ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

# SMART CITY E IOT: UNA SOLUZIONE BASATA SU LORA PER IL TRACCIAMENTO DI OGGETTI MOBILI

Tesi di laurea in Laboratorio di Applicazioni Mobili T

Relatore

Prof. Angelo Trotta

Presentata da

Massimo Giaccone

Correlatore

Prof. Federico Montori

Sessione dicembre 2024

Anno Accademico 2023/2024

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

# SMART CITY E IOT: UNA SOLUZIONE BASATA SU LORA PER IL TRACCIAMENTO DI OGGETTI MOBILI

Tesi di laurea Laboratorio di Applicazioni Mobili T

Relatore

Prof. Angelo Trotta

Presentata da

Massimo Giaccone

Correlatore

Prof. Federico Montori

---

Sessione dicembre 2024

Anno Accademico 2023/2024

*Ricordami,*

*Ora devo andare via,*

*Ripensa a me,*

*Sentendo questa melodia,*

*Uniremo con le note il cuore e le anime,*

*Il tuo amore rimarrà*

*Sempre per me*

---

Coco - Pixar, 2017

# Sommario

Questo lavoro si propone di sviluppare un sistema di tracciamento basato sulla tecnologia LoRa, utilizzato per monitorare e seguire eventi pubblici di grande portata. Dopo aver presentato le specifiche della tecnologia LoRa e introdotto il protocollo LoRaWAN, verranno presentati i dettagli architetturali e implementativi del sistema, dai microcontrollori agli algoritmi di individuazione del posizionamento ottimale. Infine, le prestazioni del sistema sono valutate mediante il confronto dei due algoritmi (Brute Force e Greedy), supportati da un'applicazione per la visualizzazione in tempo reale. I risultati dimostrano l'efficienza del sistema proposto, evidenziando il compromesso tra precisione e rapidità computazionale.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# 1 Introduzione

## 1.1 Contesto della ricerca

Negli ultimi anni, nel contesto delle città intelligenti, l'Internet of Things (IoT) ha rivoluzionato numerosi settori, introducendo soluzioni innovative per la raccolta, l'elaborazione e la trasmissione dei dati in tempo reale. Grazie alla capacità di connettere dispositivi intelligenti e automatizzare processi, l'IoT si è affermato come una tecnologia abilitante per applicazioni in qualsiasi ambito quotidiano.

Un elemento chiave nell'ecosistema IoT è la capacità di comunicazione a lungo raggio, particolarmente rilevante per applicazioni distribuite che richiedono il monitoraggio continuo di oggetti mobili su ampie aree geografiche. A questo scopo, la selezione di una tecnologia di comunicazione adeguata è fondamentale per garantire affidabilità, scalabilità ed efficienza. Tra le tecnologie emergenti in questo campo, LoRa (Long Range) e il protocollo LoRaWAN si sono distinte per le loro caratteristiche uniche: ampia copertura, basso consumo energetico e costi relativamente contenuti.

Queste tecnologie si rivelano particolarmente efficaci in scenari in cui l'infrastruttura tradizionale non è facilmente implementabile o economicamente sostenibile, come ambienti urbani densi o applicazioni su larga scala con numerosi nodi distribuiti.

Inoltre, la loro capacità di funzionare efficacemente con dispositivi alimentati a batteria per periodi prolungati senza interventi di manutenzione aggiuntivi, li rende particolarmente adatti per contesti operativi impegnativi.

## 1.2 Obiettivi tesi

Questa tesi si pone l'obiettivo di progettare e sviluppare un sistema di tracciamento capace di monitorare eventi pubblici dinamici. Gli obiettivi specifici includono:

- **Progettazione del sistema**: sviluppo di un'architettura modulare che comprenda dispositivi trasmittenti, riceventi e la logica per la raccolta ed elaborazione dei dati;
- **Ottimizzazione del posizionamento**: implementazione e confronto di algoritmi per massimizzare la copertura del percorso dell'evento;

- **Interazione utente**: creazione di un'applicazione che consenta di visualizzare in tempo reale la posizione degli oggetti monitorati;

## 1.3 Struttura del documento

La tesi è strutturata in modo da fornire una visione completa e articolata del lavoro svolto, partendo dalle basi teoriche fino ad arrivare all'implementazione pratica e all'analisi dei risultati.

Nel Capitolo 2 viene introdotta la tecnologia LoRa e il protocollo LoRaWAN, con una panoramica dettagliata delle loro principali caratteristiche, vantaggi e limitazioni. Questa parte offre una comprensione del contesto tecnologico, evidenziando la loro applicabilità per i sistemi IoT a lungo raggio.

Il Capitolo 3 descrive l'architettura del sistema proposto, spiegate le configurazioni dei dispositivi trasmittenti e riceventi, con un'attenzione particolare alla modularità del sistema e alla sua capacità di adattarsi a contesti applicativi differenti.

Nel Capitolo 4 viene analizzata in dettaglio l'architettura del sistema, includendo i particolari implementativi. Vengono descritti i dispositivi utilizzati, il modulo di deploy preliminare, la selezione degli algoritmi di ottimizzazione e lo sviluppo dell'applicazione finale.

Nel Capitolo 5 vengono approfonditi gli algoritmi di ottimizzazione Brute Force e Greedy utilizzati per il posizionamento dei dispositivi riceventi, evidenziandone i dettagli implementativi, le differenze concettuali e i rispettivi vantaggi e svantaggi. Successivamente, si analizzano le prestazioni del sistema attraverso i risultati degli esperimenti condotti, corredati da un confronto grafico tra le soluzioni proposte dai due algoritmi. Infine, viene presentata l'applicazione mobile sviluppata, descrivendone le principali schermate.

Infine, nel Capitolo 6 vengono tratte le conclusioni e si discutono gli sviluppi futuri. In questa parte si analizzano i risultati ottenuti, valutando le prestazioni complessive del sistema e identificando le principali limitazioni. Vengono inoltre proposte possibili direzioni per il miglioramento e l'estensione del lavoro, sia dal punto di vista tecnologico che applicativo.

## 1.4 La Pasqua di Comiso

Tra i numerosi eventi che si prestano a un simile approccio tecnologico, ho scelto la tradizionale processione pasquale di Comiso. Questo evento è uno dei momenti più sentiti della città iblea. La tradizione della "Paci" è una delle poche in Sicilia che

**Figura 1.1.** Uscita dei simulacri dalla basilica

combina elementi di origine bizantina con influenze catalane, retaggio della dominazione aragonese. Questo mix culturale ha reso la Pasqua di Comiso un evento unico, capace di attrarre visitatori da tutta l'isola e oltre.

Sin dal Medioevo, la città ha sviluppato un profondo culto mariano, consolidato nei secoli attraverso celebrazioni che hanno trovato il loro centro spirituale nella Basilica dell'Annunziata. Questa devozione alla Madonna Annunziata, insieme alla tradizione pasquale, dove avviene l'incontro tra i simulacri di Gesù Risorto e della Madonna, è profondamente radicata nella cultura religiosa e sociale di Comiso.

La processione si svolge lungo tutta la giornata della domenica di Pasqua e rappresenta un momento esemplificativo della devozione della comunità. Il rito dell'incontro tra i simulacri, accompagnato dal canto dei bambini vestiti da "angioletti", evoca un forte simbolismo di pace e resurrezione.

La processione non è solo un evento religioso, ma anche un momento di grande partecipazione comunitaria. Generazioni di fedeli hanno contribuito a mantenere viva questa tradizione, che continua a essere un simbolo di identità culturale e spirituale per gli abitanti di Comiso. La devozione alla Madonna Annunziata si manifesta non solo attraverso la processione, ma anche con le celebrazioni liturgiche e le confraternite che si sono sviluppate nei secoli per sostenere e tramandare queste tradizioni(4).

Il lungo percorso della processione attraversa le vie principali di Comiso, fermandosi davanti ad ogni chiesa e in luoghi simbolici della città. La logistica dell'evento richiede un notevole coordinamento, con la collaborazione delle autorità locali, poichè muove la

partecipazione di migliaia di persone.

Questo evento rappresenta un'occasione ideale per implementare un sistema di monitoraggio tecnologico. Attraverso dispositivi basati su tecnologie come LoRa, è possibile tracciare in tempo reale il movimento dei simulacri e visualizzare la loro posizione su una mappa interattiva. Questa innovazione non solo arricchisce l'esperienza dei partecipanti, ma consente una gestione più efficiente del flusso della processione, migliorando la sicurezza e la comunicazione con i fedeli.

# 2
# Letteratura

## 2.1 LoRa

La tecnologia **LoRa** (acronimo di **Lo**ng **Ra**nge) è un sistema di comunicazione wireless a lungo raggio che utilizza una modulazione a spettro espanso[1] chiamata Chirp Spread Spectrum (CSS). Questa utilizza un segnale sinusoidale (chirp), la cui frequenza varia in maniera lineare in relazione al tempo per trasmettere segnali su lunghe distanze. L'uso del chirp permette di operare nell'intera larghezza di banda, distribuendo il segnale su una vasta gamma di frequenze, garantendo robustezza contro il degrado del canale di comunicazione da interferenze, come ad esempio l'in-band jamming e l'effetto Doppler.

LoRa opera all'interno della banda **ISM** (Industrial, Scientific, Medical), cioè la porzione di spettro elettromagnetico riservato dall'ITU (International Telecommunication Union) alle applicazioni di radiocomunicazioni non commerciali, ma per uso industriale, scientifico e medico. In Europa, la banda ISM utilizzata per le comunicazioni LoRa è principalmente quella a 433 MHz e 868 MHz, che garantisce un buon compromesso tra portata della trasmissione e capacità di attraversamento del segnale.

### 2.1.1 Modulazione

A differenza di altre modulazioni come la Frequency Shift Keying (FSK), che utilizza frequenze discrete per rappresentare i bit, LoRa sfrutta un "chirp", che varia gradualmente in frequenza durante un periodo di tempo.

Lo **Spreading Factor** (SF) è un parametro fondamentale della modulazione LoRa. Esso rappresenta il numero di bit codificati in ciascun simbolo trasmesso, con

$$SF \in \{7, 8, 9, 10, 11, 12\}$$

dove valori più alti aumentano la distanza di comunicazione, ma di contro riducono il tasso di trasmissione dei dati (**Data Rate**, DR). Lo SF rappresenta quindi un compromesso tra la robustezza del segnale e la velocità di trasmissione dei dati: con un SF più alto, la sensibilità del ricevitore aumenta, ma aumenta anche il tempo di trasmissione del pacchetto, con conseguente maggiore consumo energetico.

[1] Tecnica di trasmissione: l'informazione viene trasmessa in un intervallo più grande del necessario, con l'obiettivo di migliorare il rapporto segnale/rumore, diminuendo notevolmente le interferenze.

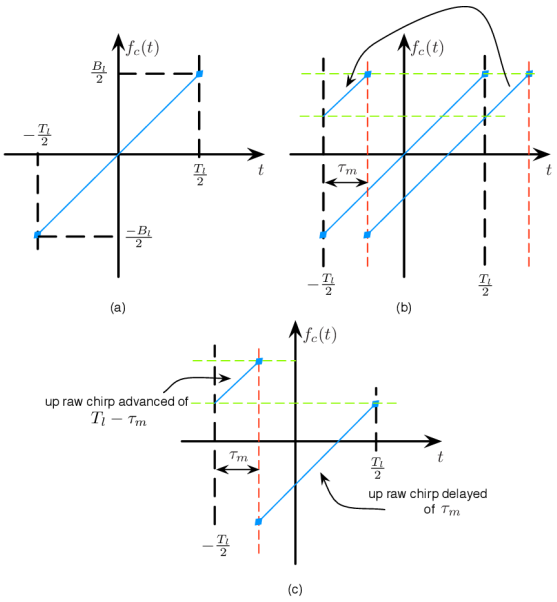


**Figura 2.1.** Rappresentazione temporale di un chirp in modulazione CSS: (a) Chirp grezzo, (b) Chirp ritardato di $\tau_m$, (c) Chirp avanzato di $T_s - \tau_m$ (1)

La relazione tra SF e il numero di chip per simbolo ($N$) è espressa dalla seguente formula:

$$SF = \log_2(N)$$

Pertanto, il numero di simboli possibili è dato da:

$$N = 2^{SF}$$

Successivamente, ad ogni simbolo $n$, dove $n \in \{0, ..., N-1\}$, viene associato un chirp. I chirp vengono definiti *up-chirp* quando la frequenza aumenta linearmente, e *down-chirp* quando la frequenza diminuisce. La larghezza di banda del chirp è pari alla larghezza di banda $B$ del segnale CSS. In LoRa, la larghezza di banda può assumere i seguenti valori:

$$B \in \{125\,\text{MHz}, 250\,\text{MHz}, 500\,\text{MHz}\}$$

Il chirp associato al simbolo $n$ viene ottenuto applicando un ritardo $\tau_n = \frac{n}{B}$ a un chirp grezzo, ovvero un chirp up o down su tutto il periodo $T_s$. Il chirp grezzo al di fuori dell'intervallo $[-T_s/2, T_s/2]$ viene riportato ciclicamente nell'intervallo $[-T_s/2, T_s/2+\tau_m]$, come mostrato nella Figura 2.1, dove

$$f_c(t) = \pm \left( \frac{B}{T_s} \right) \cdot t$$

Quindi, il chirp modulato relativo alla trasmissione del simbolo $n$ può essere scritto in due parti:

- Per $t \in \left[ -\frac{T_s}{2}, -\frac{T_s}{2} + \tau_m \right]$, la rampa (in salita o in discesa) del chirp grezzo anticipata nel tempo $T_s - \tau_m$;
- Per $t \in \left[ -\frac{T_s}{2} + \tau_m, -\frac{T_s}{2} \right]$, la rampa (in salita o in discesa) del chirp grezzo anticipata nel tempo $\tau_m$.

