Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Magistrale

STEP-APP: PROGETTAZIONE E IMPLEMENTAZIONE DI UN SISTEMA DI LOCALIZZAZIONE INDOOR BASATO SU SENSOR FUSION

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Stefano Traini

Sessione III
Anno Accademico 2016-2017

Ai miei genitori ...

Sommario

Grazie al rapido avanzamento delle nuove tecnologie si è posta sempre più attenzione sui servizi di localizzazione. Tuttavia, data la natura fisica dei segnali emessi, i sistemi di geolocalizzazione tradizionali non riescono ad attraversare ostacoli come le mura degli edifici. ***StepApp*** *è un sistema di localizzazione indoor, scalabile e a basso costo, che fonde la tecnica WiFi-Fingerprinting (basata sugli RSS provenienti dagli Access Point) e Pedestrian Dead Reckoning (basata sui sensori inerziali dello smartphone) al fine di migliorare l'accuratezza del posizionamento all'interno degli edifici. Lo scopo di StepApp è anche quello di fornire al suo interno tre differenti algoritmi di localizzazione tramite WiFi e quattro differenti modalità di localizzazione tramite sensori e in ultimo di inglobare cinque algoritmi di fusione delle due tecniche. Il sistema è stato testato in due differenti ambienti indoor con diverse caratteristiche e dall'analisi dei dati si sono registrati ottimi risultati in termini di accuratezza, raggiungendo una precisione media nella localizzazione superiore al 90% per quasi tutti gli algoritmi di fusione.*

Introduzione

Grazie al rapido avanzamento delle nuove tecnologie e all'altrettanto rapido inserimento di queste nella vita delle persone registrato nell'ultimo decennio a livello globale, si è posta sempre più attenzione sui servizi di localizzazione. I tradizionali sistemi di localizzazione utilizzati quotidianamente dagli utenti si basano su piattaforme satellitari, come *GPS*, *GLONASS* e *Galileo*, che permettono di restituire la posizione sul globo terrestre di un utente attraverso un dispositivo capace di triangolare il segnale radio emesso dai satelliti. Tuttavia, data la natura fisica dei segnali emessi, essi non riescono ad attraversare ostacoli come le mura degli edifici. Le applicazioni di un sistema di localizzazione indoor, che quindi forniscono la posizione all'interno di un edificio, possono riguardare diversi ambiti. Solo per citare qualche esempio, il sistema, all'interno di un museo potrebbe guidare l'utente verso le opere di maggior interesse, all'interno di una struttura ospedaliera potrebbe guidare il paziente verso l'ambulatorio nel quale ha un appuntamento, in un padiglione fieristico potrebbe portare un visitatore in uno stand in particolare, in un aeroporto indicherebbe ad un viaggiatore la strada per il gateway corretto, in una grande struttura potrebbe fornire la posizione di estintori e uscite di sicurezza ed infine, nel caso in cui si subisse un malore, si potrebbe inviare una segnalazione di soccorso con la posizione esatta. Per questi motivi, data la grande utilità, la comunità scientifica sta progettando e sviluppando nuove tecniche per permettere di localizzare un utente all'interno di uno scenario indoor.

Tra le varie tecniche sviluppate, appartenenti alla letteratura scientifica in

oggetto, emergono due approcci che hanno riscontrato un discreto successo tra gli addetti ai lavori: si tratta del metodo **Pedestrian Dead Reckoning** e del metodo **WiFi-Fingerprinting**. La prima tecnica nominata prevede di localizzare un utente attraverso l'uso di sensori inerziali come *accelerometro, giroscopio* e *magnetometro*. L'altra tecnica utilizza i segnali in radiofrequenza emessi dagli Access Point presenti all'interno dell'ambiente di riferimento. Tuttavia queste tecniche se applicate da sole, pur avendo una buona accuratezza, possono presentare degli errori dovuti a condizioni tecniche e ambientali anomale. Basti pensare ad esempio ad un edificio con un'elevata quantità di metalli, al suo interno si riscontrerebbe un campo magnetico influenzato che porterebbe ad una errata localizzazione.

L'obiettivo di questa tesi è quello di presentare **StepApp**, un sistema di localizzazione indoor, scalabile e a basso costo, che fonde le due tecniche descritte al fine di migliorare l'accuratezza del posizionamento all'interno degli edifici. Il sistema è composto da un'architettura *Client-Server* dove la parte server si occupa di implementare la tecnica *WiFi-Fingerprinting* e il client, oltre ad implementare la tecnica *Pedestrian Dead Reckoning*, si occupa di fondere i dati provenienti dai due metodi e calcolare la posizione dell'utente. Per la parte server è stato scelto di utilizzare un tradizionale elaboratore collegato alla rete locale. Dal momento che si è voluto costruire un sistema a basso costo e che la quasi totalità delle persone che si sono approcciate alla tecnologia sono in possesso di uno smartphone, per la parte client si è scelto di sviluppare un'applicazione per questi dispositivi perché al proprio interno dispongono di tutto l'hardware necessario.

Lo scopo di StepApp è anche quello di fornire al suo interno tre differenti algoritmi di localizzazione tramite WiFi, quattro differenti modalità di localizzazione tramite sensori e in ultimo di inglobare cinque algoritmi di fusione delle due tecniche. Inoltre è stato implementato un sistema di *mapping* originale e modulare, infatti in una sua futura versione sarebbe facile aggiungere altre tecniche di localizzazione, come ad esempio le tecniche di prossimità quali quelle basate sui popolari *BLE Beacon*.

Il sistema è stato testato in due differenti ambienti indoor con diverse caratteristiche, infatti uno dei due presenta un campo magnetico anomalo e proprio su di esso si sono visti i maggiori benefici della fusione dei dati. Dall'analisi dei dati provenienti dai test si sono registrati ottimi risultati in termini di accuratezza, raggiungendo una precisione media nella localizzazione superiore al 90% per quasi tutti gli algoritmi di fusione.
La tesi è così suddivisa: nel primo capitolo si parla dello stato dell'arte delle tecnologie di localizzazione indoor, nel secondo capitolo si mostra la progettazione del sistema giustificando le scelte fatte, nel capitolo terzo si spiegano le tecniche implementative adottate per realizzare il sistema e nel capitolo finale vengono mostrati i dati relativi ai test su tutte le tecniche sviluppate nel sistema.

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

1.1 Introduzione allo stato dell'arte

Prima di inoltrarsi nel lungo percorso che ha portato alla realizzazione dell'applicazione sviluppata durante il periodo di tesi e di spiegarne il funzionamento è doveroso mostrare lo stato dell'arte relativo alle tecnologie utilizzate, al fine di giustificare le scelte implementative.
L'accuratezza della localizzazione dipende principalmente dall'uso che si necessita in un particolare contesto. La posizione espressa può essere classificata in 4 categorie [1]:

- Posizione fisica (**PL**-Physical location): identifica un punto in una mappa 2D/3D attraverso le coordinate spaziali;
- Posizione simbolica (**SL**-Symbolic location): identifica una posizione attraverso il linguaggio naturale, come ad esempio "ufficio", "sala riunioni";
- Posizione assoluta (**AL**-Absolute location): identifica la posizione degli oggetti attraverso una griglia di riferimento;
- Posizione Relativa (**RL**-Relative location): identifica una posizione basata sulla posizione conosciuta di altri oggetti, come ad esempio un Access Point;

Come anticipato nel capitolo introduttivo l'applicazione sviluppata ingloba due tecniche di localizzazione indoor: la *Pedestrian Dead Reckoning* (PDR) e la *WiFi-Fingerprinting*.
In questo capitolo vengono presentati articoli scientifici, pubblicati su riviste del settore o presentati durante le conferenze, riguardanti le varie tecniche utilizzate nel corso degli anni dagli esperti in questo campo. Inizialmente vengono illustrati lavori riguardanti la localizzazione indoor attraverso i sensori inerziali presenti sui moderni smartphone o su dispositivi indossabili, successivamente vengono descritti alcuni sistemi che utilizzano le radiofrequenze per localizzare un dispositivo in uno scenario indoor ed infine sono presentate tecniche che utilizzano entrambe le soluzioni sopracitate.

1.2 Sistemi di localizzazione attraverso sensori inerziali

I sistemi di Dead Reckoning utilizzano i **sensori inerziali**, come *accelerometro*, *giroscopio* e *magnetometro*, presenti sui moderni *smartphone* o sui nuovi dispositivi *wearable* per stimare la posizione all'interno di un ambiente indoor della persona che li indossa. Il vantaggio di questi sistemi è che per poter essere realizzati non necessitano di un'architettura complessa o, a volte, non ne fanno uso affatto.

1.2.1 Tipologie di PDR

Nel survey presentato da R. Harle [2] si fa una distinzione tra due tipologie di PDR, *Inertial Navigation Systems* (INSs) e *Step-and-Heading Systems* (SHSs). INSs è un sistema che stima la posizione attraverso la *traiettoria fornita dai sensori sui tre assi* in ogni istante, mentre SHS stima la posizione dell'utente ad ogni passo effettuato conoscendo la *posizione attuale*, la *lun-*

ghezza del passo e la *direzione*.

I due sistemi appena presentati ruotano attorno al concetto di **Gait Cycle**, ossia l'alternanza tra la fase di *stance* e la fase di *swing* del piede di un utente durante una camminata. La fase di *stance* è il periodo nel quale il piede dell'utente è attaccato al terreno, mentre la fase di *swing* è il periodo nel quale il piede non tocca il terreno e oscilla in avanti.

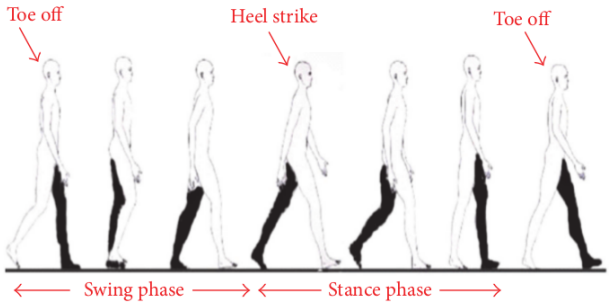

Figura 1.1: Fase di stance e di swing [3].

1.2.2 Riconoscimento del Gate Cycle

Il primo obiettivo è quindi quello di trovare un metodo affidabile per il riconoscimento del Gate Cycle e, a tal proposito, dai ricercatori nel campo sono state analizzate diverse soluzioni. La prima è quella di un algoritmo che percepisca la fase di *stance*, il che comporta che il sensore debba essere indossato nella scarpa dell'utente. Questa prima tecnica è risultata essere efficace per il conteggio dei passi ma non fornisce ulteriori dati utili [2]. La seconda tecnica analizzata è quella relativa al riconoscimento di cicli nei sensori, scaturiti dai movimenti ripetitivi durante la camminata. Anche questa metodologia non risulta essere molto efficace per essere utilizzata nella localizzazione indoor perché, per poter essere applicata, i sensori devono essere applicati nella scarpa dell'utente e si è notato che il "rimbalzo" del piede sul terreno può provocare delle accelerazioni anomale sui sensori, il che comporta un riconoscimento non affidabile del ciclo o dei falsi positivi.

Figura 1.2: Sensore indossato sul piede [3].

Altre tecniche sono state quindi analizzate al fine di riconoscere il Gate Cycle posizionando i sensori in una qualsiasi altra parte del corpo. Una di queste è la *peak detection* che punta a riconoscere una camminata attraverso lo studio dei picchi nei dati ottenuti dai sensori, in particolare dall'accelerometro, sull'asse verticale [4] [5]. Frequentemente a questa tecnica viene applicato un *low-pass filter*, un filtro che taglia le frequenze intorno ai 20 Hz, per ridurre il rumore presente dei sensori [6]. Alla *peak detection* viene applicata un'ulteriore tecnica definita *template matching* che permette di confrontare i picchi ottenuti con dei template già noti in letteratura al fine di individuare un passo [5]. Il *template matching* viene anche utilizzato per individuare una camminata su un terreno inclinato oppure un utente che sta salendo o scendendo una rampa di scale e in questi casi si utilizzeranno template differenti [7].

Analizzati alcuni metodi di riconoscimento del Gate Cycle si entra nel dettaglio delle tipologie di *Pedestrian Dead Reckoning*.

1.2.3 Inertial Navigation Systems

Come anticipato precedentemente, INS fornisce una localizzazione continua basata sui tre assi, realizzata utilizzando i dati provenienti dall'accelerometro. L'utilizzo di questo sensore nella navigazione indoor non è banale,

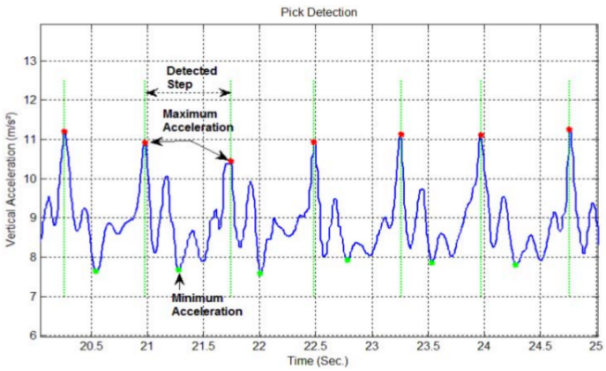

Figura 1.3: Esempio di peak detection [8].

infatti durante una camminata esso è soggetto a diversi fattori che possono indurre degli errori di misurazione. Uno dei fattori fondamentali è la forza di gravità che influisce sulle misurazioni; è quindi opportuno tenere conto della sua influenza. Un altro fattore molto influente è che durante un percorso a piedi il sensore non è perfettamente allineato con gli assi reali, esso ruota continuamente seguendo i movimenti del corpo di chi lo indossa, e per questo, nelle implementazioni, viene sempre affiancato un altro sensore, il giroscopio, che è in grado di misurare le rotazioni subite. In aggiunta, alcuni ricercatori [9], hanno provato ad utilizzare un ulteriore sensore per filtrare al meglio i dati dell'accelerometro: il *magnetometro*. Questi due sensori di "appoggio" all'accelerometro hanno caratteristiche complementari: il giroscopio fornisce un orientamento a lungo termine, mentre il magnetometro percepisce il cambiamento di orientamento a breve termine. Inserendo i dati di questi tre sensori all'interno di un **Filtro di Kalman**, filtro ricorsivo che aiuta a ridurre gli errori nelle misurazioni e che è stato adattato al contesto si sono ottenuti ottimi risultati in termini di accuratezza [2].

1.2.4 Step-and-Heading Systems

In molti scenari indoor la traiettoria in 3D dei sensori non è necessaria ed è sufficiente una navigazione in 2D nel piano parallelo al terreno usando dei

vettori di spostamento; questo concetto è alla base della tipologia SHS. Essa, come anticipato precedentemente, si basa su tre informazioni fondamentali: *posizione iniziale o attuale*, *lunghezza del passo* e *direzione del passo*. La *posizione iniziale* è un prerequisito fondamentale per tutte le tecniche di PDR, poiché non c'è alcun modo per dedurre questa informazione. Discorso diverso per quanto riguarda la *posizione attuale*, infatti conoscendo la posizione iniziale e applicando SHS in modo ricorsivo è possibile ottenere la posizione attuale [2]. La *lunghezza del passo* è il punto cruciale di questo metodo poiché ogni individuo cammina in modo diverso e con velocità diverse, a tal proposito in letteratura sono state analizzate alcune tecniche. La più semplice prevede di fissare una lunghezza del passo statica per ogni utente dal momento che sono state trovate delle correlazioni tra l'altezza dell'individuo e la lunghezza del passo [3]. Un'altra tecnica, derivata da uno studio in ambiente medico, prevede l'utilizzo di dati, come la frequenza e la velocità del passo, per determinarne l'ampiezza [10] [11]. Un altro metodo, difficilmente realizzabile per la complessità dell'hardware utilizzato, si può applicare posizionando dei sensori ad ultrasuoni nel fronte e nel retro di ogni scarpa [12]. Infine, la soluzione per calcolare la lunghezza del passo più accurata, più utilizzata in letteratura e più idonea all'utilizzo nell'applicazione implementata è quella proposta da H. Xing e altri [3], X. Wang e altri [13] e H. Weinberg [6]. Il loro algoritmo prevede di analizzare i dati provenienti dall'accelerometro e, ogni qualvolta il dispositivo percepisce un passo, è possibile ottenere la distanza percorsa dal punto precedente attraverso la formula:

$$step\_length = k * \sqrt[4]{A_{max} - A_{min}}$$

dove k è un parametro di conversione tra le unità di misurazione (ove necessario), A_{max} e A_{min} sono rispettivamente l'accelerazione massima e minima che l'accelerometro rileva sull'asse verticale durante il compimento di un passo. Un altro metodo efficace è quello di utilizzare una rete neurale BP-ANN (Back Propagation Artificial Neural Network) [3] che tuttavia necessita di una infrastruttura per poter essere messo in atto. Infine l'ultimo parametro fondamentale della tipologia SHS, la *direzione del passo*, può essere stimata

costruendo una bussola utilizzando i dati provenienti dal magnetometro e dal giroscopio [2] [8].

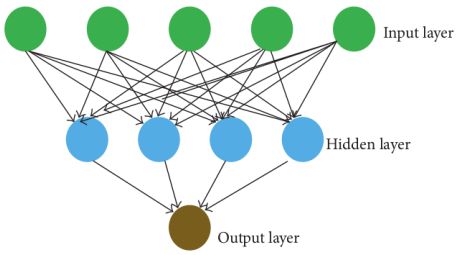

Figura 1.4: Classica architettura BP-ANN [3].

