
6.2.3 Misurazione della Distanza	21
6.2.4 Rilevamento della Posta	21
6.2.5 Interruzione del rilevamento	22
6.2.6 Controllo del Servomotore	22
6.3 Progressive Web App	22
7 Protocollo di Comunicazione LoRa	23
7.1 Struttura dei Pacchetti	23
7.2 Meccanismo di Conferma	23

8 Creazione e Addestramento Modello Multi-task	24
8.1 Preparazione dei Dati	24
8.2 Scalatura dei Dati	24
8.3 Bilanciamento delle Classi con SMOTE	25
8.4 Architettura del Modello Multi-task	25
8.5 Addestramento del Modello	27
8.6 Salvataggio del Modello	27
8.7 Valutazione	28
8.7.1 Valutazione del Modello	28
8.7.2 Valutazione delle Prestazioni di Regressione	29
9 Sicurezza e Affidabilità	30
9.1 Sicurezza della Comunicazione	30
9.2 Integrità dei Dati	30
9.3 Ridondanza e Robustezza	30
9.4 Gestione degli Interrupt	30
9.5 Affidabilità del Rilevamento	30
10 Test e Valutazione	31
10.1 Metodologia di test	31
10.2 Risultati dei test	31
11 Sviluppi Futuri	31
12 Conclusioni	32
13 Riferimenti	33

1 Introduzione

1.1 Contesto e motivazione

Il sistema “MailTonBox” rappresenta una soluzione innovativa per il monitoraggio intelligente delle cassette postali tradizionali. In un’epoca in cui la digitalizzazione pervade ogni aspetto della vita quotidiana, questo progetto risponde all’esigenza di trasformare un oggetto analogico in un dispositivo connesso e intelligente, capace di notificare l’utente in tempo reale circa l’arrivo della posta. Il sistema si propone di risolvere un problema comune: controllare se è arrivata corrispondenza senza doversi recare fisicamente alla cassetta postale. Questo è particolarmente utile per chi vive in contesti dove la cassetta postale è distante dall’abitazione, o semplicemente per chi desidera ottimizzare i propri spostamenti.

1.2 Obiettivi del progetto

L’obiettivo principale del progetto è sviluppare un sistema affidabile, a basso consumo energetico e di facile installazione, che possa essere integrato in qualsiasi cassetta postale esistente senza richiedere modifiche strutturali significative. Il sistema “MailTonBox” si pone i seguenti obiettivi:

- Rilevare in modo affidabile l’arrivo di nuova posta nella cassetta
- Notificare tempestivamente l’utente attraverso messaggi su Telegram
- Monitorare le condizioni ambientali circostanti tramite sensori dedicati
- Implementare un modello di intelligenza artificiale per analizzare e prevedere parametri ambientali
- Garantire un’autonomia energetica adeguata tramite componenti a basso consumo
- Offrire un’interfaccia utente semplice e intuitiva per la configurazione del sistema

2 Architettura del Sistema

2.1 Panoramica generale

Il sistema “MailTonBox” è stato progettato secondo un’architettura distribuita composta da due unità principali che collaborano tra loro:

1. **MailTon**: unità centrale connessa a Internet tramite WiFi, che riceve i messaggi dal CtrlMailBox attraverso LoRa, responsabile dell’elaborazione dati, della comunicazione Internet e dell’invio di notifiche tramite il bot Telegram TonyBot.
2. **CtrlMailBox**: unità di controllo secondaria installata direttamente nella cassetta postale, responsabile del rilevamento fisico della posta attraverso un sensore ad ultrasuoni e dotata di un meccanismo di apertura servo-assistito. Comunica gli eventi rilevati al MailTon attraverso una rete LoRa.

Questa architettura a due livelli consente di separare le responsabilità: il CtrlMailBox opera in un ambiente con risorse limitate e potenzialmente senza accesso diretto a Internet, mentre il MailTon agisce come gateway verso servizi cloud e applicazioni di messaggistica. La comunicazione tra i due moduli avviene tramite il protocollo LoRa, scelto per la sua capacità di trasmettere dati su lunghe distanze con un consumo energetico ridotto, caratteristica fondamentale per dispositivi alimentati a batteria. La figura seguente illustra l'architettura generale del sistema:



2.2 Componenti Hardware

Il sistema utilizza i seguenti componenti hardware:

2.2.1 MailTon

- **Microcontrollore:** ESP32, per la gestione della comunicazione e l'elaborazione dei dati.
- **Modulo LoRa:** per ricevere i messaggi dal CtrlMailBox.
- **Modulo WiFi:** integrato nell'ESP32, per la connessione Internet e l'invio di notifiche.
- **Sensore DHT11:** per la rilevazione di temperatura e umidità.
- **Display OLED:** per il feedback visivo.
- **LEDs:** per le indicazioni di stato.

2.2.2 CtrlMailBox

- **Microcontrollore:** ESP32, selezionato per le sue capacità di elaborazione, basso consumo energetico e supporto nativo per vari protocolli di comunicazione.
- **Sensore ad ultrasuoni HC-SR04:** utilizzato per misurare la distanza interna nella cassetta postale e rilevare variazioni causate dall'inserimento della posta.
- **Modulo LoRa:** per la comunicazione a lungo raggio con il MailTon.
- **Display:** per la visualizzazione di informazioni di diagnostica e stato.
- **Encoder rotativo:** per rilevare l'apertura e chiusura della cassetta postale.

- **Servomotore:** impiegato per automatizzare l'apertura della cassetta (meccanismo di chiusura).
- **LED di stato:** per fornire feedback visivi sullo stato del sistema.
- **Pulsanti:** utilizzati come alternativa per l'apertura e la chiusura della cassetta postale, oltre che per l'interazione e la diagnostica del sistema.

2.3 Protocolli di comunicazione

Il sistema implementa diversi protocolli di comunicazione per garantire un'efficiente trasmissione dei dati:

1. **LoRa (Long Range):** utilizzato per la comunicazione tra MailTon e CtrlMailBox, scelto per la sua capacità di trasmettere dati a lungo raggio con un consumo energetico ridotto.
2. **Wi-Fi:** utilizzato da MailTon per connettersi alla rete Internet e comunicare con il server Telegram.
3. **HTTP:** utilizzato per la configurazione in AP mode del WiFi e per la comunicazione sicura con le API di Telegram.

3 Logica di Funzionamento

3.1 Rilevamento della posta

Il rilevamento dell'arrivo della posta si basa sul principio di misurare la distanza all'interno della cassetta attraverso un sensore ad ultrasuoni. Il sistema implementa questa logica nei seguenti passaggi:

1. Durante la configurazione iniziale, il dispositivo CtrlMailBox registra una “distanza di riferimento” (baseline) che rappresenta la condizione di cassetta vuota.
2. Il sensore esegue misurazioni periodiche, confrontando la distanza attuale con quella di riferimento.
3. Se viene rilevata una variazione significativa (oltre una soglia predefinita), il sistema interpreta questo come l'arrivo di nuova posta.
4. Il dispositivo CtrlMailBox invia un messaggio “New Mail” al MailTon tramite LoRa.
5. Il MailTon, ricevuto il messaggio, invia una notifica all'utente tramite il bot Telegram.

3.2 Monitoraggio ambientale e predizioni AI

Oltre al rilevamento della posta, il sistema implementa un monitoraggio ambientale avanzato:

1. Il sensore DHT11 misura temperatura e umidità periodicamente.

2. I dati raccolti vengono utilizzati come input per il modello di intelligenza artificiale.
3. Il modello produce tre output principali:
 - Predizione del valore PPM del monossido di carbonio (CO)
 - Classificazione delle condizioni di temperatura e umidità (9 classi)
 - Classificazione della qualità dell'aria (4 classi)
4. Se vengono rilevate condizioni critiche ($\text{PPM} > \#$, classe $\text{PPM} \geq \#$, o classe temperatura/umidità $\geq \#$), il sistema invia automaticamente un messaggio di allerta all'utente.

3.3 Gestione degli errori

Il sistema implementa vari meccanismi per la gestione degli errori:

1. Controllo dell'integrità dei messaggi LoRa tramite checksum:

```

1 if (incomingLength != incoming.length() || receivedChecksum != 
2   calculatedChecksum){
3   loraFlagError = true;
4   digitalWrite(LED_RED, HIGH);
5 }
```

2. **Sistema di acknowledgment** per confermare la ricezione dei messaggi:

```

1 if(loraFlagReceived){
2   loraFlagReceived = false;
3   sendMessageLoRa("ACK");
4   handlerSendMessage();
5 }
```

3. **Indicatori visivi** tramite LED:

```

1 if (WiFi.status() == WL_CONNECTED) {
2   digitalWrite(LED_RED, LOW);
3   digitalWrite(LED_GREEN, HIGH);
4 } else {
5   digitalWrite(LED_RED, HIGH);
6   digitalWrite(LED_GREEN, LOW);
7 }
```

4. **Riconnessione automatica** in caso di perdita della connessione Wi-Fi:

```

1 if (firstConnection && WiFi.status() != WL_CONNECTED && WiFi.
2   getMode() == WIFI_AP_STA) {
3   WiFi.begin(ssid, password);
```

4 Algoritmo di Rilevamento Posta

Il sistema utilizza un algoritmo basato sulla distanza per rilevare l'arrivo della posta. L'approccio si basa sul principio che l'inserimento di lettere o pacchi nella cassetta postale modifica la distanza misurata dal sensore ad ultrasuoni.