Grazie a questa divisione, si ottiene un segnale chirp che è continuo e ciclico, garantendo che non vi siano discontinuità che possano ridurre la qualità della trasmissione. La continuità del segnale è molto importante per la robustezza contro il rumore e le interferenze, rendendo la comunicazione LoRa efficace anche su lunghe distanze. Inoltre, questa struttura ciclica facilita la sincronizzazione tra trasmettitore e ricevitore, permettendo una decodifica più efficiente e una maggiore sensibilità del segnale. Complessivamente, questa tecnica assicura un compromesso ottimale tra la distanza di trasmissione e il consumo energetico, garantendo affidabilità e stabilità nella comunicazione.

In aggiunta, il livello fisico di LoRa utilizza il **Codice di Hamming** come schema di correzione degli errori, inserendo dei bit di parità che garantiscono che la differenza tra i vari messaggi sia abbastanza grande da poter correggere gli errori. In questo modo vengono prodotti dei messaggi ridondanti, ma sicuri e affidabili.

### 2.1.2 Frame LoRa

L'azienda Semtech, membro fondatore di LoRa Alliance, ha specificato un formato fisico del frame che viene utilizzato nei trasmettitori e ricevitori LoRa.

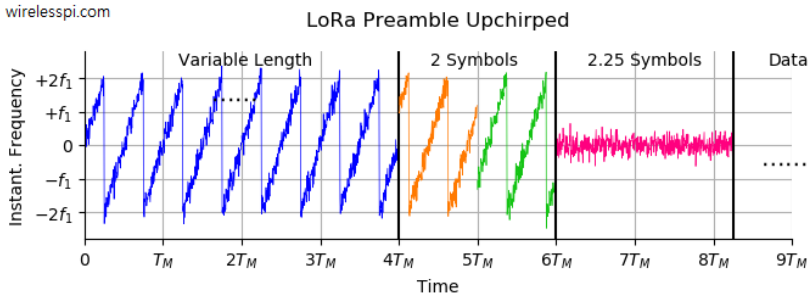


**Figura 2.2.** Esempio di preambolo con up-chirped (2)

Un frame inizia con un preambolo: in Figura 2.2, una sequenza di otto chirp in salita e costanti che coprono l'intera banda di frequenza. Gli ultimi due chirp sono quelli che codificano la **sync word**, un byte utilizzato per distinguere le reti LoRa che utilizzano la stessa banda di frequenza. Dopo l'ottetto di up-chirp seguono due più un quarto di down-chirp, per una durata di 2,25 simboli, che permette la sincronizzazione della trasmissione nel tempo. La durata totale del preambolo può essere configurata tra 10,25 e 65539,25 simboli. A seguire c'è la trasmissione dei simboli che identificano il payload, e quindi i dati da trasmettere.

Dopo il preambolo, c'è un header opzionale che, se presente, viene trasmesso tramite un code rate di 4/8, che definisce la dimensione del dato (in byte). L'header include anche un tasso di codifica usato per la trasmissione e, se presente, un **CRC** (Cyclic Redundancy Check) di 16 bit per il payload, tecnica di controllo per verificare l'integrità dei dati.

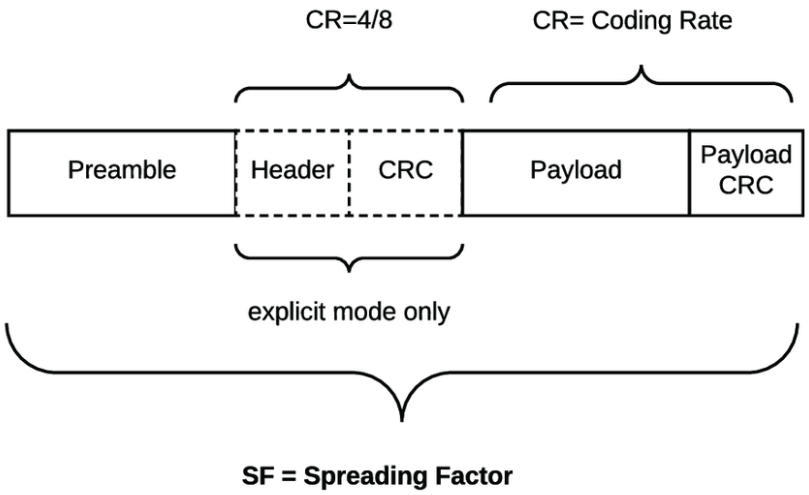


**Figura 2.3.** Struttura del frame LoRa (3)

### 2.1.3 Parametri utili per la comunicazione

La tecnologia LoRa offre quindi una soluzione versatile per la comunicazione a lungo raggio e a basso consumo energetico. La comprensione dei seguenti parametri è necessaria per comprendere la modulazione che utilizza LoRa e per ottimizzare le prestazioni di rete, adattandosi alle diverse condizioni di comunicazione.[2]

[2]LoRa for the Internet of Things(5)

- **RSSI (Received Signal Strength Indicator)**
  Il **RSSI** è un indicatore della potenza del segnale ricevuto, misurato in **milliwatt** (mW) ma comunemente espresso in **dBm**. Questo valore fornisce una misura della qualità del segnale che il ricevitore sta ricevendo, infatti è direttamente proporzionale alla qualità di comunicazione. Valori tipici di RSSI in LoRa variano da -30 dBm (segnale forte) a -120 dBm (segnale molto debole);

- **TP (Trasmission Power)**
  Il **TP** o **potenza di trasmissione** è la potenza con cui il trasmettitore invia il segnale. Un valore più elevato di TP aumenta la portata della comunicazione, ma comporta anche un maggiore consumo di energia. Il controllo della potenza di trasmissione è importante per ottimizzare l'efficienza energetica nei dispositivi LoRa, che spesso operano con alimentazione a batteria;

- **CF (Carrier Frequency)**
  La **Carrier Frequency (CF)** è la frequenza centrale attorno alla quale il segnale LoRa viene modulato. La frequenza dipende dalla banda di operazione scelta e varia a seconda della regione geografica. Ovviamente, la scelta della frequenza influisce direttamente sulla portata del segnale e sulla suscettibilità alle interferenze;

- **SNR (Signal to Noise Ratio)**
  Il **Signal to Noise Ratio (SNR)** è il rapporto tra la potenza del segnale ricevuto e il livello di rumore di fondo.

## 2.2 LoRaWAN

**LoRaWAN** (Long Range Wide Area Network) è un protocollo di comunicazione LPWAN (Low Power Wide Area Networking) progettato per la trasmissione di dati tra dispositivi dell'**Internet of Things**[3]. Basato sulla tecnologia di modulazione LoRa, LoRaWAN opera nello strato MAC (Media Access Control) dello stack OSI e fornisce una rete di ampia copertura, scalabile, adatta per le applicazioni in cui la durata della batteria di un dispositivo e le lunghe distanze rappresentano una criticità. Questo protocollo definisce le modalità di accesso al canale e la gestione della comunicazione tra i dispositivi IoT e la rete. La sua principale caratteristica è la gestione di un sistema di accesso al canale che permette a più dispositivi di condividere lo stesso mezzo di comunicazione senza interferire tra loro.

[3]Che cos'è LoRaWAN, AWS

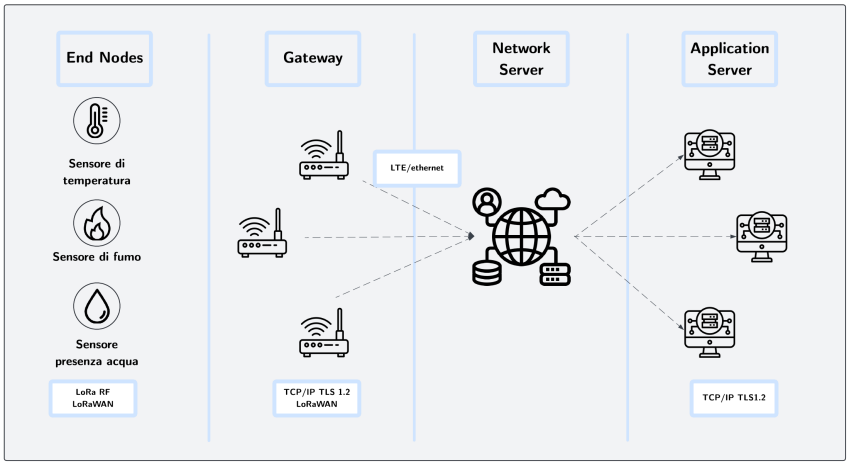


**Figura 2.4.** Architettura LoRaWAN

### 2.2.1 Architettura della rete LoRaWAN

La Figura 2.4 rappresenta l'architettura che sta alla base di tutte le applicazioni IoT che utilizzano questo protocollo. In una rete LoRaWAN, la comunicazione avviene tra diversi elementi che interagiscono per consentire la corretta esecuzione del sistema.

I dispositivi finali **end nodes** sono gli elementi principali in cui avviene il monitoraggio. Questi sono tipicamente localizzati in posizioni remote e hanno il compito di raccogliere i dati sull'ambiente circostante. Per questo motivo, sono progettati per funzionare in autonomo e con un basso consumo energetico.

Il **gateway** LoRaWAN è il dispositivo che riceve le comunicazioni dai punti finali e le inoltra all'infrastruttura di supporto di rete, cioé la parte della rete che collega la rete LoRa ad altri sistemi. Questa può essere costituita da una rete Ethernet, cellulare o qualsiasi altro tipo di connessione, sia cablata che wireless. I gateway LoRa sono connessi al server di rete LoRa tramite connessioni IP standard, il che significa che i dati viaggiano usando protocolli comuni e possono essere integrati in qualsiasi rete di telecomunicazioni, pubblica o privata.

Il **server di rete** coordina l'intera rete. Il suo ruolo è cruciale per garantire che la rete sia efficiente e che i dati siano instradati correttamente dal gateway ai dispositivi finali.

Infine, l'**application server** in una rete LoRaWAN è la parte del sistema che riceve i dati elaborati dal network server, li analizza o li utilizza per eseguire azioni specifiche. È un componente fondamentale per garantire che i dati raccolti dai dispositi-

vi IoT possano essere utilizzati per scopi concreti e interagire con gli utenti o altri sistemi.

La topologia di una rete LoRaWAN segue quindi il modello "stars-of-stars", dove i dispositivi finali comunicano con i gateway, che a loro volta inoltrano i dati al server di rete centrale. La comunicazione con i dispositivi finali è generalmente bi-direzionale, ossia i dispositivi possono inviare dati al server, ma anche ricevere comandi o risposte da esso. Inoltre, è possibile supportare operazioni **multicast**, che sono utili per inviare aggiornamenti software o altri messaggi di massa a più dispositivi contemporaneamente.

# 3
# Architettura

## 3.1 Scenario applicativo

I grandi eventi pubblici, come parate, cortei e manifestazioni, coinvolgono un'enorme mole di partecipanti che si muovono attraverso le zone urbane lungo percorsi stabiliti. In questi contesti, la necessità di seguire e monitorare il flusso di queste persone in tempo reale è fondamentale per una migliore esperienza dell'evento.

La capacità di tracciare oggetti mobili in contesti urbani rientra nelle applicazioni moderne delle **smart-city**, dove l'utilizzo e l'integrazione di sensori, reti di comunicazione e complessi sistemi di elaborazione consente una gestione più efficace delle risorse.

Il monitoraggio di oggetti mobili in un contesto di smart-city sfrutta delle tecnologie innovative per recuperare, trasmettere e analizzare i dati in modo rapido ed efficace. Queste tecnologie trovano applicazioni in numerosi scenari, come la gestione del traffico, della logistica urbana e, appunto, il tracciamento di eventi dinamici.

L'obiettivo primario è garantire una visione costante e aggiornata in tempo reale del movimento di persone e/o oggetti, in maniera da ottimizzare il coordinamento urbano e migliorare la risposta a situazioni in continuo cambiamento.

## 3.2 Componenti del sistema

Per raggiungere questo obiettivo, come si vede in Figura 3.1, l'architettura si compone di diversi moduli interconnessi che gestiscono il flusso di dati dalla raccolta delle coordinate alla visualizzazione nell'applicazione.

- **Dispositivi di tracciamento**
  Gli oggetti mobili sono equipaggiati con dispositivi che consentono di rilevare le coordinate GPS e trasmetterle in tempo reale, garantendo un corretto funzionamento per l'intera durata dell'evento.

- **Ponti**
  I ponti hanno il compito di ricevere le coordinate GPS trasmesse dai dispositivi. Inoltre, sono collegati ad Internet, per permettere il trasferimento dei dati verso il server.

