DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE TELECOMUNICAZIONI

Tesi di Laurea Triennale

INGEGNERIA ELETTRONICA E TECNOLOGIE DELL'INFORMAZIONE

Febbraio 2026

Sistema di controllo e monitoraggio di cisterne d'acqua interconnesse

Candidato: Zahi Abou Chakra

Relatore: Prof. Riccardo Berta

Correlatori: Dr. Matteo Fresta
Dr. Luca Lazzaroni

Sommario

Il presente elaborato ha l'obiettivo di descrivere la progettazione e la realizzazione di un sistema embedded per il monitoraggio remoto e il controllo di un sistema di cisterne interconnesse. Il sistema acquisisce i dati da un sensore ad ultrasuoni, li elabora localmente mediante un microcontrollore e valuta lo stato di funzionamento. Le informazioni relative al livello dell'acqua e allo stato del sistema vengono successivamente trasmesse al framework Measurify tramite una Application Programming Interface (API), qualora sia disponibile una connessione ad Internet. Infine, vengono visualizzate con un'interfaccia grafica che garantisce un'interpretazione chiara e intuitiva.

Il progetto si inserisce nel contesto delle moderne applicazioni Internet of Things (IoT), che prevedono l'integrazione tra dispositivi fisici, sistemi di comunicazione e interfacce grafiche. In particolare, il sistema proposto risponde all'esigenza di monitorare in modo efficiente le risorse idriche sia in ambito domestico sia in quello industriale, sfruttando le tecnologie dei sistemi embedded e la connettività remota. [1][2]

Indice

1. Introduzione

Il presente progetto consiste nella realizzazione di un sistema in grado di automatizzare il funzionamento di diverse cisterne d'acqua interconnesse, fornendo all'utente informazioni relative allo stato operativo e ai livelli d'acqua delle cisterne.

L'idea alla base del progetto nasce da una problematica riscontrata in ambito domestico, legata alla discontinuità e alla bassa pressione dell'erogazione idrica comunale, insufficienti a garantire il riempimento di un serbatoio situato al terzo piano dell'edificio. Per ovviare a tale criticità, è stato installato un contenitore di accumulo al piano terra, alimentato lentamente da una sorgente naturale. Una volta raggiunto il livello massimo, l'acqua viene trasferita al serbatoio superiore mediante una pompa elettrica.

Attualmente, l'intero processo richiede un intervento manuale sia per il monitoraggio dei livelli sia per l'attivazione della pompa.

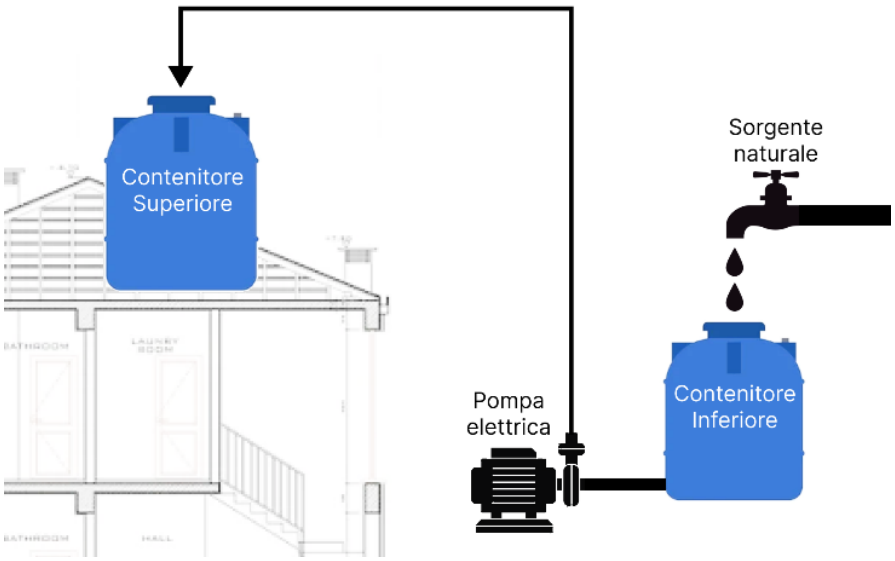

Figura 1: Configurazione delle cisterne.

Con la grande diffusione dei sistemi IoT (Internet of Things) e la loro evoluzione, che li ha resi sempre più accessibili, è stato realizzato un sistema basato su queste tecnologie al fine di automatizzare l'intero processo e fornire all'utente i dati sul proprio dispositivo mobile.

Il sistema è stato progettato per operare all'interno di una configurazione idrica costituita da una cisterna dedicata all'alimentazione idrica di una struttura edilizia come un edificio o un palazzo, e da una cisterna destinata all'accumulo dell'acqua.

2. Metodi e Strumenti utilizzati

Il seguente schema presenta i componenti che formano il sistema complessivo e i collegamenti tra di loro:

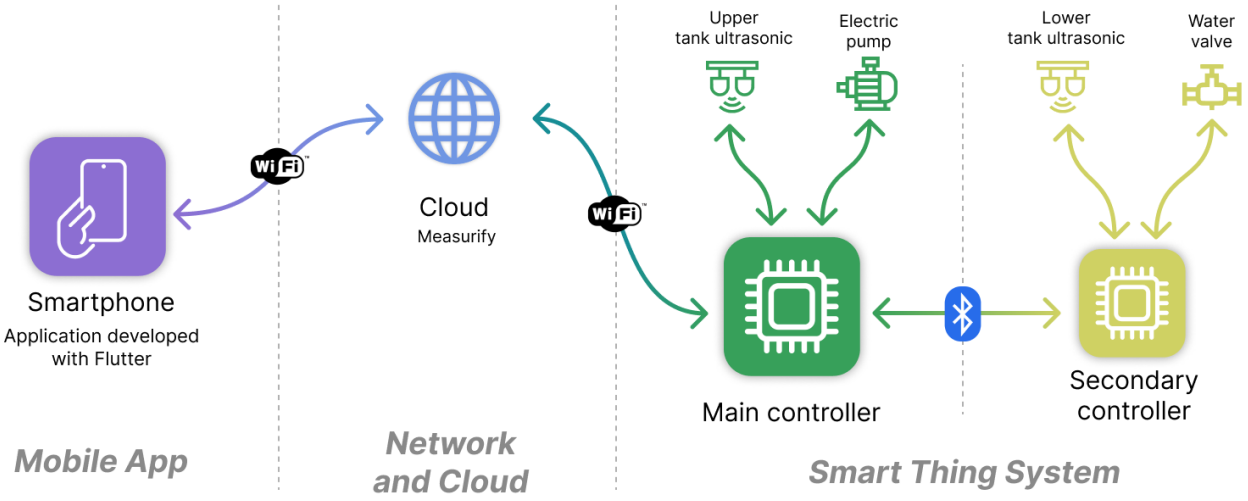

Figura 2: Schema del sistema.