Per quanto riguarda la tipologia SHS non rimane altro che illustrare come utilizzare questi dati, posizione iniziale/attuale, lunghezza del passo e direzione del passo per poter stimare la posizione di un utente. Come anticipato precedentemente, uno dei metodi che è possibile utilizzare è quello di inserire i dati ottenuti in un *Filtro di Kalman*. Per conoscere la posizione corrente di un utente dopo aver effettuato un passo si inseriscono i dati collezionati nel seguente sistema di equazioni:

$$X_i = \begin{pmatrix} x_{i-1} \\ y_{i-1} \\ \beta_{i-1} \end{pmatrix} + \begin{pmatrix} \cos\beta_{i-1} \\ \sin\beta_{i-1} \\ 0 \end{pmatrix} * u_{i-1}, \tag{1.1}$$

dove x_{i-1} è la coordinata est della posizione precedente, y_{i-1} è la coordinata nord della posizione precedente, β_{i-1} è la direzione verso la quale si intende effettuare il passo successivo i, $\cos\beta_{i-1}$ è lo spostamento verso est, $\sin\beta_{i-1}$ è lo spostamento verso nord e u_{i-1} è la lunghezza del passo. Il risultato sarà il nuovo stato dopo aver effettuato un passo [8].
Dal momento che ci si muove in uno scenario 2D, la nuova posizione di un utente dopo un passo può essere espressa attraverso la coppia di coordinate (x, y) posizionate in un piano Cartesiano. Un metodo semplice per stimare queste coordinate è utilizzare il *filtro di Kalman* attraverso le seguenti

equazioni:

$$x_i = x_{i-1} + D_n * \cos\Theta_n, \tag{1.2}$$

$$y_i = y_{i-1} + D_n * \sin\Theta_n, \tag{1.3}$$

dove x_{i-1} e y_{i-1} sono le coordinate (x, y) al passo precedente, D_n è la lunghezza del passo e Θ_n è la direzione del passo [6].
Un metodo per ridurre il rumore dei sensori in un sistema SHS è quello di applicare la tecnica del **Particle Filters** (filtro particellare) che è un'approssimazione numerica del filtro di Bayes [14]. Essa è composta da *particelle*, ognuna delle quali rappresenta una possibile posizione in 2D. Alcune di queste posizioni sono più probabili di altre, quindi ad ognuna di esse viene assegnato un *peso* direttamente proporzionale alla probabilità che un utente si trovi in quella posizione calcolata in base alle informazioni fornite. Questa tecnica è iterativa ed è composta da tre passi per ogni iterazione:

1. **Update**: ogni particella è posizionata nello scenario 2D basandosi sul modello costruito con i movimenti precedenti (lunghezza del passo precedente, direzione del passo precedente);

2. **Correct**: ad ogni particella viene assegnato un peso in base alle similarità tra gli spostamenti precedenti e il vettore di movimento stimato del passo corrente (lunghezza del passo stimata, direzione del passo stimata);

3. **Resample**: un nuovo gruppo di particelle viene generato tenendo conto del gruppo di particelle correnti con il rispettivo peso.

Per ogni particella corrente con stato (x_t, y_t, Θ_t) nel passo 2 vengono assegnati i pesi per ogni particella del gruppo successivo attraverso il sistema di equazioni:

$$x_{t+\delta t} = x_t + (l + n_l)\cos(\delta\Theta + n_\Theta), \tag{1.4}$$

$$y_{t+\delta t} = y_t + (l + n_l)\sin(\delta\Theta + n_\Theta), \tag{1.5}$$

$$\Theta_{t+\delta t} = \Theta_t + \delta\Theta + n_\Theta, \tag{1.6}$$

dove l è la lunghezza del passo, $\delta\Theta$ è la variazione di direzione del passo, n_l è il rumore presente nel modello di stima della lunghezza del passo e n_Θ è il rumore presente nel modello di stima della direzione del passo [2].

1.2.5 In sintesi

In questa sezione si sono visti alcuni dei sistemi di localizzazione basati su sensori inerziali integrati all'interno dei moderni dispositivi mobili oppure indossati direttamente dall'utente nei propri indumenti. Si sono visti due differenti tipologie di PDR: INS, che fornisce una localizzazione in 3D, e SHS, che fornisce una localizzazione in 2D. Sono stati descritti alcuni metodi di riconoscimento del *Gate Cycle*: *stance detection* e *peak detection* applicando *low-pass filter* e *template matching*. Sono state mostrate anche le varie metodologie utilizzate per la *stima della lunghezza del passo* e *stima della direzione del passo*. Infine si sono visti tre metodi di stima della posizione, due con riduzione del rumore e uno senza riduzione del rumore.

1.3 Sistemi di localizzazione attraverso radiofrequenze

Le tecnologie basate su **radiofrequenze** occupano una posizione ormai stabile all'interno della vita di tutte le persone che utilizzano i dispositivi di ultima generazione. Giusto per citarne alcune, basti pensare alla tecnologia *WiFi* o alla *rete dati cellulare* per la connessione alla rete Internet, alla tecnologia *Bluetooth* per lo scambio di file tra diversi dispositivi. Oltre che in applicazioni end-user, queste tecnologie, sono entrate anche nei meccanismi industriali (ad esempio nel controllo della qualità dei prodotti), medici (apparecchiature che utilizzano radiofrequenze per degli esami clinici, ecc...), finanziari (sistemi di pagamento contactless, ecc...), logistici (accertamento del contenuto di un contanier attraverso RFID, ecc...), della pubblica sicu-

rezza (sistemi di monitoraggio e di comunicazione, ecc...).

1.3.1 Topologie di localizzazione indoor basate su radiofrequenze

Nel corso degli ultimi anni, a quelli sopracitati, si è aggiunto un ulteriore campo di applicazione, quello della indoor localization attraverso le radiofrequenze.
Un sistema di localizzazione indoor basato su radiofrequenze è generalmente composto da due tipi di dispositivi: uno o più device fissi e uno mobile. Queste componenti possono assumere due diversi stati: emittente o ricevente. Da queste classificazioni possono scaturire quattro combinazioni che rappresentano le topologie di localizzazione [15]:

1. **Remote positioning system**: in questo caso la parte emittente è il nodo mobile e la parte ricevente è rappresentata dai nodi fissi. L'emittente trasmette il proprio segnale, esso viene catturato e immagazzinato dai ricevitori e successivamente viene passato ad una *master station* che calcola la posizione del dispositivo mobile;

2. **Self positioning system**: in questa seconda topologia il nodo mobile assume lo stato di ricevente e i nodi fissi sono gli emittenti. I dispositivi fissi inviano il segnale nell'ambiente e il nodo mobile è capace di catturare questi segnali e calcolare la propria posizione;

3. **Indirect remote positioning**: il funzionamento è simile al *self positioning system* ma in questo caso le misurazioni effettuate dal nodo mobile vengono inviate alla *master station* che calcolerà la posizione;

4. **Indirect self positioning**: il funzionamento è simile al *remote positioning system* ma in questa situazione le misurazioni effettuate dai dispositivi fissi vengono inviate al dispositivo mobile che calcola la posizione.

1.3.2 Tecnologie di localizzazione indoor basate su radiofrequenze

Come anticipato precedentemente, esistono diverse tecnologie che possono essere utilizzate per implementare un sistema di localizzazione indoor basato su radiofrequenze.
La prima tecnologia analizzata è quella basata sulla rete dati cellulare, in particolare quella denominata **Long Term Evolution**. LTE è una tecnologia di telecomunicazioni a cavallo tra la terza e la quarta generazione, fa parte dello standard 3GPP (Third Generation Partnership Project, accordo di collaborazione fra enti che si occupano di standardizzare sistemi di telecomunicazioni) ed è presentata come evoluzione delle tecnologie UMTS/HSPDA di terza generazione per la connessione alla rete Internet di dispositivi mobili. Si basa sulla tecnologia *OFDM* e può raggiungere velocità di 300MBPS in downlink e 75MBPS in uplink. Grazie alle sue caratteristiche può supportare applicazioni multimediali, giochi, mobile TV [16] e la localizzazione.
La seconda tecnologia analizzata è la IEEE 802.11 conosciuta commercialmente con il nome di **WiFi**. Basata sul protocollo di accesso multiplo con prevenzione delle collisioni *CSMA/CA*, ha un raggio di azione che si aggira intorno ai 20m, è soggetta alle attenuazioni provocate dagli ostacoli e permette l'accesso alla rete Internet ai dispositivi mobili forniti di questa tecnologia e agganciati agli Access Point (dispositivi statici equipaggiati con un'interfaccia di rete WiFi). Una sua variante, *WiFi Direct*, permette di scambiare dati direttamente tra dispositivi senza passare per un Access Point.
L'ultima tecnologia che viene presa in analisi in questo paragrafo è la IEEE 802.15.1 che prende il nome di **Bluetooth** (BT). È uno standard di comunicazione wireless per reti personali (WPAN - Wireless Personal Area Network) e permette di scambiare dati direttamente tra due dispositivi. Prende vita a cavallo tra gli anni 1999 e 2000 e inizialmente veniva utilizzata quasi esclusivamente per lo sharing di file multimediali tra due telefoni cellulari. Negli ultimi anni ha subito diverse evoluzioni e recentemente viene utilizzata maggiormente per scambiare dati tra lo smartphone e i dispositivi indossabili

come smartwatch e smartband. La sua ultima versione **Bluetooth 4.0** o **Bluetooth Low Energy** (BLE) ha portato un miglioramento significativo in termini di risparmio energetico, per questo si è pensato di utilizzarlo come tecnologia per l'indoor localization.

1.3.3 Algoritmi di triangolazione del segnale

Non è facile modellare la propagazione delle radiofrequenze in uno scenario indoor perché subiscono spesso attenuazioni, rifrazioni e rimbalzi causati dagli oggetti/mura presenti. In sostanza è quasi impossibile ottenere una **Line-of-Sight** (linea di "vista") tra l'emittente e il ricevente.
La prima tecnica è detta **triangolazione** perché sfrutta le proprietà geometriche del triangolo per stimare la posizione dell'obiettivo [1]. Ci sono due algoritmi di calcolo che derivano dalla triangolazione: *laterazione* e *angolazione*.

Laterazione

La **laterazione** stima la posizione di un dispositivo misurando la distanza tra diversi punti di riferimento. Le unità di misura utilizzate possono essere diverse:

- **TOA** - Time Of Arrival: la distanza di un dispositivo da un punto di riferimento è direttamente proporzionale al tempo di propagazione. Per localizzarsi in uno scenario 2D il numero minimo di *base-station* da prendere in considerazione è tre. Data la posizione sconosciuta del terminale mobile (x_0, y_0) che trasmette un segnale al tempo t_0, le N *base-station* posizionate nelle coordinate $(x_1, y_1), ..., (x_N, y_N)$ che ricevono il segnale ai tempi $t_1, ..., t_N$, la posizione del terminale è ottenibile minimizzando la seguente funzione:

$$F(\bar{x}) = \sum_{i=1}^{N} \alpha_i^2 f_i^2(\bar{x}) \tag{1.7}$$

dove α_i è scelto in base all'affidabilità del segnale ricevuto con quantità misurata i, e $f_i(x)$ è calcolato con la seguente equazione:

$$f_i(\bar{x}) = c(t_i - t) - \sqrt{(x_i - x)^2 + (y_i - y)^2}, \qquad \forall i = 1, ..., N \tag{1.8}$$

dove c è la velocità della luce e $\bar{x} = (x, y, t)^T$. Tuttavia TOA presenta una criticità non banale, ossia tutti i dispositivi (mobili e *base-station*) debbono essere perfettamente sincronizzati [1].

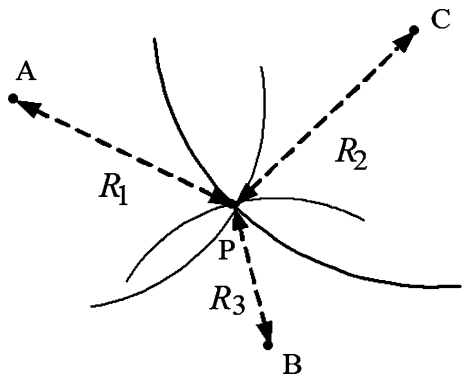

Figura 1.5: Localizzazione basata su TOA/RTOF [1].

- **TDOA** - Time Difference Of Arrival: l'idea è quella di determinare la posizione relativa del nodo mobile esaminando la differenza dei tempi di arrivo attraverso misurazioni multiple nel tempo. Per ogni segnale emesso dal nodo mobile, ogni *base-station* costruisce un'iperbole con un raggio direttamente proporzionale al tempo di arrivo e la posizione del dispositivo è stimata nell'intersezione delle iperbole. Date le coordinate dei ricevitori i e j rispettivamente (x_i, y_i, z_i) e (x_j, y_j, z_j), e le coordinate del nodo mobile (x, y, z) l'equazione per costruire l'iperbole è [1]:

$$R_{i,j} = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \quad - \\ \sqrt{(x_j - x)^2 + (y_j - y)^2 + (z_j - z)^2} \tag{1.9}$$

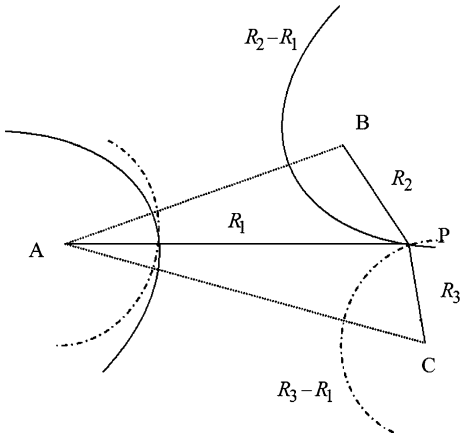

Figura 1.6: Localizzazione basata su TDOA [1].

- **RSS** - Received Signal Strength: è una delle tecniche più utilizzate nel campo della localizzazione. RSS si basa sull'attenuazione derivata dalla propagazione del segnale che viene calcolata sia teoricamente che empiricamente. Data la potenza del segnale emesso $p(R_0)$, la potenza del segnale ricevuto $p(R)$ viene calcolata secondo la formula:

$$p(R) = p(R_0) - 10n \log(\frac{R}{R_0}) \tag{1.10}$$

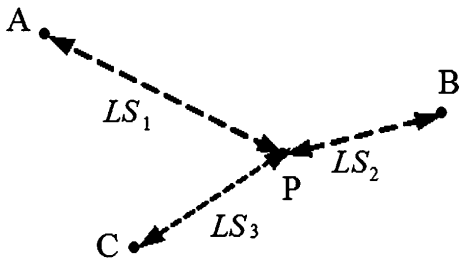

Figura 1.7: Localizzazione basata su RSS [1].

- **RTOF** - Roundtrip Time Of Flight: questo metodo misura il tempo che impiega un segnale per andare dall'emettitore al ricevitore e tornare

indietro fino all'emettitore. La distanza è calcolabile con l'equazione:

$$distance = \frac{(T_{arrival} - T_{sent}) * c}{2} \tag{1.11}$$

dove c è pari alla velocità della luce, $T_{arrival}$ e T_{sent} sono rispettivamente l'istante di arrivo e di partenza del segnale. Grazie a questa tecnica si risolverebbero i problemi di sincronizzazione introdotti nel metodo TOA, tuttavia soggiunge un'ulteriore problematica: infatti nella *base-station* potrebbe presentarsi una coda di segnali da inviare e quindi non si riuscirebbe a garantire l'inoltro immediato del segnale ricevuto.

Angolazione

Le tecniche di **angolazione** (AOA - *Angulation of Arrival*) misurano la posizione di un oggetto attraverso l'intersezione di diverse coppie di *linee di direzione angolari*, ognuna di esse composta dal raggio che si forma partendo dalla *base-station* fino al nodo obiettivo. Per applicare questo metodo sono sufficienti due dispositivi fissi, A e B, e due angoli calcolati, Θ_1 e Θ_2. Le tecniche di angolazione non necessitano di sincronizzazione tra dispositivi, tuttavia è difficile applicarle perché è necessario un hardware specifico e costoso capace di calcolare le angolazioni [1].

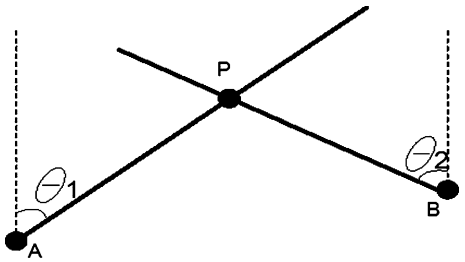

Figura 1.8: Localizzazione basata su AOA [1].

1.3.4 Algoritmi di posizionamento con sistemi di fingerprinting

Come visto nella sezione precedente esistono diverse tecnologie che possono essere utili per sviluppare un servizio di localizzazione indoor. La quasi totalità dei lavori presenti in letteratura utilizzano un semplice ma efficace metodo per localizzare un dispositivo attraverso le radiofrequenze emesse che prende il nome di **Radio Fingerprinting**. Esso si compone in due fasi distinte: **offline phase** e **online phase**.

Offline phase

Durante la fase offline viene creata una *Radio-Frequency Map* dello scenario nel quale si intende applicare la localizzazione [17]. Indipendentemente dalla tecnologia utilizzata, questa mappa viene creata posizionandosi in dei specifici punti di riferimento chiamati **Reference Point** (RPs) e campionando le radiofrequenze emesse dai trasmettitori posizionati in delle postazioni fisse che si trovano all'interno dello scenario e che sono percepibili da tale punto.

Online phase

Durante la fase online un utente, equipaggiato con un dispositivo mobile che integra la tecnologia utilizzata durante la fase offline, si muove nello scenario d'interesse, campiona le radiofrequenze da una posizione sconosciuta (o viceversa come nel caso del *remote positioning system*) e le compara con i dati raccolti durante la fase offline per stimare la posizione dell'utente [17].

La fase offline è quella più lunga e delicata durante un algoritmo di fingerprinting: per questo motivo alcuni ricercatori hanno tentato di velocizzare questa procedura utilizzando un approccio crowdsensing [18] [19] [20] su uno schema di pattern matching e spesso sviluppando un approccio di Machine

Learning [21] [22].
In questa sezione si descrivono alcuni metodi di posizionamento basati sulla tecnica del *fingerprinting*.

Metodi probabilistici

Il primo metodo considera il posizionamento come un problema di classificazione. Supponendo di avere n locazioni candidate L_1, ..., L_n, e s è la potenza del segnale rilevata durante la fase online, può essere ottenuta la seguente regola di decisione [1]:

$$\text{Scegliere } L_i \text{ se } P(L_i|s) > P(L_j|s), \text{ per } i, j = 1, ..., n \text{ e } i \neq j \tag{1.12}$$

dove $P(L_i|s)$ è la probabilità che il nodo mobile si trovi nella posizione L_i avendo ricevuto la potenza di segnale s. Tuttavia questa tecnica può fornire una *posizione simbolica* ma per ottenere una *posizione fisica* si possono ricavare le probabilità delle coordinate (x, y) attraverso la seguente formula [1]:

$$(x, y) = \sum_{i=1}^{N} (P(L_i|s)(x_{L_i}, y_{L_i})) \tag{1.13}$$

KNN (K-Nearest-Neighbor)

Questa tecnica fa uso degli RSS raccolti durante la fase online per trovare le k posizioni conosciute più vicine e precedentemente immagazzinate in un database durante la fase offline. Applicando il calcolo della distanza Euclidea tra queste k posizioni conosciute è possibile ricavare la posizione del nodo mobile [1].

Reti neurali

Durante la fase offline vengono immagazzinate alcune posizioni con i relativi RSS misurati e vengono forniti come input per il training di una rete neurale. Per il sistema di posizionamento viene di solito utilizzata una rete *Multilayer Perceptron* con un livello nascosto. Durante la fase online vengono

collezionati gli RSS dal nodo mobile e vengono moltiplicati per i pesi ottenuti durante la fase di addestramento della rete neurale. I risultati vengono poi passati alla funzione di trasferimento del livello nascosto. L'output di questa operazione viene moltiplicato per la matrice dei pesi ottenuta dall'allenamento del livello nascosto. Il risultato del sistema è un vettore di due elementi che rappresenta la posizione stimata [23] [24].

SVM (Support Vector Machine)

È un nuovo approccio che si basa sulla classificazione. Consiste in un tool che combina analisi statistiche e *Machine Learning* [1].