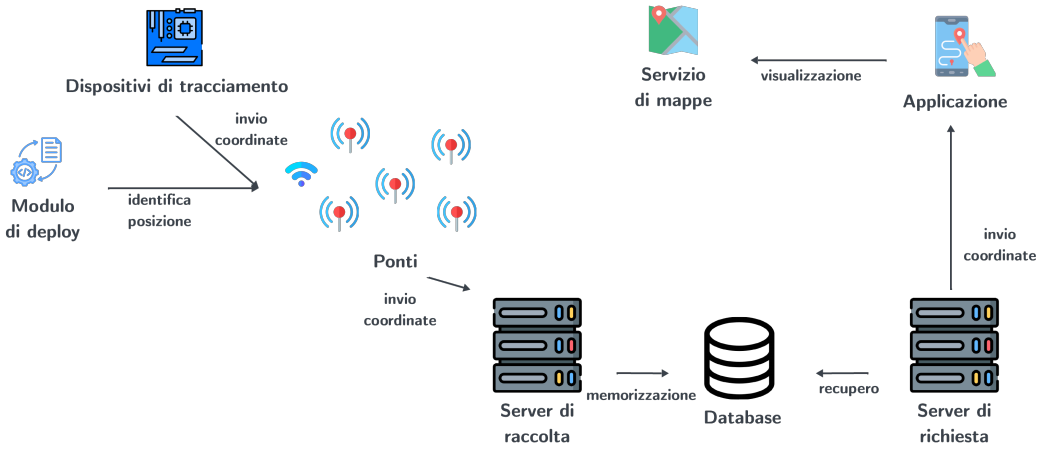


**Figura 3.1.** Architettura del sistema

- **Server di Raccolta**
  Server che riceve le coordinate GPS dai ponti, le processa e le memorizza in un database centralizzato. Questo è il primo responsabile della gestione dei dati, utilizzati per aggiornamenti in tempo reale.

- **Database**
  Il database memorizza tutte le coordinate GPS ed è fondamentale per la registrazione continua delle informazioni e per garantire che i dati siano sempre disponibili.

- **Server di Richiesta**
  Il server di richiesta rappresenta l'elemento che permette l'interazione tra applicazione e database. Ha il compito di recuperare i dati GPS dal database e inviarli all'applicazione, garantendo aggiornamenti in tempo reale.

- **Applicazione mobile**
  L'applicazione offre la possibilità di visualizzare la posizione dell'evento su una mappa in tempo reale. Gli utenti possono seguirlo, visualizzare il percorso e ricevere aggiornamenti su eventi significativi.

- **Servizio di mappe**
  Integrato nell'applicazione mobile, il servizio di mappe permette di graficare in una mappa la posizione dell'utente e dell'evento.

- **Modulo di deploy dei ponti**
  Test preliminari sono stati effettuati per individuare i punti migliori per il posizionamento, basandosi su parametri come altitudine e percentuale di ricezione.

# 4

# Implementazione

Per la realizzazione di un sistema di monitoraggio per oggetti che si muovono in contesti urbani si è scelto di utilizzare il protocollo LoRa, che opera a livello fisico per consentire la comunicazione a lungo raggio e a basso consumo energetico. Questa tecnologia è particolarmente adatta per scenari in cui è necessario trasmettere informazioni su distanze significative senza compromettere la durata della batteria dei dispositivi. Secondo quanto concesso dal **Piano nazionale di ripartizione delle frequenze (PNRF)**[1](6), la banda utilizzata per questo progetto è stata 433MHz.

Il sistema utilizza il protocollo LoRa tramite una serie di microcontrollori e sensori con la capacità di operare in maniera ottimale anche in condizioni ambientali critiche. L'architettura progettata garantisce che i dati raccolti dai sensori, in questo caso coordinate GPS, possano essere inviati in modo stabile e sicuro, supportando una gestione centralizzata delle informazioni.

Nelle sezioni seguenti verranno illustrati in dettaglio i componenti tecnici dell'integrazione del protocollo LoRa, con particolare attenzione alle specifiche dei microcontrollori e dei sensori impiegati. Verranno poi descritti: il codice sviluppato in Arduino per la gestione dei dispositivi, il modulo di deployment preliminare per la corretta disposizione dei dispositivi riceventi e, infine, l'applicazione finale che interagisce con l'utente.

L'obiettivo è presentare una soluzione robusta ed efficiente, fondamentale per applicazioni di tracciamento in tempo reale.

## 4.1 Microcontrollori e Sensori

In questo sistema, l'elemento principale è rappresentato da un' **ESP32**, microcontrollore molto utilizzato nelle applicazioni di Internet of Things, data la sua versatilità e le sue performance. L'ESP32 possiede una connessione Wi-Fi e Bluetooth, utile per l'interazioni con Internet, ma deve essere affiancato ad un modulo LoRa, per poter trasmettere o ricevere dati da lunghe distanze, non trascurando il consumo.

Il modulo LoRa scelto è un ricetrasmettitore **E32** a 433 MHz, prodotto dall'azienda EByte, a cui è stata collegata un'antenna SMA per migliorare la qualità e la stabilità

[1]Il PNRF, o Piano Nazionale di Ripartizione delle Frequenze, è il risultato della pianificazione dello spettro radio a livello nazionale, mirata a ottimizzare l'efficienza e garantire un utilizzo armonizzato delle risorse spettrali.

del segnale. Il modulo E32, collegato all'alimentazione fornita dall'ESP32, è in grado di operare sia come trasmettitore che come ricevitore. Per configurare le modalità di trasmissione, il modulo utilizza i pin M0 e M1. A seconda della loro configurazione, è possibile selezionare diverse modalità operative[2]:

- **Normale (M0 = 0, M1 = 0)**: In questa modalità, il modulo E32 può sia trasmettere che ricevere dati, funzionando in modalità di comunicazione trasparente;
- **Risparmio Energetico (M0 = 0, M1 = 1)**: Il modulo non trasmette, e la comunicazione seriale è disattivata. Tuttavia, se riceve un segnale specifico, noto come WOR (Wake Up on Radio), inviato da un trasmettitore, si risveglia per gestire la comunicazione;
- **Configurazione (M0 = 1, M1 = 1)**: Permette di configurare i parametri del modulo, come la frequenza e la velocità di trasmissione, mentre la ricezione e la trasmissione sono interrotte;
- **Wake-Up (M0 = 1, M1 = 0)**: Invia un segnale WOR (Wake Up on Radio) per risvegliare un modulo che si trova in modalità sleep.

Il modulo dispone anche di un pin **AUX**, che fornisce informazioni sul proprio stato operativo al microcontrollore. Altri pin importanti sono **TXD** (trasmissione) e **RXD** (ricezione), utilizzati per comunicare con l'ESP32.

Per rispettare le normative imposte dal PNRF(6) e ottimizzare l'efficienza energetica, il sistema utilizza un duty cycle del 10%, il che significa che il modulo LoRa trasmette solo per il 10% del tempo totale, seguito da un periodo di inattività. Inoltre, il modulo è configurato per operare alla potenza massima di trasmissione di 500 mW (ca 27 dBm), garantendo una comunicazione affidabile a lunga distanza.

Ultimo modulo necessario e fondamentale per il sistema è il **GPS NEO 6M**, un modulo GPS che fornisce le coordinate di localizzazione in tempo reale. Con un'accuratezza di circa 3 m e il basso consumo energetico, rappresenta la scelta ideale per questo progetto. Come il modulo LoRa, è alimentato dall'ESP32 e comunica con esso tramite i pin TX e RX per trasmettere i dati relativi alla posizione.

Di seguito verranno mostrati gli schemi elettrici dei dispositivi riceventi e trasmettitori. In Figura 4.1 è rappresentato lo schema elettronico del dispositivo che si occupa

[2]E32 LoRa Module Datasheet(7)

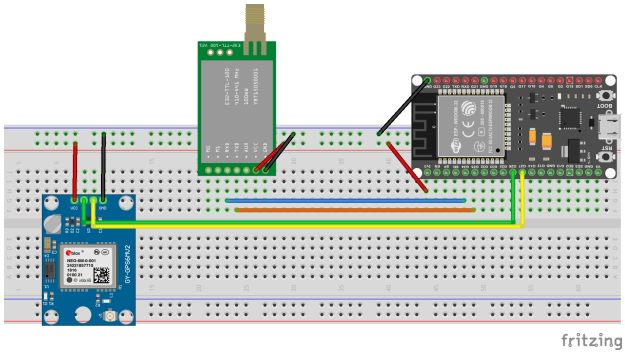


**Figura 4.1.** Schema elettronico Trasmettitore

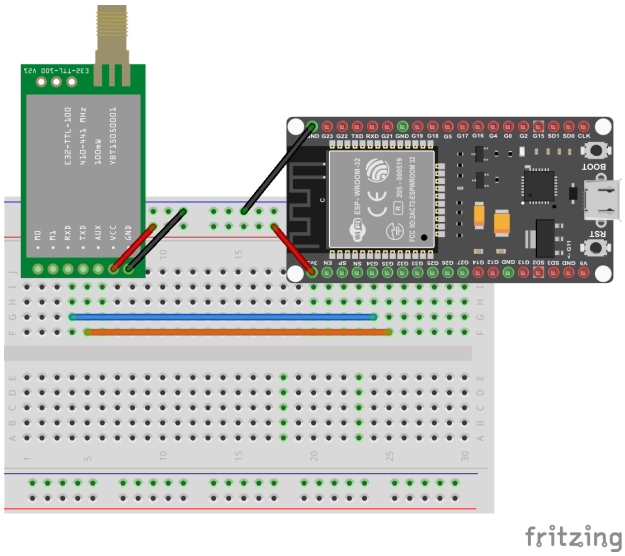


**Figura 4.2.** Schema elettronico Ricevitore

della trasmissione. L'ESP32 alimenta sia il modulo GPS NEO-6M che il modulo LoRa E32, fornendo una tensione di 3.3V a entrambi. I pin di massa (GND) di ciascun modulo sono connessi al GND dell'ESP32 per completare il circuito elettrico.

Il modulo GPS è collegato all'ESP32 attraverso il pin TX del GPS, che trasmette i dati delle coordinate al pin RX dell'ESP32, permettendo al microcontrollore di raccogliere informazioni di localizzazione.

Il modulo LoRa è connesso all'ESP32 tramite i pin TX e RX, che gestiscono rispettivamente l'invio e la ricezione dei dati tra i due componenti. I pin M0 e M1 del modulo LoRa sono collegati per configurare le modalità operative, mentre il pin AUX fornisce feedback sullo stato del modulo, segnalando all'ESP32 quando è pronto per la trasmissione o ricezione.

In Figura 4.2, lo schema elettronico del dispositivo ricevente è molto simile a quello

del trasmettitore. L'unica differenza significativa è l'assenza del modulo GPS. In questo caso, il modulo LoRa riceve i dati e li trasmette tramite comunicazione seriale all'ESP32, che si occupa di processarli.

### 4.1.1 Implementazione hardware: codice Arduino e configurazione

In questa sezione viene descritta l'implementazione hardware del sistema, con un focus sul codice Arduino sviluppato. L'obiettivo principale è garantire la corretta interazione tra il microcontrollore, il modulo GPS e il modulo LoRa, ottimizzando la trasmissione e la ricezione dei dati in tempo reale.

**Codice 4.1.** Codice Arduino per dispositivo trasmettitore

```
#include "Arduino.h"
#include "LoRa_E32.h"
#include "TinyGPSPlus.h"

// Definizione dei pin per LoRa e GPS
#define RX_PIN 34 // RX LoRa --> TX ESP32
#define TX_PIN 35 // TX LoRa --> RX ESP32
#define GPS_RX 26 // RX GPS --> TX ESP32
#define GPS_TX 27 // TX GPS --> RX ESP32

// Oggetto per il modulo LoRa
LoRa_E32 e32ttl(RX_PIN, TX_PIN);

// Oggetto per il modulo GPS
HardwareSerial GPS_Serial(1); // Usare la Serial1 per il GPS
TinyGPSPlus gps;

void setup() {
  Serial.begin(115200); // Serial per monitoraggio
  e32ttl.begin();     // Inizializzazione del modulo LoRa

  // Configura la porta seriale per il GPS
  GPS_Serial.begin(9600, SERIAL_8N1, GPS_RX, GPS_TX);
  delay(500);

  // Configura la potenza di trasmissione del modulo LoRa
  e32ttl.setPower(POWER_20); // Configura la potenza massima di trasmissione
      a 500 mW (20 dBm)
}
```

```
29
30 void loop() {
31   // Variabili per memorizzare le coordinate GPS
32   float latitude = 0.0;
33   float longitude = 0.0;
34   bool newData = false;
35
36   // Leggi i dati dal modulo GPS e aggiorna il parser
37   while (GPS_Serial.available()) {
38     if (gps.encode(GPS_Serial.read()))
39       newData = true;
40   }
41
42   if (newData && gps.location.isValid()) {
43     // Recupera le coordinate GPS
44     latitude = gps.location.lat();
45     longitude = gps.location.lng();
46
47     // Prepara il messaggio con le coordinate GPS
48     char msg[50];
49     snprintf(msg, sizeof(msg), "Lat:%.6f;Lon:%.6f", latitude, longitude);
50
51     // Invia il messaggio tramite LoRa E32
52     ResponseStatus rs = e32ttl.sendBroadcastFixedMessage(6, msg);
53
54     // Stampa il messaggio e lo stato della trasmissione
55     Serial.println(msg);
56     Serial.println(rs.getResponseDescription());
57   } else if (!gps.location.isValid()) {
58     Serial.println("Posizione GPS non valida.");
59   }
60
61   // Rispetta il duty cycle del 10%: 1 secondo di trasmissione -> 9 secondi
       di attesa
62   delay(9000); // Aspetta 9 secondi per rispettare il duty cycle
63 }
64
65 }
```