È stato suddiviso in tre sotto-parti: Smart Thing System, Network and Cloud e Mobile App.

2.1. Smart Thing System

Lo Smart Thing System rappresenta i dispositivi fisici, cioè la configurazione dei microcontrollori, sensori, pompa elettrica, elettrovalvola e le tecniche di comunicazione usate tra di loro.

Per suddividere questo sottosistema in piccoli sotto-problemi, è stato adottato il concetto leader-follower. Ogni cisterna è dotata di un microcontrollore in grado di misurare il livello d'acqua, mandare le misure al resto del sistema e controllare una pompa o una valvola. In questo caso un leader è una cisterna che è dotata di un microcontrollore principale (Main Controller) e un follower rappresenta una cisterna con un microcontrollore secondario (Secondary Controller), che è anche la cisterna di accumulo.

L'unica differenza tra un controllore principale e uno secondario (cioè un leader e un follower) è che quello principale, o il leader, non espone il suo livello all'altra cisterna e invece lo manda al framework Measurify, che, in questo caso, è un database raggiungibile tramite Internet. Invece, la cisterna secondaria (follower) manda le misure alla cisterna principale, e quella principale condivide tutte le misure con il framework.

Per poter scegliere i microcontrollori adatti all'applicazione, sono stati studiati i propri ruoli. Il microcontrollore principale, per esempio, deve:

1. Controllare la pompa, ovvero deve prendere decisioni cambiando lo stato di un pin (ON/OFF).
2. Calcolare il dato del sensore, che vuol dire manipolare un insieme di pin per fare le misure.
3. Ricevere i dati dalla cisterna secondaria connettendosi ad essa.
4. Mandare tutti i dati al cloud, quindi deve essere in grado di connettersi ad una rete WiFi.

Nella configurazione del problema iniziale domestico, l'area di copertura del WiFi include la cisterna principale, ma non copre la posizione della cisterna secondaria. Per quello è stato necessario trovare una tecnica di comunicazione alternativa. È stata scelta la tecnologia **BLE (Bluetooth Low Energy)** come mostrato in figura 2**,** che, come il nome suggerisce, è di basso consumo di energia e offre un'area di copertura maggiore rispetto a quella del WiFi. Il BLE è capace di stabilire una connessione per una distanza di circa 100 metri in linea di vista, che si riduce a 10-30 metri in presenza di ostacoli. E nel caso si usassero trasmettitori ad alta potenza, l'area di copertura può arrivare oltre ad un kilometro. [3]

I controllori secondari devono avere le stesse specifiche tranne quella del WiFi. A questo punto si procede alla scelta del microcontrollore.

Esistono diversi microcontrollori che soddisfano le specifiche del progetto, è stato scelto l'**ESP32-WROOM-32** da Espressif Systems, riportato in figura a destra.

Figura 3: Modulo ESP32-WROOM-32.

Questo modulo è stato scelto per la sua accessibilità e il suo basso costo. Inoltre, offre un MCU (Microcontroller Unit, l'ESP32) abilitato sia di WiFi sia di BLE, ed è capace di effettuare calcoli e prendere decisioni logiche semplici. [4]

L'ESP32-WROOM-32 è usato sia nella cisterna principale sia in quella secondaria, ed è stato programmato con l'Arduino IDE, spiegato successivamente.

Per misurare il livello d'acqua dentro alla cisterna si usa un sensore ad ultrasuoni (figura 4) che viene operato tramite due pin: il Trigger, che manda un'onda sonora ad alta frequenza, e l'Echo che riceve la riflessione dell'onda sulla superficie dell'acqua. Il microcontrollore, a questo punto, misura il tempo impiegato dall'onda per viaggiare e rientrare nel sensore, calcolando la distanza fra l'acqua e il sensore. (moltiplicando il tempo impiegato per la velocità del suono e dividendo per due…) [5]

Figura 4: Sensore ad ultrasuoni (HC-SR04)

La pompa elettrica invece, viene operata tramite un pin digitale, connesso direttamente ad un relè elettromeccanico, che consente di controllore una corrente ad alta potenza con un segnale a bassa tensione (accende o spegne la tensione di 220 volt con quella proveniente dal microcontrollore di 5 volt). Per poter effettuare i test in laboratorio, la pompa elettrica ad alta potenza è stata sostituita con una piccola di 5 volt.

L'elettrovalvola, quella che controlla il flusso d'acqua alle cisterne di accumulo, è controllata in modo simile ed è solo usata nelle cisterne secondarie.

Figura 5: Micro-pompa elettrica (3-5 volt)

Dopo aver fatto le scelte del hardware, passiamo allo sviluppo software. Per ottenere i risultati desiderati dal progetto, la metodologia usata per lo sviluppo del programma dei microcontrollori è fondamentale.

Per il controllore principale, il codice sviluppato con Arduino IDE è sequenziale ed è diviso in un file principale chiamato mainController.ino, e altri header files (.h) per mantenere una struttura modulare:

1. config.h:

Questo header file definisce tutti i parametri del sistema che vengono usati nel processo di controllo (#define…). Sono definiti:

- I numeri dei pin del microcontrollore usati per la pompa e il sensore (TRIG e ECHO)
- La misura del sensore quando la cisterna è piena (distanza tra il sensore e la superficie d'acqua).
- La misura del sensore quando la cisterna è vuota (distanza tra il sensore e il fondo della cisterna).
- Il numero di misure da fare (usato per ottenere una misura precisa, spiegato dopo…)
- Il livello minimo di acqua nella cisterna secondaria per accendere la pompa,
- Il livello minimo accettabile d'acqua nella cisterna principale.
- Il tempo fra i controlli del funzionamento corretto del sistema.
- Il tempo fra gli aggiornamenti del cloud.
- SENSOR_TOLERANCE: usato per offrire flessibilità nel controllo.

Questi valori sono definiti per una specifica configurazione (taglia delle cisterne, tipo di microcontrollore, sensore…) e non vengono modificati nell'esecuzione del programma. Inoltre, offrono flessibilità nel caso viene cambiata la configurazione del sistema.

2. globalVariables.h:

Insieme a globalVariables.cpp, raccoglie tutte le variabili necessarie al corretto funzionamento del programma. Ad esempio, le variabili usate per il conteggio del tempo trascorso, per memorizzare un livello d'acqua precedente (usato per il confronto con quello attuale), per memorizzare lo stato della cisterna, per memorizzare gli errori (error flags). Inoltre, sono definiti e memorizzati i vari identificativi del BLE e WiFi (UUIDs, SSID, Password…).