SMP (Smalles M-vertex Polygon)

Con l'uso di questa tecnica vengono raccolti gli RSS provenienti dalle *base-station* e vengono scelti i *reference point* più probabili tenendo conto della potenza del segnale ricevuto prendendo singolarmente ogni nodo fisso. Viene creato un poligono con M-vertici scegliendo almeno una *base-station* per ogni RP. Per ricavare la posizione del nodo mobile si calcola la media dei segnali ricevuti prendendo i vertici del poligono con l'area minore [25].

1.3.5 Algoritmi di prossimità

Gli algoritmi di prossimità forniscono una posizione simbolica di un nodo mobile. Per applicare questa tecnica vengono disposte delle antenne a corto raggio (BLE, RFID, ...) nello scenario di interesse: ogni volta che un dispositivo mobile aggancia una delle antenne, ossia percepisce il *beacon* inviato da esse, si considera che il nodo si trovi molto vicino (nel raggio di qualche metro) da quella posizione [1].
Come già menzionato *Bluetooth Low Energy* è una tecnologia performante per applicare questo tipo di algoritmo. Dato il corto raggio del segnale radio emesso da queste antenne è logico che lo scenario debba essere molto denso di dispositivi. I ricercatori si sono concentrati maggiormente sul trovare un

algoritmo che permettesse di avere una buona copertura di segnale cercando di non far diventare questa tecnica troppo dispendiosa in termini di costi [26].

1.3.6 Metriche di performance

In letteratura si sono rinvenute le seguenti metriche di performance dei sistemi [1]:

- **Accuratezza**: misura quanto la posizione stimata dal sistema sia vicina a quella reale: generalmente viene calcolata attraverso la distanza Euclidea tra le due posizioni. Più la distanza è breve e più il sistema è accurato, più il sistema è accurato e più il sistema è migliore;
- **Precisione**: misura la forza dell'accuratezza del sistema, ossia per quanti metri/passi il sistema costruito riesce a mantenere una buona accuratezza;
- **Complessità**: misura la complessità nella costruzione del sistema in termini di software, hardware e costo computazionale;
- **Robustezza**: misura la forza del sistema nel tempo, ossia se l'algoritmo è capace di funzionare anche dopo la modifica dello scenario (ad esempio: oggetti spostati, *base-station* rimosse;
- **Scalabilità**: misura quanto il sistema sia scalabile, ossia ci dice quanto un sistema è applicabile in base alle dimensioni dello scenario di interesse;
- **Costo**: misura i costi in termini di risorse umane, tempo, energia, denaro ecc...

1.3.7 In sintesi

In questa sezione si sono descritte le tecnologie principali che permettono la localizzazione indoor tramite radiofrequenze come WiFi, Bluetooth e

LTE. Si sono mostrate le quattro topologie di localizzazione indoor basate su radiofrequenze (Remote positioning system, Self positioning system, Indirect remote positioning system e Indirect self positioning system). Si sono anche descritti i principali algoritmi che permettono di realizzare questi sistemi (triangolazione, fingerprinting e prossimità) e per ognuno di essi sono stati forniti dettagli accurati su come modellarli. Alla fine della sezione, invece, sono stati riportate le metriche più utilizzate in letteratura per comparare sistemi differenti.

1.4 Sistemi ibridi: dead reckoning e radiofrequenze

Quelli descritti nei paragrafi precedenti sono i due approcci maggiormente utilizzati in letteratura per sviluppare un sistema di localizzazione indoor. Questi sistemi tuttavia, se utilizzati in modo atomico, non hanno soddisfatto alcune delle metriche di valutazione presentate all'inizio di questo capitolo Solo per citare qualche esempio, la tecniche basate sui sensori inerziali sono deficitarie in termini di *accuratezza* e *precisione*, infatti uno dei maggiori problemi di questi approcci è che al crescere del percorso compiuto l'accumulo di errori porta ad una localizzazione inaccurata. D'altro canto le tecniche basate su radiofrequenze non risultano essere performanti dal punto di vista della *robustezza*, dei *costi* e della *scalabilità*: se il sistema utilizzato è quello del *WiFi-Fingerprinting* o *LTE-Fingerprinting* ogni piccola modifica dell'ambiente target può causare una localizzazione inaccurata. Inoltre il costo energetico è elevato a causa del continuo *scanning* per localizzare le *base-station* presenti da parte della scheda di rete del nodo mobile; se il sistema invece utilizza *BLE-Firngerprinting* si ottiene una buona robustezza ma i costi di realizzazione per poter ottenere una copertura minima su tutto lo scenario salgono in maniera direttamente proporzionale alla grandezza del contesto causando così una limitata scalabilità del sistema.
Per questi motivi, nel corso degli ultimi anni, i ricercatori hanno iniziato ad

approcciarsi a questo problema costruendo sistemi ibridi, che unissero tecniche di *Dead Reckoning* e tecniche di *radio-fingerprinting*, al fine di creare sistemi quanto più performanti e che potessero porre rimedio alle lacune dei singoli approcci. In questo paragrafo vengono descritti alcuni sistemi ibridi presenti in letteratura.

1.4.1 Dynamic subarea e positioning fusion

Il primo lavoro preso in considerazione è stato sviluppato da ricercatori provenienti dall'Università Jiaotong di Xi'an con la collaborazione del reparto di ricerca di Huawei Technologies [27]. Il loro sistema ibrido utilizza i due metodi, PDR e WiFi-Fingerprinting, per migliorare principalmente l'accuratezza del posizionamento. La loro ricerca si basa su due punti chiave: **dynamic subarea** e **positioning fusion**.

Dynamic subarea

La tecnica della *dynamic subarea* è un metodo ideato ed implementato dagli stessi ricercatori che hanno redatto questo articolo. L'algoritmo KNN visto in precedenza seleziona i k *Reference Points* basandosi solamente sulla distanza Euclidea minima calcolata attraverso gli RSS. Considerando che gli RSS subiscono alterazioni dovute agli oggetti presenti nello scenario è possibile che vengano considerati tra i "più vicini" degli RP che si trovano molto distanti dalla posizione reale del nodo mobile e questo può portare a problemi di accuratezza. L'idea dei ricercatori è stata quindi quella di costruire una sotto-area di ricerca dei k RP più vicini intorno alla posizione reale dell'utente che cambia con il cambiare della posizione dell'utente (per questo definita dinamica), in modo da ridurne la complessità computazionale e migliorarne l'accuratezza. Dal momento che la posizione reale non è conosciuta, essa viene rimpiazzata dalla posizione stimata attraverso il metodo PDR.

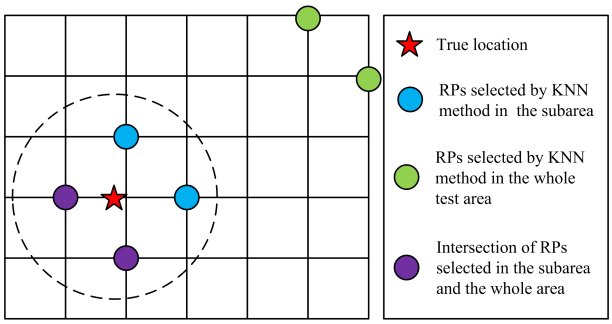

Figura 1.9: Metodo della dynamic subarea [27].

Positioning fusion

Il punto cruciale del lavoro dei ricercatori è stato quello di creare un algoritmo che ottimizzi la fusione dei due metodi menzionati precedentemente. Il loro lavoro si basa, oltre che sulla *dynamic subarea*, su un particolare accorgimento: dal momento che la lunghezza del passo è generalmente inferiore ad $1m$, allora la distanza tra due posizioni stimate consecutive deve necessariamente essere inferiore a tale valore. Se la posizione stimata corrente è distante oltre $1m$ dalla precedente allora vuol dire che si è verificato un errore da parte dei sensori. Da questo accorgimento si può dedurre che l'accuratezza del posizionamento è inversamente proporzionale alla distanza tra la posizione corrente e la posizione precedente. L'algoritmo finale è quindi:

1. **Inizializzazione dei parametri**: viene inserita la posizione iniziale dell'utente $L_0 = (x_0, y_0)$, dove x_0 rappresenta la coordinata x e y_0 rappresenta la coordinata y.

2. **Posizione PDR**: quando un passo, di lunghezza D_n e di direzione Θ_n, viene percepito dai sensori viene calcolata la posizione $L_{pdr,n} = (x_{pdr,n}, y_{pdr,n})$ che si basa sulla posizione precedente attraverso le formule:

$$x_{pdr,n} = x_{n-1} + D_n * \cos(\Theta_n), \tag{1.14}$$

$$y_{pdr,n} = y_{n-1} + D_n * \sin(\Theta_n). \tag{1.15}$$

3. **Posizione WiFi**: si calcola la posizione stimata attraverso il metodo *WiFi-Fingerprinting* con l'ausilio della tecnica della *dynamic subarea* impostando la posizione precedente $L_{n-1} = (x_{n-1}, y_{n-1})$ come centro dell'area di ricerca dei k RP vicini e ottenendo la posizione corrente $L_{wifi,n} = (x_{wifi,n}, y_{wifi,n})$

4. **Stima della posizione**: viene calcolata la posizione dell'utente assegnando dei pesi ai due metodi tenendo conto degli accorgimenti descritti precedentemente. Inizialmente si calcola la distanza tra la posizione attuale stimata dal metodo PDR e quella precedente attraverso la formula della distanza Euclidea:

$$dis_{pdr} = \sqrt{(x_{pdr,n} - x_{n-1})^2 + (y_{pdr,n} - y_{n-1})^2}. \tag{1.16}$$

Successivamente viene calcolata la distanza tra la posizione attuale stimata dal metodo WiFi-Fingerprinting e quella precedente con la stessa formula:

$$dis_{wifi} = \sqrt{(x_{wifi,n} - x_{n-1})^2 + (y_{wifi,n} - y_{n-1})^2}. \tag{1.17}$$

Si ottiene poi il peso r del metodo WiFi-Fingerprinting attraverso la formula:

$$r = \frac{\frac{1}{dis_{wifi}}}{\frac{1}{dis_{wifi}} + \frac{1}{dis_{pdr}}}, \tag{1.18}$$

e si ottengono le coordinate della posizione ibrida stimata $L_n = (x_n, y_n)$:

$$x_n = (1 - r) * x_{pdr,n} + r * x_{wifi,n}, \tag{1.19}$$

$$y_n = (1 - r) * y_{pdr,n} + r * y_{wifi,n}. \tag{1.20}$$

5. Tornare al **passo 2** e ripetere l'algoritmo.

Algoritmo migliorato

Per migliorare l'accuratezza di questo algoritmo un'ulteriore considerazione è stata fatta dai ricercatori: dal momento che gli RSS captati dai sensori

in due posizioni adiacenti possono essere identici, la posizione stimata dal metodo WiFi-Fingerprinting può risultare essere la stessa del passo precedente. Dato questo accorgimento si sono resi conto che quando si compiono due passi adiacenti la fusione dei due metodi introduce un errore nella stima della posizione finale, perché il percorso fatto risulta essere più corto rispetto a quello reale. Per questo motivo se durante il passo 3 si ottiene $L_{wifi,n-1} = L_{wifi,n}$ allora si può saltare la fusione dei due metodi nel **passo 4** e la posizione finale verrà stimata unicamente dal metodo PDR.

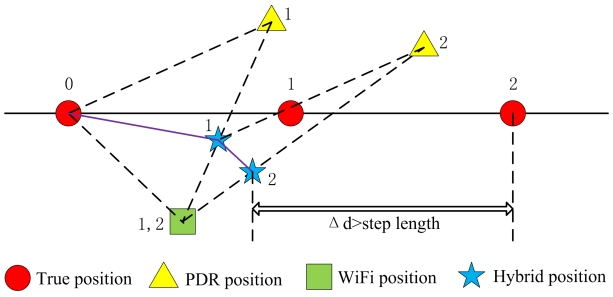

Figura 1.10: Esempio di due posizioni WiFi identiche [27].

Performance

Gli esperimenti condotti dai ricercatori sulle performance di questo sistema ibrido hanno portato a buoni risultati, infatti l'algoritmo ibrido con *dynamic subarea* ha ridotto gli errori di posizione del 58% se confrontato con il metodo WiFi-Fingerprinting, del 42% se confrontato con il metodo PDR e del 28% se confrontato con un algoritmo ibrido uguale a quello descritto ma senza l'utilizzo della *dynamic subarea*. Inoltre l'algoritmo migliorato ha portato una riduzione del 26% degli errori se confrontato con l'algoritmo ibrido non migliorato.

1.4.2 Sistema ibrido per l'aggiornamento automatico delle mappe radio: AcMu

Il secondo lavoro che interseca il funzionamento di due approcci visti precedentemente è quello stilato da alcuni ricercatori dell'Università Tsinghua di Pechino con la collaborazione di un ricercatore dell'Università del Michigan [20]. Il loro lavoro punta a migliorare la *robustezza* del metodo WiFi-Fingerprinting creando un sistema che sia capace, grazie ai dati provenienti dal metodo PDR, di aggiornare la mappa radio di un ambiente indoor.

Mappatura e misurazioni preliminari

Lo studio condotto dai ricercatori ha portato all'evidenza che, come anticipato nei paragrafi precedenti, i segnali RSS provenienti dalle *base-station* campionati nello stesso posto possono subire delle variazioni a distanza di pochi giorni derivate da vari fattori ambientali (temperatura, umidità, ecc.) o da fattori umani (spostamento di oggetti, apertura e chiusura delle porte, ecc.). Generalmente durante la fase offline del metodo *fingerprinting* viene immagazzinata una mappa $L = (l_1, ..., l_n)$ dove n è il numero di *reference point* scelti per il campionamento e ognuno di essi è caratterizzato dalle coordinate $l_i = (x_i, y_i)$ per $1 \leq i \leq n$. Vengono immagazzinati i campionamenti delle frequenze $F = (f_1, ..., f_n)$ dove $f_i = (f_{i1}, ..., f_{ip})$ è il fingerprint relativo alla posizione l_i, f_{ij} denota il valore RSS della $j - esima\ base\ station$ per $1 \leq j \leq p$, e p è il numero totale di *base station* percepibili nel determinato RP. Nel sistema prodotto dai ricercatori viene aggiunto un ulteriore livello: vengono immagazzinate nel database le mappe che mutano durante il tempo. La mappa radio RM_{i-1} immagazzinata al tempo t_{i-1} viene rimpiazzata dalla mappa RM_i creata al tempo t_i cosicché gli utenti possano beneficiare sempre degli ultimi aggiornamenti delle radiofrequenze nel contesto in esame.

Algoritmo di aggiornamento della mappa radio

Come anticipato, la mappa radio viene aggiornata grazie al metodo PDR e si illusta in questo paragrafo il funzionamento dell'algoritmo. I dati che permettono alla mappa di essere aggiornata vengono raccolti in ogni momento dagli utenti che si muovono nello scenario di interesse. Quando un utente cammina viene stimata la sua posizione grazie al metodo PDR calcolando la traiettoria percorsa e, in tempo reale, vengono raccolti i segnali provenienti dalle *base station*. Quando l'utente è fermo per un determinato lasso di tempo, i dati raccolti vengono inviati a un server che si occupa di analizzare i dati della traiettoria e gli RSS percepiti durante il percorso. Lo stesso server si occupa di aggiornare la mappa e renderla disponibile a tutti gli utenti.

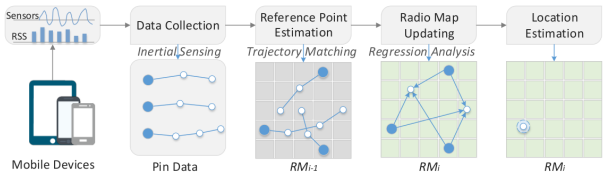

Figura 1.11: Architettura del sistema AcMu [20].

1.5 Altri sistemi di localizzazione indoor

Quelli presentati fino ad ora sono i sistemi di localizzazione indoor più utilizzati ma esistono altre tecniche altrettanto valide in questo contesto. In questa sezione si vedranno, non in modo approfondito, due tecniche di localizzazione indoor alternative a quelle presentate precedentemente che tuttavia non vengono prese in considerazione nella realizzazione del sistema in esame in questa tesi.

Tecniche basate su ultrasuoni

La prima delle due tecniche è quella che basa il suo funzionamento sugli **ultrasuoni** per calcolare la distanza tra una postazione fissa e un nodo mobile [28]. Per poter realizzare questo sistema sono necessari numerosi ricevitori, equipaggiati con la tecnologia ad *ultrasuoni* e *onde radio*, sincronizzati l'uno con l'altro. Per calcolare la distanza tra i nodi fissi e il nodo mobile vengono sfruttate le caratteristiche fisiche dei due tipi di segnale. Dal momento che le onde radio viaggiano alla velocità della luce, che è più alta rispetto alla velocità del suono utilizzata dalla tecnologia ad *ultrasuoni*, un dispositivo emittente (mobile) lancia nello stesso istante un segnale radio e uno ad ultrasuoni e il ricevente calcola il tempo che trascorre tra l'arrivo del segnale radio e quello sonoro. In base alla differenza di arrivo si può calcolare la distanza tra i due dispositivi e utilizzare le tecniche viste nei paragrafi precedenti per stimare la posizione dell'utente. Questa tecnica ha un'accuratezza nell'ordine del *centimetro* [28] e non è molto dispendiosa in termini di costo, tuttavia presenta alcune criticità. Il primo problema è dovuto alla scarsa *robustezza* del sistema, infatti cambi di temperatura e di umidità influenzano notevolmente la velocità degli ultrasuoni, così da corromperne l'accuratezza. Il secondo problema è che questi dispositivi hanno un raggio di azione che varia da 1 a 5 metri, ciò comporta una scarsa scalabilità del sistema.

Tecniche basate sulla visione artificiale

La seconda tecnica analizzata è quella che fa uso della **visione artificiale** per capire la posizione dell'utente che tiene in mano un dispositivo equipaggiato con questa tecnologia [8]. L'algoritmo che porta allo sviluppo di questo sistema è composto da due parti principali. La prima parte è l'*esplorazione dell'ambiente*. Durante questa prima fase vengono catturate le immagini dell'ambiente di interesse e successivamente vengono studiate le immagini in cerca di punti di riferimento particolari al fine di riconoscere una determinata posizione. La seconda fase è quella della *localizzazione*. Durante questa fase un utente con a bordo un dispositivo avente la tecnologia in

questione si muove nell'ambiente indoor, la camera del suo dispositivo cattura le immagini e cerca i punti di riferimento conosciuti. Dal momento che questa operazione è troppo onerosa, dal punto di vista computazionale, per un dispositivo mobile, le immagini catturate durante la seconda fase vengono inviate ad un server e processate. Una volta terminata questa scansione la parte remota comunica al dispositivo il risultato contenente la posizione [8].