**Codice 4.2.** Codice Arduino per dispositivo ricevente

```
#include "Arduino.h"
#include "LoRa_E32.h"
#include <WiFi.h>
#include <HTTPClient.h>

// Definizione dei pin per LoRa
#define RX_PIN 34 // RX LoRa --> TX ESP32
#define TX_PIN 35 // TX LoRa --> RX ESP32

// Credenziali WiFi
const char* ssid = "nome-rete";
const char* password = "psw";
const char* serverName = "http://192.168.1.1:5000/esp32";

// Oggetto per il modulo LoRa
LoRa_E32 e32ttl100(RX_PIN, TX_PIN);

void setup() {
  Serial.begin(115200); // Serial per monitoraggio
  e32ttl100.begin();  // Inizializzazione del modulo LoRa
  delay(500);
  Serial.println("Ricevitore LoRa E32 pronto.");

  // Connessione WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  delay(2000);
}

void loop() {
  // Controlla se ci sono dati disponibili dal modulo LoRa
  if (e32ttl100.available() > 0) {

    // Ricevi il messaggio
    ResponseContainer rc = e32ttl100.receiveMessage();
    String data = rc.data;
```

```
41
42      // Se c'e' un errore, stampa la descrizione dell'errore
43      if (rc.status.code != 1) {
44        Serial.println(rc.status.getResponseDescription());
45      } else {
46        // Invia i dati al server se la connessione WiFi e' attiva
47        if (WiFi.status() == WL_CONNECTED) {
48          HTTPClient http;
49          http.begin(serverName);
50          http.addHeader("Content-Type", "text/plain");
51
52          // Invia il messaggio tramite HTTP POST
53          int httpResponseCode = http.POST(data);
54
55          if (httpResponseCode > 0) {
56            String response = http.getString();
57            Serial.println("HTTP Response code: " + String(httpResponseCode));
58            Serial.println("Server Response: " + response);
59          } else {
60            Serial.print("Errore nella richiesta HTTP: ");
61            Serial.println(httpResponseCode);
62          }
63          http.end(); // Libera risorse
64        } else {
65          Serial.println("Errore nella connessione WiFi");
66        }
67      }
68    }
69    delay(10000); // Aspetta dieci secondi prima di controllare di nuovo
70  }
```

## 4.2 Modulo deployment

In questa sezione verranno discusse le motivazioni che hanno portato alla scelta di specifiche posizioni come possibili "stazioni riceventi". Prima del deployment finale, sono stati condotti test in cui il trasmettitore inviava messaggi al ricevitore, posizionato ad un'altezza definita. In questi test, si è tenuto conto sia della distanza dal trasmettitore, sia del **PDR** (Packet Delivered Ratio), ovvero la percentuale di ricezione di pacchetti. Di seguito sono riportate le tabelle con i risultati ottenuti nei test preliminari, in cui il

ricevitore era posizionato prima ad altezza uomo, poi al quinto piano di un edificio, e infine all'ottavo piano.

| Distanza TX-RX | PDR |
|---|---|
| 0m | 100% |
| 100m | 99% |
| 200m | 95% |
| 300m | 92% |
| 350m | 90% |
| 500m | 88% |
| 700m | 70% |
| 800m | 40% |

**Tabella 4.1.** Risultati test svolti con ricevitore fermo ad'altezza uomo, meteo sereno.

| Distanza TX-RX | PDR |
|---|---|
| 0m | 100% |
| 100m | 100% |
| 200m | 99% |
| 300m | 80% |
| 350m | 75% |
| 450m | 63% |
| 600m | 62% |
| 750m | 51% |

**Tabella 4.2.** Risultati test svolti con ricevitore posto in un palazzo al quinto piano, meteo sereno, leggermente trafficato.

| Distanza TX-RX | PDR |
|---|---|
| 0m | 100% |
| 100m | 100% |
| 250m | 90% |
| 300m | 78% |
| 350m | 75% |
| 450m | 70% |
| 550m | 67% |
| 700m | 65% |

**Tabella 4.3.** Risultati test svolti con ricevitore posto in un palazzo all'ottavo piano, meteo piovoso con forte vento.

Con l'insieme di dati raccolti, l'obiettivo è stato quello di trovare un'approssimazione mediante una funzione che potesse calcolare la Percentuale di Pacchetti Ricevuti (PDR)

in relazione alla distanza tra i due dispositivi. Dopo un'attenta analisi, la funzione scelta come punto di partenza è stata la seguente:

$$PDR(x) = \frac{100}{1 + \left(\frac{x}{a}\right)^{2b}} \tag{4.1}$$

In questa espressione, diversi parametri giocano ruoli chiave nel modellare il comportamento della curva di decrescita del PDR rispetto alla distanza $x$:

- $a$: controlla il punto di massimo decadimento della curva: definisce la distanza a cui la ricezione dei pacchetti inizia a scendere rapidamente;
- $b$: controlla la forma della curva, ovvero quanto è ripida la discesa dopo aver superato la distanza critica;
- *Fattore* 100: è un moltiplicatore necessario per garantire che, quando la distanza $x$ è pari a zero (ossia, quando i dispositivi sono molto vicini), il PDR sia al 100%. Questo fattore assicura che la funzione sia calibrata in maniera da riflettere correttamente il comportamento ideale del sistema.

Successivamente, per determinare i valori ottimali dei parametri $a$ e $b$, sono stati sviluppati script in Python. Questi script hanno utilizzato un metodo di fitting in cui si fornivano i risultati dei test preliminari e la funzione, per determinare la migliore approssimazione dei due valori. Di seguito, saranno presentati i codici Python utilizzati per questa operazione, insieme alle tre diverse versioni delle funzioni ottenute grazie ai test e alle simulazioni.

**Codice 4.3.** Script Python per calcolare i valori di a e b

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Dati di distanza (in metri) e PDR (in percentuale) -- ALTEZZA UOMO --
distanze = np.array([0, 100, 200, 300, 350, 500, 700, 800])
pdr = np.array([100, 99, 95, 92, 90, 88, 70, 40])

# Dati di distanza (in metri) e PDR (in percentuale) ---- QUINTO PIANO ----
# distanze = np.array([0, 100, 200, 300, 350, 450, 600, 750])
# pdr = np.array([100, 100, 99, 80, 75, 63, 62, 51])

# Dati di distanza (in metri) e PDR (in percentuale) ---- OTTAVO PIANO ----
# distanze = np.array([0, 100, 250, 300, 350, 450, 550, 700])
```

```
# pdr = np.array([100, 100, 90, 78, 75, 70, 67, 65])

def bell_shape_decay(x, a, b):
    return 100 / (1 + (x / a) ** (2 * b))

# Fit del modello ai dati con stime iniziali per L, k, e x0
parametri, covarianza = curve_fit(
    bell_shape_decay,
    distanze,
    pdr,
    p0=(750, 0.8), # Stime iniziali per a, b
    bounds=([100, 0.5], [1800, 2.5]),
    maxfev=20000
)

# Otteniamo i parametri a, b
a, b = parametri
```

Dopo aver eseguito gli script, ogni volta utilizzando i dati relativi all'altezza considerata, sono state ricavate le seguenti funzioni che modellano il comportamento del PDR in relazione alla distanza:

$$PDR_{altezza-uomo}(x) = \frac{100}{1 + \left(\frac{x}{792{,}23}\right)^{2 \cdot 2{,}29}} \tag{4.2}$$

In Fig. 4.3, la funzione Eq. (4.2) rappresenta il PDR calcolato per la distanza quando il ricevitore è posizionato ad altezza uomo. Si nota che i parametri trovati portano ad una curva con una rapida discesa, indicando che la ricezione di pacchetti diminuisce in modo significativo oltre una certa distanza.

$$PDR_{quinto-piano}(x) = \frac{100}{1 + \left(\frac{x}{731{,}63}\right)^{2 \cdot 0{,}86}} \tag{4.3}$$

In Fig. 4.4 viene mostrato l' Eq. (4.3), calcolata con il ricevitore posizionato al quinto piano di un edificio. In questo caso, il valore più basso del parametro $b$ indica una curva meno ripida, suggerendo una transizione più graduale nella perdita di pacchetti con l'aumento della distanza.

$$PDR_{ottavo-piano}(x) = \frac{100}{1 + \left(\frac{x}{994{,}87}\right)^{2 \cdot 0{,}63}} \tag{4.4}$$

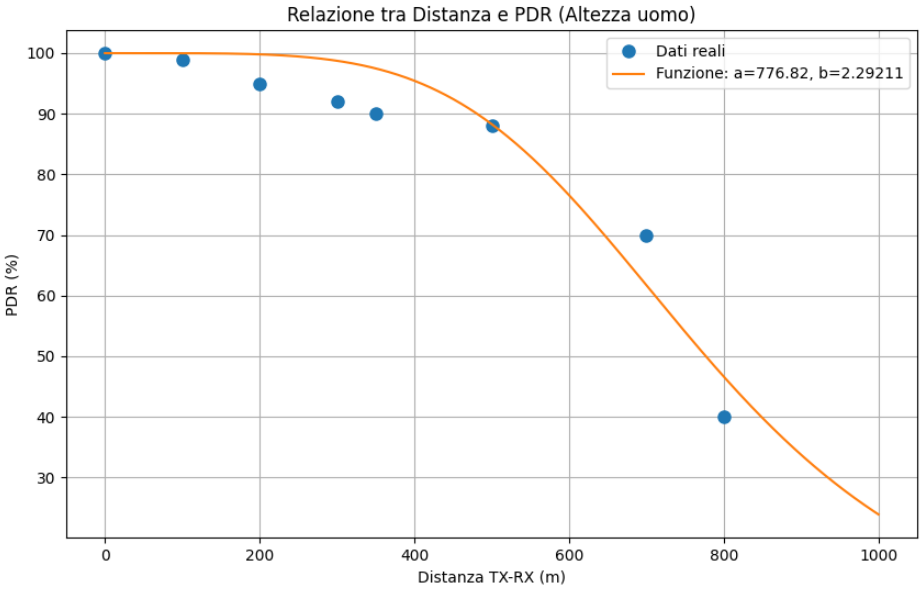


**Figura 4.3.** Rappresentazione Eq. (4.2)

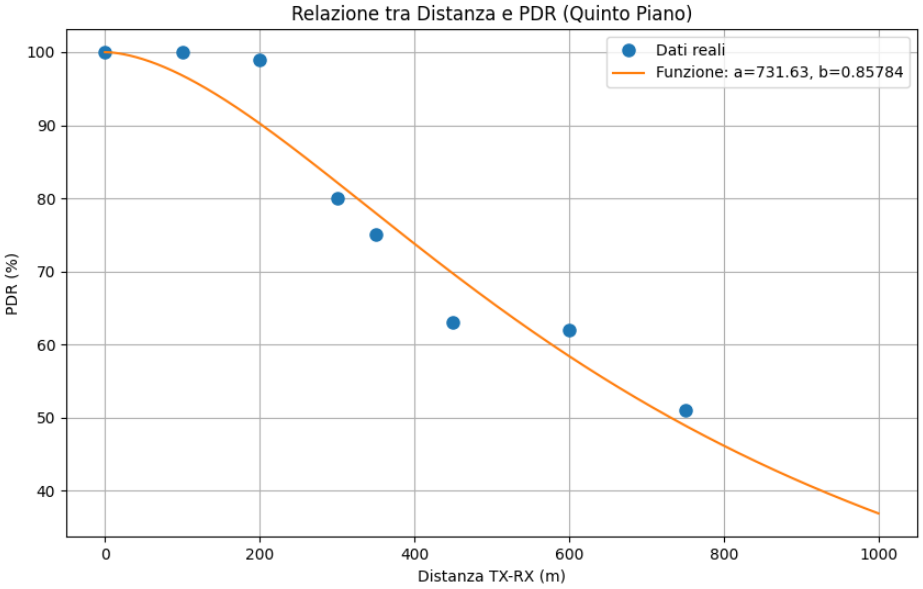


**Figura 4.4.** Rappresentazione Eq. (4.3)

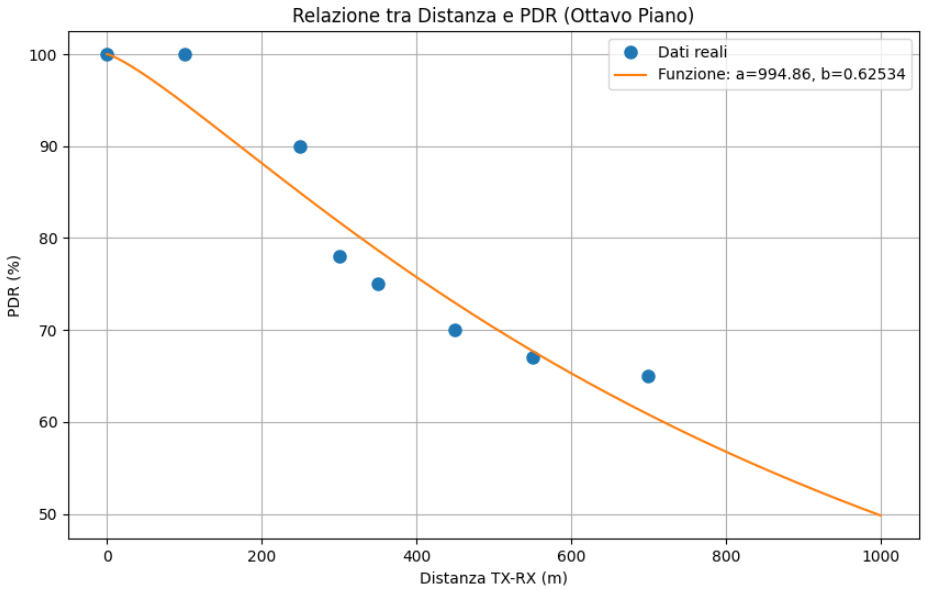


**Figura 4.5.** Rappresentazione Eq. (4.4)

Infine, in Fig. 4.5 la funzione Eq. (4.4) descrive il PDR per la distanza con il ricevitore collocato all'ottavo piano. Il parametro $a$ maggiore riflette una distanza critica più alta, mentre il valore di $b$ più basso rispetto agli altri casi indica una curva ancora più dolce. Ciò suggerisce che l'effetto della distanza sul PDR è meno drastico, con una perdita di pacchetti che avviene più gradualmente.

Queste funzioni permettono di caratterizzare e confrontare l'efficacia della ricezione del segnale a diverse altezze, evidenziando come il posizionamento del ricevitore influenzi significativamente la qualità della comunicazione. Esse verranno utilizzate come base per due diversi algoritmi, progettati per individuare le posizioni ottimali dei ricevitori. L'obiettivo degli algoritmi sarà massimizzare l'efficienza della ricezione del segnale, tenendo conto delle caratteristiche di decadimento della curva del PDR a seconda della distanza e dell'altezza del ricevitore.

Gli algoritmi riceveranno in input le posizioni candidate per i dispositivi riceventi, il numero di dispositivi da collocare, i punti strategici lungo il tragitto della processione pasquale di Comiso, in corrispondenza di tappe chiave dell'evento, e una soglia di PDR minimo. Questi parametri saranno fondamentali per determinare quali posizioni offrano la copertura migliore, assicurando che il PDR superi il valore soglia per garantire un'efficace comunicazione.

### 4.2.1 Algoritmo Brute Force

Il primo approccio è un algoritmo di tipo **brute force**. Gli algoritmi di forza bruta propongono un approccio diretto per risolvere problemi che si basano sull'esplorazione di ogni possibilità, utilizzando la sola potenza computazionale(8). Questo metodo valuta tutte le possibili combinazioni di posizioni per i ricevitori e calcola il PDR corrispondente per ciascuna combinazione, utilizzando le funzioni precedentemente calcolate.

L'algoritmo è stato sviluppato per individuare le posizioni ottimali dei dispositivi riceventi, in modo da garantire una copertura ottimale dei punti lungo un percorso predefinito e assicurando che la Percentuale di Pacchetti Ricevuti (PDR) sia sempre superiore a una soglia minima del 50%.

Per ogni combinazione di dispositivi e posizioni possibili, l'algoritmo calcola il PDR per ciascun punto del percorso. Innanzitutto, si combinano i PDR di tutti i dispositivi attraverso un approccio basato sulla probabilità complementare; se il PDR totale per un punto supera la soglia del 50%, quel punto viene considerato coperto, e il PDR medio viene aggiornato per i punti coperti.

Nonostante la complessità dell'algoritmo brute force, esso rappresenta uno strumento potente per ottenere la soluzione ottimale e fornisce un benchmark ideale per confrontare la performance di algoritmi più efficienti, come l'algoritmo greedy.

**Codice 4.4.** Script Python per algoritmo Brute Force