In questo file, sono anche descritti gli stati della cisterna: WAITING: attesa della cisterna secondaria prima di attivare la pompa. FILLING: pompa attiva. BLOCKING_ERROR: quando c'è un errore critico. Questi stati sono usati solo all'interno del programma e non vengono trasmesse al cloud.

3. control.h:

Con il relativo control.cpp, definiscono le funzioni di basso livello:

- `get_level_sup()`: calcola il livello d'acqua nella propria cisterna. Manda un segnale TRIG al sensore e calcola il tempo impiegato dall'onda per riflettere sulla superficie dell'acqua e rientrare nel sensore. Usando questo dato, viene calcolata la distanza tra il sensore e la superficie d'acqua, poi la percentuale. Infine, il valore viene memorizzato e usato da altre funzioni.
- `start_pump() e stop_pump()`: usate per attivare o disattivare la pompa, cambiando il valore del pin digitale relativo.
- `check_pumping()`: verifica il corretto funzionamento della pompa confrontando il livello dell'acqua precedente con quello corrente. Nel caso non si riscontri un incremento del livello, viene segnalata una condizione di errore alla funzione chiamante (return -1).

Le funzioni definite nel presente header file, operanti a questo livello di controllo, non prevedono la modifica degli error flags o dello stato del sistema, che sono controllati a livelli superiori del programma.

4. logic.h

insieme a logic.cpp, definisce le funzioni di livello più alto rispetto a quelle in control.h. Le funzioni definite sono quelle in grado di prendere decisioni logiche e cambiare lo stato del sistema:

- `handle_upper_tank()`: questa funzione è alla base dell'operazione della pompa (oppure della valvola, per una cisterna secondaria). In questa fase, vengono verificati i livelli d'acqua sia nella cisterna secondaria sia in quella principale, sulla base dei quali viene determinata l'attivazione o la disattivazione della pompa.
  Viene disattivata qualora il livello d'acqua nella cisterna secondaria non risulti sufficiente secondo i parametri definiti in config.h, per evitare il sovraccarico della pompa. Inoltre, viene disattivata nel caso la cisterna principale risulti completamente riempita.
- `handle_error()`: verifica il corretto funzionamento di tutti i componenti del sistema. In particolare, controlla la misura corrente del livello d'acqua e segnala un errore qualora la funzione di basso livello (`get_level_sup()`) restituisca -1. Successivamente, viene verificata l'eventuale presenza di errori di comunicazione BLE, e in tal caso, viene avviata la procedura di riconnessione.
- `log()`: fatta per gli sviluppatori per visualizzare i cambiamenti e gli errori del sistema nel Serial Monitor.

5. connectivity.h

Dichiara le funzioni responsabili della gestione della parte di comunicazione del sistema. Ad esempio, la comunicazione tramite BLE viene gestita in questa sezione, dove vengono definite le principali funzioni necessarie al suo corretto funzionamento (notify, onConnect, onDisconnect…). Inoltre, è definita la funzione `search_and_connect(…)` usata nello stack del BLE nelle fasi di connessione iniziali per la ricerca e la connessione alla cisterna secondaria. Tale funzione viene anche chiamata in caso di errore BLE, per ristabilire la comunicazione.

In questo file, viene anche gestita la comunicazione col cloud tramite WiFi. La procedura iniziale della connessione ad un Access Point viene gestita una sola volta nella funzione `setup()` presente nel file mainController.ino. In caso di errore WiFi, l'ESP32 gestisce la riconnessione automaticamente. Le funzioni responsabili dell'aggiornamento dei dati su Measurify sono le seguenti:

- **`update_cloud()`**: Measurify è una piattaforma di tipo RESTful che usa JSON per rappresentare e scambiare i dati. In questa fase, viene creato un oggetto JSON che contiene i seguenti dati: data e ora della misura, percentuale dell'acqua nella cisterna e flag bits che contengono dati relativi allo stato del sistema. La struttura dell'oggetto JSON è descritta in dettaglio nella sezione 2.2.

  A questo punto, viene preparato l'header del protocollo HTTP nel quale sono specificati il tipo di messaggio (in questo caso application/json) e le informazioni di autorizzazione, cioè il token.

  Questa procedura viene ripetuta all'interno della stessa funzione anche per i dati relativi alla cisterna secondaria.

- **`WiFi_error_handler(…)`**: questa funzione viene usata dallo stack WiFi per determinare il processo relativo agli eventi relativi alla connessione WiFi (connessione non riuscita, connessione

interrotta...). In particolare, in questa fase vengono acquisite la data e l'ora correnti tramite una richiesta all'URL "pool.ntp.org", qualora sia disponibile una connessione WiFi.

6. mainController.ino:

mainController è il file principale del programma dove vengono eseguite le funzioni `setup()` e `loop()`.

In seguito, sono riportate le operazioni di ciascuna funzione, in ordine di esecuzione:

`setup()`:

- preparazione della comunicazione seriale.
- Indicazione del ruolo dei pin.
- Misurazione iniziale del livello dell'acqua, per assicurare la correttezza dei valori.
- Procedura di ricerca e connessione della cisterna secondaria tramite BLE.
- Preparazione della connessione WiFi, usando SSID e password specificati in globalVariables.h.

Questa funzione viene eseguita solo all'accensione del microcontrollore. Nel caso di anomalia, viene segnalato l'errore negli error flags.

La funzione `loop()` viene eseguita ciclicamente dopo l'esecuzione di `setup()`:

`loop()`:

- Misurazione del livello dell'acqua, per garantire l'aggiornamento del valore ad ogni ciclo di esecuzione.
- Controllo degli errori tramite `handle_error()`, responsabile della gestione delle condizioni di errore e dell'aggiornamento dello stato del sistema.
- Controllo della pompa tramite `handle_upper_tank()`, responsabile della logica di controllo della cisterna principale.
- Aggiornamento dei dati su Measurify con `update_cloud()`.
- Per gli sviluppatori viene alla fine eseguita `log()`, che indica lo stato del sistema tramite la porta seriale.

2.2. Network and Cloud

Il cloud costituisce l'infrastruttura usata per poter immagazzinare e manipolare i dati ricevuti dal Smart Thing System.

Come piattaforma, è stato usato **Measurify** basato su REST, che permette di salvare e manipolare dati provenienti da sensori in formato JSON, tramite chiamate API. [8]

Con HTTP, il microcontrollore manda i propri dati e quelli ricevuti tramite BLE, alla piattaforma Measurify, dove vengono salvati e visualizzati in grafici e numeri.