Capitolo 2

Progettazione

2.1 Introduzione alla progettazione

L'obiettivo di questa tesi è stato quello di costruire un sistema che fornisca una localizzazione indoor accurata ma che al tempo stesso non sia eccessivamente costoso in termini di realizzazione e di risorse necessarie. Basandosi su quanto appreso durante lo studio dello stato dell'arte si è deciso di realizzare un sistema ibrido che unisca due tecniche di localizzazione indoor illustrate precedentemente: **localizzazione attraverso sensori inerziali** e **localizzazione attraverso radiofrequenze**. Pur avendo a disposizione molteplici strade da intraprendere per sviluppare il sistema, la scelta dei dispositivi e delle tecnologie da utilizzare è stata guidata dall'obiettivo prefissato. La necessità di acquisire dati provenienti da sensori inerziali e fonderli con dati provenienti da fonti radio ha fatto ricadere la scelta sullo sviluppo di un'*applicazione mobile* e un *servizio web* uniti in un'architettura **client-server** che sia capace di eseguire le tecniche **Pedestrian Dead Reckoning** e **WiFi-Fingerprinting**. In questo capitolo, in particolare, viene trattato il funzionamento del sistema e la sua **progettazione** specificando la sua architettura e le componenti utilizzate.

2.2 Architettura

In questa sezione viene presentata l'**architettura** su cui si basa il sistema sviluppato che è di tipo *Client-Server*.

2.2.1 Architettura Client-Server

L'architettura *Client-Server* appartiene allo stile architetturale a due livelli, denominati appunto *Client* e *Server*, ed è uno dei più utilizzati all'interno della rete Internet. In questa architettura uno o più **Server** mette a disposizione dei servizi che possono essere utilizzati da uno o più **Client** che fa/fanno parte della stessa rete del/dei Server.
Ognuno dei due livelli ha delle funzionalità differenti che deve soddisfare:

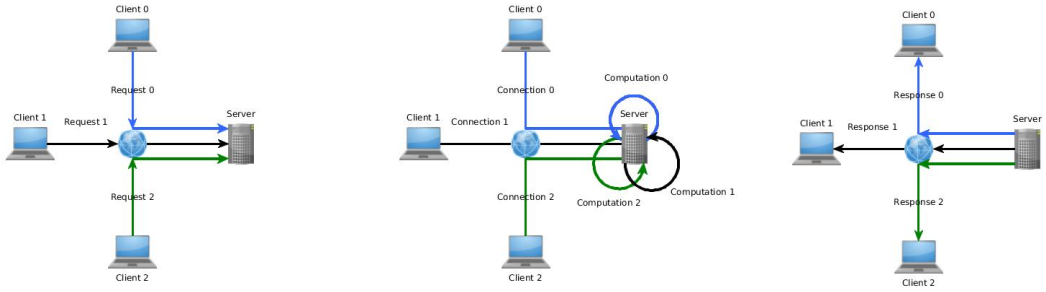

Figura 2.1: Architettura Client-Server.

- Client:
  - richiede un servizio attraverso l'interfaccia messa a disposizione dal server;
  - permette all'utente di interagire con il sistema attraverso degli input (grafici o testuali);
  - processa i dati inseriti dall'utente e li comunica al server;
  - riceve l'esito dell'esecuzione dal server.

- Server:
  - mette a disposizione un'interfaccia per poter ricevere richieste dai client;
  - fornisce dei servizi che possano essere usati dai client;
  - riceve e processa i dati forniti dai client;
  - invia i risultati della computazione ai client.

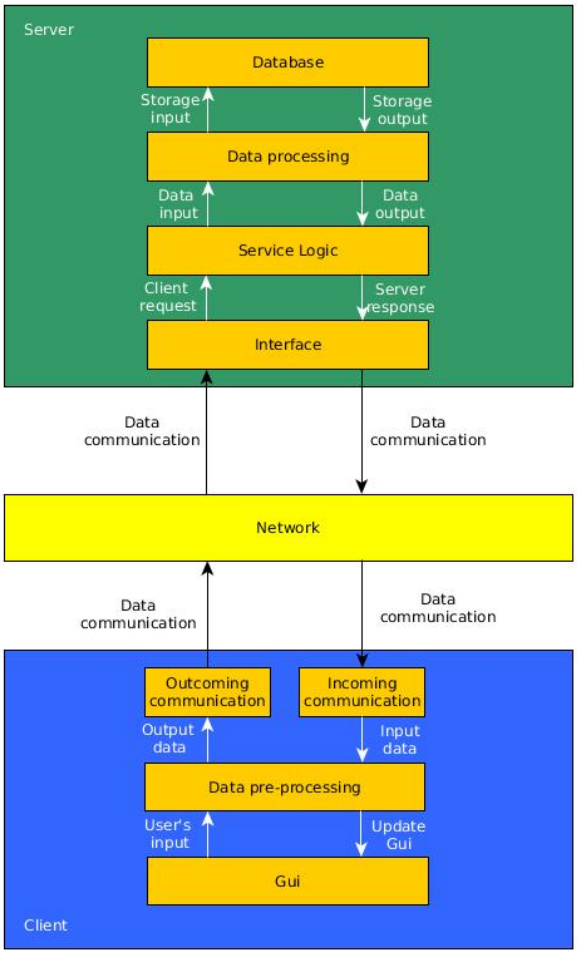

Figura 2.2: Funzionalità dei livelli

Un'architettura Client-Server può essere quindi utile in diversi contesti, la situazione più tradizionale si verifica quando più client necessitano di condividere una certa risorsa o un gruppo di risorse. Queste risorse posso essere appartenenti sia al mondo hardware, come stampanti e hard drive, che al mondo software, come un database o un determinato programma. Lo sharing delle risorse però non è l'unico scenario nel quale è utile l'utilizzo di un'architettura di questo tipo; infatti dei client, con poca capacità computazionale, possono scegliere di far eseguire al server delle operazioni onerose in termini di prestazioni. Nel caso di questo sistema, come si vedrà più avanti nel capitolo implementativo, l'architettura *Client-Server* è utile per entrambe le situazioni.

2.3 Client

Come appena anticipato in questo sistema agiscono due componenti principali: il client e il server. In questa parte del capitolo si illustrano tutti i ruoli ricoperti da ognuna delle parti, gli algoritmi e le tecniche utilizzate per acquisire e per fondere i dati.
Il primo ruolo presentato è quello del *client* che, come anticipato precedentemente, è interpretato da un'*applicazione mobile*. Questa scelta è stata dettata dal fatto che i moderni smartphone sono i dispositivi tecnologici più utilizzati dagli utenti medi, hanno un costo relativamente contenuto e sono equipaggiati con tutto l'hardware necessario per poter implementare il sistema. Le caratteristiche sono perfettamente conformi alle necessità che il sistema deve soddisfare.
Il client progettato si occupa principalmente di sei funzionalità:

1. inserimento e registrazione delle planimetrie dell'ambiente indoor di interesse;

2. raccolta dei dati provenienti dai sensori inerziali come accelerometro, giroscopio e magnetometro;

3. calcolo della posizione prendendo in input i dati forniti dai sensori;

4. raccolta dei dati provenienti dalle fonti radio presenti all'interno dell'ambiente nella fase offline;

5. raccolta dei dati provenienti dalle fonti radio presenti all'interno dell'ambiente nella fase online;

6. ricezione dell'esito della localizzazione radio effettuata dal server e fusione con i dati della localizzazione calcolata precedentemente dal client stesso.

Nei sottoparagrafi seguenti si illustrano nel dettaglio tutti i compiti eseguiti dal client.

2.3.1 Planimetrie

Dal momento che, come si evince dalla lista precedentemente stilata, il client si occupa della localizzazione indoor attraverso il sistema **Pedestrian Dead Reckoning** in maniera monolitica, ossia senza l'ausilio di sorgenti di dati esterne, è opportuno inserire al suo interno le informazioni riguardanti l'ambiente indoor nel quale l'utente deve essere localizzato. Premesso che il sistema utilizza una tecnica di *localizzazione* di tipo bidimensionale, usando delle coordinate (x, y) per localizzare un utente dentro l'ambiente, anche la planimetria dello scenario deve soddisfare questo requisito. La tecnica di *mapping* dell'ambiente può essere quindi riassunta nei seguenti punti:

1. acquisire la planimetria cartacea o digitale dell'ambiente di interesse;

2. sezionare la planimetria acquisita in celle di dimensioni fissate $1m$x$1m$ adottando la scala indicata nella planimetria;

3. fissare il punto $(0, 0)$ con il vertice del quadrato che si trova all'estremo inferiore sinistro ottenendo così un piano cartesiano di coordinate (x, y) positive;

4. individuare le varie stanze all'interno della planimetria e appuntare la rispettiva grandezza attraverso le coordinate (x_{min}, y_{min}), (x_{max}, y_{max}); se ci si trova di fronte ad una stanza di forma irregolare è opportuno scomporla in più figure regolari;

5. individuare i punti di interesse all'interno della planimetria (come ad esempio la posizione di estintori, uscite di emergenza ecc..) e indicarne le coordinate (x, y);

6. inserire i dati all'interno del client.

I dettagli implementativi vengono trattati nel capitolo seguente.

2.3.2 Raccolta dati dei sensori inerziali

Come si è visto nel capitolo dello stato dell'arte la tecnica *Pedestrian Dead Reckoning* si basa sui sensori inerziali come accelerometro, giroscopio e magnetometro per poter registrare gli spostamenti di un utente. Fornendo al sistema una posizione iniziale di partenza e tracciando tutti gli spostamenti dall'inizio della camminata è possibile conoscere la posizione corrente stimata dell'utente. Il client si occupa quindi delle seguenti mansioni:

1. riconoscimento del passo;

2. stima della lunghezza del passo;

3. calcolo della direzione del passo.

Riconoscimento del passo

Il sistema di riconoscimento del passo utilizzato in questa tesi è quello della **peak detection** illustrata nel paragrafo 1.2.2. Questa tecnica utilizza i dati delle accelerazioni sull'asse verticale registrati dall'accelerometro e vengono poi confrontati con dei pattern già noti in letteratura. Quando i dati seguono un andamento riconducibile a quelli dei pattern conosciuti allora un passo è stato percepito.

Stima della lunghezza del passo

Riguardo alla stima della lunghezza del passo sono stati utilizzati due algoritmi differenti provenienti dalla letteratura.
Il primo metodo, in questa tesi denominato **Static Step Length** (SSL), si basa su uno studio effettuato che dimostra una correlazione tra l'altezza e il sesso di un utente con la sua lunghezza del passo [2]. La funzione utilizzata per stimare la lunghezza del passo è:

$$SSL = \alpha * h, \qquad \alpha = \begin{cases} 0,415 & \text{se l'utente è di sesso maschile} \\ 0,413 & \text{se l'utente è di sesso femminile} \end{cases} \tag{2.1}$$

con h che rappresenta l'altezza dell'utente. Per questo motivo il client deve quindi essere in grado di registrare le caratteristiche dell'utente. Come si evincerà dai risultati sperimentali questa tecnica non risulta essere accurata poiché assume una lunghezza del passo costante e quindi accumula velocemente errori nel caso in cui l'utente abbia un'andatura non regolare. Il secondo metodo, in questa tesi denominato **Dynamic Step Length** (DSL), si basa su una ricerca che dimostra come si possa risalire alla lunghezza del passo analizzando i dati delle accelerazioni provenienti dall'accelerometro [3] [13][6]. Ogni volta che un passo viene percepito, con il metodo della *peak detection*, vengono registrati i dati dell'accelerometro e vengono individuate le accelerazioni massima (A_{max}) e minima (A_{min}) relative a quel passo e inserite nella seguente equazione:

$$DSL = k * \sqrt[4]{A_{max} - A_{min}}$$

dove k è un parametro di conversione tra le unità di misurazione se necessario.

Direzione del passo

In merito alla stima della direzione del passo il client sfrutta i dati provenienti dal giroscopio e dal magnetometro per determinare la direzione del

dispositivo prendendo come punto di riferimento il *nord magnetico*.
Un problema riguardante la direzione del passo è derivato dal fatto che durante la fase di mappatura dell'ambiente l'asse verticale del piano cartesiano potrebbe non essere rivolto verso il nord. Per questo motivo il client si deve occupare di una calibrazione iniziale del magnetometro in modo tale da creare un **nord locale** che potrebbe non coincidere con il nord magnetico.

2.3.3 Calcolo della posizione attraverso i sensori

L'ultimo passo riguardante la tecnica *Pedestrian Dead Reckoning* è quello del calcolo della posizione dell'utente. Come anticipato nel primo capitolo uno dei requisiti fondamentali di questa tecnica è quello di conoscere la posizione iniziale da cui iniziare a tracciare gli spostamenti dell'utente, quindi al momento dell'inserimento della planimetria è necessario specificare nel client quale sia la posizione iniziale.
Una volta conosciuta la posizione iniziale, ogni qualvolta l'utente compie un passo e viene percepito dal client, vengono letti i dati provenienti dai sensori attraverso i metodi descritti precedentemente e viene calcolata la nuova posizione dell'utente attraverso le equazioni:

$$x_i = x_{i-1} + D_n * \cos\Theta_n, \tag{2.2}$$

$$y_i = y_{i-1} + D_n * \sin\Theta_n, \tag{2.3}$$

dove x_{i-1} e y_{i-1} sono le coordinate (x, y) al passo precedente, D_n è la lunghezza del passo e Θ_n è la direzione del passo [6].

2.3.4 Raccolta dei dati provenienti dalle fonti radio: fase offline

La topologia della localizzazione è una variante del tipo **Indirect Remote Positioning**, ossia il client cattura le frequenze emesse dagli access

point e le invia ad una *master station*, che in questo caso si tratta del server, che si occuperà di applicare gli algoritmi di localizzazione. Dato questo presupposto è opportuno che il server sia a conoscenza della mappa radio dell'ambiente perché, come visto nel primo capitolo, la tecnica del **WiFi-Fingerprinting** si suddivide in due fasi, *offline* e *online*. Durante la fase *offline* il client deve essere in grado di catturare le frequenze emesse dagli access point e inviarli al server che si occuperà di immagazzinarli in un *database*. Per applicare questa fase vengono scelti dei **Reference Point**(RPs) sulla planimetria, ognuno distante dall'altro di una quantità costante, dove il client deve essere posizionato per poter iniziare la scansione. Il numero di Reference Point che devono essere individuati è derivabile dalla seguente equazione [17]:

$$|P_{rp}| = \lceil (\frac{P_{tside}}{P_{rpside}})^2 \rceil \tag{2.4}$$

dove P_{rp} è il numero di *Reference Point* in un determinato piano di un determinato edificio, P_{tside} è l'area del piano e P_{rpside} è la grandezza di ogni cella con il quale è stata suddivisa la planimetria. Per ogni RP i, viene effettuata una lettura dello spetto WiFi e trovati gli Access Point corrispondenti a quel RP. Considerando $|AP^{off}|$ come il gruppo di AP trovati nel RP_i e considerando $AP_j \in |AP^{off}|$ si immagazzinano le seguenti informazioni [17]:

$$R_{ij}^{off} =< BSSID_j^{off}, RSS_{ij}^{off}, \mu_{ij}^{off}, \sigma_{ij}^{off} > \tag{2.5}$$

dove $BSSID_j^{off}$ è l'indirizzo MAC di APj, RSS_{ij}^{off} è l'insieme dei campionamenti RSS, μ_{ij}^{off} e σ_{ij}^{off} sono rispettivamente la media e la varianza dei valori RSS.

2.3.5 Raccolta dei dati provenienti dalle fonti radio: fase online

Una volta completata la fase offline si passa alla fase online durante la quale il dispositivo effettua una scansione dello spettro WiFi memorizzando

le seguenti informazioni [17]:

$$R_j^{on} =< BSSID_j^{on}, RSS_i^{on}, \mu_i^{on}, \sigma_i^{on} >, \qquad \forall j \in AP_u^{on} \tag{2.6}$$

dove AP_u^{on} è l'insieme degli Access Point rilevati nella posizione u. Raccolte queste informazioni, esse vengono inviate al server e il client rimane in attesa della risposta.

2.3.6 Ricezione dell'esito della localizzazione e fusione dei dati

L'ultimo compito che deve soddisfare il client è quello di fondere i dati della localizzazione attraverso i sensori inerziali e di quella attraverso radio frequenze proveniente dal server. Entrambe le localizzazioni restituiscono una coppia di coordinate (x, y) sulla planimetria. A tale scopo sono stati progettati cinque algoritmi di fusione dei dati:

1. Only WiFi;
2. Midpoint;
3. Midpoint + threshold;
4. Proporzionale;
5. Proporzionale2.

Only WiFi

Il primo algoritmo, denominato *Only WiFi*, prevede di effettuare una localizzazione provvisoria con i dati provenienti dai sensori nelle coordinate (x_{pdr}, y_{pdr}) e, una volta effettuata la localizzazione tramite *Wi-Fi Fingerprinting*, aggiornare la posizione stimata dell'utente nel punto corrispondente al *Reference Point* candidato di coordinate (x_{wifi}, y_{wifi}).

Midpoint

Il secondo algoritmo, come si può dedurre dalla denominazione, prevede di aggiornare la posizione stimata dell'utente nel punto medio (x_{med}, y_{med}) tra le coordinate (x_{pdr}, y_{pdr}) fornite dal metodo *Pedestrian Dead Reckoning* e le coordinate (x_{wifi}, y_{wifi}) del metodo *WiFi Fingerprinting* comunicate dal server. Le equazioni da applicare sono le seguenti:

$$x_{med} = \frac{x_{pdr} + x_{wifi}}{2}, \tag{2.7}$$

$$y_{med} = \frac{y_{pdr} + y_{wifi}}{2}, \tag{2.8}$$

Midpoint + threshold

Il terzo algoritmo è stato studiato per ovviare ad una problematica del metodo *PDR* molto ricorrente in letteratura. Questo problema si verifica quando ci si trova in uno scenario magneticamente disturbato, ossia quando il magnetometro acquisisce dei dati errati causati da condizioni ambientali avverse, un esempio può essere l'effetto *gabbia di Faraday* causato dall'eccessiva presenza di metalli nell'edificio. L'errore presente nel magnetometro può letteralmente alterare l'intera localizzazione tramite *PDR* in maniera non predicibile poiché le direzioni calcolate dal sensore non rispecchiano quelle effettivamente intraprese dall'utente. Per questo motivo si è inizialmente calcolato, attraverso test specifici consultabili nel capitolo 4 di questa tesi, l'errore medio λ del metodo PDR. Quando il client riceve la localizzazione WiFi dal server calcola la distanza δ tra il punto (x_{wifi}, y_{wifi}) e il punto (x_{pdr}, y_{pdr}) attraverso l'equazione:

$$\delta = \sqrt{(x_{pdr} - x_{wifi})^2 + (y_{pdr} - y_{wifi})^2}. \tag{2.9}$$

Successivamente verifica se $\delta < \lambda$ allora applica l'algoritmo *Midpoint*, altrimenti applica l'algoritmo *Only WiFi*.