```
import numpy as np
from itertools import combinations

def bell_shape_decay(x, a, b):
    return 100 / (1 + (x / a) ** (2 * b))

# Distanza tra due punti
def distanza(p1, p2):
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def valuta_combinazione(combinazione_dispositivi):
    copertura_totale = 0
    pdr_medio_totale = 0
    num_punti_coperti = 0
    pdr_per_punto = [] # Lista per salvare il PDR massimo per ogni punto del
        percorso

    # Per ogni punto del percorso, calcola il PDR combinato ricevuto dai
```

```
        dispositivi
    for punto in punti_percorso:
        pdr_complementare = 1 # Iniziamo con 1 e moltiplichiamo per (1 - P_i)

        for dispositivo in combinazione_dispositivi:
            dist = distanza(dispositivo, punto)
            # Seleziona la funzione corretta in base all'altitudine del
                dispositivo
            a, b = seleziona_func(dispositivo[2])
            pdr_attuale = bell_shape_decay(dist, a, b)
            pdr_attuale = max(0, min(pdr_attuale, 100)) # Assicuriamoci che il
                PDR sia tra 0 e 100%
            pdr_complementare *= (1 - pdr_attuale / 100)

        # Calcola la probabilita' totale combinata
        pdr_totale = 1 - pdr_complementare
        pdr_totale = pdr_totale * 100 # Convertiamo di nuovo in percentuale
        pdr_per_punto.append(pdr_totale)

        # Verifica se il PDR combinato e' sopra la soglia
        if pdr_totale >= soglia_pdr:
            copertura_totale += 1
            pdr_medio_totale += pdr_totale
            num_punti_coperti += 1

    # Calcola PDR medio per i punti coperti
    if num_punti_coperti > 0:
        pdr_medio_totale /= num_punti_coperti

    # Ritorna la percentuale di punti coperti e il PDR medio, insieme ai PDR
        per punto
    percentuale_copertura = (copertura_totale / len(punti_percorso)) * 100
    return percentuale_copertura, pdr_medio_totale, pdr_per_punto

# Trova la migliore combinazione di dispositivi
migliore_copertura = 0
miglior_pdr_medio = 0
migliore_combinazione = None
migliori_pdr_per_punto = []

for combinazione in combinations(posizioni_candidate, numero_dispositivi):
```

```
55      percentuale_copertura, pdr_medio, pdr_per_punto =
            valuta_combinazione(combinazione)
56      if percentuale_copertura > migliore_copertura or (percentuale_copertura
            == migliore_copertura and pdr_medio > miglior_pdr_medio):
57          migliore_copertura = percentuale_copertura
58          miglior_pdr_medio = pdr_medio
59          migliore_combinazione = combinazione
60          migliori_pdr_per_punto = pdr_per_punto
```

### 4.2.2 Algoritmo Greedy

Il secondo approccio è un algoritmo di tipo **greedy**. Questo metodo adotta una strategia euristica che seleziona le posizioni migliori in modo incrementale, basandosi su una scelta locale ottimale a ogni passo. In particolare, l'algoritmo greedy valuta le posizioni una alla volta e sceglie quella che offre il miglior incremento di PDR complessivo, fino a coprire l'area desiderata. Anche se non garantisce la soluzione globale ottimale, l'algoritmo greedy rappresenta una soluzione pratica per scenari più complessi.

In questo caso, si inizia considerando tutte le posizioni candidate per i dispositivi e, iterativamente, selezionando quella che fornisce la maggiore percentuale di copertura. Ogni volta che viene posizionato un dispositivo, la posizione selezionata viene rimossa dall'insieme delle posizioni candidate, e il processo continua fino a posizionare il numero totale di dispositivi richiesto.

Per ciascun punto del percorso, viene calcolato il PDR complessivo considerando sia i dispositivi già posizionati sia la nuova posizione candidata. L'efficacia di ciascuna configurazione è determinata dalla percentuale di punti del percorso coperti, cioè quelli per cui il PDR supera una soglia minima stabilita al 50%.

Pur senza garantire il raggiungimento della soluzione ottimale, l'algoritmo greedy rappresenta una migliore scelta in situazioni in cui prestazioni e rapidità sono cruciali, risultando pratico e scalabile per problemi con un grande numero di dispositivi o punti del percorso.

**Codice 4.5.** Script Python per algoritmo Greedy

```
1  import numpy as np
2
3  # Funzione che calcola la distanza euclidea tra due punti (p1 con altitudine
       inclusa e p2 senza)
4  def distanza(p1, p2):
5      return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)
```

```
6
7  def bell_shape_decay(x, a, b):
8      return 100 / (1 + (x / a) ** (2 * b))
9
10 # Funzione che valuta una singola posizione candidata con i dispositivi
       attuali
11 def valuta_posizione(posizione_candidata, dispositivi_posizionati,
       punti_percorso):
12     copertura_totale = 0
13     pdr_per_punto = []
14
15     for punto in punti_percorso:
16         pdr_complementare = 1
17
18         # Calcoliamo la copertura con i dispositivi gia' posizionati
19         for dispositivo in dispositivi_posizionati:
20             dist = distanza(dispositivo, punto)
21             a, b = seleziona_func(posizione_candidata[2])
22             pdr_attuale = bell_shape_decay(dist, a, b)
23             pdr_attuale = max(0, min(pdr_attuale, 100))
24             pdr_complementare *= (1 - pdr_attuale / 100)
25
26         # Calcoliamo la copertura aggiuntiva con il nuovo dispositivo candidato
27         dist = distanza(posizione_candidata, punto)
28         a, b = seleziona_func(posizione_candidata[2])
29         pdr_attuale = bell_shape_decay(dist, a, b)
30         pdr_attuale = max(0, min(pdr_attuale, 100))
31         pdr_complementare *= (1 - pdr_attuale / 100)
32
33         # PDR totale per il punto
34         pdr_totale = 1 - pdr_complementare
35         pdr_totale = pdr_totale * 100 # Convertiamo in percentuale
36
37         if pdr_totale >= soglia_pdr:
38             copertura_totale += 1
39
40         pdr_per_punto.append(pdr_totale)
41
42     percentuale_copertura = (copertura_totale / len(punti_percorso)) * 100
43     return percentuale_copertura, pdr_per_punto
44
```

```python
# Algoritmo greedy per posizionare i dispositivi
def greedy_posizionamento_dispositivi(posizioni_candidate, punti_percorso, numero_dispositivi):
    dispositivi_posizionati = []

    for i in range(numero_dispositivi):
        migliore_copertura = 0
        miglior_pdr_per_punto = []
        migliore_posizione = None

        # Trova la migliore posizione candidata
        for posizione in posizioni_candidate:
            percentuale_copertura, pdr_per_punto = valuta_posizione(posizione, dispositivi_posizionati, punti_percorso)
            if percentuale_copertura > migliore_copertura:
                migliore_copertura = percentuale_copertura
                migliore_posizione = posizione
                miglior_pdr_per_punto = pdr_per_punto

        dispositivi_posizionati.append(migliore_posizione)
        posizioni_candidate.remove(migliore_posizione) # Rimuoviamo la posizione selezionata

    return dispositivi_posizionati, migliore_copertura, miglior_pdr_per_punto

# Esegui l'algoritmo
dispositivi_posizionati, copertura_totale, pdr_per_punto = greedy_posizionamento_dispositivi(posizioni_candidate, punti_percorso, numero_dispositivi)
```

## 4.3 Sviluppo applicazione Flutter

### 4.3.1 Introduzione a Flutter

Flutter è un framework open-source sviluppato da Google per lo sviluppo di applicazioni mobili cross-platform, utilizzando un unico codice sorgente(9). Una delle sue principali caratteristiche è l'utilizzo del linguaggio di programmazione Dart, che offre un ambiente altamente performante e un sistema di rendering basato su widget. In Flutter, tutto è un widget, ovvero blocchi che costituiscono l'interfaccia utente.

Una distinzione fondamentale in Flutter è quella tra **StatelessWidget** e **StatefulWidget**(10):

- **StatelessWidget**: Rappresenta un widget che non cambia nel tempo. È utilizzato per creare componenti che rimangono statici una volta costruiti. In questi widget, tutte le proprietà vengono inizializzate una volta sola e vengono aggiornati solo se qualcosa è cambiato al loro interno.

- **StatefulWidget**: Rappresenta un widget che può cambiare dinamicamente in risposta ad eventi. Sono composti da due parti: il widget stesso e uno stato separato che viene mantenuto durante il ciclo di vita del widget.

### 4.3.2 Struttura

L'applicazione Flutter è stata progettata con un'architettura modulare per facilitare la manutenibilità del codice. La cartella principale `lib` è suddivisa in diverse sottosezioni per una gestione efficiente dei componenti e della logica dell'app.

Il file `main.dart` rappresenta il punto di ingresso dell'applicazione che avvia il widget principale `AppContainer`. Esso gestisce la navigazione tra le diverse pagine dell'applicazione utilizzando una barra di navigazione personalizzata, implementata nel file `bottom_navigation_bar.dart`. Le destinazioni di navigazione sono definite nel file `navigation_items.dart`, che fornisce icone e etichette per le sezioni principali, ovvero: Mappa, Galleria e Programma.

L'applicazione utilizza diverse pagine per offrire funzionalità specifiche:

- **MapPage**: È un `StatefulWidget` poiché gestisce la visualizzazione della mappa interattiva. Questa pagina implementa il tracciamento della posizione GPS dell'utente e aggiorna periodicamente la posizione di un marker basato sui dati ricevuti da un server backend.

- **GalleryPage**: È un `StatefulWidget` poiché carica e visualizza un elenco di immagini da una sorgente remota. Utilizza un `FutureBuilder` per gestire il caricamento asincrono delle immagini, e il suo stato viene aggiornato man mano che le immagini vengono scaricate.

- **ProgramPage**: Anche questo è un `StatefulWidget` poiché gestisce un calendario interattivo che consente all'utente di selezionare date e visualizzare eventi associati. Il widget mantiene lo stato per gestire le date selezionate e il formato del calendario.

- **InfoPage**: Questo componente è un `StatelessWidget` poiché mostra informazioni statiche come il nome dell'utente e i dettagli di contatto. Non ha bisogno di aggiornarsi dinamicamente e quindi non richiede uno stato.

La `MyBottomNavigationBar` permette all'utente di spostarsi facilmente tra queste pagine, mentre la `MyAppBar` fornisce un'interfaccia personalizzata per la parte superiore dell'app con un pulsante per accedere alle informazioni aggiuntive.

### 4.3.3 Librerie e dipendenze utilizzate

Lo sviluppo di applicazioni Flutter trae significativo vantaggio dall'ecosistema ricco di librerie e pacchetti della comunità. Queste risorse consentono l'implementazione veloce di funzionalità complesse, ottimizzando il processo di sviluppo e migliorando l'esperienza complessiva dell'utente. Di seguito, verranno presentate le librerie principali, analizzando le loro specifiche funzionalità e modalità di utilizzo nell'applicazione:

- **`http`**: Utilizzata per la comunicazione tramite chiamate HTTP con il backend. Utilizzata per ottenere le coordinate del marker che si muove(11).
- **`google_maps_flutter`**: Fornisce l'integrazione delle mappe interattive di Google. È impiegata per visualizzare la posizione dell'utente e i marker aggiornati in tempo reale(12).
- **`geolocator`**: Gestisce l'ottenimento della posizione GPS dell'utente e la richiesta dei permessi relativi. È fondamentale per le funzionalità di geolocalizzazione e per aggiornare la posizione sulla mappa(13).
- **`permission_handler`**: Usata per gestire le autorizzazioni richieste dall'app, come l'accesso alla posizione GPS. Garantisce che l'app operi in conformità con le policy di sicurezza di iOS e Android(14).
- **`table_calendar`**: Un pacchetto per la visualizzazione di calendari interattivi. È stato implementato per mostrare eventi relativi alle date selezionate(15).

Queste librerie hanno contribuito a rendere lo sviluppo dell'applicazione più rapido ed efficiente, fornendo soluzioni già realizzate.

### 4.3.4 Gestione dati GPS e integrazione con il backend

La gestione dei dati GPS e l'integrazione con il backend rappresentano uno degli aspetti centrali del sistema. Questa funzionalità consente di recuperare le coordinate GPS

memorizzate nel database lato server, inviate precedentemente dal dispositivo che segue l'evento. Il recupero delle coordinate avviene tramite la funzione `fetchMarkerPosition`, definita nel file `map_page.dart`. La funzione utilizza i costrutti asincroni `Future` e `await` offerti da Flutter, che permettono di eseguire richieste HTTP in modo non bloccante. Ogni dieci secondi, una chiamata HTTP viene effettuata per recuperare le coordinate aggiornate dal server. In caso di risposta corretta, l'oggetto JSON viene decodificato, ricavando latitudine e longitudine, e la posizione del marker sulla mappa viene aggiornata utilizzando `setState`. Inoltre, il controller della mappa `mapController` anima la visualizzazione per centrare la nuova posizione, garantendo un'esperienza utente fluida e reattiva. Questo approccio asincrono è in linea con le esigenze di ottimizzazione del duty cycle, mantenendo un'efficienza energetica elevata e minimizzando l'uso delle risorse.

**Codice 4.6.** Codice Dart per il recupero e l'aggiornamento delle coordinate GPS