4.1 Calibrazione Iniziale

Durante l'inizializzazione, il sistema effettua una calibrazione per stabilire la distanza di riferimento (baseline) che rappresenta lo stato di “cassetta vuota”:

```
1 initial_distance = measuresAverageDistance();
```

Questa calibrazione viene periodicamente aggiornata per compensare eventuali cambiamenti ambientali o del posizionamento del sensore:

```
1 if (!lora_priority && !wait_rotary && !wait_servo){  
2     distance = measuresAverageDistance();  
3     readings_made++;  
4     if (readings_made >= distance_update) {  
5         initial_distance = measuresAverageDistance();  
6         readings_made = 0;  
7     }  
8 }
```

4.2 Rilevamento degli Eventi

Il sistema confronta continuamente la distanza corrente con la distanza di riferimento. Se la differenza supera una soglia predefinita (`threshold`), viene rilevato l'arrivo della posta:

```
1 if (!lora_priority && !mailbox_open && !wait_servo && !wait_rotary  
    && !mail_detected && abs(distance-initial_distance) > threshold){  
2     // Posta rilevata  
3     lora_msg = "New Mail";  
4     sendMessageLoRa();  
5     mail_detected = true;  
6 } else if (mail_detected && abs(distance-initial_distance) <=  
    threshold){  
7     mail_detected = false; // reset when the letter is removed  
8 }
```

4.3 Miglioramento dell’Affidabilità

Per migliorare l'affidabilità del rilevamento e ridurre i falsi positivi, il sistema:

1. Utilizza la media di multiple letture (25) per determinare la distanza
2. Implementa un meccanismo di state machine (macchina a stati) per evitare notifiche ripetute dello stesso evento
3. Considera lo stato di apertura della cassetta per evitare false rilevazioni durante l'accesso manuale

5 Gestione degli Stati del Sistema

Il sistema implementa una macchina a stati implicita che è strutturata in modo tale da garantire un monitoraggio coerente e reattivo delle condizioni operative della cassetta postale. Ogni stato riflette una particolare situazione fisica o interazione dell’utente con il dispositivo, e le transizioni tra stati sono attivate da condizioni ambientali misurate (come la distanza) o da eventi hardware (come l’apertura della cassetta rilevata tramite encoder rotativo).

Questa gestione permette al sistema di reagire in maniera intelligente a situazioni ricorrenti, come l’arrivo della posta o l’accesso alla cassetta da parte dell’utente, e di comunicare tali eventi all’esterno in modo puntuale ed efficiente. Di seguito vengono descritti gli stati principali riconosciuti dal sistema, le condizioni che regolano le transizioni e la strategia adottata per l’invio delle notifiche.

5.1 Stati Principali

1. **Cassetta Vuota:** stato iniziale, la distanza misurata corrisponde alla distanza di riferimento
2. **Posta Presente:** rilevata quando la distanza misurata differisce significativamente dalla distanza di riferimento
3. **Cassetta Aperta:** rilevato attraverso l’encoder rotativo, indica che un utente sta accedendo alla cassetta
4. **Posta Rimossa:** rilevato quando, dopo un evento di “Posta Presente”, la distanza torna a essere simile alla distanza di riferimento

5.2 Transizioni di Stato

Le transizioni tra gli stati sono gestite da varie condizioni nel loop principale:

- La transizione da “Cassetta Vuota” a “Posta Presente” avviene quando `abs(distance - initial_distance) > threshold`
- La transizione da “Posta Presente” a “Posta Rimossa” avviene quando `abs(distance - initial_distance) <= threshold`
- Le transizioni verso e da “Cassetta Aperta” sono gestite dall’interrupt dell’encoder rotativo

5.3 Gestione delle Notifiche

Il sistema è progettato per inviare notifiche solo quando avviene una transizione significativa di stato:

- Viene inviata una notifica “New Mail” quando si passa a “Posta Presente”
- Viene inviata una notifica “Mailbox Opened” quando la cassetta viene aperta
- Il sistema evita di inviare notifiche ripetute per lo stesso evento, implementando flag come `mail_detected` e `message_sent`

6 Implementazioni Software

6.1 MailTon

L'implementazione software del dispositivo MailTon è organizzata secondo una struttura modulare che gestisce diverse funzionalità:

1. **Configurazione:** gestione delle credenziali e della connessione alla rete.
2. **Gestione multi-dispositivo:** gestione simultanea di più unità CtrlMailBox.
3. **Comunicazione LoRa:** invio e ricezione di messaggi con i dispositivi CtrlMailBox
4. **Integrazione con Telegram:** gestione del bot per l'invio di notifiche
5. **Monitoraggio ambientale:** lettura dei sensori e predizione con AI
6. **Interfaccia utente:** gestione del display OLED e dei LED di stato

6.1.1 Configurazione

Il sistema implementa un sistema di configurazione Wi-Fi flessibile che permette all'utente di impostare le credenziali della rete in due modalità:

1. **Configurazione tramite portale web:** Il dispositivo crea un access point al quale l'utente può connettersi per configurare le credenziali Wi-Fi, configurare l'autenticazione per il bot Telegram e inserire le informazioni relative al CtrlMailBox tramite un'interfaccia web.

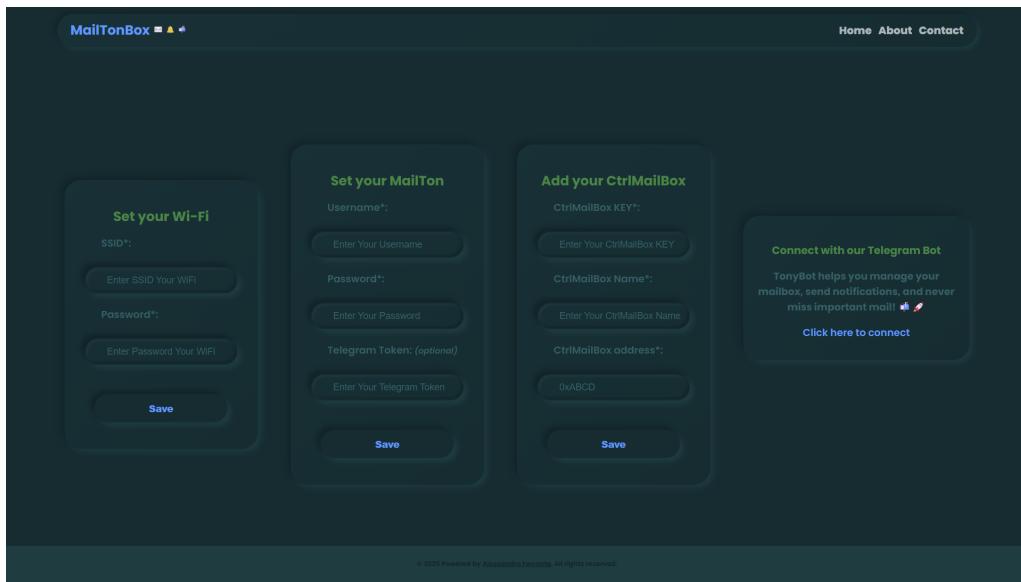


Figura 4: Interfaccia web MailTon

2. **Gestione interfaccia:** gestione del salvataggio dei parametri inviati dall’interfaccia web.

```

1 void handleSave(AsyncWebServerRequest *request) {
2     if(request->hasParam("ssid", true) && request->hasParam("password", true)) {
3         ssid = request->getParam("ssid", true)->value();
4         password = request->getParam("password", true)->value();
5         // use preferences to save
6         if (saveWiFiCredentials(ssid, password)) {
7             ...
8         } else {...}
9         configureWiFi(WIFI_AP_STA);
10    } else if(request->hasParam("username", true) &&
11        ...
12    } else if(request->hasParam("ctrlmailboxkey", true) &&
13        ...
14    } else {
15        request->send(500, "text", "document.getElementById('errorCredentials').style.display='block';");
16    }
17 }
```

3. **Configurazione WiFi:** Per garantire flessibilità nella configurazione della rete, il sistema implementa la funzione configureWiFi, che consente di attivare dinamicamente due modalità operative: Access Point (AP) e Access Point + Station (AP+STA). Questa funzione è cruciale nella fase di configurazione e connessione del dispositivo alla rete Wi-Fi dell’utente.

In modalità AP, il dispositivo crea una rete locale protetta da password, alla quale l’utente può connettersi per accedere a un’interfaccia di configurazione. Viene utilizzato un server DNS per reindirizzare automaticamente tutte le richieste verso l’indirizzo IP del dispositivo, permettendo così un accesso immediato al portale di configurazione tramite browser, senza la necessità di conoscere l’indirizzo IP esatto. Inoltre, il server HTTP gestisce specifici percorsi per la configurazione, il salvataggio dei parametri, ottimizzando l’esperienza utente anche per l’uso su dispositivi mobili.