Proporzionale

Il quarto algoritmo introduce un *peso* α per preferire una localizzazione piuttosto che l'altra in base ai dati provenienti dai due metodi. Il punto scelto durante la fusione si troverà nel segmento che unisce i due punti e la sua vicinanza ad un punto piuttosto che all'altro sarà dettata dal *peso*:

$$\alpha = \frac{dist_{wifi}}{dist_{pdr} + dist_{wifi}} * (1 - (\frac{N}{N_{max}})^{\gamma}), \tag{2.10}$$

dove $N \in \mathbb{N}$ è il numero di Access Point percepiti durante la fase online, $N_{max} \in \mathbb{N}$ è una costante rappresenta il numero di Access Point sufficienti per una buona localizzazione, $\gamma \in \mathbb{N}$, con dis_{wifi} rappresenta la distanza tra il punto precedente e quello attuale calcolato tramite il metodo *WiFi Fingerprinting* e dis_{pdr} che rappresenta la distanza tra il punto precedente e quello attuale calcolato tramite il metodo *PDR* calcolate attraverso le seguenti equazioni:

$$dis_{pdr} = \sqrt{(x_{pdr,n} - x_{n-1})^2 + (y_{pdr,n} - y_{n-1})^2}, \tag{2.11}$$

$$dis_{wifi} = \sqrt{(x_{wifi,n} - x_{n-1})^2 + (y_{wifi,n} - y_{n-1})^2}. \tag{2.12}$$

Una volta calcolato il peso da assegnare ai due metodi si calcola la fusione della localizzazione nelle coordinate (x_{prop}, y_{prop}):

$$x_{prop} = (x_{pdr} * \alpha) + (x_{wifi} * (1 - \alpha)), \tag{2.13}$$

$$y_{prop} = (y_{pdr} * \alpha) + (y_{wifi} * (1 - \alpha)). \tag{2.14}$$

Proporzionale2

L'ultimo algoritmo applicato per la fusione delle due localizzazioni è quello illustrato in [27]. Anche in questo algoritmo viene applicato un peso:

$$\beta = \frac{\frac{1}{dis_{wifi}}}{\frac{1}{dis_{wifi}} + \frac{1}{dis_{pdr}}}, \tag{2.15}$$

con $dist_{wifi}$ e con $dist_{pdr}$ come sopra.
Si ottengono le coordinate della posizione ibrida stimata (x_{prop2}, y_{prop2}):

$$x_{prop2} = (1 - \beta) * x_{pdr} + \beta * x_{wifi}, \tag{2.16}$$

$$y_{prop2} = (1 - \beta) * y_{pdr} + \beta * y_{wifi}. \tag{2.17}$$

2.4 Server

Il secondo attore di questo sistema è il **server** che come anticipato precedentemente può essere interpretato da qualsiasi dispositivo che sia capace di collegarsi alla rete LAN e di compiere le operazioni descritte in questo paragrafo.
Questo componente entra in funzione solamente durante la **WiFi-Fingerprinting localization** e non opera nella fase *Pedestrian Dead Reckoning*. Principalmente il server si deve occupare di due funzioni:

1. immagazzinamento dei dati provenienti dal client nella fase offline;
2. applicazione degli algoritmi di localizzazione nella fase online.

2.4.1 Immagazzinamento dati: fase offline

Come spiegato nella descrizione del client, al server arrivano le informazioni riguardanti gli Access Point trovati in ognuno dei Reference Point di un determinato piano di un edificio, quindi il server deve mettere a disposizione del client delle interfacce per poter inviare i dati. Una volta ricevuti i dati dal client il server si deve occupare di immagazzinarli all'interno di un database.

2.4.2 Localizzazione: fase online

Durante la fase online, così come nella fase offline, il server riceve i dati riguardanti la scansione dal client e determina quale dei Reference Point immagazzinati nel database è il più vicino alla posizione dell'utente attraverso

degli algoritmi di **pattern matching**. Gli algoritmi di localizzazione sono tre:

1. BSSID Based (BSB);
2. Nearest Neighbour (NN);
3. BSSID Based + Nearest Neighbour (BSBNN).

BSSID Based

Questo algoritmo calcola una funzione obiettivo $S(i)$ per ogni RP_i definito come il numero di Access Point che riesce ad individuare che fanno parte sia del gruppo di AP individuati nella fase online che quelli immagazzinati nella fase offline:

$$S(i) = |AP_i^{off} \cap AP_i^{off}|. \tag{2.18}$$

Il Reference Point con il punteggio più alto è selezionato come il candidato ad essere quello esatto, se si dovesse arrivare ad una situazione di incertezza nella quale vengono selezionati più di un Reference Point, ne viene scelto uno in modo casuale.

Nearest Neighbout

Questo algoritmo calcola una funzione obiettivo $S(i)$ per ogni RP_i definito come la *distanza Euclidea* tra la l'RSS media in RP_i e quella catturata nella posizione dell'utente u:

$$S(i) = \sqrt{\sum_j (\mu_{ij}^{off} - \mu_j^{on})^2}, \qquad \forall j \in AP_i^{off} \cap AP_u^{on} \tag{2.19}$$

Si noti che se $|AP_i^{off}| > |AP_u^{on}|$ è opportuno assegnare una penalità a causa dell'errore in BSB:

$$S(i) = S(i) + (|AP_i^{off}| - |AP_u^{on}|) * \gamma \tag{2.20}$$

dove γ è un valore costante. Il Reference Point che risulterà avere un errore minore sarà candidato come possibile posizione dell'utente.

BSSID Based + Nearest Neighbour

Questa soluzione utilizza entrambi gli algoritmi precedenti. Inizialmente si tenta con l'algoritmo BSB e se ci si trova di fronte ad una situazione di incertezza allora si utilizza anche l'approccio NN.
Nelle immagini seguenti sono riassunti i ruoli del client e del server durante le due fasi.

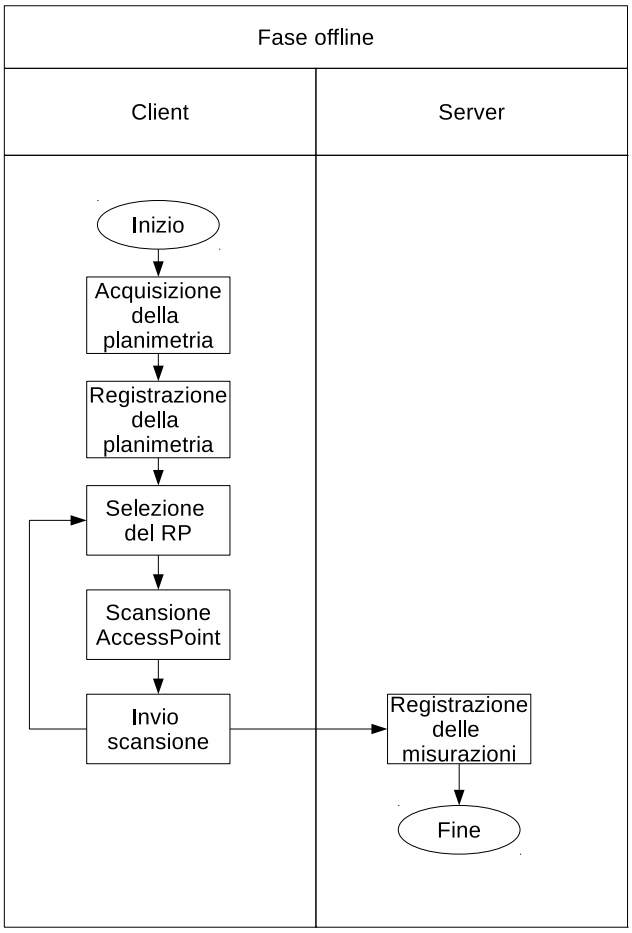

Figura 2.3: Fase offline

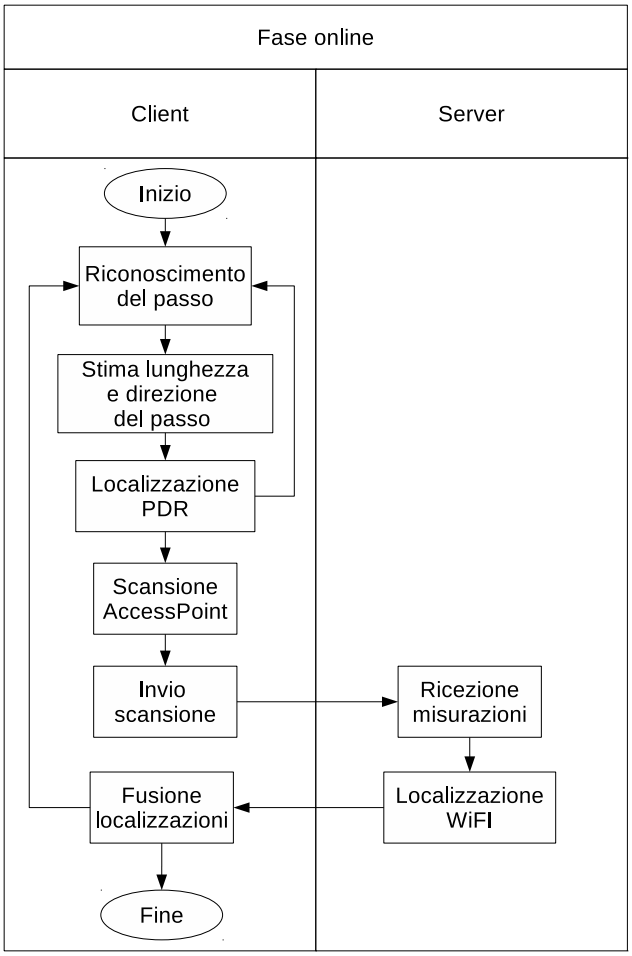

Figura 2.4: Fase online

Capitolo 3

Implementazione

3.1 Introduzione all'implementazione

Come si è descritto nel capitolo della *progettazione* questo sistema è composto da un client che risiede su un dispositivo mobile e una parte server che può essere eseguita su un elaboratore generico. In questo capitolo vengono illustrate le scelte implementative che hanno portato alla realizzazione del sistema esplicando tecnologie, software, linguaggi, librerie e algoritmi utilizzati.

3.2 Tecnologie

Le tecnologie utilizzate in questo sistema sono diverse. In questa sezione, come avvenuto per il capitolo precedente, vengono mostrate inizialmente le tecnologie usate dal client, dal server e le tecnologie di comunicazione tra queste due componenti.

3.2.1 Client

Per i motivi spiegati esaustivamente in questa tesi è stato scelto come ruolo del client uno smartphone di nuova generazione poiché è in grado di

soddisfare tutte le richieste, sia hardware che software, necessarie alla realizzazione di un sistema di localizzazione indoor. Per questo sistema è stato scelto di implementare un'applicazione **Android**.

Android

Android è un sistema operativo per dispositivi mobili nato nel 2003 dalle menti di *Andy Rubin*, *Rich Miner*, *Nick Sears* e *Chris White* e acquisito nel 2005 da **Google Inc**. È un software **Open Source** distribuito con licenza libera **Apache 2.0** e sviluppato dall'Android Open Source Project. È basato su **kernel Linux** anche se non può essere considerato come una distribuzione *GNU/Linux* perché la quasi totalità delle utilità *GNU* sono state rimpiazzate dal software **Java**. Attualmente esistono diversi forking del sistema operativo che hanno portato la sua installazione anche su dispositivi come automobili, televisori e dispositivi indossabili ed è presente nell'85% di questi device, ricerca di IDC (International Data Corporation) sui dati del secondo quadrimestre 2017, dominando di fatto il mercato.

Sensori

I dispositivi *Android* sono equipaggiati con numerosi sensori che possono essere hardware o software. I sensori hardware sono quelli che effettuano delle vere e proprie misurazioni dell'ambiente:

- Sensore di luminosità: misura l'illuminazione presente nell'ambiente in *lux*;
- Sensore di temperatura: misura la temperatura dell'ambiente in *gradi Celsius*;
- Sensore di pressione: misura la pressione dell'aria presente nell'ambiente in *milliBar*;
- Sensore di prossimità: misura la presenza di un oggetto vicino al device in *centimetri*;

- Sensore di umidità: misura l'umidità presente nell'ambiente in *percentuale.*

Oltre ai sensori ambientali esistono altri sensori hardware che invece sono destinati a misurare i movimenti compiuti dal device:

- Magnetometro: è sia un sensore ambientale che di movimento, misura le onde magnetiche presenti nell'ambiente sugli assi (x, y, z) ed è capace di percepire la direzione verso il quale è rivolto il dispositivo;
- Giroscopio: misura la rotazione del dispositivo sui tre assi (x, y, z) in $\frac{radianti}{secondo}$;
- Accelerometro: misura l'accelerazione che subisce il dispositivo sui tre assi (x, y, z) in $(\frac{metri}{secondo})^2$, compresa la forza di gravità.

Questi ultimi tre sensori, quelli di movimento, sono stati utilizzati per l'implementazione della tecnica Pedestriand Dead Reckoning.

I sensori software sono quelli che uniscono i dati provenienti da più sensori

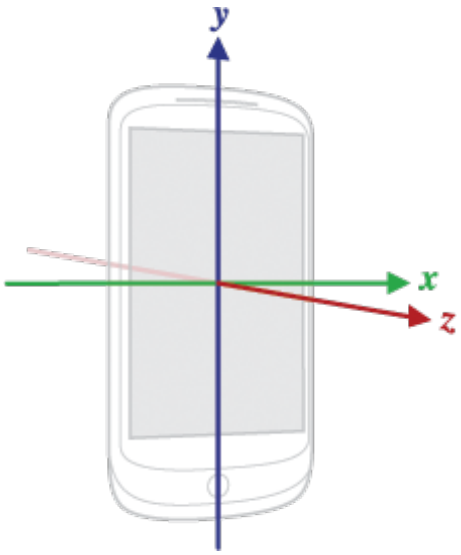

Figura 3.1: Sistema di assi dell'accelerometro [29]

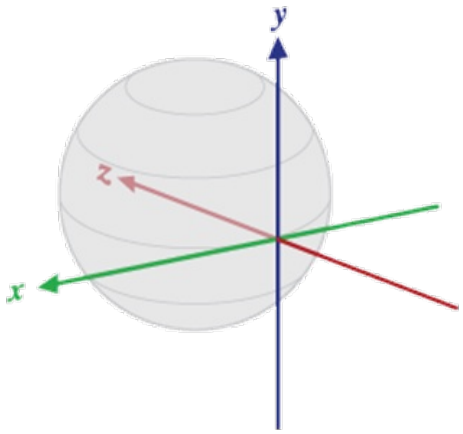

Figura 3.2: Sistema di assi del giroscopio [29]

hardware per determinare una misurazione e sono presenti nelle librerie di *Android*:

- Step Detector: è un sensore software che è in grado di rilevare e notificare il compimento di un passo;
- Step Counter: è un sensore software che rileva e conta il numero di passi effettuati dall'ultimo avvio del dispositivo.

Questi sono solo alcuni dei sensori software presenti nelle librerie, ma sono gli unici che sono utilizzati nell'applicazione sviluppata.

3.2.2 Server

Per quanto riguarda la parte server è stato utilizzato il *Web Server* **Apache2** che viene eseguito su una distribuzione **GNU/Linux Debian 9 Stretch** affiancato dal database **MongoDB**.

Apache2

Apache2 è un Server Web *open source* sviluppato dalla *Apache Software Foundation*. Il suo primo rilascio avvenne nel 1995 e ad oggi è la piattaforma più diffusa per quanto riguarda questa categoria, toccando quota 60

milioni di installazioni presenti in Internet pari al 69,32% del mercato, ed è disponibile sui maggiori sistemi operativi desktop e server. Il suo scopo è quello di realizzare le funzioni di trasporto delle informazioni, di collegamento in un'architettura *client-server* e quindi gestire le richieste con protocollo *HTTP*. L'esecuzione di *Apache* è gestita da un demone o da un servizio e presenta un'architettura modulare, quindi ad ogni richiesta del client vengono svolte le funzioni da ogni **modulo** di cui è composto in maniera atomica e indipendente ed il controllo dei moduli è gestito dal **core**. Al di sopra del ciclo del core un demone esegue un ciclo di polling, attraverso il quale vengono interrogate continuamente le linee logiche da cui possono pervenire messaggi di richiesta. Il core passa poi la richiesta ai vari moduli in modo sequenziale, usando i parametri di uscita di un modulo come parametri di ingresso per il successivo, creando così l'impressione di una comunicazione orizzontale fra i moduli realizzando di fatto una *pipeline software*. Le principali fasi di cui è composto il ciclo sono:

- Translation: traduce la richiesta del client;
- Access Control: controlla le richieste in base ai criteri di autorizzazione;
- MIME Type: identifica il tipo di contenuto e decide quali moduli possono contribuire a soddisfare la richiesta;
- Response: invia la risposta al client;
- Logging: tiene traccia di tutta l'esecuzione.

MongoDB

MongoDB, dall'unione delle parole "humongous" (enorme) e "database", è un *DBMS* non relazionale nato da un progetto **Open Source** e distribuito con licenza libera **GNU Affero General Public License**. La sua struttura *NoSql* gli permette di abbandonare la struttura tradizionale dei database relazionali strutturati in tabelle per favorire una gestione in documenti in stile *JSON* e ciò gli conferisce una maggiore scalabilità e agilità.

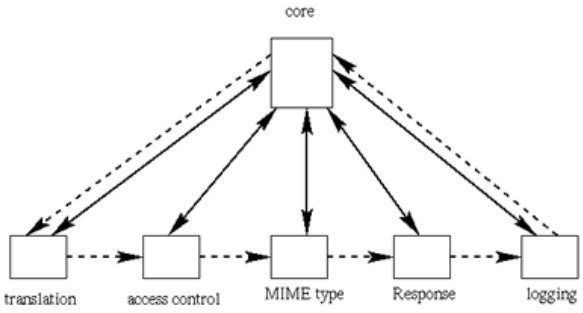

Figura 3.3: Ciclo di esecuzione di Apache

3.3 Sviluppo del Client

Nelle prossime due sezioni si entra nel merito dello sviluppo dei due livelli dell'architettura Client-Server. Si illustrano i software, i linguaggi, gli ambienti di sviluppo, gli algoritmi e le librerie usate per ognuno delle due componenti.
Il primo livello trattato è il client che, come anticipato, è un'applicazione *Android* interamente progettata e realizzata con l'IDE ufficiale *Android Studio* disponibile su tutte le piattaforme e rilasciato gratuitamente da *JetBrains* e *Google*. L'ambiente di sviluppo mette a disposizione tutto l'occorrente per poter realizzare un'applicazione, dalle librerie Java a quelle Android, da un emulatore di dispositivi a un simulatore di sensori per il test delle varie funzionalità.
Il client sviluppato, denominato **StepApp**, è un'applicazione **Android** con *API Level 21* composta da 45 tra classi Java e *Android Activity*, in Figura 3.4. è mostrato il *Class Diagram* dell'intera applicazione. Come è possibile vedere dalla figura il progetto ha dimensioni impossibili da descrivere tutte insieme, per questo motivo è stato scelto di dividere la descrizione dell'implementazione del client in due parti.