```dart
void _startFetchingMarkerPosition() {
  _timer = Timer.periodic(Duration(seconds: 10), (Timer timer) async {
    await _fetchMarkerPosition();
  });
}

Future<void> _fetchMarkerPosition() async {
  try {
    // Esegui la chiamata HTTP per ottenere la nuova posizione
    final response = await
        http.get(Uri.parse('http://192.168.1.3:5001/latest'));
    if (response.statusCode == 200) {
      var data = json.decode(response.body);
      double latitude = data['latitude'];
      double longitude = data['longitude'];
      setState(() {
        asyncMarker = Marker(
          markerId: MarkerId('async_marker'),
          position: LatLng(latitude, longitude),
          infoWindow: InfoWindow(title: 'Updated Position'),
        );
      });
      if (mapController != null) {
        mapController!.animateCamera(CameraUpdate.newLatLng(LatLng(latitude,
            longitude)));
      }
```

```
    } else {
      print('Failed to fetch marker position');
    }
  } catch (e) {
    print('Error fetching marker position: $e');
  }
}
```

Per supportare l'applicazione Flutter, il backend è stato sviluppato utilizzando Python e Flask, con MongoDB come sistema di gestione del database. I due server si occupano uno di ricevere i dati GPS dal dispositivo e memorizzarli nel database e l'altro di fornire le coordinate più recenti all'applicazione Flutter quando richiesto. Di seguito, sono presentati gli script Python utilizzati per implementare queste funzionalità.

**Codice 4.7.** Script per aggiungere dati al database

```
from flask import Flask
import certifi
from pymongo import MongoClient
from flask_cors import CORS
import json, re

app = Flask(__name__)
CORS(app) # Abilita CORS per tutte le origini

# Configurazione della connessione MongoDB
client = MongoClient(
    'mongodb+srv://user:pws@esempio0.m76yo.mongodb.net',
    tlsCAFile=certifi.where()
)
db = client['tesi']
collection = db['coordinate']

@app.route('/esp32', methods=['POST'])
def receive_data():
    try:
        data = request.data.decode('utf-8')
        # Utilizzare una regex per estrarre la latitudine e la longitudine
        match = re.search(r"Lat:\s*([\d.]+),\s*Lng:\s*([\d.]+)", data)
        if match:
            latitude = float(match.group(1))
            longitude = float(match.group(2))
```

```
27
28            # Creare il documento da inserire in MongoDB
29            document = {
30                "latitude": latitude,
31                "longitude": longitude
32            }
33
34            # Inserire il documento nel database
35            collection.insert_one(document)
36            return f"Data received and stored: {data}", 200
37        else:
38            return "Invalid data format", 400
39    except Exception as e:
40        return f"An error occurred: {str(e)}", 500
41
42 if __name__ == '__main__':
43    app.run(host='0.0.0.0', port=5001, debug=True)
```

**Codice 4.8.** Script Python per recuperare le coordinate dal database

```
1  from flask import Flask, request, jsonify
2  from pymongo import MongoClient
3  from flask_cors import CORS
4  import certifi
5  import re
6
7  app = Flask(__name__)
8  CORS(app) # Abilita CORS per tutte le origini
9
10 # Configurazione della connessione MongoDB
11 client = MongoClient(
12     'mongodb+srv://user:pws@esempio0.m76yo.mongodb.net',
13     tlsCAFile=certifi.where()
14 )
15
16 db = client['tesi']
17 collection = db['coordinate']
18
19 @app.route('/latest', methods=['GET'])
20 def latest():
21     try:
```

```
        # Recupera l'ultimo documento inserito
        latest_doc = collection.find().sort('_id', -1).limit(1)

        # Verifica se ci sono documenti
        latest_data = list(latest_doc)
        if latest_data:
            # Estrai solo latitude e longitude
            data_to_return = {
                'latitude': latest_data[0].get('latitude'),
                'longitude': latest_data[0].get('longitude')
            }
            return jsonify(data_to_return), 200
        else:
            return jsonify({'error': 'No data found'}), 404