Measurify definisce una struttura a quattro livelli:

- Thing: definisce una “cosa” che può essere un oggetto o una persona su cui si fanno le misure. Nel nostro caso è la cisterna.
- Feature: La misurazione stessa, cioè il livello d’acqua.
- Device: il dispositivo che effettua le misure, cioè l’ESP32-WROOM-32
- Measurement: la misura effettuata dal dispositivo su una determinata Thing.

Per questo progetto, è stato fornito un “tenant” di Measurify, dove si possono memorizzare i dati, insieme a due Things: “CisternaA” e “CisternaB”, e una Feature per ciascuna chiamata “Livello_acqua”. Inoltre, è stato fornito un token per ogni cisterna, per riuscire a fare chiamate POST e GET con HTTP.

A questo punto, il microcontrollore principale, connesso a WiFi, è capace di fare chiamate HTTP (POST) per salvare le misure e gli stati di ogni cisterna su Measurify.

Nel sistema di controllo, la funzione `update_cloud()` presente nel file connectivity.cpp è responsabile della preparazione dell’oggetto JSON che verrà spedito a Measurify.

L’oggetto ha il formato seguente:

```
{
    "timestamp": "1769295600004",
    "values": [84, 8]
}
```

dove “timestamp” rappresenta la data e l’ora secondo l’epoch UNIX in millisecondi e “values” rappresenta due “Features”: il primo è Livello_acqua che è una percentuale del livello d’acqua, e il secondo rappresenta lo stato e gli errori del sistema (tipo di errore, attesa, riempimento…), codificato in formato binario come segue:

$0_{10} = 0000_2$: Nessun errore. Il sistema è in attesa o la cisterna è piena.

$1_{10} = 0001_2$: Errore del sensore (livello d’acqua fuori scala).

$2_{10} = 0010_2$: Errore della pompa (indicato da `check_pumping()`).

$4_{10} = 0100_2$: Errore del BLE (connessione alla cisterna secondaria fallita).

$8_{10} = 1000_2$: Pompa attiva.

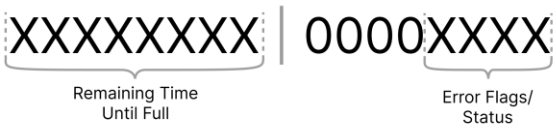

Figura 6: Rappresentazione del secondo "Feature"

Inoltre, è stato aggiunto nel secondo byte meno significativo, il tempo rimasto per riempire la cisterna (se calcolabile) (nella figura 6 a sinistra), sempre in formato binario, che consente di memorizzare un tempo massimo di 255 minuti (8 bits).

Nelle fasi di sviluppo, è stata utilizzata la piattaforma Postman che permette allo sviluppatore di mandare chiamate HTTP al server. In altre parole, rende possibile modificare, cancellare e aggiungere dati al tenant di Measurify, accedendo col username e password, che è stato rilevante nella fase di test.

2.3. Mobile App

L'utente deve riuscire a visualizzare i livelli nelle cisterne e lo stato del sistema dal suo dispositivo mobile. Per questo, è stata sviluppata un'interfaccia utente (UI) con Flutter (nel linguaggio Dart) supportata dai sistemi operativi più diffusi (Android, IOS, MacOS e Windows). [9][10]

Figura 7: Icona dell'app.

Per rendere la lettura dei dati efficace all'utente, l'applicazione, chiamata Water Monitor, è stata studiata a livello di design di interfaccia usando Figma (strumento di prototipazione), seguendo i principi di design adottati nelle applicazioni moderne.

L'applicazione presenta tre pagine accessibili tramite una barra di navigazione posizionata nella parte inferiore dell'interfaccia:

2.3.1. Home Screen

È la pagina iniziale che contiene una finestra (widget) per ciascuna cisterna. All'interno della finestra è riportata la percentuale d'acqua, lo stato della cisterna, eventuali commenti, e a destra, una grafica di una cisterna che mostra il livello di riempimento:

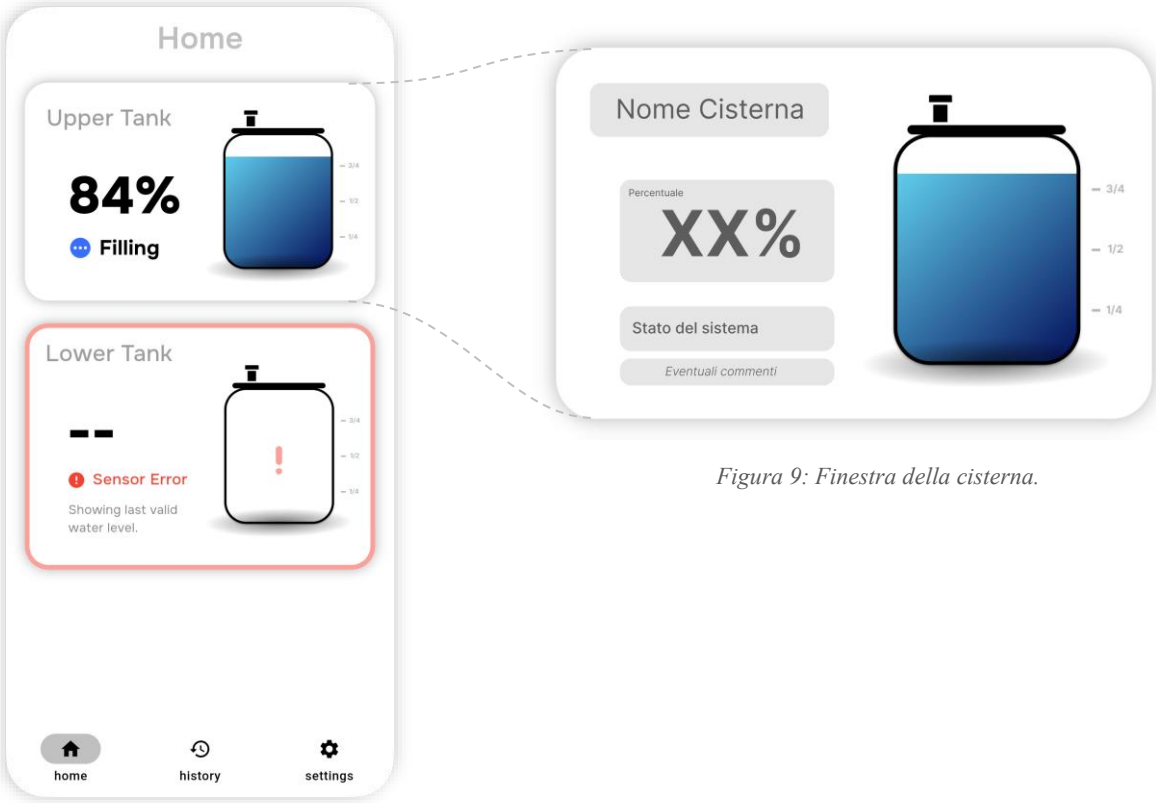

Figura 8: Home Screen

Figura 9: Finestra della cisterna.