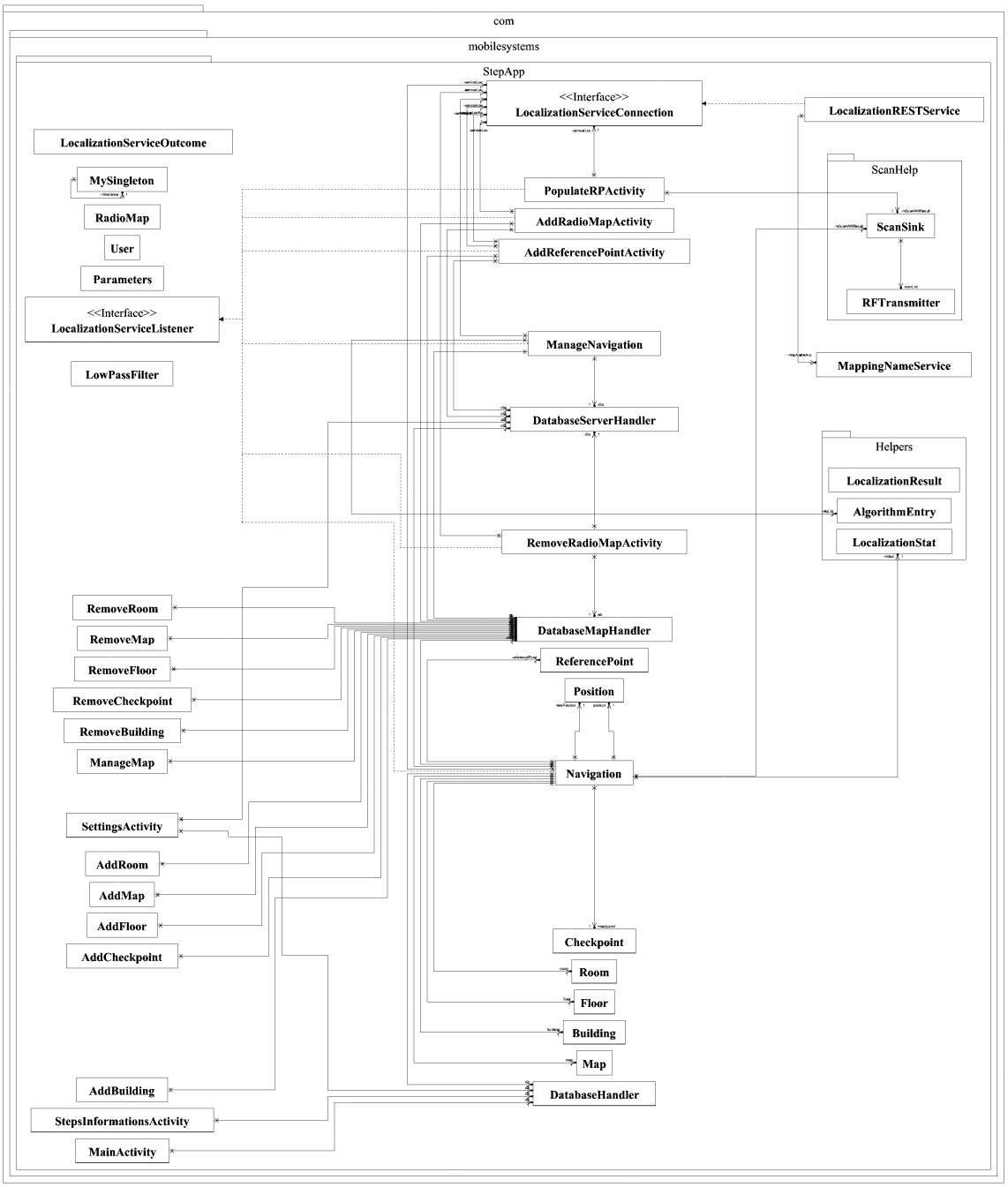

Figura 3.4: Class Diagram

3.3.1 Implementazione PDR

La prima delle due parti che vengono analizzate separatamente è quella relativa all'implementazione della tecnica *Pedestrian Dead Reckoning*. Questa prima parte è a sua volta suddivisa in tre fasi come spiegato nel capitolo della progettazione:

1. riconoscimento del passo;
2. stima della lunghezza del passo;
3. direzione del passo;

Le classi riguardanti l'implementazione della tecnica PDR sono visibili in Figura 3.5.

Riconoscimento del passo

Per il riconoscimento del passo è stata utilizzata la tecnica della *peak detection* e fortunatamente le *API Android* mettono già a disposizione del programmatore due librerie che inglobano questa funzionalità. Le due librerie sono state scritte sotto forma di *sensori software* che possono essere richiamati dal codice con un *Sensor Event*. I due sensori software sono **Step Counter** e **Step Detector** e si basano entrambi sui dati provenienti dall'accelerometro ma presentano alcune differenze sostanziali:

- Step Detector: per poterlo utilizzare deve essere attivato un listener all'occorrenza poiché esso produce un evento ogni volta che un utente compie un passo, quindi per contare i passi effettuati basta aumentare di una unità una variabile ogni qualvolta si verifica un "evento passo";
- Step Counter: il sensore è sempre attivo all'avvio dello smartphone perché conta tutti i passi compiuti dall'utente partendo dall'ultimo boot del dispositivo, quindi per contare i passi effettuati basta accedere alla memoria del sensore e leggere il valore.

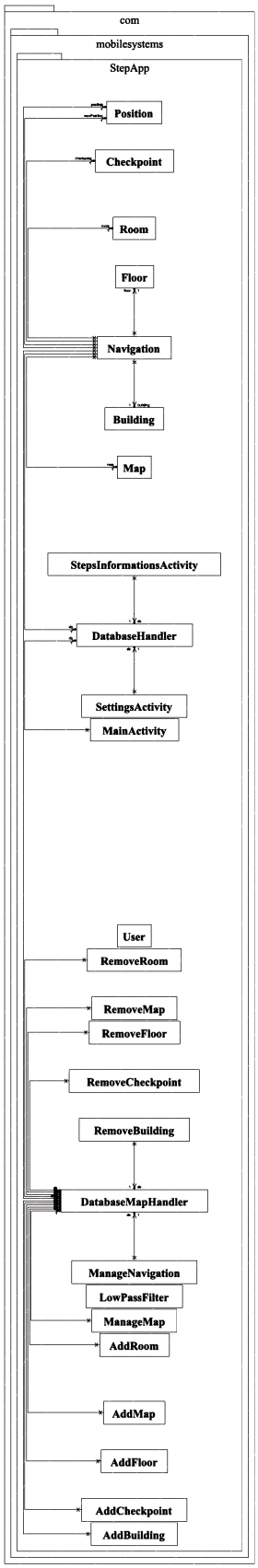

Figura 3.5: Class diagram PDR

Si può dunque dire che i due sensori software siano equivalenti poiché entrambi si basano sui medesimi sensori hardware ma, come si vedrà nel capitolo delle valutazioni sperimentali, questa affermazione non è del tutto vera. Qui in basso si mostra come vengono utilizzati i sensori software:

```
public class StepsInformationsActivity extends AppCompatActivity implements
    SensorEventListener {
    [...]
    Sensor stepDetectorSensor;
    Sensor stepCounterSensor;
    SensorManager smSteps;

    //Step Detector
    private int stepDetected = 0;

    //Step Counter
    private float stepsInSensor = 0;
    private float stepsAtReset;
    private String MY_PREFS_NAME;
    private float stepsBeforeStop;
    private float stepsAfterStop;
    private float stepsDuringStop;
    [...]

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_steps_informations);
        [...]
        SharedPreferences prefs = getSharedPreferences(MY_PREFS_NAME, MODE_PRIVATE);
        stepsAtReset = prefs.getFloat("stepsAtReset", 0);
        stepsBeforeStop = prefs.getFloat("stepsBeforeStop", 0);
        stepsAfterStop = prefs.getFloat("stepsAfterStop", 0);
        stepsDuringStop = prefs.getFloat("stepsDuringStop", 0);

        smSteps = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        stepCounterSensor = smSteps.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
        stepDetectorSensor = smSteps.getDefaultSensor(Sensor.TYPE_STEP_DETECTOR);
        [...]
    }

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        if (sensorEvent.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
            stepsInSensor = Float.valueOf(sensorEvent.values[0]);
        }
        [...]

        if (sensorEvent.sensor.getType() == Sensor.TYPE_STEP_COUNTER){
            SharedPreferences sp = getSharedPreferences(MY_PREFS_NAME, MODE_PRIVATE);
            Float stepsSinceReset = stepsInSensor - sp.getFloat("stepsAtReset", -1) - sp
                .getFloat("stepsDuringStop", -1);
            [...]
        } else {
            if (sensorEvent.sensor.getType() == Sensor.TYPE_STEP_DETECTOR) {
                stepDetected ++;
                [...]
            }
        [...]
        }
        [...]
    }
```

```
    [...]
}
```

Come è possibile vedere dal codice, dal momento che *Step Counter* è sempre attivo, è stato opportuno maneggiare il caso in cui il dispositivo sia in pausa o l'utente non voglia registrare gli spostamenti utilizzando **SharedPreferences** che permette il salvataggio di dati di tipo *chiave-valore*.

Lunghezza del passo

Per la stima della lunghezza del passo sono stati utilizzati due differenti algoritmi denominati **Static Step Length** e **Dynamic Step Length** come spiegato nel capitolo della progettazione.

- **Static Step Length**: Vengono presi in input i dati di altezza e sesso dell'utente che sta utilizzando il dispositivo e viene calcolata la lunghezza del passo con la formula:

$$SSL = \alpha * h, \qquad \alpha = \begin{cases} 0,415 & \text{se l'utente è di sesso maschile;} \\ 0,413 & \text{se l'utente è di sesso femminile.} \end{cases}$$

- **Dynamic Step Length**: Vengono presi in input i dati dell'accelerazione percepiti dall'accelerometro e viene stimata la lunghezza del passo come spiegato nel capitolo precedente con la formula: $DSL = k * \sqrt[4]{A_{max} - A_{min}}$.

Per quanto riguarda i dati personali dell'utente sono stati raccolti in una apposita *Android Activity* e immagazzinate nel database **SQL Lite** interno al dispositivo. Invece per le accelerazioni massime e minime le *API* di **Android** non mettono a disposizione queste informazioni che servono per calcolare la lunghezza del passo nel secondo algoritmo e per questo motivo è stato opportuno ideare una soluzione ad hoc per implementare questa funzionalità. L'idea è stata quella di utilizzare un array bidimensionale per registrare in una dimensione i valori dell'accelerazione percepita dall'accelerometro ad ogni intervallo predefinito di tempo e nell'altra l'istante temporale relativo

all'accelerazione. Successivamente, ogni volta che un passo viene compiuto, si confronta il timestamp dell'evento "passo" con la seconda dimensione dell'array per trovare l'istante registrato più vicino. Da quell'indice si apre una finestra denominata **acceleration_window** che va a ritroso di una quantità x (valore ottenuto attraverso prove empiriche) di posizioni e all'interno di questo intervallo si selezionano l'accelerazione massima e minima. Il codice relativo alla stima della lunghezza del passo è mostrato qui di seguito:

```
public class StepsInformationsActivity extends AppCompatActivity implements
    SensorEventListener {
    DatabaseHandler db = new DatabaseHandler(this); //The DB handler
    //User Information
    private double genderValue;
    private boolean genderType;
    private double stepLenght;
    private double heightNumber;
    //DynamicSteplenght
    private Long[] timestamps = new Long[300];
    private Float[] accelerations = new Float[300];
    //microseconds depends on the setted sensor's delay
    private int microseconds = 20000000;
    private long timestamp;
    private int indexArray;
    private double aMax;
    private double aMin;
    private double acceleration;
    private double distance;
    private long tsMin;
    private double feetToMeters;
    [...]
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        User selectedUser = db.getSelectedUser();
        //for user'sinformation
        heightNumber = selectedUser.getHeight();
        genderType = selectedUser.getGender();
        aMax = Double.MIN_VALUE;
        aMin = Double.MAX_VALUE;
        timestamp = Long.MIN_VALUE;
        indexArray = 0;
        if(genderType)
            genderValue = 0.415;
        else
            genderValue = 0.413;

        feetToMeters = (1 - 0.3048); //conversion
        //Dynamic Step lenght
        for(int i=0; i < 300; i++){
            timestamps[i] = 0l;
            accelerations[i] = 0f;
        }
        [...]
    }

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        [...]
        if (switchStepLenght.isChecked()){ //if DynamicStepLength
```

```
            timestamp = sensorEvent.timestamp;
            tsMin = Math.abs(timestamps[0] - timestamp);
            for (int i = 1; i < 300; i++){
                if(Math.abs(timestamps[i] - timestamp) < tsMin){
                    tsMin = Math.abs(timestamps[i] - timestamp);
                    indexArray = i;
                }
            }
            if (indexArray  + 3 > 299)
                indexArray = 296;
            if (indexArray - 7 < 0)
                indexArray = 7;
            aMax = Double.MIN_VALUE;
            aMin = Double.MAX_VALUE;
            for(int i = indexArray - 7; i < indexArray + 3; i++){
                if(accelerations[i] > aMax )
                    aMax = accelerations[i];
                if(accelerations[i] < aMin)
                    aMin = accelerations[i];
            }
            //for StepCounter
            double dist_temp_counter = distance;
            //for StepDetector
            //double dist_temp_counter = distance;
            stepLenght = (java.lang.Math.sqrt(java.lang.Math.sqrt((aMax)-(aMin))));
            distance += stepLenght *  feetToMeters;
            //for StepCounter
            double stepL_counter = distance - dist_temp_counter;
            //for StepDetector
            //double stepL_counter = distance - dist_temp_counter;
        } else { //if StaticStepLenght
            stepLenght = heightNumber * genderValue;
            //for StepCounter
            distance = (stepsSinceReset * stepLenght) / 100;
            //for StepDetector
            //distance = (stepsDetected * stepLenght) / 100;
        }
        [...]
    }
    [...]
}
```

Direzione del passo

La direzione del passo si basa sui dati provenienti dal magnetometro. Grazie alle API Android è facile costruire una *bussola* in grado di fornire la direzione verso la quale il dispositivo è orientato. Il codice fornisce l'orientamento in gradi prendendo come punto di riferimento il *nord magnetico* ed è mostrato di seguito:

```
public class StepsInformationsActivity extends AppCompatActivity implements
    SensorEventListener {
    private double degree;
    private float currentDegree= 0f;
    private float[] gravity = new float[3];
    private float[] geomagnetic = new float[3];
```

```
    private float[] rotation = new float[9];
    private float[] orientation = new float[3];
    private float[] smoothed = new float[3];
    private Sensor sensorGravity;
    private Sensor sensorMagnetic;
    private SensorManager sensorManager;
    [...]
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        [...]
        sensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
        sensorGravity = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorMagnetic = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
        [...]
    }
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        [...]
        //compass
        if (sensorEvent.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            smoothed = LowPassFilter.filter(sensorEvent.values, gravity);
            gravity[0] = smoothed[0];
            gravity[1] = smoothed[1];
            gravity[2] = smoothed[2];
        }
        if (sensorEvent.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
            smoothed = LowPassFilter.filter(sensorEvent.values, geomagnetic);
            geomagnetic[0] = smoothed[0];
            geomagnetic[1] = smoothed[1];
            geomagnetic[2] = smoothed[2];
        }
        if (gravity != null && geomagnetic != null){
            // get rotation matrix to get gravity and magnetic data
            SensorManager.getRotationMatrix(rotation, null, gravity, geomagnetic);
            // get bearing to target
            SensorManager.getOrientation(rotation, orientation);
            // east degrees of true North
            degree = orientation[0];
            // convert from radians to degrees
            degree = (Math.toDegrees(degree));
            if (degree < 0)
                degree = degree + 360;
            currentDegree = (float) -degree;
        }
        [...]
    }
    [...]
}
```

Come è possibile vedere dal codice è stato implementato un Low-Pass Filter per levigare i dati provenienti dal magnetometro, il codice del filtro è mostrato qui in seguito:

```
class LowPassFilter {
    private static final float ALPHA = 0.2f;
    private LowPassFilter(){
    }

    public static float[] filter(float[] input, float[] output){
        if (input == null){
            return input;
        }
```

```
        for (int i = 0; i< input.length; i++){
            output[i] = output[i] + ALPHA * (input[i] - output[i]);
        }
        return output;
    }
}
```

Fusione delle tre tecniche

Tutte le tecniche e gli algoritmi implementati sono stati inseriti all'interno di una *Android Activity* per testare il corretto funzionamento del codice scritto. La classe responsabile di tale scopo è StepsinformationsActivity la quale viene mostrata in Figura 3.6. Come è possibile vedere dall'immagine essa comprende tutto il necessario per poter testare la tecnica *Pedestrian Dead Reckoning*. All'avvio dell'applicazione devono essere registrati tutti i

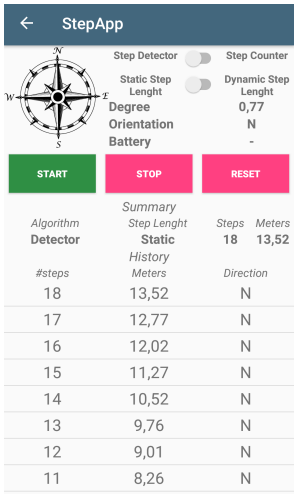

Figura 3.6: StepInformationsActivity

Listener dei sensori ed ogni volta che viene interrotta devono essere rilasciati seguendo le specifiche del ciclo di vita di una applicazione Android visibile in Figura 3.7. Il codice responsabile di tale compito è il seguente:

```
public class StepsInformationsActivity extends AppCompatActivity implements
    SensorEventListener {
    [...]
    protected void onResume() {
        super.onResume();
        if (stepCounterSensor != null && stepDetectorSensor != null
```

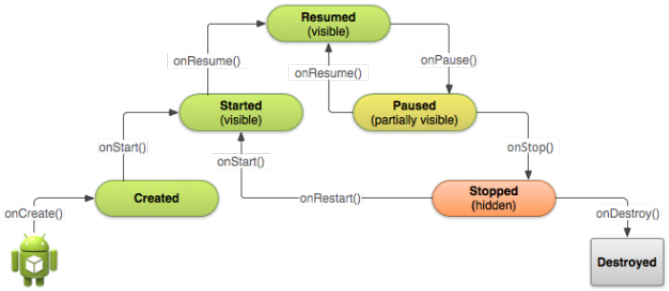

Figura 3.7: Ciclo di vita di un'app

```
                && sensorGravity != null && sensorMagnetic != null) {
            //Step Counter and Detector
            smSteps.registerListener(this, stepDetectorSensor, smSteps.SENSOR_DELAY_GAME
                );
            smSteps.registerListener(this, stepCounterSensor, smSteps.SENSOR_DELAY_GAME)
                ;
            //Compass
            sensorManager.registerListener(this, sensorGravity, SensorManager.
                SENSOR_DELAY_GAME);
            sensorManager.registerListener(this, sensorMagnetic, SensorManager.
                SENSOR_DELAY_GAME);
        } else {
            Toast.makeText(this, "Sensors not found", Toast.LENGTH_SHORT).show();
        }
    }
    protected void onPause() {
        super.onPause();
        smSteps.unregisterListener(this, stepCounterSensor);
        smSteps.unregisterListener(this, stepDetectorSensor);
        sensorManager.unregisterListener(this, sensorGravity);
        sensorManager.unregisterListener(this, sensorMagnetic);
    }
    [...]
}
```

Sulla sinistra si trova l'immagine di una bussola che ruota in base ai dati provenienti dal magnetometro. Il codice responsabile di questa funzione è il seguente:

```
public class StepsInformationsActivity extends AppCompatActivity implements
    SensorEventListener {
    ImageView image;
    [...]
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        [...]
        // how long the animation will take place
        ra.setDuration(210);
        // set the animation after the end of the reservation status
        ra.setFillAfter(true);
        // Start the animation
        image.startAnimation(ra);
        currentDegree = (float) -degree;
```

```
    }
    [...]
}
```

Sulla destra invece si trovano degli Android Switch che permettono il passaggio tra i vari algoritmi di riconoscimento del passo e di stima della lunghezza del passo spiegati precedentemente. Al di sotto degli switch si trovano le informazioni riguardanti la direzione del dispositivo in maniera testuale e la percentuale di batteria consumata durante il test. Nella parte centrale sono presenti tre bottoni per l'avvio, l'arresto o il reset dei test. Nella parte bassa sono visualizzate in *Listview* i dati relativi al test come numero di passi effettuati, metri percorsi e direzione di ogni passo. I risultati dei test della tecnica PDR sono mostrati nel capitolo seguente.