    except Exception as e:
        app.logger.error(f"Error occurred: {e}")
        return jsonify({'error': 'Internal Server Error'}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001, debug=True)
```

Questi due componenti assicurano una gestione efficace dei dati GPS, permettendo all'app di ricevere aggiornamenti puntuali e affidabili. La combinazione di un backend solido e un client Flutter ben progettato garantisce una soluzione robusta per applicazioni di tracciamento in tempo reale.

# 5

# Valutazioni performance

In questo capitolo verranno presentati i risultati ottenuti attraverso l'implementazione dei due algoritmi, Brute Force e Greedy, evidenziando le loro prestazioni e i principali punti di forza e debolezza. Successivamente, saranno illustrate le schermate principali dell'applicazione sviluppata in Flutter, fornendo una panoramica dell'interfaccia utente.

## 5.1 Confronto algoritmi

L'obiettivo è stato valutare l'efficacia e l'efficienza di ciascun algoritmo nell'individuare le posizioni ottimali per i dispositivi riceventi, tenendo conto delle loro complessità computazionali e delle risorse richieste.

Nonostante la sua precisione, l'algoritmo di Brute Force non rappresenta sempre la scelta giusta. Infatti, sebbene questa strategia assicuri la ricerca della soluzione ottimale, la sua complessità computazionale risulta notevole. Nell'algoritmo alla Sezione 4.2.1, il numero totale di operazioni effettuate è dato da:

$$NumOperazioni = \binom{N}{k} \times M = \frac{N!}{k! \cdot (N-k)!} \times M \tag{5.1}$$

dove $\binom{N}{K}$ rappresenta tutte le possibili combinazioni tra posizioni candidate $N$ e dispositivi$k$, $M$ rappresenta il numero di punti del percorso che si vuole seguire.

Questo prodotto dimostra come la complessità cresca molto velocemente, rendendo l'algoritmo poco pratico per scenari con un gran numero di posizioni o punti del percorso. Anche per valori moderati di N, k, e M, l'algoritmo richiede una quantità significativa di tempo di calcolo.

A discapito della perfezione, la caratteristica che differenzia l'algoritmo greedy dal brute force è la sua efficienza, poiché, evitando un'elevata complessità computazionale, si riesce a selezionare la posizione che massimizza localmente la copertura. Questo porta ad una riduzione drastica della mole computazionale, riducendo le operazioni a:

$$Operazioni = N \times M \times k \tag{5.2}$$

con $N$ numero di posizioni candidate, $M$ numero di punti del percorso e $k$ numero di dispositivi utilizzati. Di conseguenza, la complessità computazionale dell'algoritmo cresce linearmente con il numero di dispositivi e il numero di punti del percorso, ma rimane significativamente inferiore rispetto alla crescita dell'algoritmo brute force.

Nella Tabella 5.1, sono mostrati i tempi di calcolo dei due algoritmi, ottenuti tramite simulazioni sul mio computer personale. Le condizioni iniziali prevedono 100 punti del percorso e 5 dispositivi, mentre il numero di posizioni candidate è stato variato.

**Tabella 5.1.** Confronto tra algoritmi Brute Force e Greedy

| **Posizioni candidate($k$)** | **Brute Force (s)** | **Greedy (s)** |
|---|---|---|
| 25 | 24.01 | 0.05 |
| 30 | 65.62 | 0.1 |
| 50 | 295.67 | 0.9 |
| 200 | $NaN$ | 9 |
| 300 | $NaN$ | 15 |

Per evidenziare ulteriormente le differenze tra l'algoritmo Brute Force e l'algoritmo Greedy, sono stati realizzati dei grafici che mostrano le scelte effettuate da ciascun algoritmo rispetto alla disposizione dei dispositivi riceventi. Questi grafici rappresentano visivamente il risultato delle strategie adottate dai due algoritmi, permettendo di confrontare la qualità delle soluzioni generate.

I grafici Fig. 5.1, Fig. 5.5, Fig. 5.3 mostrano le soluzioni proposte dall'algoritmo di Brute Force con 5 dispositivi, 40 punti del percorso, 30 posizioni candidate. La soluzione ottimale proposta è:

$$[(1000, -200, 25), (1600, -400, 25), (3000, -200, 25), (3200, 400, 25), (3600, -500, 25)]$$

Questa disposizione consente una copertura del 100% dei punti del percorso, con un PDR medio per punto pari al 93,49%.

I grafici Fig. 5.2, Fig. 5.6, Fig. 5.4 rappresentano, invece, i risultati ottenuti dall'algoritmo Greedy, mantenendo gli stessi parametri iniziali. La soluzione trovata è:

$$[(100, -100, 25), (1000, -200, 25), (1200, 800, 15), (3400, 600, 15), (3600, -500, 25)]$$

Anche in questo caso, l'algoritmo garantisce una copertura completa dei punti del percorso, ma il PDR medio per punto risulta inferiore, pari al 68,73%. Questa differenza è attribuibile alla natura dell'algoritmo Greedy, che seleziona localmente le posizioni ottimali senza considerare l'interazione globale tra le scelte successive.

Dall'analisi dei grafici e delle soluzioni proposte dai due algoritmi emergono alcune osservazioni chiave:

- **Qualità della soluzione**: l'algoritmo di Brute Force garantisce sempre la soluzione ottimale, con un PDR medio più elevato rispetto all'algoritmo Greedy.
- **Efficienza computazionale**: l'algoritmo Greedy riduce significativamente il tempo di esecuzione, rendendolo praticabile in scenari con un elevato numero di posizioni candidate e punti del percorso. Tuttavia, questa efficienza viene raggiunta sacrificando parte della qualità del risultato.
- **Distribuzione delle posizioni selezionate**: entrambi gli algoritmi tendono a preferire posizioni situate ai piani superiori. Questo comportamento era prevedibile, poiché i dispositivi posizionati più in alto beneficiano di una minore interferenza causata da ostacoli. L'algoritmo Greedy, tuttavia, mostra una maggiore variabilità nella scelta delle altezze, selezionando in alcuni casi dispositivi a livelli intermedi.
- **Impatto sul PDR medio**: nonostante il PDR medio ottenuto dall'algoritmo Greedy sia inferiore, esso supera comunque la soglia richiesta per garantire la trasmissione affidabile dei dati. Questo lo rende una scelta accettabile in contesti in cui il tempo di calcolo è un fattore critico.

I risultati evidenziano un chiaro compromesso tra precisione ed efficienza. L'algoritmo di Brute Force rappresenta la soluzione ideale per ottenere la disposizione ottimale, ma il suo utilizzo è limitato a scenari con un numero moderato di posizioni candidate e punti del percorso. Al contrario, l'algoritmo Greedy offre una soluzione più rapida e pratica, sacrificando parte dell'ottimalità in cambio di una maggiore efficienza. In contesti reali, la scelta tra i due algoritmi dipenderà dai requisiti specifici dell'applicazione, in termini di accuratezza richiesta e risorse computazionali disponibili.

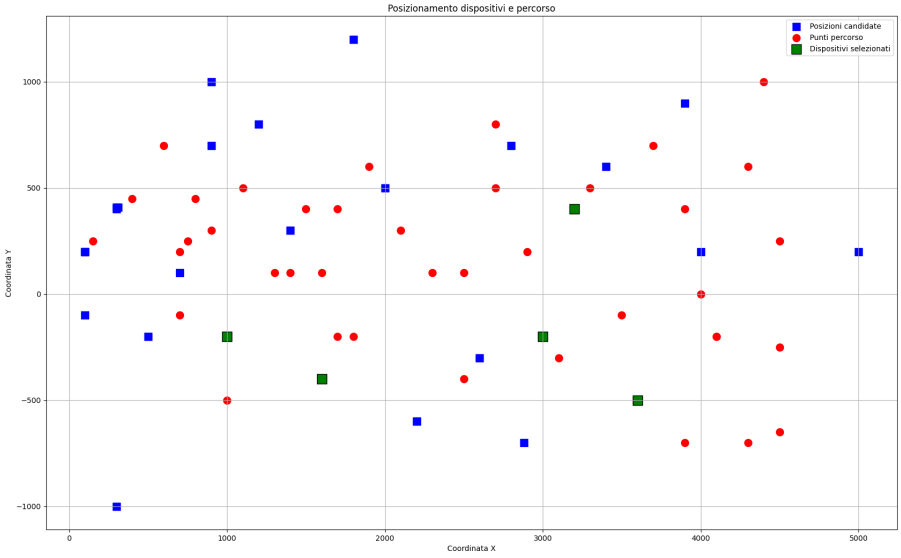


**Figura 5.1.** Posizionamento 2D dispositivi e percorso - Brute Force

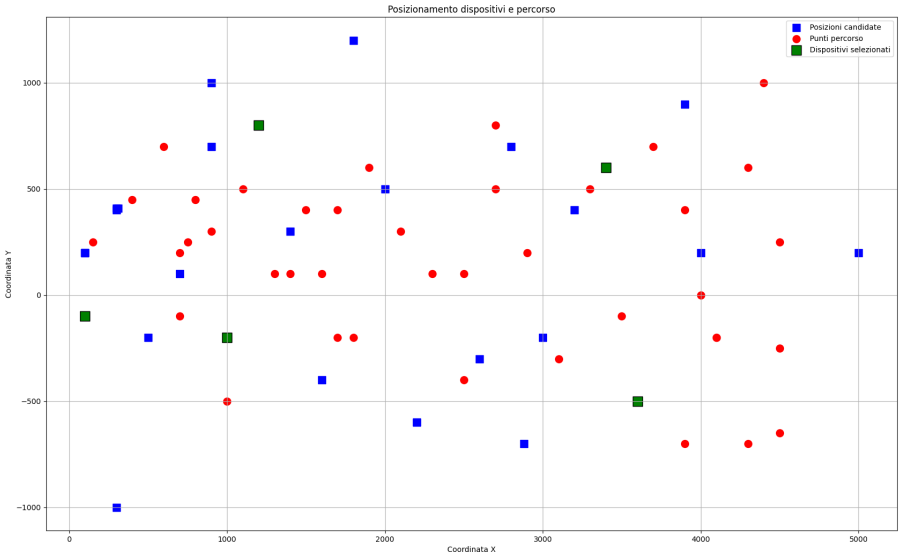


**Figura 5.2.** Posizionamento 2D dispositivi e percorso - Greedy

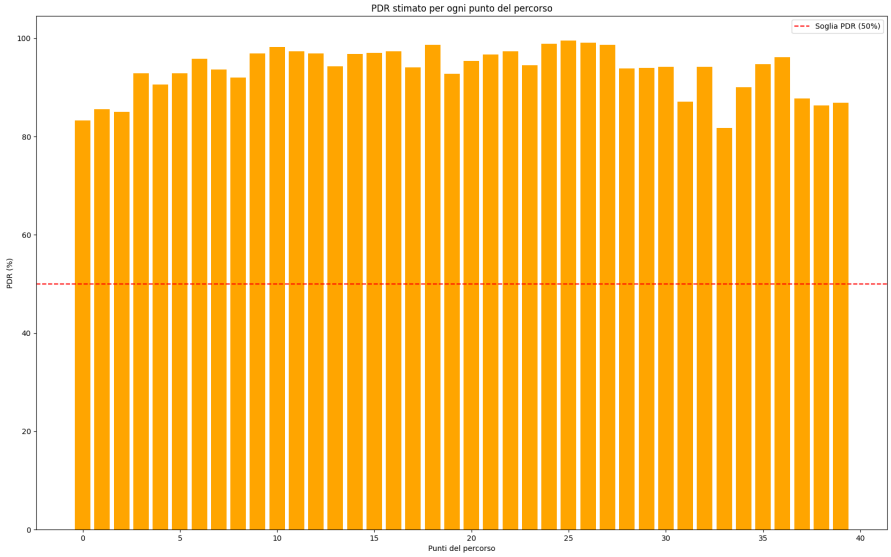


**Figura 5.3.** PDR stimato per ogni punto del percorso - Brute Force

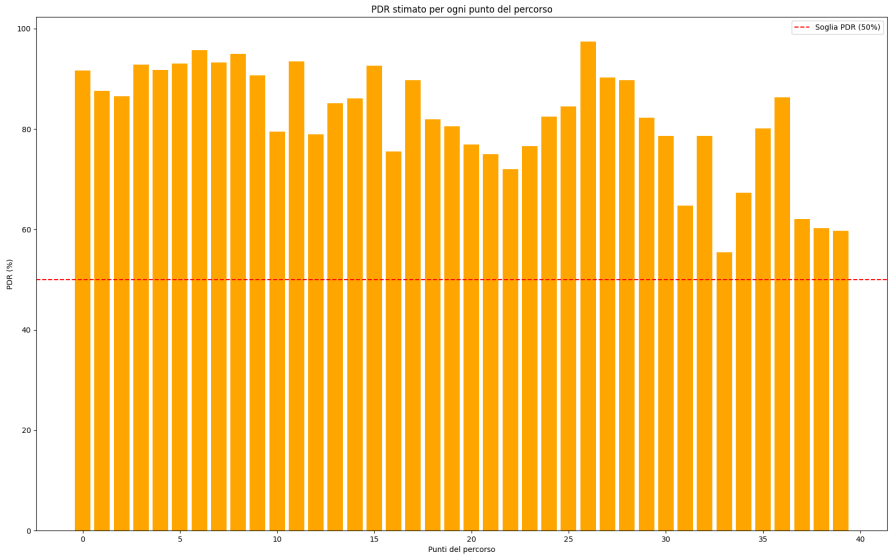


**Figura 5.4.** PDR stimato per ogni punto del percorso - Greedy

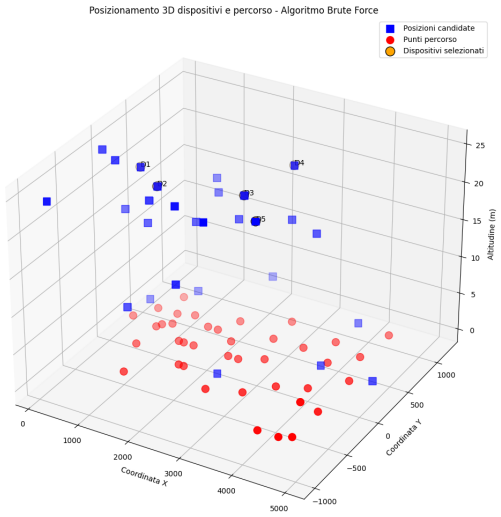


**Figura 5.5.** Posizionamento 3D dispositivi e percorso - Brute Force

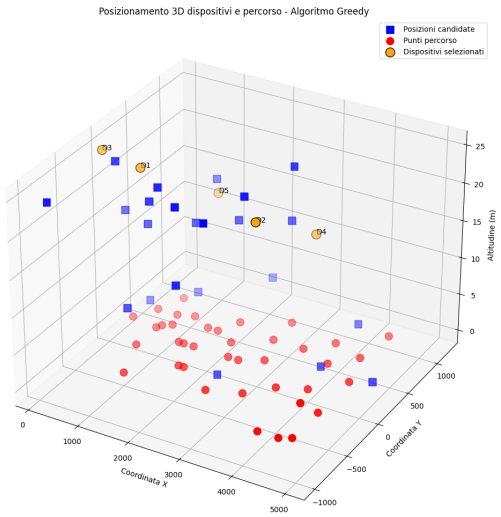


**Figura 5.6.** Posizionamento 3D dispositivi e percorso - Greedy

## 5.2 Schermate principali app

L'applicazione Flutter sviluppata è stata progettata per offrire un'interfaccia semplice e intuitiva, facilitando l'interazione dell'utente con le funzionalità principali. Di seguito sono presentate le schermate principali che compongono l'applicazione.

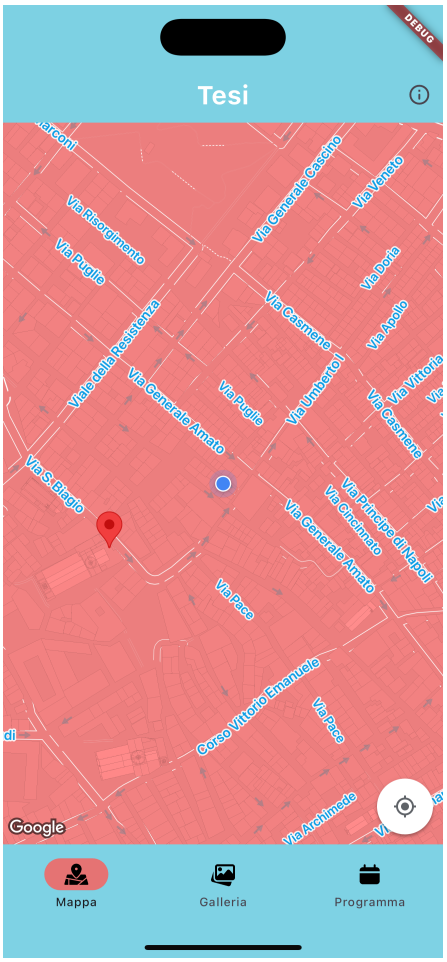


**Figura 5.7.** Schermata iniziale

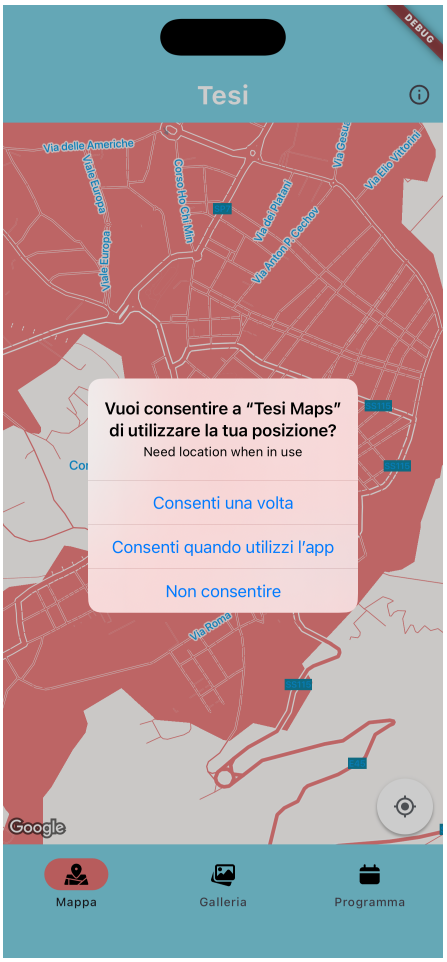


**Figura 5.8.** Permessi necessari

Come mostrato in Fig. 5.7, la schermata iniziale dell'applicazione è costituita dalla mappa interattiva, che rappresenta il punto di ingresso principale. Questa schermata consente di visualizzare in tempo reale la posizione degli oggetti tracciati, garantendo un accesso immediato alle informazioni essenziali. Da qui, l'utente può anche navigare verso altre sezioni dell'app, attraverso un'interfaccia semplice e intuitiva.

Per garantire il corretto funzionamento dell'applicazione, l'utente deve concedere i permessi necessari per accedere alla posizione GPS. La Fig. 5.8 mostra la schermata in

cui viene richiesta l'autorizzazione, fondamentale per garantire la funzionalità prioritaria.

La schermata di galleria, mostrata in Fig. 5.9, consente di visualizzare una serie di immagini, fornendo ulteriori dettagli relativi all'evento o al contesto dell'applicazione.

In Fig. 5.10, viene presentata la schermata relativa al programma. Essa fornisce informazioni dettagliate sugli eventi pianificati, incluse date, orari e descrizioni. Rappresenta una funzionalità chiave per gli utenti che necessitano di una panoramica chiara e organizzata.

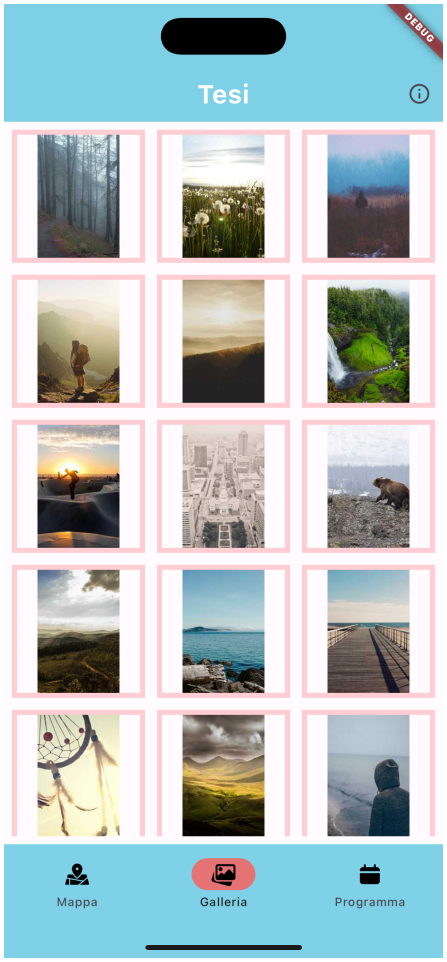


**Figura 5.9.** Schermata galleria

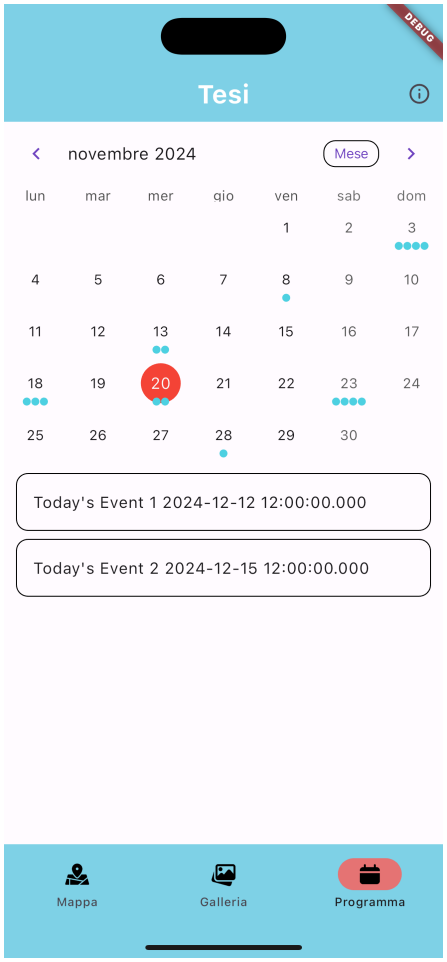


**Figura 5.10.** Schermata con il programma relativo all'evento

Infine, la Fig. 5.11 mostra una schermata dedicata alle informazioni sull'applicazione. Questa sezione può includere dettagli sull'evento, i contatti o informazioni aggiuntive utili per l'utente.

L'interfaccia è stata realizzata per essere semplice e molto utilizzabile, assicurando

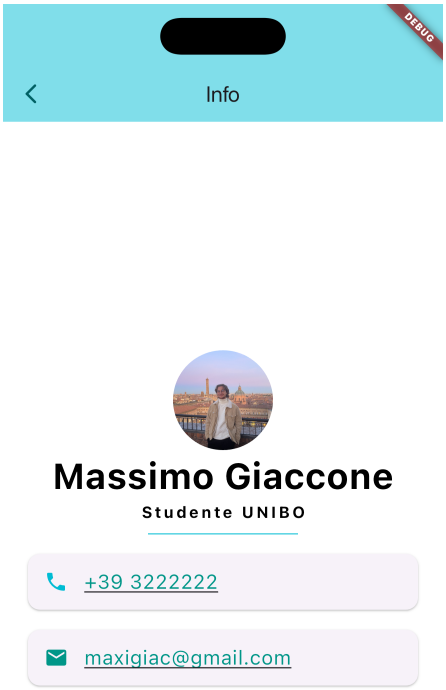


**Figura 5.11.** Schermata informativa

un'esperienza fluida anche agli utenti meno esperti.

# 6
# Conclusioni

## 6.1 Sintesi dei risultati

Questo progetto ha approfondito lo sviluppo e l'implementazione di un sistema innovativo per il tracciamento di oggetti mobili sulla tecnologia LoRa, applicabile in contesti smart-city. L'obiettivo principale della tesi era progettare un'architettura a moduli e implementare un sistema efficiente in grado di monitorare in tempo reale eventi dinamici. Il caso di studio scelto, la processione pasquale di Comiso, ha fornito un esempio concreto e rappresentativo del sistema.