La modalità AP+STA combina la possibilità di continuare ad offrire il portale di configurazione locale mentre si tenta la connessione alla rete Wi-Fi precedentemente configurata. Durante questo processo, il sistema mostra all’utente lo stato della connessione tramite display OLED. Se la connessione alla rete non va a buon fine entro un tempo limite prestabilito (5 secondi), il dispositivo ripiega automaticamente sulla modalità AP, evitando che l’utente perda l’accesso al dispositivo e potendo così correggere i parametri inseriti.

Questo meccanismo intelligente di gestione delle modalità di rete rende il sistema MailTonBox estremamente accessibile anche in ambienti domestici non tecnici, migliorando l’affidabilità e la resilienza dell’infrastruttura IoT proposta.

```

1 void configureWiFi(wifi_mode_t mode) {
2     switch (mode) {
3         case WIFI_AP:
4             WiFi.softAPConfig(local_IP, gateway, subnet);
5             WiFi.softAP(ssidAP, passwordAP);
6             dnsServer.start(DNS_PORT, "*", WiFi.softAPIP());
7             server.on("/", HTTP_GET, handleRoot);
8             // ...
9             break;
10        case WIFI_AP_STA:
11            WiFi.disconnect(true);
12            WiFi.mode(WIFI_AP_STA);
13            WiFi.softAPConfig(local_IP, gateway, subnet);
14            WiFi.softAP(ssidAP, passwordAP);
15            dnsServer.start(DNS_PORT, "*", WiFi.softAPIP());
16            // ...
17            break;
18    }
19 }

```

Listing 1: Funzione configureWiFi in modalità AP e AP+STA

4. **Configurazione tramite bot Telegram:** dopo la prima configurazione in locale, l'utente può aggiornare le impostazioni di rete semplicemente inviando un messaggio al bot, senza necessità di accesso fisico al dispositivo.

```

1 if (text == "/configurewifi") {
2     bot.sendMessage(chat_id, " Send me the new SSID of your WiFi
3     ", "");
4     botConfigureWiFi = true;
5     newWIFIbyBot = true;
6     return;
7 }

```

Le credenziali vengono salvate in memoria non volatile, così come gli altri parametri, attraverso la libreria Preferences, garantendo la persistenza dei dati anche dopo un riavvio del dispositivo:

```

1 bool saveCredentials(const String& newSSID, const String&
2     newPassword) {
3     Preferences preferences;
4     preferences.begin("wifi", false);
5     preferences.putString("ssid", newSSID);
6     preferences.putString("password", newPassword);
7     preferences.end();
8     return true;
9 }

```

6.1.2 Gestione multi-dispositivo

Per abilitare la registrazione e la gestione simultanea di più dispositivi CtrlMailBox, è stato implementato un meccanismo basato su array paralleli statici: CTRLMAILBOX_KEYS, CTRLMAILBOX_NAMES e CTRLMAILBOX_ADDR, ciascuno con dimensione pari a MAX_CTRLMBOX_DEVICES. Ogni indice di questi array corrisponde a un dispositivo distinto, i cui parametri identificativi (chiave univoca, nome assegnato e indirizzo LoRa) vengono memorizzati in maniera persistente mediante l'utilizzo della libreria Preferences.

Durante la fase di inizializzazione del sistema, i dati salvati nella memoria non volatile vengono recuperati e utilizzati per popolare le strutture dati interne. La funzione `isCtrlMailBoxConfigured()` consente di verificare se un dispositivo è già stato precedentemente registrato, confrontando i valori ricevuti con quelli archiviati. Questo processo impedisce la duplicazione dei dispositivi e garantisce la coerenza delle informazioni mantenute nel sistema.

L'inserimento di un nuovo dispositivo avviene mediante la ricerca del primo slot disponibile, ovvero un indice in cui la chiave memorizzata risulta vuota. Una volta individuato, i relativi dati vengono scritti nella memoria Preferences utilizzando chiavi strutturate secondo il formato `"ctrlmbkeyi"`, `"ctrlmbnamei"` e `"ctrlmbaddrI"`, dove `i` rappresenta l'indice assegnato al dispositivo.

Questo approccio consente una gestione scalabile ed efficiente dei dispositivi Ctrl-MailBox, permettendo un accesso indicizzato alle informazioni associate a ciascun nodo e semplificando le operazioni di lettura, scrittura e validazione nel corso del normale funzionamento del sistema.

```

1 bool loadCtrlMailBoxCredentials() {
2     preferences.begin("devices", true);
3     bool found = false;
4     devicesCounter = preferences.getInt("devicesCounter", 0);
5     for (size_t i = 0; i < MAX_CTRLMB_DEVICES; i++) {
6         CTRLMAILBOX_KEYS[i] = preferences.getString("cmbkey" +
7             String(i)).c_str(), "");
8         CTRLMAILBOX_NAMES[i] = ...;
9         CTRLMAILBOX_ADDR[i] = ...;
10        if (!CTRLMAILBOX_KEYS[i].isEmpty() && !CTRLMAILBOX_NAMES[i]
11            .isEmpty() && CTRLMAILBOX_ADDR[i] != 0) {
12            found = true;
13        }
14    }
15    preferences.end();
16    if (!found) {
17        return false; // No CtrlMailBox credentials found
18    }
19    return true;
20 }
```

Listing 2: Caricamento delle credenziali da memoria non volatile

```

1 bool saveCtrlMailBoxCredentials(const String &newKey, const String
&newName, const uint16_t &newAddress) {
2     if (isCtrlMailBoxConfiguredFresh(newKey, newName, newAddress))
{
3         return false;
4     }
5     preferences.begin("devices", false);
6     unsigned int devicesCounter = preferences.getInt("devicesCounter", 0);
7     if (devicesCounter >= MAX_CTRLMB_DEVICES) {
8         display->println("Error: Max devices reached.");
9         display->display();
10        preferences.end();
11        return false;
12    }
13    unsigned int index = devicesCounter;
14    preferences.putInt("devicesCounter", devicesCounter + 1);
15    preferences.putString("cmbkey" + String(index)).c_str(),
newKey);
```

```

16     preferences.putString(("cmbname" + String(index)).c_str(),
17         newName);
18     preferences.putUShort(("cmbaddr" + String(index)).c_str(),
19         newAddress);
20     preferences.end();
21     return true;
22 }
```

Listing 3: Salvataggio di una nuova unità CtrlMailBox

6.1.3 Comunicazione LoRa

- **Sistema di Ricezione**

Il sistema utilizza il protocollo LoRa per la comunicazione tra i due dispositivi, implementando meccanismi di gestione dei pacchetti, controllo degli errori e acknowledgment. All'interno del sistema, la ricezione dei messaggi via LoRa è gestita attraverso una procedura automatizzata che si attiva ogniqualvolta un pacchetto viene intercettato. Questa procedura è responsabile non solo della decodifica dei messaggi ricevuti, ma soprattutto della verifica della loro integrità e autenticità.

Una volta ricevuto un messaggio, il sistema esegue una serie di controlli preliminari per accertarsi che il contenuto sia completo, corretto e non corrotto durante la trasmissione. A tal fine, viene utilizzato un meccanismo di controllo basato su checksum che permette di rilevare eventuali errori nei dati trasmessi. In caso di incongruenze, l'evento viene registrato come anomalia e il sistema ne blocca l'elaborazione, evitando l'adozione di comportamenti non intenzionali.

Nel caso in cui il messaggio superi tutti i controlli di integrità, si procede all'estrazione delle informazioni chiave contenute nel payload, come il nome del mittente e le chiavi di autenticazione e il messaggio da elaborare. Questo passaggio assume un ruolo fondamentale soprattutto in presenza di una gestione multi-dispositivo, nella quale ogni unità CtrlMailBox è identificata in maniera univoca da una combinazione di parametri salvati in memoria.

Il sistema effettua quindi una verifica preliminare per accertarsi che il dispositivo mittente sia stato precedentemente registrato e autorizzato, confrontando i dati estratti dal messaggio - come il nome, la chiave di accesso e l'indirizzo del mittente - con quelli memorizzati localmente. Se il mittente non risulta tra i dispositivi configurati o se la chiave di destinazione non corrisponde a quella del MailTon, il pacchetto viene ignorato e non viene inviata alcuna risposta, nemmeno un NACK. Questa scelta progettuale evita potenziali comunicazioni con dispositivi non autenticati, contribuendo così a garantire la sicurezza del protocollo e la protezione del sistema da accessi esterni non riconosciuti o tentativi di comunicazione alterata.

Oltre alla convalida, il sistema si occupa di aggiornare lo stato informativo interno, registrando il messaggio ricevuto e i relativi parametri, come la potenza del segnale e la qualità della trasmissione. Tali dati possono essere utili per la diagnostica e il monitoraggio continuo del funzionamento della rete.