3.3.2 Implementazione WiFi Fingerprinting

L'implementazione della tecnica **Wifi-Fingerprinting** è, come detto in precedenza, suddivisa nella parte client e nella parte server, in questa sezione si descrive solamente la prima che può essere ulteriormente scomposta in due sottoproblemi: misurazioni attraverso l'interfaccia WiFi e comunicazione con il server. Le classi che si occupano di queste funzionalità sono mostrate nella Figura 3.8.

Misurazioni attraverso l'interfaccia WiFi

Il primo problema da risolvere riguarda il compito del client di effettuare delle misurazioni sia per mappare l'ambiente indoor, sia per localizzarsi all'interno dell'ambiente. Questa fase deve essere effettuata sia durante la mappatura dell'ambiente sia durante la localizzazione. In entrambi i casi si utilizza un WiFi-Manager messo a disposizione con le *API Android* che permette di scansionare gli *Access Point* presenti nei dintorni del dispositivo che utilizza questa funzionalità e di memorizzarne alcune informazioni come il *MAC Address* e la potenza trasmissiva **RSS**. La classe che imple-

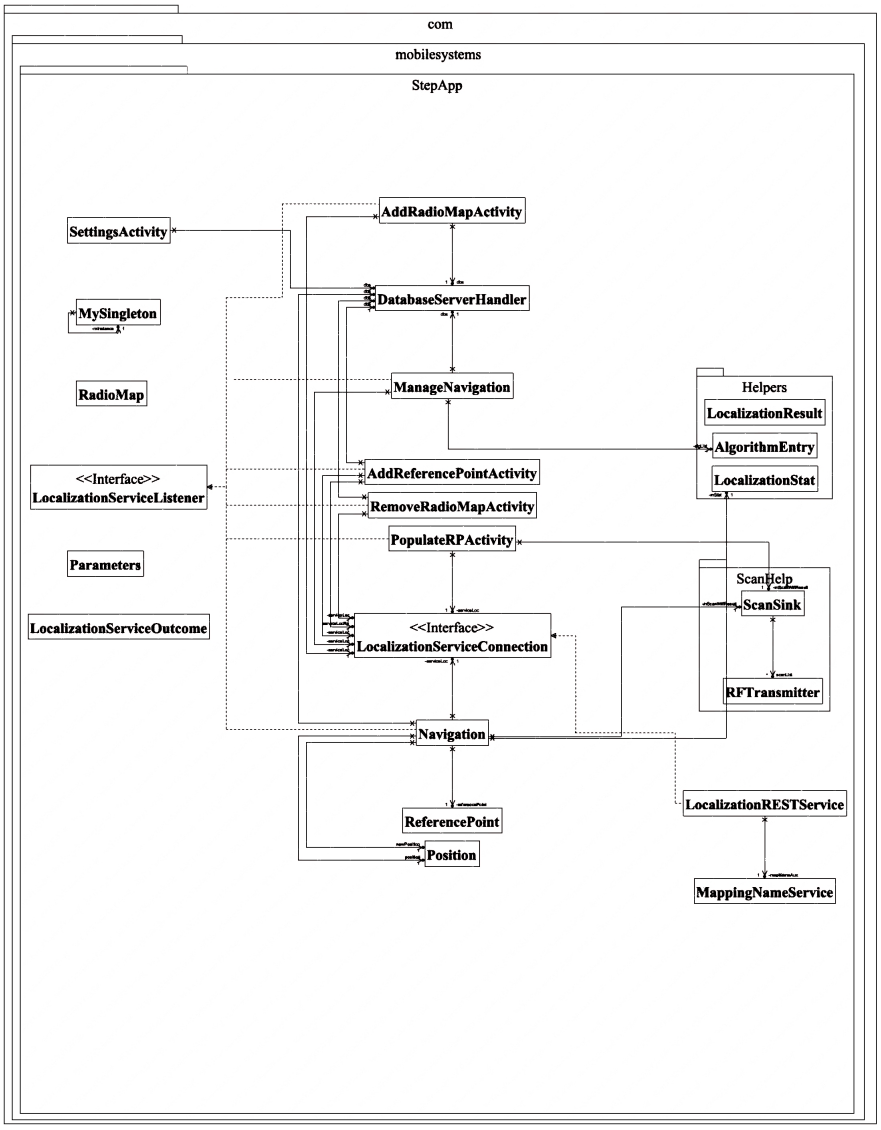

Figura 3.8: Class Diagram WiFi-Fingerprinting

menta le funzionalità di mappatura, e quindi quella responsabile di popolare i *Reference Point*, è la PopulateRPActivity visibile in Figura 3.9.

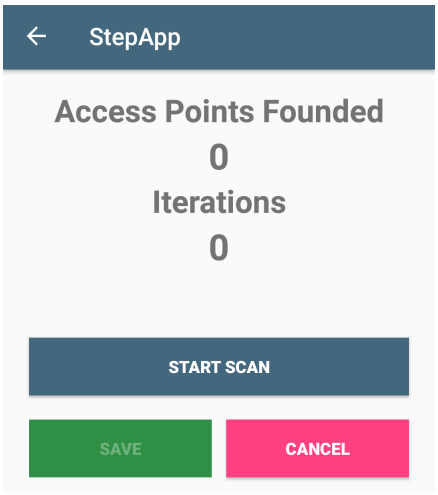

Figura 3.9: PopulateRPActivity

Le parti più importanti del codice sono quelle presenti di seguito:

```
public class PopulateRPActivity extends AppCompatActivity implements
     LocalizationServiceListener{
    [...]
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_populate_rp);
        [...]
        mScanWifiResult = new ScanSink();
        mWifi = (WifiManager) getApplicationContext().getSystemService(Context.
             WIFI_SERVICE);
        if (checkWiFi) {
            mWifiReceiver = new BroadcastReceiver() {
                @Override
                public void onReceive(Context c, Intent intent) {
                    List<ScanResult> results = mWifi.getScanResults();
                    parseScan(results);
                    if (!wifiDone)
                        mWifi.startScan();
                }
            };
        }
    }
    private void parseScan(List<ScanResult> list) {

        for (int i = 0; i < list.size(); i++){
            ScanResult mScan=(ScanResult) list.get(i);
            if (mScan.level!=0)
                mScanWifiResult.insert(mScan.BSSID, mScan.level);
        }
    }
```

```
    public void startScan(View view) {
        android.os.Handler mHandler=new android.os.Handler();
        if (checkWiFi) {
            registerReceiver(mWifiReceiver, new IntentFilter(WifiManager.
                SCAN_RESULTS_AVAILABLE_ACTION));
            mWifi.startScan();
        }
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                if (checkWiFi)
                    unregisterReceiver(mWifiReceiver);
                wifiDone=true;
                checkEnd();
            }
        }, numTotalScan*1000);
    }
    private void checkEnd() {
        if (wifiDone)
            btnSave.setEnabled(true);
    }
    [...]
}
```

Comunicazione con il server

Per quanto riguarda la comunicazione con il server sono state effettuate delle chiamate REST attraverso la libreria Volley di Android. Questa funzionalità viene usata due volte all'interno del client: per inserire un *Reference Point* o una Radio Map nel database del server, per comunicare i dati degli *Access Point* percepiti e ricevere da server la posizione calcolata. La classe che principalmente si occupa di tale compito è **LocalizationRESTService**. Essa è composta da vari metodi in grado di comunicare al server le informazioni e alcune di esse sono mostrate nel codice di seguito:

```
public class LocalizationRESTService implements LocalizationServiceConnection {
    public LocalizationRESTService(String server, Context context) {
        serverIP=server;
        mapNameAux=new MappingNameService(serverIP);
        mcontext=context;
        RequestQueue queue = MySingleton.getInstance(context.getApplicationContext()).
            getRequestQueue();
    }
    [...]
    //RP infos add
    @Override
    public void addRefPoint(final LocalizationServiceListener listener, ReferencePoint
         referencePoint) {
        String restMethod=mapNameAux.mapPUTRefPoints();
        String jsonParam=null;
        jsonParam="{ \"name\":\""+referencePoint.getName()+"\", \"radiomap\":\""+
            referencePoint.getRadioMap()+
                "\", \"x\":\""+referencePoint.getX()+"\", \"y\":\""+referencePoint.getY
                    ()+
```

```
            "\"}";
        JSONObject Jparam=null;
        try {
            Jparam=new JSONObject(jsonParam);
        } catch (JSONException e) {
            e.printStackTrace();
            return;
        }
        JsonObjectRequest jsObjRequest = new JsonObjectRequest(com.android.volley.
            Request.Method.POST, restMethod, Jparam, new Response.Listener<JSONObject>()
             {
            @Override
            public void onResponse(JSONObject response) {
                try {
                    int result = response.getInt("result");
                    if (result==1)
                        listener.RefPointPUTComplete(new LocalizationServiceOutcome(true
                            ,null));
                } catch(org.json.JSONException ex) {
                    listener.RefPointPUTComplete(new LocalizationServiceOutcome(false,"
                        JSON parsing error"));
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                listener.RefPointPUTComplete(new LocalizationServiceOutcome(false,"
                    Network error"));

            }
        });
        MySingleton.getInstance(mcontext).addToRequestQueue(jsObjRequest);
    }

    //RP measurement add
    @Override
    public void addRPMeasurement(final LocalizationServiceListener listener, String
        rpName, String radioMap, ScanSink sink) {
        String restMethod=mapNameAux.mapPUTMeasurement();
        String jsonParam="{ \"rp\":\""+rpName+"\", \"radiomap\":\""+radioMap+"\", \"
            measurements\":[";
        jsonParam=jsonParam+ sink.JSONencode("wifi");
        jsonParam=jsonParam+"]";
        jsonParam=jsonParam+"}";
        JSONObject Jparam=null;
        try {
            Jparam=new JSONObject(jsonParam);
        } catch (JSONException e) {
            e.printStackTrace();
            return;
        }
        JsonObjectRequest jsObjRequest = new JsonObjectRequest(com.android.volley.
            Request.Method.POST, restMethod, Jparam, new Response.Listener<JSONObject>()
             {
            @Override
            public void onResponse(JSONObject response) {
                try {
                    int result = response.getInt("result");
                    if (result==1)
                        listener.RPMeasurementPUTComplete(new LocalizationServiceOutcome
                            (true,null));
                } catch(org.json.JSONException ex) {
                    listener.RefPointPUTComplete(new LocalizationServiceOutcome(false,"
                        JSON parsing error"));
```

```
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {

                listener.RefPointPUTComplete(new LocalizationServiceOutcome(false,"
                    Network error"));
            }
        });
        MySingleton.getInstance(mcontext).addToRequestQueue(jsObjRequest);
    }
    [...]
}
```

Come possibile vedere da questi due esempi riportati, i metodi costruiscono un file con estensione **Json** e lo inviano al server indicato all'inizio della classe che è contenuto in un database interno al client i quali record possono essere modificati attraverso la **SettingsActivity**

3.3.3 L'applicazione StepApp

Dopo aver mostrato gli algoritmi principali che hanno permesso di implementare le tecniche *Pedestrian Dead Reckoning* e *Wifi-Fingerprinting* si mostrano alcune delle immagini delle schermate dell'applicazione progettata e implementata delle quali viene spiegato il funzionamento e lo scopo.

Settings Activity

La **Settings Activity**, come si può dedurre dal suo nome, si occupa di raccogliere alcuni dati utili all'applicazione. Come è possibile vedere in Figura 3.10. possono essere inseriti i dati dell'utente relativi al sesso e all'altezza per essere poi usati per calcolare la *Static Step Length* durante la navigazione. Inoltre possono essere immessi i dati relativi all'indirizzo IP del server, al numero di scansioni che devono essere effettuate durante la mappatura della *radio map* e il numero di scansioni durante la navigazione per poi essere utilizzati nella tecnica *WiFi-Fingerprinting*. Tutte queste informazioni vengono immagazzinate all'interno di un database **SQL Lite** che risiede sul client.

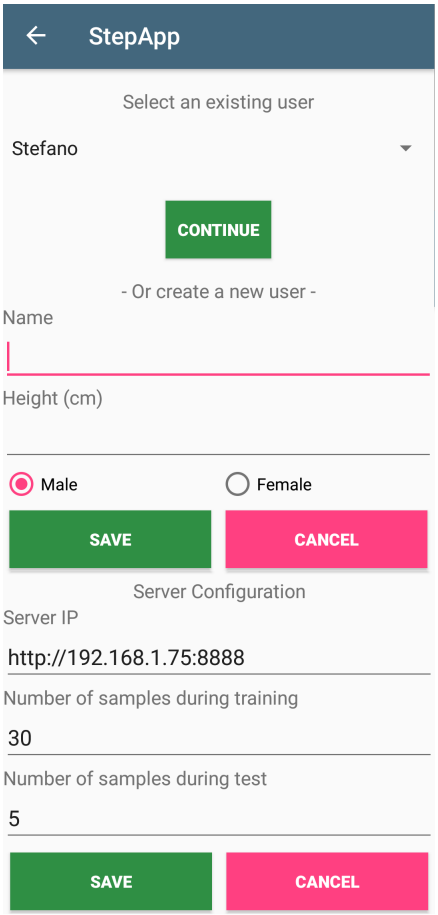

Figura 3.10: SettingsActivity

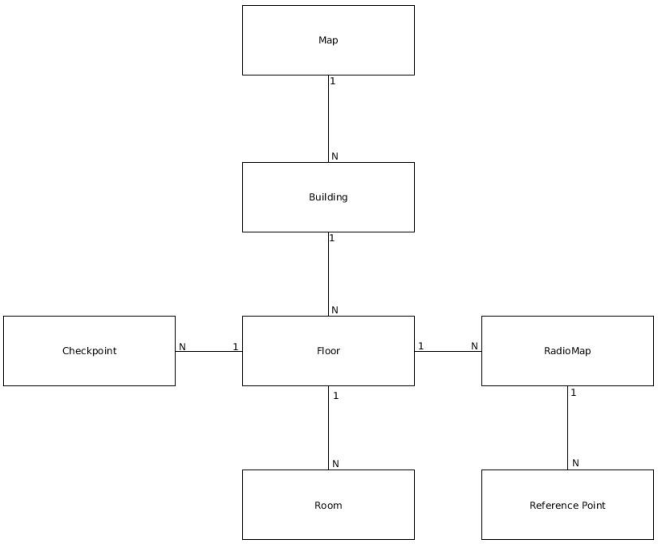

Figura 3.11: Schema della mappatura

Manage Map

L'*Android Activity* **ManageMap** permette all'utente di muoversi nelle varie schermate di creazione ed eliminazione delle mappe reali e delle mappe radio. La logica di mappatura è riassunta nello schema di Figura 3.11. Come è possibile vedere una **Mappa** può essere composta da più edifici ognuno dei quali è composto da più **Piani**. In ogni piano possono trovarsi N **Stanze** e M **Reference Point**. Ogni piano può anche avere K **RadioMap** associate ad esso poiché è possibile costruire più radio mappe contenenti un numero differente di **Reference Point** che si riferiscono allo stesso piano.

3.4 Sviluppo del server

Si passa ora a mostrare il modo nel quale è stato implementato il server che, come si è già detto in precedenza è un'estensione dell'architettura **Wi-Lo** [17], si basa su un *web server* **Apache2** con pagine dinamiche scritte in **PHP 7.0**. Il lato server è stato interamente scritto attraverso l'uso

dell'*IDE PHP Storm* rilasciato per tutte le piattaforme dalla software house *JetBrains*. Il database utilizzato è di tipo *noSql* ed è *MongoDB*. Per poter interfacciare PHP con MongoDB è stato opportuno installare un *driver* che ha permesso la comunicazione tra i due. Il server mette a disposizione i servizi, che vengono richiamati dal client, per poter eseguire la localizzazione tramite *WiFi-Fingerprinting*. Tra di essi ci sono due tipi diversi di servizi: i primi servono per inserire le *radiomap* nel database mentre gli altri forniscono la localizzazione.

3.4.1 Servizi di inserimento delle Radiomap

I primi servizi di cui si tratta in questo documento sono quelli relativi all'inserimento delle **Radiomap**, dei **Reference Point** e delle misurazioni e di seguito viene mostrato il codice dei principali servizi.