Il lavoro si è articolato in più fasi. Inizialmente, sono stati analizzate la caratteristiche della tecnologia LoRa e del protocollo LoRaWAN. Successivamente, si è sviluppata l'architettura che integra componenti hardware e software, che comprende dispositivi trasmettitori e riceventi, server per l'eleborazione dei dati e un'applicazione mobile per la visualizzazione in tempo reale delle informazioni sull'oggetto in movimento.

Un'aspetto chiave è stato lo sviluppo e la sperimentazione di due algoritmi (Brute Force e Greedy), utilizzati per ottimizzare il posizionamento dei dispositivi ricevitori, individuando la posizione più adatta in base ai vincoli del contesto operativo.

I test condotti hanno dimostrato l'efficacia del sistema proposto. La combinazione di LoRa con una logica algoritmica ottimizzata ha permesso di massimizzare la copertura del percorso monitorato. Il confronto tra gli algoritmi ha evidenziato i punti di forza e i limiti di ciascun approccio:

- **Brute Force**: ha garantito la massima precisione nella disposizione dei dispositivi riceventi, con una copertura completa e un PDR medio per punto molto elevato. Tuttavia, il costo computazionale ne limita l'applicazione a scenari con un numero moderato di posizioni candidate.

- **Greedy**: si è rivelato più adatto a scenari reali grazie alla sua efficienza computazionale. Nonostante un PDR medio per punto inferiore, ha comunque soddisfatto i requisiti minimi richiesti.

Le simulazioni sul Packet Delivery Ratio (PDR) hanno confermato la robustezza del modello e la capacità di adattarsi a diverse condizioni ambientali e logistiche.

## 6.2 Miglioramenti e sviluppi futuri

L'architettura sviluppata offre numerose possibilità di espansione:

- Scalabilità: estensione del sistema per monitorare eventi più complessi, con un maggior numero di dispositivi;
- Integrazione con tecnologie emergenti: sperimentazione con 5G e reti ibride per aumentare la capacità di trasmissione e la sicurezza dei dati;
- Integrazione del protocollo LoRaWAN: questa integrazione migliorerebbe ulteriormente la robustezza, la sicurezza e la scalabilità del sistema, rendendolo più adatto per applicazioni in contesti urbani o industriali.
- Esperienza utente: potenziamento dell'applicazione mobile con funzionalità avanzate, come notifiche in tempo reale e statistiche dettagliate.

In conclusione, questo lavoro ha dimostrato come una combinazione di tecnologie IoT e algoritmi di ottimizzazione possa offrire soluzioni pratiche per monitorare eventi complessi, fornendo un valido contributo alla digitalizzazione delle tradizioni e alla gestione delle smart-city.

# Bibliografia


[1] G. Ferré and A. Giremus, "LoRa Physical Layer Principle and Performance Analysis," in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 65–68, 2018.

[2] Q. Chaudhari, "Understanding LoRa PHY (Long-Range Physical Layer)." https://wirelesspi.com/.

[3] M. A. Ertürk, M. A. Aydin, T. Büyükakkaslar, and H. Evirgen, "A Survey on LoRaWAN Architecture, Protocol and Technologies," *Future Internet*, vol. 11, p. 216, 10 2019.

[4] Mons. Giovanni Battaglia, *Pietre Vive*.

[5] M. C. Bor, J. Vidler, and U. Roedig, "LoRa for the Internet of Things," in *Ewsn*, vol. 16, pp. 361–366, 2016.

[6] M. delle Imprese e del Made in Italy, "Piano nazionale di ripartizione delle frequenze (PNRF)." https://www.mimit.gov.it/it/digitale/gestione-spettro-radio/piano-nazionale-ripartizione-frequenze.

[7] EByte, *E32 LoRa Module Datasheet*. EByte Electronic Technology Co., Ltd., 2023.

[8] freeCodeCamp.org, "Brute Force Algorithms Explained." https://www.freecodecamp.org/news/brute-force-algorithms-explained/.

[9] Google, "Flutter." https://flutter.dev.

[10] Google, "Flutter." https://docs.flutter.dev/get-started/fundamentals/state-management.

[11] DartDev, "HTTP Package." https://pub.dev/packages/http.

[12] Google, "Google Maps for Flutter." https://pub.dev/packages/google_maps_flutter.

[13] BaseFlow, "Geolocator for Flutter." https://pub.dev/packages/geolocator.

[14] BaseFlow, "Permission Handler for Flutter." https://pub.dev/packages/permission_handler.

[15] https://pub.dev/packages/table_calendar.

# Ringraziamenti

Di recente ho partecipato a una conferenza sul diritto d'autore, in particolare su chi detenga i diritti di un'opera generata tramite intelligenza artificiale. Tra le tante tesi emerse, una mi ha colpito profondamente: ogni opera ha un artista e una modalità di realizzazione, che sia un'AI, la pittura a olio, o il lavoro con martello e scalpello. Ma ciò che conta davvero è il contesto in cui quell'artista ha vissuto: le persone che ha incontrato, i luoghi che ha frequentato, le esperienze che lo hanno formato. Se Van Gogh fosse stato mio compagno di corso e ci fossimo presi una birra insieme ogni sabato, chissà se avrebbe dipinto le sue meraviglie. Ma perché dico questo? Non sto delirando (anche se scrivere alle 2 di notte ascoltando *Perché lo fai* di Marco Masini potrebbe suggerire il contrario). Credo fermamente che il contesto in cui sono nato e cresciuto abbia giocato un ruolo fondamentale nello sviluppo del mio lavoro e della mia persona. Ed è proprio per questo che sento il bisogno di ringraziarlo per l'ispirazione che mi ha regalato.

Innanzitutto, il primo grazie va alla mia famiglia. Al mio papà, il mio primo supereroe e il mio più grande fan. Sei sempre capace di fare in modo che tutto vada per il meglio, sostenendomi in ogni sogno e augurandomi sempre il massimo. Le persone che ci conoscono dicono che abbiamo spesso gli stessi atteggiamenti; questo mi riempie il cuore di gioia, perché essere come te è esattamente ciò che desidero. Alla mia mamma, grazie perché da te ho imparato cosa vuol dire lottare e mai arrendersi davanti alle difficoltà. Mi sei sempre stata accanto nel tuo modo unico e speciale, anche senza tanti bacini e abbraccini, ma con tante tante tante calorose coccole. Ti ringrazio perché è grazie a te se sono la persona che sono oggi. Anche qui, chi ci conosce mi dice che: "mi si u stissu a to ma" e io sbam: 3 metri sopra il cielo: d'altronde si sa che siamo "troppo bellissimi". A Gianni, fratellone che spesso diventa fratellino, sopratutto quando alza la mano destra in macchina per chiedere scusa o far passare le persone. Grazie perché sei sempre pronto a schierarti dalla mia parte.

Ovviamente, le persone da ringraziare mica finiscono qui: avendo ventordici mila zii e cinque trilioni di cugini non posso tirarmi indietro dal menzionare anche loro. Grazie

a tutti voi perché ognuno di voi ha lasciato qualcosa in me: chi perché "*chi lo vuole un soldino*", chi perché i pranzi al sabato dopo scuola, chi perché viva il Milan, chi perché "*Filippo Maglione*". In tutti i momenti più belli di cui ho memoria, c'è sempre qualcuno di voi, o addirittura tutti. Grazie.

Un grazie particolare va anche al mio nido lontano da casa. Max, Rosaria, Fede e Leti, grazie perché quando mi sento giù so che posso contare su una lasagna, una partita a Fifa o Mario Kart, ma sopratutto, un'immancabile pizza al Setaccio.

Ai miei più cari amici: Ciccio, Sam, Nunzio e Davide. Sin dal momento in cui ci siamo conosciuti, ho capito che la nostra amicizia sarebbe durata per sempre. Anche se ci sentiamo poco e ci vediamo ancora meno, so con certezza che, così come io ci sarò sempre per voi, anche voi ci sarete sempre per me.

Grazie a Mario, anzi, a Don Mario. Ogni giorno che passa sono sempre più convinto della scelta che ho fatto. Ti ringrazio perché sei una presenza speciale e luminosa nella mia vita, una guida che mi fa sentire più vicino a Dio. Grazie alla famiglia Russo, perché la serata "**pini figliocci**" è un evento prioritario e fondamentale ogni volta che scendo.

Voglio ringraziare anche chi ha fatto crescere la mia aura di onnipotenza di ingegnere: Marco e Carlo siete stati gli artificieri che hanno fatto scoppiare in me un enorme passione per quello che studio e che vorrei fare nella vita. Siete una fonte di ispirazione sia per quello che fate sia per il modo in cui lo fate; spero di fare almeno la metà, o perché no anche il doppio, di quello che avete fatto voi.

Adesso è cambiata la location: sono sul 20, davanti piazza Maggiore. È giusto ringraziare anche la città che mi ha accolto a braccia super aperte per questi tre anni e a tutte le persone che ho trovato qui.

Prima però apro una parentesi, perché c'è un'eccezione. Infatti, non ho conosciuto il mio "carissimo Man" qui, ma è come se a Bologna fossimo ripartiti non da zero ma da un po' più di zero. Grazie perché con te le giornate non sono mai noiose, sempre ricche di "UAU", di "perché mi odi", di infiniti turpiloqui che includono domande e risposte. Grazie perché il "Forza man" prima di un esame c'è sempre stato, perché il sabato è "pizza e birruni" rigorosamente Moretti, grazie perché non sei solo un semplice coinquilino, ma sei e rimarrai sempre il "carissimo Man".

Sono sceso dal 20 e sto andando a sedermi nel nostro posticino dove, forse, tutto è iniziato e continua ad andare. Liu-Jo e poi all'improvviso sei arrivata, non so chi l'ha deciso, e mi hai preso sempre più. Grazie perché con te posso essere un pagliaccio, un bambino, un cuoco stellato e sopratutto una persona migliore.

Ad Andrea, Antonio, Andrea, Gaia, Toby, Mea, Gaia, Maria e Chiara. Accenti

diversi, personalità diverse, tutti diversi, ma sempre a partire quando Massimo organizza qualcosa e sempre pronti a mangiare quando cucina qualcosa. Grazie perché avete reso splendidi questi tre anni con un caffé, con una birra, con i tiramisù, con esami copiati, con Maranathà e taaante altre cose.

Infine, non per importanza, le persone a cui ho voluto dedicare questo lavoro. A nonna Rosaria: i tuoi "ma iu nun ti vuogghiu disturbari" fanno pensare tanto il contrario, ma è per questo che sono molto apprezzati.

Grazie a tutti.