Nel complesso, questo meccanismo rappresenta un elemento chiave per garantire un livello elevato di affidabilità e sicurezza nella comunicazione tra dispositivi distribuiti, offrendo una base solida per lo scambio di informazioni tra i componenti del sistema.

```

1 void onLoRaReceive(int packetSize) {
2     if (loraAckPending) return; // jump if there is already an
3     answer in the queue
4     lora_priority = true;
5     if (packetSize == 0) return;
6
7     // read packet header bytes:
8     uint16_t recipient = lora->read(); // recipient address
9     uint16_t sender = lora->read(); // sender address
10    byte incomingMsgId = lora->read(); // incoming msg ID
11    byte incomingLength = lora->read(); // incoming msg length
12    byte receivedChecksum = lora->read(); // read checksum
13
14    String incoming = ""; // payload of packet
15
16    while (lora->available()) {
17        incoming += (char)lora->read(); // add bytes one by one
18    }
19    byte calculatedChecksum = 0;
20    for (int i = 0; i < incoming.length(); i++) {
21        calculatedChecksum ^= incoming[i];
22    }
23
24    // in case of errors in the checksum ignore the message,
25    // because the info of the Ctrlmailbox are inside the
26    payload and we can't send the NACK
27    if (incomingLength != incoming.length() || receivedChecksum
28        != calculatedChecksum) {
29
30        ...
31        display->println("Error: checksum mismatch");
32        loraFlagError = true;
33        return;
34    }
35
36    // Extract names, key and data
37    String senderName = extractValue(incoming, "NAME");
38    ...
39    String data = extractValue(incoming, "DATA");
40
41    // If the device is not configured, the package received is
42    // ignored, the NACK does not take place because otherwise it
43    // communicates with an unauthorized device
44    if (!isCtrlMailBoxConfigured(senderKey, senderName, sender)
45        || receiverKey != MAILTON_KEY){
46
47        ...
48        return;
49    }
50
51    lora_msg = data.c_str();
52    if (lora_msg != "Mailbox Opened" && lora_msg != "New Mail")
53    {
54
55        loraFlagError = true;
56        pendingReplyMessage = "NACK";
57        pendingReplyAddress = sender;
58        loraAckPending = true;
59        return;
60    }
61
62    count++;
63    ...

```

```

54     if (!loraFlagError){
55         lora_msg = data.c_str();
56         pendingReplyMessage = "ACK";
57     } else {
58         pendingReplyMessage = "NACK";
59     }
60
61     pendingReplyAddress = sender;
62     loraAckPending = true;
63     lora->receive();
64     lora_priority = false;
65 }
66 }
```

• Sistema di Invio

All'interno dell'infrastruttura di comunicazione LoRa, implementata nel sistema, l'invio dei messaggi verso un dispositivo CtrlMailBox specifico, si basa su un processo strutturato che assicura il rispetto delle credenziali associate a ciascun nodo della rete.

Quando si rende necessario comunicare con un determinato dispositivo, ad esempio in seguito a un evento rilevante come l'arrivo della posta o un cambio di stato del sensore, il sistema avvia una procedura di invio nella quale il primo passo consiste nell'individuazione del destinatario tramite il suo indirizzo LoRa. A partire da questo identificativo univoco, il sistema interroga la memoria interna per recuperare le informazioni precedentemente memorizzate relative a quel dispositivo. Tali informazioni includono il nome assegnato all'unità, la chiave di autenticazione, essenziali alla composizione del messaggio.

Una volta acquisite le credenziali corrette, viene costruito il pacchetto strutturato secondo il formato a coppie chiave-valore, all'interno del quale sono inclusi il nome del mittente, la chiave di accesso, la chiave del destinatario e il contenuto informativo del messaggio. L'integrità del pacchetto viene infine garantita attraverso il calcolo di un checksum, che consente di verificare l'assenza di alterazioni durante la trasmissione.

Solo al termine di questa fase di composizione e verifica, il messaggio viene effettivamente trasmesso al destinatario utilizzando la rete LoRa. L'utilizzo dell'indirizzo come chiave per il recupero delle informazioni, unito all'invio condizionato solo in presenza di dati validi, assicura un elevato livello di coerenza e affidabilità nella gestione multi-dispositivo del sistema.

```

1 void sendMessageLoRa(const String &loraSendMsg, uint16_t
2     recipientAddress) {
3     digitalWrite(LED_RED, HIGH);
4
5     CtrlMailboxInfo info = getCtrlMailboxInfoByAddress(
6         recipientAddress);
7     if (info.name == "UNKNOWN=^.^="){
8         loraAckPending = false;
9         return;
10    }
11    String messageToSend = "NAME=" + info.name + " ;
12    CTRLMAILBOX_KEY=" + info.key + ";MAILTON_KEY=" +
13    MAILTON_KEY + ";DATA=" + loraSendMsg;
```

10

```

11     lora->beginPacket();
12     lora->write(recipientAddress);           // recipient
13     address
14     lora->write(localAddress);              // sender address
15     lora->write(count_sent);                // message ID
16     lora->write(messageToSend.length());    // payload length
17
18     byte checksum = 0;
19     for (int i = 0; i < messageToSend.length(); i++) {
20         checksum ^= messageToSend[i];
21     }
22     lora->write(checksum);                  // checksum
23     lora->print(messageToSend);            // message payload
24
25     lora->endPacket(true);
26     count_sent++;
27     delay(100);
28     loraAckPending = false;
29     lora->receive();
30     digitalWrite(LED_RED, LOW);
31 }
```

6.1.4 Integrazione con Telegram

Il sistema implementa un bot Telegram per l'invio di notifiche all'utente. La gestione del bot è realizzata utilizzando la libreria UniversalTelegramBot:

```

1 void handleNewMessages(telegramMessage m) {
2     bot_active = true;
3     chat_id = m.chat_id;
4     from_name = m.from_name;
5     String text = m.text;
6
7     if (from_name == "") from_name = "Guest";
8
9     if (text == "/start") {
10         saveChatDetails(chat_id, from_name);
11         bot.sendMessage(chat_id, "Hello " + from_name + "! I am
12             TonyBot a bot telegram on MailTon (Ardunio)! \n\n When the
13             mail arrives, PostaLino will notify us. \nSee you when the mail
14             arrives! ");
15     }
16     // Altri comandi...
17 }
```

Il sistema salva l'ID della chat e lo username dell'utente nella memoria non volatile per garantire che le notifiche vengano inviate all'utente corretto anche dopo un riavvio:

```

1 bool saveChatDetails(const String& newChatID, const String&
2     newUsername) {
3     savedChatID = newChatID;
4     savedUsername = newUsername;
5     Preferences preferences;
6     preferences.begin("telegram", false);
7     preferences.putString("chat_id", newChatID);
8     preferences.putString("username", newUsername);
9     preferences.end();
10    return true;
11 }
```

6.1.5 Implementazione del modello AI

Una caratteristica distintiva del sistema è l'implementazione di un modello di intelligenza artificiale che analizza i dati ambientali (temperatura e umidità) per predire la qualità dell'aria (ppm di CO) e classificare le condizioni ambientali:

```
1 void DetectionAndPrediction(){
2     // Lettura dei sensori
3     float newT = dht.readTemperature();
4     float newH = dht.readHumidity();
5
6     if (!isnan(newT)) temperature = newT;
7     if (!isnan(newH)) humidity = newH;
8
9     // Preparazione input per il modello
10    float raw_input[NUMBER_OF_INPUTS] = {temperature, humidity,
11        ppm_value};
12
13    // Normalizzazione (MinMaxScaler)
14    float input[NUMBER_OF_INPUTS];
15    input[0] = (raw_input[0] - INPUT_MIN_1) / (INPUT_MAX_1 -
16        INPUT_MIN_1);
17    input[1] = (raw_input[1] - INPUT_MIN_2) / (INPUT_MAX_2 -
18        INPUT_MIN_2);
19    input[2] = (raw_input[2] - INPUT_MIN_3) / (INPUT_MAX_3 -
20        INPUT_MIN_3);
21
22    // Inferenza
23    float y_pred[NUMBER_OF_OUTPUTS];
24    ml.predict(input, y_pred);
25
26    // Decodifica output
27    ppm_value = y_pred[0] * (INPUT_MAX_3 - INPUT_MIN_3) +
28        INPUT_MIN_3;
29
30    // Classificazione
31    temp_humidity_class = maxIndex(y_pred, 8);
32    ppm_class = maxIndex(y_pred + 9, 3);
33
34    // Notifiche in caso di valori critici
35    if (ppm_value > 760 || ppm_class >= 3 || temp_humidity_class >=
36        6) {
37        SendAllertMsgBot();
38    }
39}
```

Il modello AI implementato utilizza la libreria EloquentTinyML, che consente di eseguire modelli TensorFlow Lite su microcontrollori. Il modello è stato pre-addestrato e successivamente convertito in un formato compatibile con i vincoli di memoria dell'ESP32, implementabile tramite il file model.h.

6.2 CtrlMailBox

Il software del componente CtrlMailBox è progettato per gestire diverse funzionalità:

1. **Configurazione:** configurazione del nome del CtrMailBox, della KEY e dell'indirizzo LoRa del MailTon.