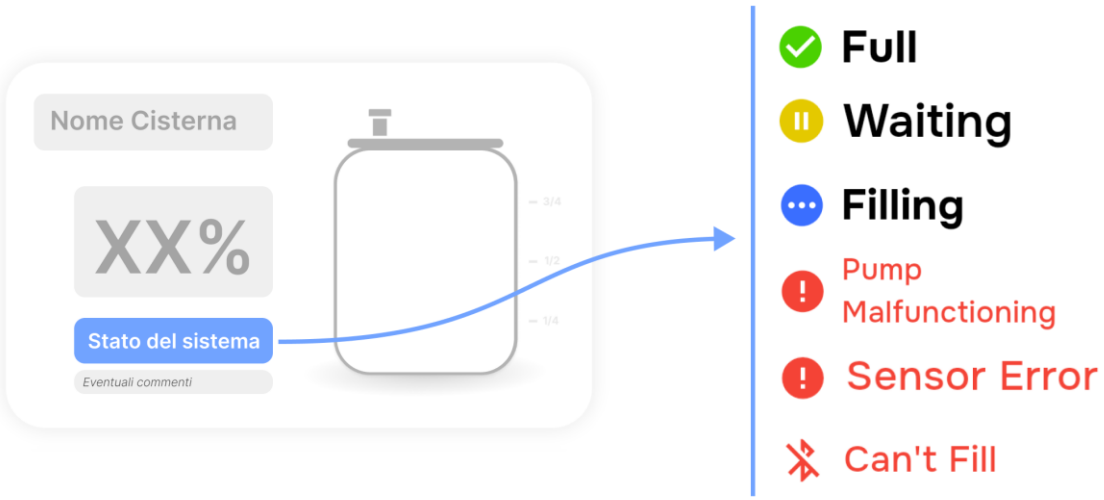

Figura 10: Possibili stati della cisterna.

Nella figura 10 sono mostrati i possibili output dello stato del sistema:

- **Full**: cisterna piena, pompa non attiva.
- **Waiting**: sistema in attesa. Il livello d'acqua nella cisterna secondaria non è sufficiente per attivare la pompa (attesa del riempimento di quella cisterna).
- **Filling**: pompa attiva / riempimento della cisterna.
- **Pump Malfunctioning**: errore di pompa. La pompa è stata attivata mentre il livello d'acqua è rimasto invariato.
- **Sensor Error**: i valori misurati dal sensore sono fuori scala.
- **Can't Fill**: la connessione alla cisterna secondaria è persa. Il livello d'acqua in quella cisterna non è più disponibile, e la pompa non può essere attivata.

Per una cisterna secondaria, le possibili uscite sono "**Full", "Sensor Error"**, **"Can't Accumulate"** e **"Filling"**, tra cui "**Can't Accumulate**" indica un problema con la sorgente naturale (la sorgente naturale è attiva mentre il livello d'acqua rimane invariato).

Nella sezione dei commenti, qualora lo stato sia "**Sensor Error**", viene visualizzato il commento "***Showing last valid water level.***" (mostrato in figura 8) per indicare che l'applicazione mostra l'ultima misura valida del livello d'acqua. Per lo stato "Filling", viene invece mostrato il tempo residuo per il riempimento della cisterna, qualora il dato sia disponibile. ("***xx minutes until full.***")

2.3.2. History

Nella presente pagina sono visualizzati grafici che rappresentano la variazione nel tempo del livello d'acqua. È stata adottata la stessa struttura delle finestre (widgets) che contengono il grafico, lo stato della cisterna, la percentuale del livello d'acqua e la data dell'ultima acquisizione del dato:

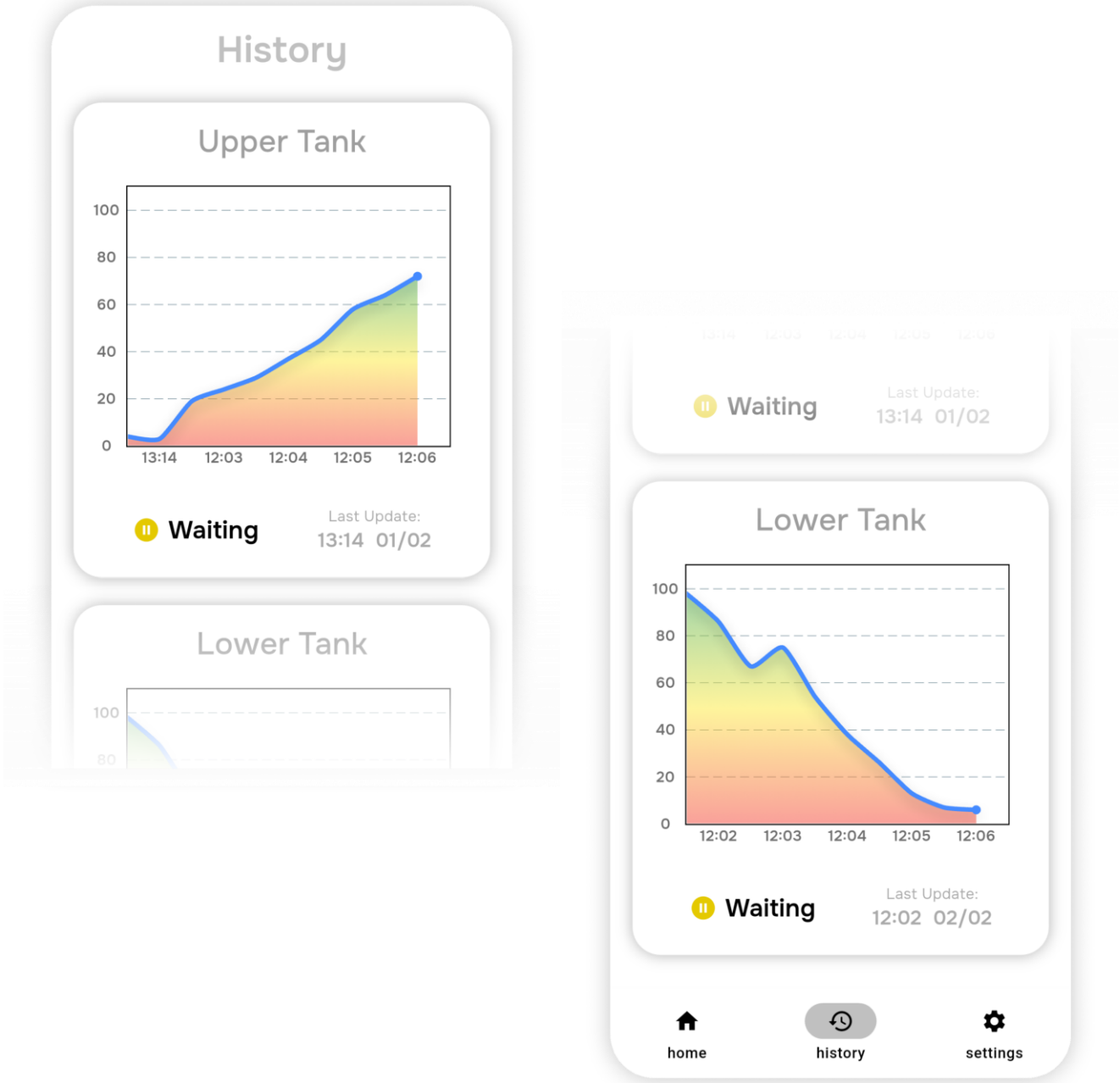

Figura 11: Pagina History

Il grafico relativo alla cisterna riporta sull'asse delle ordinate la percentuale del livello d'acqua, e sull'asse delle ascisse è indicato l'orario dell'acquisizione. Quest'ultimo è determinato solo dal momento in cui è stata aggiornata l'informazione sul cloud, e non coincide necessariamente con l'ora di recupero del dato dall'applicazione. Questa scelta progettuale viene utile nel caso di errori (soprattutto errori WiFi) poiché consente di individuare l'ultima data in cui il sistema ha funzionato correttamente.

La parte inferiore della finestra riporta lo stato del sistema e la data e l'ora dell'ultimo aggiornamento da parte della cisterna.

2.3.3. Settings

Rappresenta la pagina delle impostazioni, che viene utile nel caso il progetto venga esteso.

Possibili impostazioni:

- Controllo del sistema dal dispositivo mobile (indicare orari di lavoro, fermare immediatamente il sistema, reset del sistema…)
- Modalità della visualizzazione dei dati (app theme, colori, sfondo…)
- Impostazioni avanzate, che permettono allo sviluppatore di cambiare gli indirizzi http e token ed indicare nuove dimensioni delle cisterne, senza dover riprogettare l'intera applicazione.

3. Sperimentazione e Risultati

Per garantire il corretto funzionamento del sistema, è stato valutato il comportamento dei diversi componenti che lo costituiscono.

La cisterna è stata simulata mediante una vasca trasparente, sulla quale vengono fissati il microcontrollore e il sensore. Invece per riprodurre il funzionamento della pompa, è stata impiegata una micro-pompa da 5 volt (comunemente utilizzata negli acquari domestici).

I microcontrollori e la micro-pompa sono stati alimentati tramite adattatori 5V (USB).

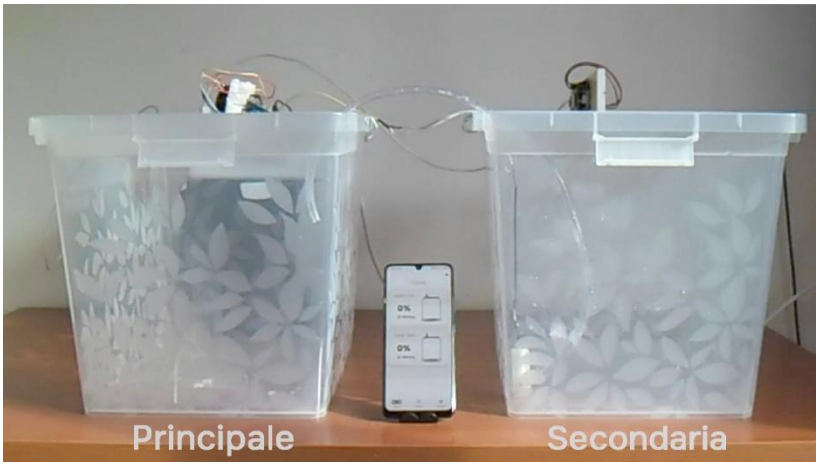

Figura 12: Simulazione delle cisterne per testare il funzionamento.

3.1. Problematiche e Miglioramenti:

Come illustrato nel capitolo 2, l'Arduino IDE è stato adottato come l'ambiente di sviluppo per la realizzazione del codice del programma di controllo. Inoltre, la funzione `log()`, insieme alle altre funzioni, mette a disposizione tutte le operazioni del microcontrollore tramite la porta seriale, mediante il Serial Monitor integrato nell'ambiente di sviluppo. Questo approccio ha permesso di monitorare il funzionamento del programma e di individuare errori di logica nel codice.

Per la configurazione del sistema, le misurazioni acquisite rappresentano la distanza tra il sensore e la superficie dell'acqua. Quindi, per ricavare da questa informazione la percentuale d'acqua nella cisterna, è necessario sapere le dimensioni della cisterna. Successivamente, viene calcolata la percentuale come segue:

$$\frac{SUP\_SENSOR\_HI - distance}{SUP\_SENSOR\_HI \ - SUP\_SENSOR\_LO} \times 100$$

Dove SUP_SENSOR_HI rappresenta la misurazione della cisterna vuota (oppure la distanza tra il sensore e il fondo della cisterna, in centimetri), SUP_SENSOR_LO la misurazione della cisterna piena (cioè la distanza tra la superficie d'acqua e il sensore, quando la

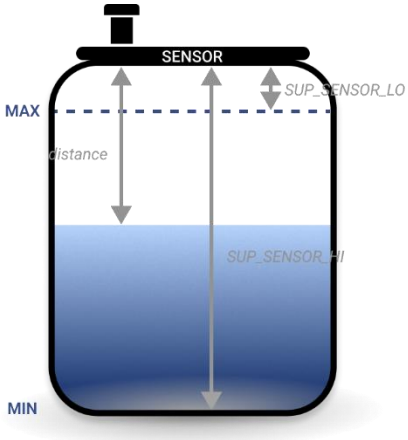

Figura 13: Misurazione nella cisterna.

cisterna è piena, sempre in centimetri) e $distance$ la misurazione corrente del sensore (distanza tra sensore e superficie dell'acqua).

Il primo problema riscontrato nelle fasi di test riguarda le misurazioni effettuate dal sensore ultrasonico. I valori ottenuti dal calcolo della distanza tra il sensore e la superficie dell'acqua risultavano instabili con la presenza di picchi e risultati non coerenti con il livello d'acqua reale. Tale comportamento è causato dai disturbi presenti sulla superficie del liquido, in particolare dalle onde generate durante il riempimento o lo svuotamento della cisterna.