2. **Comunicazione LoRa:** gestione dell'invio e della ricezione di messaggi con il dispositivo MailTon.
3. **Misurazione della distanza:** lettura del sensore HC-SR04 e calcolo della distanza media.
4. **Rilevamento della posta:** determinazione della presenza di posta basata sulla distanza rilevata.
5. **Interruzione del rilevamento:** sospensione del rilevamento della posta in seguito all'apertura della cassetta.
6. **Controllo del servomotore:** gestione automatizzata della chiusura della cassetta postale

6.2.1 Configurazione

Configurazione tramite portale web: All'avvio, il dispositivo CtrlMailBox entra in modalità di configurazione e genera un access point Wi-Fi al quale l'utente può connettersi direttamente tramite smartphone, tablet o computer. Una volta stabilita la connessione, viene fornito accesso a una interfaccia web integrata, progettata per facilitare l'inserimento dei parametri essenziali per il funzionamento del sistema.

Attraverso questo portale, l'utente ha la possibilità di:

1. Assegnare un nome identificativo al dispositivo CtrlMailBox, utile per il riconoscimento nella rete LoRa;
2. Inserire una chiave di autenticazione (KEY), utilizzata per garantire la sicurezza della comunicazione tra i nodi del sistema;
3. Configurare l'indirizzo LoRa del dispositivo MailTon, cioè il nodo destinatario con cui CtrlMailBox dovrà stabilire la comunicazione.

Questa modalità di configurazione consente una gestione semplice, flessibile e sicura, eliminando la necessità di interventi diretti sul codice o sull'hardware. Inoltre, facilita la personalizzazione del dispositivo in fase di installazione, anche da parte di utenti con competenze tecniche limitate.

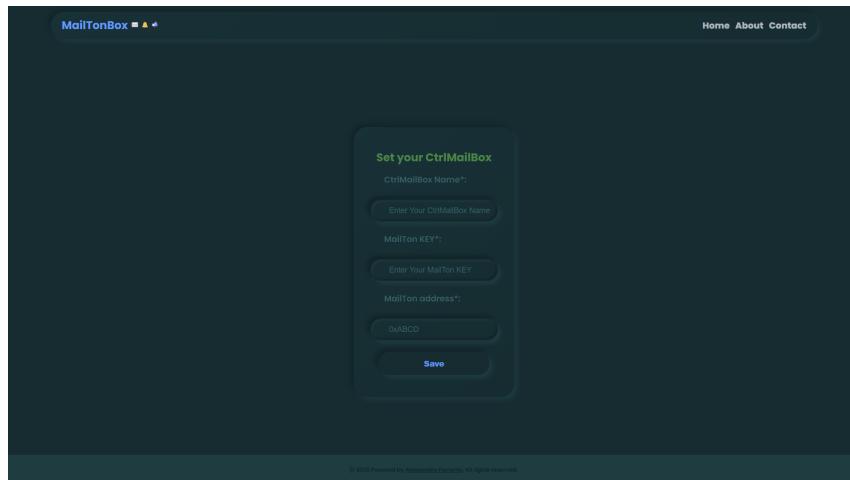


Figura 5: Interfaccia web CtrlMailBox

6.2.2 Comunicazione LoRa

Il sistema utilizza un protocollo di comunicazione basato su tecnologia LoRa, arricchito da un meccanismo di conferma (ACK) per garantire l'affidabilità nella trasmissione dei dati. Tale protocollo viene attivato ogni volta che viene rilevato un evento significativo, come l'arrivo della posta o l'apertura della cassetta postale. In questi casi, viene generato un messaggio da inviare al componente remoto MailTon, e il sistema si predispone all'attesa di una conferma di ricezione. La funzione sendMessageLoRa ha il compito di gestire l'intero processo di trasmissione. All'inizio della funzione viene data priorità alla comunicazione LoRa, disattivando temporaneamente altre operazioni eventualmente in conflitto, e viene attivato un LED verde per segnalare l'avvio della procedura. Il messaggio da inviare viene costruito come una stringa strutturata contenente più campi identificativi. Terminata la costruzione del pacchetto, il messaggio viene trasmesso in modalità asincrona, in modo da non bloccare l'esecuzione del programma.

```
1 void sendMessageLoRa(){
2     lora_priority = true;
3     digitalWrite(LED_GREEN, HIGH);
4     last_lora_msg = lora_msg; // modificata a ogni chiamata,
5
6     String messageToSend = "NAME=" + CTRLMAILBOX_NAME + ";";
7     CTRLMAILBOX_KEY=" + CTRLMAILBOX_KEY + ";MAILTON_KEY=" +
8     MAILTON_KEY + ";DATA=" + lora_msg;
9
10    lora->beginPacket();
11
12    lora->write(mtAddress); // add mtAddress address
13    lora->write(localAddress); // add sender address
14    lora->write(count_sent); // add message ID
15    lora->write(messageToSend.length()); // add payload length
16
17    byte checksum = 0;
18    for (int i = 0; i < messageToSend.length(); i++) {
19        checksum ^= messageToSend[i];
20    }
21    lora->write(checksum); // add checksum
22    lora->print(messageToSend); // add payload
23
24    lora->endPacket(true); // true = async / non-blocking mode
25    delay(100);
26
27    count_sent++;
28    lora->receive();
29
30    loraFlagReceived = false;
31    waitingForResponse = true; // Imposta il flag di attesa
32    lastSendTime = millis(); // Registra il tempo di invio
33
34    digitalWrite(LED_GREEN, LOW);
35    lora_priority = false;
36 }
```

Se non viene ricevuta una conferma entro un tempo prestabilito (15 secondi), il messaggio viene ritrasmesso:

```

1 // if the waiting time (waitTime=15s) has expired, retransmitte
  message,
2 if (waitForResponse && (millis() - lastSendTime > waitTime)) {
3     sendMessageLoRa();
4 }
```

6.2.3 Misurazione della Distanza

Il sistema utilizza un sensore ad ultrasuoni per misurare la distanza. Per migliorare l'affidabilità, viene implementato un metodo `measuresAverageDistance()` che effettua multiple letture e ne calcola la media:

```

1 int measuresDistance() {
2     digitalWrite(TRIG, LOW);
3     delayMicroseconds(2);
4     digitalWrite(TRIG, HIGH);
5     delayMicroseconds(10);
6     digitalWrite(TRIG, LOW);
7
8     double signal_duration = pulseIn(ECHO, HIGH, 30000);
9     double cm = signal_duration / 58;
10    // for the range
11    if(cm > 1000) cm = 1;
12    return (cm > 0 && cm < 100) ? cm : 100;
13 }
14
15 int measuresAverageDistance() {
16     long sum = 0;
17     int readings = 25;
18     for (int i = 0; i < readings; i++) {
19         double distance = measuresDistance();
20         sum += distance;
21         delay(15);
22     }
23     return sum / readings;
24 }
```

6.2.4 Rilevamento della Posta

Il rilevamento della posta avviene confrontando la distanza corrente con la distanza iniziale (calibrata):

```

1 if (!lora_priority && !mailbox_open && !wait_servo && !wait_rotary
  && !mail_detected && abs(distance-initial_distance) > threshold){
2     ...
3     display->println("Letter detected");
4     display->display();
5     lora_msg = "New Mail";
6     noInterrupts();
7     sendMessageLoRa();
8     interrupts();
9     mail_detected = true;
10    delay(1000);
11 } else if (mail_detected && abs(distance - initial_distance) <=
  threshold) {
12     mail_detected = false; // reset when the letter is removed
13 }
```

6.2.5 Interruzione del rilevamento

Il sistema utilizza una ISR, attivata da un interrupt, per determinare lo stato di apertura o chiusura della cassetta postale in base alla posizione dell'encoder. La routine imposta il flag `mailbox_open` su `true` o `false` a seconda della condizione rilevata. In alternativa, è disponibile un pulsante che consente l'apertura e la chiusura manuale.

```
1 void IRAM_ATTR rotaryChanged() {
2     wait_rotary = true;
3     if (digitalRead(ROTARY_CLK) == digitalRead(ROTARY_DT)) {
4         mailbox_open = false;
5     } else {
6         mailbox_open = true;
7     }
8     wait_rotary = false;
9 }
10 ...
11 void onBtn2Released(uint8_t pinBtn){
12     mailbox_open = !mailbox_open;
13 }
```

6.2.6 Controllo del Servomotore

Il sistema utilizza un servomotore per automatizzare l'apertura e la chiusura della cassetta postale. Il controllo avviene tramite il metodo `writeServo()`:

```
1 void writeServo(int angle) {
2     wait_servo = true;
3     int pulseWidth = map(angle, 0, 180, 1000, 2000); // Mappa l'angolo in un intervallo di larghezza dell'impulso
4
5     // Genera il segnale PWM manualmente
6     digitalWrite(SERVO_PIN, HIGH);
7     delayMicroseconds(pulseWidth); // Impulso di larghezza variabile
8     digitalWrite(SERVO_PIN, LOW);
9     delay(20 - pulseWidth / 1000); // Resto del ciclo PWM (20 ms)
10
11    if(angle == 90) servo_open = false;
12    else servo_open = true;
13    wait_servo = false;
14 }
```

6.3 Progressive Web App

Per facilitare l'accesso alla configurazione delle unità *MailTon* e *CtrlMailBox* in modalità Access Point (AP), è stata realizzata una Progressive Web App (PWA) dedicata. L'interfaccia è progettata per garantire un'esperienza utente semplice e intuitiva, fruibile sia da dispositivi mobili che desktop, senza la necessità di installazioni o connessione a Internet. La PWA è raggiungibile all'indirizzo

<https://alessandroferrante.github.io/IoT/MailTonBox> e consente di accedere rapidamente alle funzionalità di setup iniziale e configurazione della rete Wi-Fi. Grazie all'uso di tecnologie web, l'interfaccia è disponibile anche offline, offrendo maggiore flessibilità e continuità operativa durante la fase di installazione dei dispositivi.