Per migliorare la precisione delle misure, è stata sviluppata una nuova versione del codice che consente al sensore di effettuare misure consecutive ogni 100 millisecondi (figura 14), riducendo così la differenza tra una misura e l'altra e migliorando l'accuratezza del risultato.

Il numero di misure da effettuare è dato dal parametro `MEASURE_NUM`, definito nel file config.h che deve essere dimensionato tenendo conto del tempo impiegato per effettuare una misura completa (che è $MEASURE\_NUM \times (100ms + tempo\ di\ viaggio\ del\ suono) + tempo\ di\ calcolo$). Il tempo dell'acquisizione di una misura dipende ovviamente dal livello dell'acqua corrente, poiché la variabile $tempo\ di\ viaggio\ del\ suono$ nell'equazione precedente, aumenta con la distanza tra il sensore e la superficie dell'acqua.

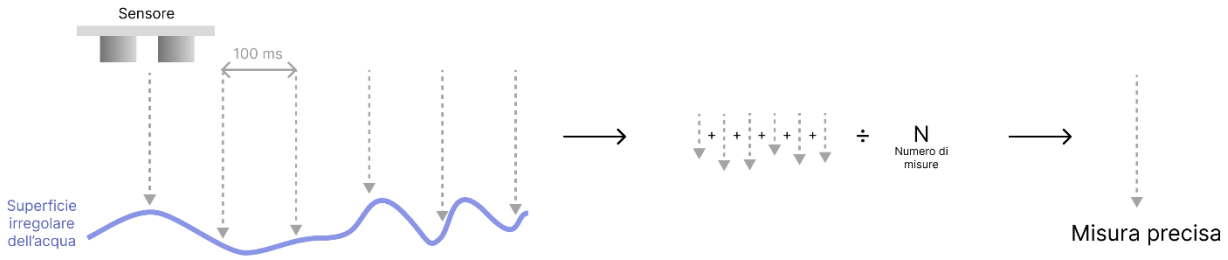

Figura 14: Calcolo della media tra le diverse misurazioni.

Un ulteriore problema riscontrato riguarda la precisione della misura fornita dal sensore. In particolare, quando la cisterna risulta completamente piena, il sensore può rilevare valori che appaiono superiori al 100% a causa delle irregolarità presenti sulla superficie dell'acqua. In questa condizione, il livello veniva interpretato dalla funzione `handle_error()` come una misura fuori scala, segnalando un errore del sensore. Per risolvere questa problematica, è stata introdotta una costante chiamata `SENSOR_TOLERANCE` che permette di definire un margine di tolleranza nella misura. Tale margine viene usato nella fase di disattivazione della pompa (quando la cisterna risulta piena), evitando segnalazioni di errori non significativi.

`SENSOR_TOLERANCE` è definito in config.h ed è una percentuale (viene sottratto dal livello massimo).

L'applicazione Water Monitor sviluppata con Flutter visualizza i dati in modo efficiente e comprensibile. Uno svantaggio riscontrato nelle fasi di test era la necessità di aggiornare le pagine manualmente. Per questo

è stata aggiunta l'istruzione `Timer.periodic()` nella funzione responsabile dell'acquisizione dei dati da Measurify, rendendo il processo di aggiornamento automatico, ad intervalli di due secondi.

3.2. Funzionamento del sistema

L'esperimento è iniziato con la simulazione di due cisterne vuote, come in figura 12. All'accensione, il microcontrollore principale (a sinistra), dotato di un sensore e relé per il controllo della micro-pompa, verifica innanzitutto la validità dei valori acquisiti dal sensore. Successivamente, procede con lo stabilimento della connessione BLE con il microcontrollore secondario (a destra).

Una volta stabilita la comunicazione tra i due dispositivi, il sistema entra nella fase di connessione ad un Access Point (WiFi). Sul dispositivo di monitoraggio nel centro vengono quindi visualizzati i livelli iniziali delle due vasche (entrambi 0%).

A questo punto, inizia il riempimento della vasca secondaria da una sorgente esterna, e si visualizza l'aumento del livello sul dispositivo mobile. Quando il livello raggiunge la soglia minima definita dal parametro `MIN_INF_TO_PUMP` (nel file config.h), la pompa viene attivata e inizia il riempimento della vasca principale.

Poiché la quantità d'acqua nella cisterna secondaria è limitata, quando il livello arriva quasi a 0% la pompa viene disattivata per evitare il suo funzionamento a secco, e di conseguenza il suo danneggiamento.

Quando la vasca secondaria viene riempita nuovamente, e il livello d'acqua diventa maggiore di `MIN_INF_TO_PUMP`, la pompa si riattiva trasferendo l'acqua alla vasca principale. Durante questa fase è stata aggiunta ulteriore acqua alla vasca secondaria in modo da mantenere la pompa attiva. È stato osservato che il microcontrollore principale disattiva la pompa non appena la vasca principale raggiunge il 100%.

Sono stati anche osservati i diversi stati di errore. Ad esempio, sollevando il sensore per simulare una lettura non valida (cisterna sovra-vuota), il sistema segnala un errore di sensore, visualizzato sul dispositivo di monitoraggio. Similmente, disattivando il microcontrollore secondario, e interrompendo la connessione BLE, l'errore viene anche visualizzato e l'operazione del sistema viene sospesa.

Gli altri stati di errore sono stati gestiti e visualizzati in modo analogo, confermando il corretto comportamento del sistema.

3.3. Limiti di funzionamento

Per garantire il corretto funzionamento, sono stati analizzati i limiti operativi del sistema. In particolare, sono state individuate le condizioni entro le quali esso funziona correttamente.

3.3.1. Limiti hardware

Il microcontrollore ESP32-WROOM-32, come qualsiasi componente elettronico, presenta limiti sulla corrente in uscita. La corrente massima supportata (Absolute Maximum Current Rating) vale 1100 mA. I

componenti alimentati dal modulo sono: il sensore ultrasonico (15mA) e il relé (circa 2mA) che in totale sono circa 17mA che è molto minore della corrente massima in uscita al modulo. [4]

La massima corrente supportata dal relé, da parte della pompa (carico ad alta potenza), vale 10A a 250V. Considerando il caso domestico, le pompe elettriche più comuni presentano una potenza pari a circa 1500W, che significa 6A a 250V, minore della corrente massima sopportata dal relé. [6]

Il sensore ad ultrasuoni invece, presenta limiti legati alla distanza di misura. Nel datasheet [5] è indicato un intervallo operativo compreso tra 2cm e 4 metri. Durante la fase di sperimentazione, è stato osservato che la distanza massima effettivamente misurabile in contesto reale si riduce a 2,5 a 3 metri. Considerando che le cisterne comunemente impiegate in ambito domestico presentano un'altezza massima di circa 2 metri, il sensore usato risulta comunque idoneo all'applicazione. Nel caso di cisterne di altezza superiore, è necessario ricorrere a moduli caratterizzati da un range di misura maggiore.

Poiché il sistema microcontrollore-sensore è installato all'interno della cisterna, è necessario garantire l'impermeabilità all'acqua, in quanto sono previsti spruzzi e condizioni di elevata umidità durante la fase di riempimento. Una possibile soluzione consiste nell'alloggiamento del sistema all'interno di una custodia stagna con grado di protezione almeno IP64 (Ingress Protection 64). Inoltre, la custodia usata deve prevedere aperture dedicate agli emettitori e ricevitori del sensore ad ultrasuoni, oltre a quelle dedicate all'alimentazione elettrica. [7]

Figura 15: Custodia Stagna per Elettronica.

3.3.2. Limiti Software

Durante la fase di sviluppo del programma di controllo, è stata analizzata la memoria occupata dallo sketch (codice in Arduino IDE). È emerso che il programma utilizza circa il 98% della memoria Flash disponibile dell'ESP32-WROOM-32 (che fornisce 4MB in totale). Il sistema, quindi, risulta in prossimità del limite di memoria. Di conseguenza, l'aggiunta di altre funzionalità porta al superamento della capacità del modulo, che rende necessario valutare microcontrollori alternativi dotati di una maggiore capacità di memoria.

Nel software di controllo sono stati trattati i diversi casi di errore. Tuttavia, il sistema adottato non consente il rilevamento diretto di guasti di natura hardware. Ad esempio, nel caso la pompa non entri in funzione, il sistema non è in grado di identificare immediatamente il malfunzionamento. Tale condizione viene rilevata indirettamente tramite la funzione `check_pumping()`, che verifica l'aumento del livello dell'acqua, con un certo ritardo. La stessa limitazione potrebbe essere riscontrata per altre tipologie di guasto (malfunzionamento del sensore o microcontrollore…).

Oltre alle limitazioni già citate, l'applicazione mobile non prevede la rappresentazione dei dati in formati alternativi. In presenza di un errore (hardware o software), l'applicazione Water Monitor non è in grado di verificare la validità dei dati acquisiti o visualizzarli. La gestione dei casi di errori è prevista esclusivamente a livello del microcontrollore.

Il Bluetooth Low Energy, come tutte le tecnologie di comunicazione wireless, prevede un range di copertura limitato. Tale limite dipende da diverse caratteristiche dell'ambiente nel quale viene usato. Nella

configurazione domestica considerata, la distanza tra la cisterna principale e quella secondaria vale circa 15 metri in linea di vista, mentre il range approssimato della comunicazione BLE in linea di vista e in presenza di ostacoli, raggiunge valori fino a 30 metri. Tuttavia, la connessione BLE non dipende esclusivamente dalla distanza fra i dispositivi, ma è influenzata anche da altri fattori ambientali e strutturali. Pertanto, non è possibile trarre conclusioni definitive senza effettuare test sperimentali in loco. [3]

4. Contributo Personale e Conclusione

L'idea alla base del presente progetto nasce dall'esigenza di realizzare un sistema embedded per il monitoraggio e la gestione automatizzata di cisterne per l'accumulo idrico in ambito domestico. L'obiettivo principale è stato quello di sviluppare una soluzione affidabile e scalabile, in grado di facilitare l'acquisizione delle informazioni sul livello dell'acqua e di rendere il controllo del sistema completamente autonomo e continuo.

Il mio contributo personale si è concentrato sulla progettazione e sull'implementazione dell'architettura software e hardware del sistema. In particolare, mi sono occupato della scelta dei componenti, dello sviluppo del firmware dei microcontrollori e della comunicazione tra i dispositivi, nonché della realizzazione di un'applicazione mobile sviluppata tramite il framework Flutter. Tale applicazione consente di visualizzare in tempo reale lo stato delle cisterne, monitorare i livelli di riempimento e disporre le operazioni autonome del sistema.

Durante la preparazione e la realizzazione del progetto sono state affrontate diverse criticità, tra cui la progettazione di un codice di controllo affidabile, la gestione delle comunicazioni wireless e l'apprendimento di nuovi strumenti e linguaggi di sviluppo, come il linguaggio Dart per la programmazione dell'applicazione mobile. Per ottenere risultati coerenti con gli obiettivi prefissati, è stato necessario passare attraverso un'attenta fase di sperimentazione e validazione del funzionamento.

Nel corso dello sviluppo ho osservato che esiste sempre un margine di miglioramento per un sistema. Qualsiasi progetto ingegneristico, non limitato al caso di studio affrontato, è intrinsecamente soggetto ad un processo di evoluzione, supportato dall'introduzione di nuove idee e tecnologie che fanno incrementare l'affidabilità e le prestazioni. Questa considerazione è in linea con quanto appreso durante il mio percorso accademico, secondo cui un sistema non può essere considerato perfetto, ma è il risultato di un compromesso tra requisiti, vincoli progettuali e soluzioni tecniche adottate.

5. Riferimenti Bibliografici

[1] Smart Water Management Market Size 2025-2029: https://www.technavio.com/report/smart-water-management-market-analysis

[2] Global Water Detection Sensors Market Size and Forecast – 2025- 2032: https://www.coherentmarketinsights.com/industry-reports/water-detection-sensors-market

[3] Bluetooth Low Energy: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy

[4] ESP32-Wroom-32 Datasheet: https://documentation.espressif.com/esp32-wroom-32_datasheet_en.pdf

[5] HC-SR04 – Sensore ad Ultrasuoni:
https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[6] Relé: https://arduinomodules.info/ky-019-5v-relay-module/

[7] IP Ratings – International Electrotechnical Commission: https://www.iec.ch/ip-ratings

[8] Measurify – Elios Lab: https://www.elios.unige.it/it

[9] Corso di Flutter su Code Academy: https://www.codecademy.com/learn/intro-to-flutter

[10] Video Introduzione a Flutter – Flutter Mapp:
https://youtu.be/3kaGC_DrUnw?si=PShjmNGqaFkNzGHg

Altri riferimenti:

- Libreria Bluetooth Low Energy: NimBLE-Arduino: https://docs.arduino.cc/libraries/nimble-arduino/
- Libreria WiFi: https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFi.h
- Postman: https://www.postman.com/
- Figma: https://www.figma.com/