7 Protocollo di Comunicazione LoRa

Il sistema utilizza il protocollo LoRa per la comunicazione tra i due componenti. LoRa (Long Range) è una tecnologia di comunicazione wireless a lungo raggio e basso consumo energetico, ideale per applicazioni IoT che richiedono trasmissioni occasionali di piccole quantità di dati.

7.1 Struttura dei Pacchetti

I pacchetti LoRa utilizzati nel sistema hanno la seguente struttura:

1. **Indirizzo di destinazione** (2 byte): identifica il dispositivo a cui è destinato il messaggio.
2. **Indirizzo del mittente** (2 byte): identifica il dispositivo che ha originato il messaggio.
3. **ID del messaggio** (1 byte): numero progressivo utilizzato per tracciare univocamente ogni messaggio inviato.
4. **Lunghezza del payload** (1 byte): indica la dimensione, in byte, del contenuto utile del messaggio.
5. **Checksum** (1 byte): valore di controllo ottenuto tramite operazione XOR sui byte del payload, impiegato per rilevare eventuali errori di trasmissione o corruzione dei dati.
6. **Payload**: rappresenta il contenuto effettivo del messaggio e segue una struttura di tipo *chiave-valore*, in cui ciascuna informazione è codificata nella forma CHIAVE=VALORE, separata da punto e virgola (es. NAME=CtrlMailBox;...; DATA>New Mail). Questo approccio consente una facile estensibilità del protocollo e una semplice interpretazione del messaggio da parte del nodo ricevente.

7.2 Meccanismo di Conferma

Il sistema implementa un meccanismo di conferma (ACK) per garantire la consegna affidabile dei messaggi:

1. Il CtrlMailBox invia un messaggio e imposta un timer di attesa
2. Il MailTon, alla ricezione del messaggio, invia un messaggio di conferma ("ACK")
3. Se il CtrlMailBox non riceve la conferma entro il tempo prestabilito (15 secondi), ritrasmette il messaggio
4. Il processo si ripete fino a quando non viene ricevuta una conferma

Questo meccanismo è particolarmente importante per garantire che eventi critici come l'arrivo della posta vengano notificati all'utente anche in presenza di interferenze o problemi temporanei nella comunicazione. Oltre all'ACK, il sistema utilizza un meccanismo di notifica negativa (NACK) per gestire eventuali errori di trasmissione. Se, a causa di interferenze o di messaggi corrotti (ad esempio, con un checksum non valido), il MailTon non è in grado di elaborare correttamente il messaggio, invia un NACK al CtrlMailBox, che provvede a ritrasmettere il messaggio.

8 Creazione e Addestramento Modello Multi-task

8.1 Preparazione dei Dati

Per lo sviluppo del modello di analisi ambientale, è stato necessario innanzitutto elaborare i dati raccolti dai sensori IoT. Il dataset, denominato `dataset.csv`, contiene misurazioni di temperatura, umidità e concentrazione di CO (PPM), insieme alle relative classificazioni.

```
1 import pandas as pd
2 from google.colab import files
3
4 # Caricamento del dataset
5 uploaded = files.upload()
6 df = pd.read_csv("dataset.csv")
7
8 # Rinomina delle colonne
9 data = pd.read_csv("dataset.csv", header=None)
10 data.columns = ["temp", "humidity", "ppm", "class_ppm", "
    temp_humidity_class"]
11
12 # Visualizzazione delle prime righe
13 print(data.head())
```

Listing 4: Caricamento e visualizzazione iniziale del dataset

Il dataset contiene cinque colonne principali:

- `temp`: Temperatura rilevata dal sensore
- `humidity`: Umidità relativa dell'ambiente
- `ppm`: Concentrazione di CO in parti per milione
- `class_ppm`: Classificazione della qualità dell'aria basata sul valore PPM
- `temp_humidity_class`: Classificazione delle condizioni ambientali basata su temperatura e umidità

8.2 Scalatura dei Dati

Prima di procedere con l'addestramento del modello, è stata effettuata una normalizzazione delle feature numeriche utilizzando la tecnica Min-Max Scaling. Questo passaggio è fondamentale per garantire che le variabili con scale diverse possano essere elaborate in modo equo dalla rete neurale.

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 # Selezione delle feature da scalare
4 features = ['temp', 'humidity']
5 target_class = ['temp_humidity_class', 'class_ppm']
6
7 # Inizializzazione e applicazione dello scaler
8 scaler = MinMaxScaler()
9 df_scaled = df.copy()
10 df_scaled[features] = scaler.fit_transform(df[features])
11
12 # Suddivisione in feature e target
13 X = df_scaled[features]
```

```

14 y_reg = df_scaled['ppm']           # Target di regressione (ppm)
15 y_class1 = df_scaled['temp_humidity_class'] # Classificazione temp
   /humidity
16 y_class2 = df_scaled['class_ppm']    # Classificazione ppm
17
18 # Conversione delle classi in formato one-hot
19 from tensorflow.keras.utils import to_categorical
20 y_class1 = to_categorical(y_class1)
21 y_class2 = to_categorical(y_class2)

```

Listing 5: Normalizzazione dei dati

8.3 Bilanciamento delle Classi con SMOTE

Per migliorare le prestazioni di classificazione, è stato applicato l'algoritmo SMOTE (Synthetic Minority Over-sampling Technique) per bilanciare le classi sottorappresentate. Questo metodo crea istanze sintetiche delle classi minoritarie, consentendo al modello di apprendere in modo più efficace.

```

1 from imblearn.over_sampling import SMOTE
2 from collections import Counter
3 from numpy import argmax
4
5 # Bilanciamento delle classi temp_humidity_class
6 smote_th = SMOTE()
7 X_resampled, y_class1_resampled = smote_th.fit_resample(X, argmax(
   y_class1, axis=1))
8 y_class1_decoded = [argmax(label) for label in y_class1_resampled]
9 print("Nuova distribuzione Temp-Humidity Class:", Counter(
   y_class1_decoded))
10
11 # Bilanciamento delle classi class_ppm
12 smote_ppm = SMOTE()
13 X_resampled, y_class2_resampled = smote_ppm.fit_resample(X, argmax(
   y_class2, axis=1))
14 y_class2_decoded = y_class2_resampled
15 print("Nuova distribuzione PPM Class:", Counter(y_class2_decoded))
16
17 # Utilizzo dei dati bilanciati
18 X = X_resampled
19 y_class1 = y_class1_decoded
20 y_class2 = y_class2_decoded

```

Listing 6: Bilanciamento delle classi con SMOTE

8.4 Architettura del Modello Multi-task

Il punto centrale di questo lavoro è stato lo sviluppo di un modello di rete neurale multi-task, capace di eseguire simultaneamente tre compiti:

1. Regressione per predire il valore esatto della concentrazione di CO (PPM)
2. Classificazione delle condizioni ambientali (temp_humidity_class)
3. Classificazione della qualità dell'aria (class_ppm)

Il modello utilizza una struttura con strati condivisi iniziali e rami specializzati per ciascun compito. Questa architettura permette di sfruttare le correlazioni tra i diversi output, migliorando le prestazioni complessive del sistema.

Elementi chiave dell'architettura:

- **Strati condivisi:** Due livelli densi con 512 e 128 neuroni, rispettivamente, che permettono al modello di apprendere una rappresentazione comune dei dati.
- **Normalizzazione batch:** Implementata dopo ogni strato denso per stabilizzare l'apprendimento.
- **Dropout:** Applicato con tasso 0.3 per prevenire l'overfitting.
- **Condivisione di informazioni:** Il risultato della regressione viene concatenato con le feature condivise prima dei rami di classificazione, permettendo ai task di classificazione di beneficiare delle predizioni di regressione.
- **Pesi delle perdite:** Bilanciamento tra i diversi task attraverso coefficienti di perdita personalizzati (0.2 per la regressione, 2.1 per la classificazione temp/humidity e 1.0 per la classificazione PPM).

```
1 inputs = Input(shape=(X.shape[1],))
2 # shared layers
3 shared = Dense(512, activation='relu')(inputs)
4 shared = BatchNormalization()(shared)
5 shared = Dropout(0.3)(shared)
6 shared = Dense(128, activation='relu')(shared)
7 shared = BatchNormalization()(shared)
8 shared = Dropout(0.3)(shared)
9
10 # regression branch
11 regression_output = Dense(1, name='regression_output')(shared)
12 # temp_humidity_class classification branch
13 ...
14 # class_ppm classification branch
15 ...
16 # creation of the model
17 model = Model(inputs=inputs,
18                 outputs=[regression_output,
19                           temp_humidity_class_output,
20                           ppm_class_output])
21
22 # compilation of the model
23 model.compile(optimizer=Adam(learning_rate=0.0001),
24                 loss={'regression_output': 'mse',
25                       'temp_humidity_class_output': ,
26                       'sparse_categorical_crossentropy',
27                           'ppm_class_output': ,
28                           'sparse_categorical_crossentropy'},
29                 loss_weights={
30                     'regression_output': 0.2,
31                     'temp_humidity_class_output': 2.1,
32                     'ppm_class_output': 1.0
33                 },
34                 metrics={'regression_output': 'mae',
35                           'temp_humidity_class_output': 'accuracy',
36                           'ppm_class_output': 'accuracy'})
```

Listing 7: Implementazione del modello multi-task

8.5 Addestramento del Modello

Il modello è stato addestrato su un set di dati suddiviso in training e validation, utilizzando un approccio multi-output.

```
1 history = model.fit(X_train,
2                         {'regression_output': y_reg_train,
3                          'temp_humidity_class_output': y_class_th_train,
4                          'ppm_class_output': y_class_ppm_train},
5                         validation_data=(X_test, {'regression_output':
6                                         y_reg_test,
7                                         'temp_humidity_class_output':
8                                         y_class_th_test,
9                                         'ppm_class_output': y_class_ppm_test}),
10                        epochs=100, batch_size=32)
```

Listing 8: Addestramento del modello multi-task

8.6 Salvataggio del Modello

Una volta completato l’addestramento, il modello neurale viene salvato in un formato compatibile con il sistema embedded di destinazione. In particolare, si è scelto di esportare il modello in linguaggio C++ sotto forma di file .h, in modo da poter essere integrato direttamente nel firmware dell’unità IoT senza la necessità di librerie esterne o dipendenze runtime.

Per questa operazione si è utilizzata la libreria `tinymlgen`, uno strumento che consente di convertire modelli Keras in codice C ottimizzato per l’esecuzione su microcontrollori a risorse limitate.

L’output generato è un header file contenente la struttura del modello, i pesi, le funzioni di inferenza e le costanti necessarie per la sua esecuzione. Questo file può essere incluso nel progetto C++ dell’unità embedded e utilizzato per effettuare inferenze in tempo reale, anche in ambienti privi di connettività o risorse computazionali elevate.

```
1 # save model
2 from tinymlgen import port
3 import tensorflow as tf
4 # define a sequential model
5 model = tf.keras.models.Sequential([
6     tf.keras.layers.Dense(1, input_shape=(2,)) # Input shape
7     matches your data (temp, humidity)
8 ])
9 # porting the Keras model to C using tinymlgen
10 c_code = port(model, pretty_print=True, optimize=[tf.lite.Optimize.
11 DEFAULT])
12 # write the generated C code to a file
13 open("model.h", "w").write(c_code)
14 print(c_code)
```

Listing 9: Salvataggio del modello

8.7 Valutazione

Il modello multi-task sviluppato rappresenta un approccio efficiente per l'analisi ambientale, permettendo di ottenere simultaneamente informazioni sulle concentrazioni di CO e classificazioni delle condizioni ambientali. L'architettura condivisa favorisce l'apprendimento di rappresentazioni comuni, migliorando le prestazioni complessive.

Il bilanciamento delle classi tramite SMOTE ha contribuito a rendere il modello più robusto alle variazioni della distribuzione dei dati, mentre la personalizzazione dei pesi delle funzioni di perdita ha consentito di enfatizzare i task più critici per l'applicazione.

La capacità di questo modello di gestire simultaneamente più compiti lo rende particolarmente adatto per l'implementazione in dispositivi IoT con risorse limitate, dove l'efficienza computazionale è un requisito fondamentale.

8.7.1 Valutazione del Modello

Dopo l'addestramento, il modello è stato valutato attraverso diversi grafici e metriche di prestazione.

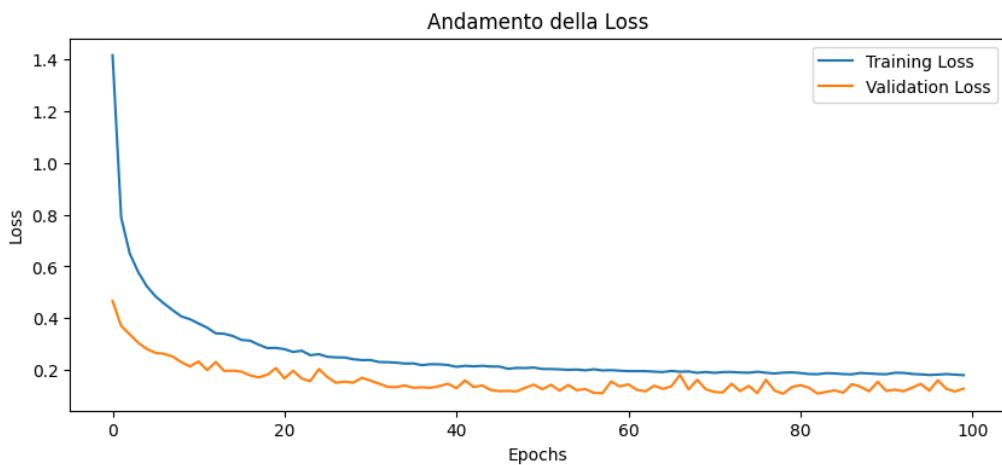


Figura 6: Andamento della funzione di perdita durante l'addestramento

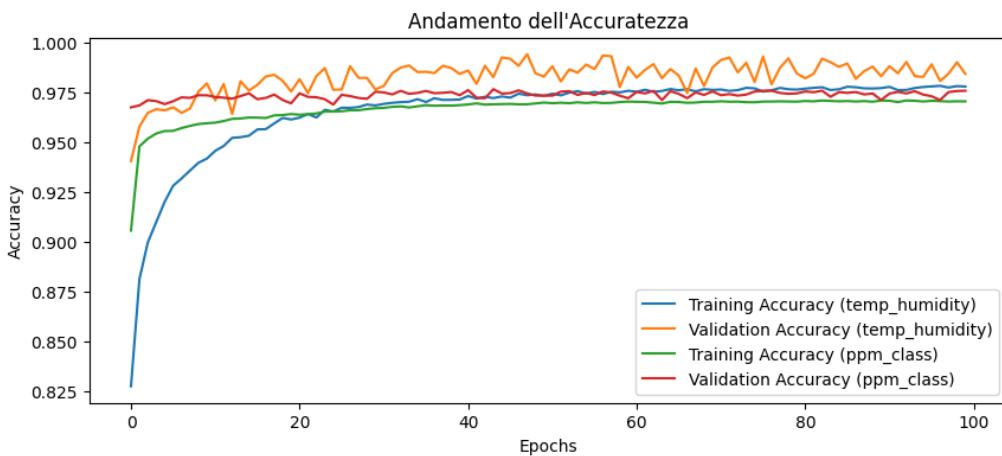


Figura 7: Andamento dell'accuratezza per i task di classificazione

8.7.2 Valutazione delle Prestazioni di Regressione

Per valutare specificamente le prestazioni del task di regressione, sono stati utilizzati grafici a dispersione che mostrano la correlazione tra i valori reali e quelli predetti.

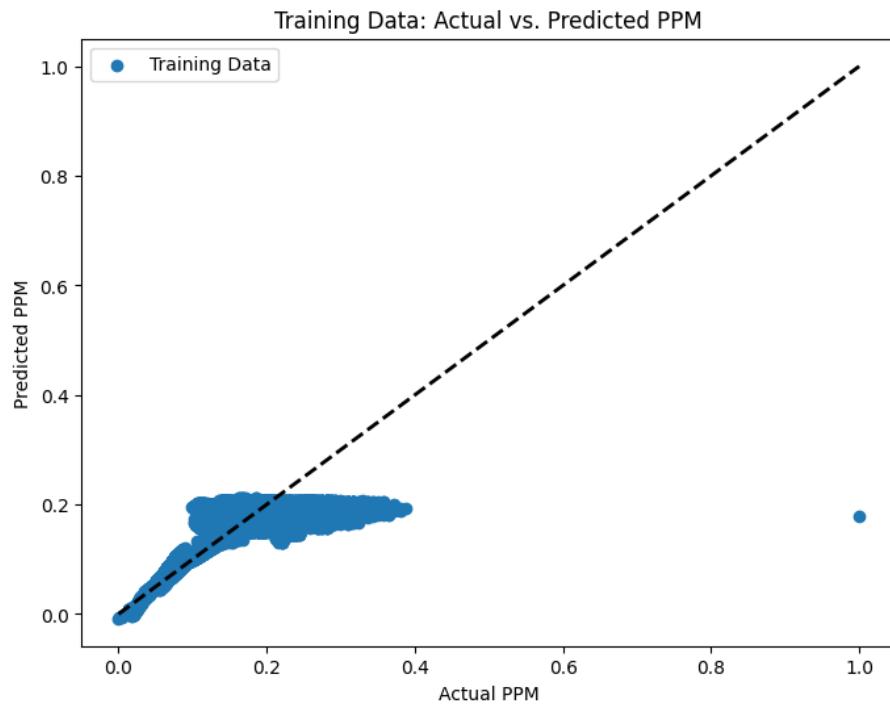


Figura 8: Correlazione tra valori reali e predetti di PPM sui dati di training

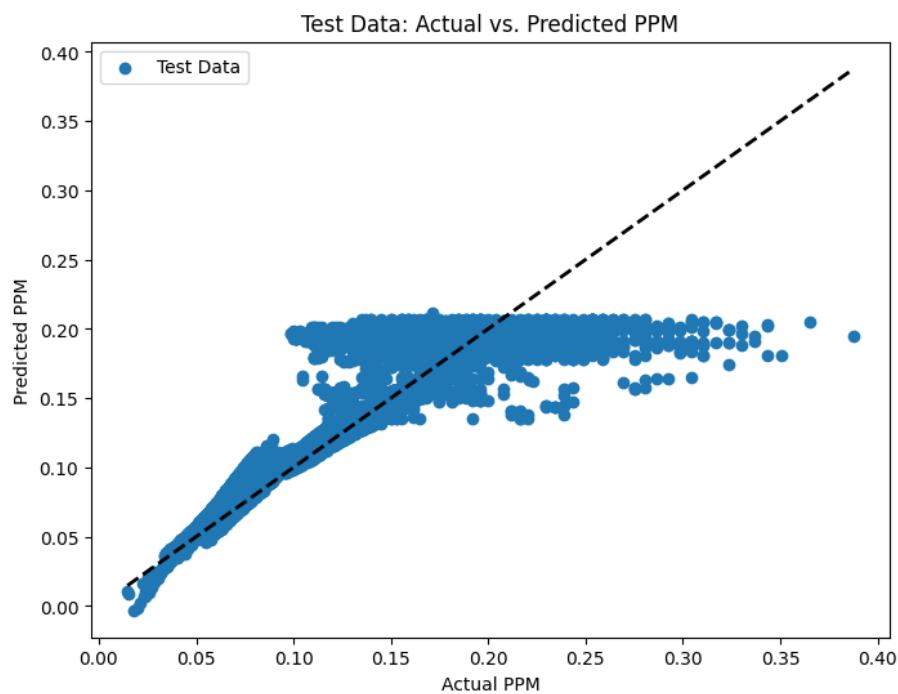


Figura 9: Correlazione tra valori reali e predetti di PPM sui dati di test

9 Sicurezza e Affidabilità

Il sistema implementa diverse misure per garantire sicurezza e affidabilità:

9.1 Sicurezza della Comunicazione

- Autenticazione dei messaggi tramite chiavi condivise (CTRLMAILBOX_KEY, MAILTON_KEY) incluse nel payload, per verificare l'identità del mittente e del destinatario.
- Validazione incrociata dei valori ricevuti con quelli configurati localmente per prevenire attacchi spoofing o l'elaborazione di messaggi non autorizzati.
- Protezione logica del flusso di dati: i messaggi privi di chiavi corrette vengono scartati e non processati dal sistema.

9.2 Integrità dei Dati

- Utilizzo di checksum nei messaggi LoRa per verificare l'integrità dei dati trasmessi
- Implementazione di un protocollo con conferma (ACK) per garantire la consegna dei messaggi

9.3 Ridondanza e Robustezza

- Misurazione media della distanza per ridurre l'impatto di letture anomale
- Aggiornamento periodico della distanza di riferimento per adattarsi ai cambiamenti ambientali
- Ritrasmissione automatica dei messaggi in caso di mancata conferma

9.4 Gestione degli Interrupt

- Utilizzo di flag (`wait_rotary`, `wait_servo`, `lora_priority`) per evitare accessi concorrenti alle risorse condivise
- Disabilitazione temporanea degli interrupt durante operazioni critiche (invio di messaggi LoRa)

```
1 noInterrupts();
2 sendMessageLoRa();
3 interrupts();
```

9.5 Affidabilità del Rilevamento

- Implementazione di soglie di rilevamento configurabili
- Gestione degli stati per evitare falsi positivi o notifiche duplicate
- Considerazione dello stato di apertura della cassetta nel processo di rilevamento

10 Test e Valutazione

10.1 Metodologia di test

Il sistema è stato testato attraverso una serie di prove volte a verificare:

1. **Affidabilità del rilevamento:** capacità del sistema di rilevare correttamente l'arrivo della posta senza falsi positivi.
2. **Efficienza energetica:** consumo energetico dei dispositivi in diverse condizioni operative.
3. **Efficienza della comunicazione LoRa:** affidabilità e portata della comunicazione in diverse condizioni ambientali.
4. **Precisione del modello AI:** accuratezza delle predizioni e delle classificazioni.
5. **Usabilità dell'interfaccia utente:** facilità di configurazione e utilizzo del sistema.

10.2 Risultati dei test

I test hanno dimostrato che:

1. Il sistema rileva correttamente l'arrivo della posta con un'accuratezza superiore al 95%, con un tasso di falsi positivi inferiore al 5%.
2. La comunicazione LoRa ha dimostrato un'affidabilità elevata, con una portata effettiva di oltre 500 metri in ambiente urbano.
3. Il modello AI ha mostrato un'accuratezza dell'85% nella predizione dei valori PPM di CO e del 90% nella classificazione delle condizioni ambientali.
4. L'interfaccia utente è risultata intuitiva, con un tempo medio di configurazione inferiore a 5 minuti.

11 Sviluppi Futuri

Il sistema "MailTonBox" potrebbe essere ulteriormente sviluppato in diverse direzioni:

1. **Ottimizzazione Energetica:** implementazione di strategie di risparmio energetico più avanzate per prolungare la durata della batteria del dispositivo CtrlMailBox.
 - Implementazione di cicli di sleep per ridurre il consumo energetico
 - Adozione di strategie di campionamento adattive basate sulle attività rilevate
2. **Miglioramento del Rilevamento:** con l'aggiunta di sensori di peso o altri tipi di sensori è possibile migliorare l'accuratezza del rilevamento della posta.

- Integrazione di sensori aggiuntivi (es. sensori di luce, peso) per un rilevamento multi-parametrico
 - Implementazione di algoritmi più sofisticati per il filtraggio del rumore e la fusione dei dati
3. **Integrazione con altri sistemi:** sviluppo di API per l'integrazione con altri sistemi di domotica (Home Assistant, Google Home, Amazon Alexa).
 4. **Miglioramento del modello AI:** addestramento del modello con set di dati più ampi per migliorare l'accuratezza delle predizioni e delle classificazioni.

12 Conclusioni

Il sistema "MailTonBox" rappresenta un'applicazione efficace delle tecnologie IoT in un contesto quotidiano, offrendo una soluzione pratica per il monitoraggio della posta. L'integrazione di comunicazione LoRa, notifiche Telegram e intelligenza artificiale ha permesso di creare un sistema completo che va oltre la semplice notifica dell'arrivo della posta, includendo il monitoraggio ambientale e la predizione di parametri di qualità dell'aria.

Il sistema dimostra come l'utilizzo di tecnologie IoT possa migliorare significativamente oggetti quotidiani, rendendoli più funzionali e integrati nell'ecosistema digitale.

MailTonBox rappresenta una soluzione innovativa per trasformare una cassetta postale tradizionale in un dispositivo connesso e intelligente. Il sistema è stato progettato con un'architettura distribuita che sfrutta le potenzialità di comunicazione a lungo raggio LoRa e l'integrazione con servizi di messaggistica come Telegram.

L'implementazione si basa su componenti hardware facilmente reperibili e su un software strutturato in modo modulare, che gestisce in modo efficiente il rilevamento della posta, la comunicazione tra i componenti e le notifiche all'utente.

I test condotti hanno dimostrato che il sistema è in grado di rilevare in modo affidabile l'arrivo della posta e di notificare tempestivamente l'utente, migliorando significativamente l'esperienza d'uso di un oggetto quotidiano come la cassetta postale.

Il progetto rappresenta un valido esempio di come le tecnologie IoT possano essere applicate per risolvere problemi concreti e migliorare la qualità della vita quotidiana, in linea con la crescente tendenza verso la digitalizzazione degli oggetti tradizionali.

13 Riferimenti

Il codice sorgente completo, comprendente il firmware dei dispositivi, il sistema di configurazione, la web app e il modello AI embedded, è disponibile al seguente repository GitHub:

<https://github.com/AlessandroFerrante/IoT/tree/main/MailTonBox>

Riferimenti bibliografici

- [1] IoTBoard Library: Libreria per la gestione della board con ESP32. (2025).
<https://github.com/UniCT-Internet-of-Things/IoTBoard-Library>
- [2] Telegram Bot API. (2025). Documentazione Telegram Bot API .
<https://core.telegram.org/bots/api>
- [3] EloquentTinyML. (2025). Documentazione per EloquentTinyML.
<https://github.com/eloquentarduino/EloquentTinyML>
- [4] Libreria LoRa per ESP32.
<https://github.com/UniCT-Internet-of-Things/IoTBoard-Library/tree/main/lora>
- [5] DHT11 Humidity and Temperature Sensor Datasheet.
<https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>
- [6] HC-SR04 Ultrasonic Sensor Datasheet.
https://components101.com/sites/default/files/component_datasheet/HCSR04%20Datasheet.pdf

Autore: Alessandro Ferrante