Inserimento di una Radiomap

Il primo servizio è l'inserimento di una radiomap che, nel client, viene richiamato nella **AddRadioMapActivity** dove vengono raccolte solamente due informazioni: il nome della mappa radio e il piano, presente nel database interno del client, a cui fa riferimento, come si è visto nello schema di Figura 3.11. Il servizio è responsabile di acquisire queste informazioni e immagazzinarle nel database. Il codice è mostrato di seguito:

```
<?php
require '../vendor/autoload.php';
$connection = new MongoDB\Client("mongodb://localhost:27017");
$collection = $connection->StepApp->Floors;
$jsonData = json_decode(file_get_contents('php://input'), true);
if ($jsonData != NULL) {
    $radioMapName=$jsonData['name'];
    $floorId=$jsonData['floorId'];
    $collection->insertOne([ 'name' => $radioMapName, 'floorId' => $floorId ] );
    $result = array("result" => 1);
} else
     $result = array("result" => 0);
echo json_encode($result);
?>
```

Inserimento di un Reference Point

In modo simile al precedente, questo servizio si occupa di inserire i dati riguardanti un *Reference Point*. Il servizio viene richiamato dalla **AddReferencePointActivity** e comunicati tramite un file *JSON*. Lo scopo è quello di immagazzinare i dati di nome del Reference Point, Radiomap a cui fa riferimento e le coordinate (x, y) nella mappa, all'interno del database. Il cordice di riferimento è il seguente:

```
<?php
require '../vendor/autoload.php';
$connection = new MongoDB\Client("mongodb://localhost:27017");
$collection = $connection->StepApp->RefPoints;
$jsonData = json_decode(file_get_contents('php://input'), true);
if ($jsonData != NULL) {
    $nameRP=$jsonData['name'];
    $radiomap=$jsonData['radiomap'];
    $x=$jsonData['x'];
    $y=$jsonData['y'];
    $result=$collection->count(['name'=> $nameRP, 'radiomap'=> $radiomap, 'x'=>$x, 'y'=>
        $y]);
    if ($result == 0) {
        $collection->insertOne([ 'name' => $nameRP, 'radiomap'=> $radiomap, 'x'=>$x, 'y'
            =>$y ] );
        $result = array("result" => 1);
    } elsedue informazioni: Il nome della mappa radio e il piano, presente nel data-
        $result = array("result" => 0);
} else
     $result = array("result" => 0);
echo json_encode($result);
?>
```

Inserimento delle misurazioni

L'ultimo servizio appartenente a quelli di inserimento nel database è quello che permette di memorizzare le misurazioni degli Access Point effettuate dal client con la **PopulateRPActivity**. Il servizio si occupa di memorizzare il codice MAC dell'Access Point, la media e la varianza della potenza trasmissiva RSS, il numero di scansioni effettuate, la mappa radio e il reference point a cui fanno riferimento. Il codice PHP del servizio è mostrato di seguito:

```
<?php
require '../vendor/autoload.php';
$connection = new MongoDB\Client("mongodb://localhost:27017");
$collection = $connection->StepApp->Measurements;
$jsonData = json_decode(file_get_contents('php://input'), true);
if ($jsonData != NULL) {
    $rp=$jsonData['rp'];
```

```
    $radiomap=$jsonData['radiomap'];
    foreach ($jsonData['measurements'] as $item){
        if (strcmp($item["type"],"wifi")==0) {
            foreach ($item['data'] as $row){
                $bbsid=$row['bssid'];
                $powerMean=$row['mean'];
                $powerVariance=$row['variance'];
                $numtest=$row['numtest'];
                try {
                    $collection->insertOne([ 'rp' => $rp, 'radiomap'=> $radiomap, 'type'
                        =>'wifi', 'bssid' => $bbsid, 'mean' => $powerMean, 'variance' =>
                         $powerVariance, 'numtest' => $numtest]);
                } catch (MongoCursorException $e) {
                    $result = array("result" => 0);
                    break;
                }
            }
        }
    }
    $result = array("result" => 1);
} else {
    $result = array("result" => 0);
}
echo json_encode($result);
?>
```

3.4.2 Servizi di localizzazione

I servizi di localizzazione sono adeguatamente illustrati nel capitolo *2. Progettazione*; in questo paragrafo viene semplicemente mostrato il codice dei tre servizi che permettono la localizzazione attraverso *WiFi-Fingerprinting* e che vengono richiamati nella NavigationActivity.

Name Based

```
<?php
include_once("algorithm.php");
include_once("entryRP.php");
class MatchNames extends AlgorithmMatching  {
    private $numMatch;
    public function __construct() {
        $this->numMatch=array();
    }
    public function computeMatch($detection,$training) {
        foreach ($training as $itemT){
            foreach ($detection as $itemD){
                $rpName=$itemT['rp'];
                if (strcmp($itemT['bssid'],$itemD['bssid'])==0) {

                    if (array_key_exists($rpName, $this->numMatch))
                        $this->numMatch[$rpName]++;
                    else
                        $this->numMatch[$rpName]=1;
                }
            }
```

```
        }
    }
    public function selectBest() {
        $maxValue=-1;
        $maxX=-999999.99;
        $maxY=-999999.99;
        $arrayResult=array();
        foreach($this->numMatch as $elem) {
            if ($elem >= $maxValue)
                $maxValue=$elem;
        }
        foreach ($this->numMatch as $key => $value) {
            if ($value==$maxValue) {
                $possibleSolution=new EntryRP($key,$maxValue,$maxX, $maxY);
                array_push($arrayResult,$possibleSolution);
            }
        }
        return $arrayResult;
    }
    public function selectBestWithTolerance($tolerance) {
        $maxValue=-1;
        $maxX=-999999.99;
        $maxY=-999999.99;
        $arrayResult=array();
        foreach($this->numMatch as $elem) {
            if ($elem >= $maxValue)
                $maxValue=$elem;
        }
        if ($maxValue!=-1) {
            $tolerance=$maxValue * $tolerance;
        }
        foreach ($this->numMatch as $key => $value) {
            $distance=abs($value-$maxValue);
            if ($distance<=$tolerance) {
                $possibleSolution=new EntryRP($key,$maxValue,$maxX,$maxY);
                array_push($arrayResult,$possibleSolution);
            }
        }
        return $arrayResult;
    }
}
?>
```

RSS Based

```
<?php
include_once("algorithm.php");
include_once("entryRP.php");
class MatchRSS extends AlgorithmMatching  {
    private $errorMatch;
    public function __construct() {
        $this->errorMatch=array();
    }
    public function computeMatch($detection,$training) {
        foreach ($training as $itemT){
            foreach ($detection as $itemD){
                $rpName=$itemT['rp'];
                if (strcmp($itemT['bssid'],$itemD['bssid'])==0) {
                    if (array_key_exists($rpName, $this->errorMatch))
                        $this->errorMatch[$rpName]+=pow($itemT['mean']-$itemD['mean'],2)
                            ;
                    else
                        $this->errorMatch[$rpName]=pow($itemT['mean']-$itemD['mean'],2);
```

```
                }
            }
        }
    }
    public function selectBest() {
        $minValue="None";
        $maxX=-999999.99;
        $maxY=-999999.99;
        $arrayResult=array();
        foreach($this->errorMatch as $elem) {
            if (($elem <= $minValue) || (strcmp($minValue,"None")==0))
                $minValue=$elem;
        }
        foreach ($this->errorMatch as $key => $value) {
            if ($value==$minValue) {
                $possibleSolution=new EntryRP($key,$minValue, $maxX, $maxY);
                array_push($arrayResult,$possibleSolution);
            }
        }
        return $arrayResult;
    }
    public function selectBestWithTolerance($tolerance) {
        $minValue="None";
        $maxX=-999999.99;
        $maxY=-999999.99;
        $arrayResult=array();
        foreach($this->errorMatch as $elem) {
            if (($elem <= $minValue) || (strcmp($minValue,"None")==0))
                $minValue=$elem;
        }
        if (strcmp($minValue,"None")!=0) {
            $tolerance=$minValue * $tolerance;
        }
        foreach ($this->errorMatch as $key => $value) {
            $distance=abs($value-$minValue);
            if ($distance<=$tolerance) {
                $possibleSolution=new EntryRP($key,$minValue, $maxX, $maxY);
                array_push($arrayResult,$possibleSolution);
            }
        }
        return $arrayResult;
    }
}
?>
```

Name + RSS Based

```
<?php
include_once("algorithm.php");
include_once("entryRP.php");
class MatchNamesRss extends AlgorithmMatching  {
    private $numMatch;
    private $errorMatch;
    public function __construct() {
        $this->numMatch=array();
        $this->errorMatch=array();
    }
    public function computeMatch($detection,$training) {
        foreach ($training as $itemT){
            foreach ($detection as $itemD){
                $rpName=$itemT['rp'];
                if (strcmp($itemT['bssid'],$itemD['bssid'])==0) {
                    if (array_key_exists($rpName, $this->numMatch)) {
```

```
                    $this->errorMatch[$rpName]+=pow($itemT['mean']-$itemD['mean'],2)
                        ;
                    $this->numMatch[$rpName]++;
                }
                else {
                    $this->errorMatch[$rpName]=pow($itemT['mean']-$itemD['mean'],2);
                    $this->numMatch[$rpName]=1;
                }
            }
        }
    }
}
public function selectBest() {
    $maxValue=-1;
    $minValue="None";
    $maxX=-999999.99;
    $maxY=-999999.99;
    $arrayResult=array();
    foreach($this->numMatch as $elem) {
        if ($elem >= $maxValue)
            $maxValue=$elem;
    }
    foreach ($this->numMatch as $key => $value) {
        if ($value==$maxValue) {
            if (($this->errorMatch[$key] <= $minValue) || (strcmp($minValue,"None")
                ==0)) {
                $minValue=$this->errorMatch[$key];
            }
        }
    }
    foreach ($this->numMatch as $key => $value) {
        if (($value==$maxValue) && ($this->errorMatch[$key]==$minValue)) {
            $possibleSolution=new EntryRP($key,$maxValue, $maxX, $maxY);
            array_push($arrayResult,$possibleSolution);
        }
    }
    return $arrayResult;
}
public function selectBestWithTolerance($tolerance) {
    $maxValue=-1;
    $minValue="None";
    $maxX=-999999.99;
    $maxY=-999999.99;
    $arrayResult=array();
    foreach($this->numMatch as $elem) {
        if ($elem >= $maxValue)
            $maxValue=$elem;
    }
    foreach ($this->numMatch as $key => $value) {
        if ($value==$maxValue) {
            if (($this->errorMatch[$key] <= $minValue) || (strcmp($minValue,"None")
                ==0)) {
                $minValue=$this->errorMatch[$key];
            }
        }
    }
    if (strcmp($minValue,"None")!=0) {
        $tolerance=$minValue*$tolerance;
    }
    foreach ($this->numMatch as $key => $value) {
        $distance=abs($this->errorMatch[$key]-$minValue);
        if (($value==$maxValue) && ($distance <=$tolerance)) {
            $possibleSolution=new EntryRP($key,$maxValue, $maxX, $maxY);
            array_push($arrayResult,$possibleSolution);
```

```
            }
        }
        return $arrayResult;
    }
}
?>
```

3.5 Fusione dei metodi di localizzazione

L'ultimo paragrafo di questo capitolo è dedicato alle due *Android Activity* responsabili della fusione dei due metodi di localizzazione indoor.

3.5.1 Preparazione alla localizzazione

ManageNavigationActivity è la prima *Activity* che viene visualizzata quando si intende esplorare uno scenario indoor mappato. Il layout è visibile in Figura 3.12. ed è così strutturato: nella parte iniziale sono presenti tre *Android Spinner* che permettono la scelta della mappa, dell'edificio e del piano da navigare presenti nel database interno al client; al di sotto è presente una *Android Chcekbox* che permette di abilitare la localizzazione tramite *WiFi-Fingerprinting* (in alternativa il sistema utilizzerebbe automaticamente solo quella tramite *PDR*) e se abilitata permette di scegliere quale *Radiomap* utilizzare tra quelle presenti nel database del server; scendendo ancora in basso è presente un *Android RadioGroup* che contiene dei *Radiobutton* che permettono la scelta tra i cinque *algoritmi di fusing* presentati nel capitolo precedente; in fondo sono presenti due bottoni, il primo serve a calibrare il *magnetometro* andando ad impostare il *Nord Locale* come spiegato in precedenza mentre il secondo permette il passaggio all'activity di navigazione e localizzazione.

Localizzazione

L'*Android Activity* principale di questo progetto è la **NavigationActivity** che racchiude tutte le funzionalità viste fino ad ora. Il layout è minimale e intuitivo ed è visibile in Figura 3.13: in alto sono presenti le coordinate (x, y) del piano sul quale l'utente si sta muovendo fornendo la *posizione fisica*; al

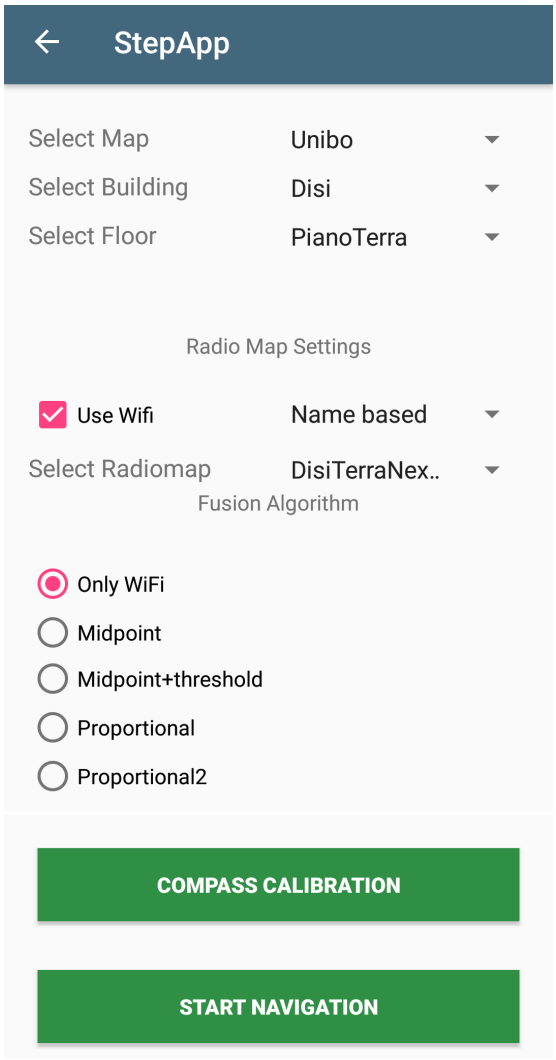

Figura 3.12: ManageNavigationActivity

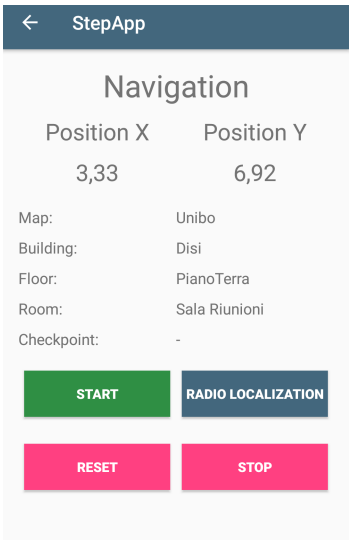

Figura 3.13: NavigationActivity

centro sono presenti le informazioni riguardanti la *posizione simbolica*; in basso sono presenti quattro bottoni per l'interazione con il dispositivo: *Start*, *Stop* e *Reset* gestiscono la *Localizzazione PDR* e agiscono in mutua esclusione, *RadioLocalization* richiede invece la *Localizzazione WiFi-Fingerprinting*. Il funzionamento è elementare: quando un utente si muove nello scenario mappato, la parte *PDR* provvede ad aggiornare le coordinate dell'utente nell'ambiente e controlla in quale delle stanze si trova e se è nei pressi di un checkpoint, quando viene premuto il bottone **RadioLocalization** si esegue la localizzazione *WiFi-FP*. Una volta ricevuta la risposta dal server il dispositivo fonde i dati calcolati da esso stesso e quelli comunicati dal server secondo l'algoritmo scelto nell'*Activity* precedente e aggiorna le coordinate.

Capitolo 4

Valutazione sperimentale

4.1 Introduzione alla valutazione sperimentale

In questo capitolo vengono descritti tutti i test che sono stati effettuati sul sistema progettato al fine di valutarne l'accuratezza in modo sperimentale. I test sono stati suddivisi in tre categorie. Inizialmente si è valutata l'accuratezza con il quale il client riesce a rilevare un passo e a calcolarne la lunghezza, successivamente si è testata la precisione del sistema utilizzando solamente la tecnica *Pedestrian Dead Reckoning* all'interno di due ambienti indoor e, in fine, si è testata la fusione delle tecniche *PDR* e *WiFi-Fingerprinting* negli stessi ambienti per valutare eventuali miglioramenti nella localizzazione. Il capitolo si conclude con un'accurata analisi dei risultati ottenuti dai test.

4.2 Test iniziali

4.2.1 Configurazione del test

Come spiegato nell'introduzione di questo capitolo i primi test sono stati effettuati per valutare l'accuratezza della tecnica *peak detection* e dei due algoritmi di stima della lunghezza del passo. I test consistono nel far percorrere ad un utente cento passi in linea retta, prendere nota di quanti metri sono

stati realmente compiuti e confrontarli con quelli percepiti dall'applicazione. Questa sessione di test è stata eseguita analizzando tutte le possibili combinazioni di utenti, metodi di riconoscimento del passo, stima della lunghezza del passo, dispositivi utilizzati e posizione dei dispositivi durante il test. I test sono stati effettuati da due utenti con caratteristiche fisiche differenti:

- *User 1*: utente di sesso maschile, destrimano e di 181 cm di altezza;
- *User 2*: utente di sesso femminile, mancino e di 165 cm di altezza.

I sensori software per il riconoscimento del passo tramite *peak detection* sono quelli introdotti nei capitoli precedenti:

- *Step Detector*;
- *Step Counter*.

Gli algoritmi di stima della lunghezza del passo sono quelli spiegati nel capitolo *2.Progettazione*:

- *Static Step Length*;
- *Dynamic Step Length*.

I dispositivi utilizzati durante i test sono due smartphone di produttori differenti equipaggiati con hardware differenti:

- *Device A*: LG Nexus5 (Android 6.0);
- *Device B*: Oneplus X (Android 6.0).

Entrambi i dispositivi sono stati testati posizionandoli in quattro differenti modalità:

- *Device tenuto con la mano destra*
- *Device tenuto con la mano sinistra*
- *Device tenuto in una borsa appoggiata sul braccio destro*
- *Device tenuto in una borsa appoggiata sul braccio sinistro*

4.2.2 Analisi dei dati

In questo paragrafo si analizzano i dati collezionati durante i test, si mettono a confronto inizialmente gli algoritmi di riconoscimento del passo e successivamente gli algoritmi di stima della lunghezza del passo.

Riconoscimento del passo

Come detto più volte nei capitoli precedenti i due sensori software che implementano la tecnica *peak detection* si basano entrambi sulla fusione dei dati provenienti dalle accelerazioni sui tre assi percepite dall'accelerometro che vengono confrontati con dei pattern già noti in letteratura. In questo caso per valutare l'accuratezza sono stati eseguiti in ogni sessione cento passi in linea retta con delle soste ogni dieci passi ed il tutto è stato ripetuto per entrambi gli utenti, con entrambi i device, posizionandoli nelle quattro modalità sopra indicate e utilizzando i due metodi di riconoscimento del passo per un totale di *tremiladuecento* passi compiuti. Data la mole importante dei dati raccolti è impossibile riportarli in modo completo, quindi si mostrano e si descrivono direttamente i risultati utilizzando degli istogrammi.
Analizzando i dati raccolti durante le sessioni di test si evince che gli algoritmi di peak detection sono confrontabili, come è possibile vedere in Figura 4.1, dal punto di vista dell'errore assoluto e quindi proporzionalmente anche dal punto di vista dell'accuratezza. Infatti entrambi gli algoritmi riescono a percepire oltre il 90% dei passi compiuti con un piccolo vantaggio nei confronti di Step Counter.

Tuttavia si può notare come l'accuratezza sia sensibilmente influenzata dalle differenze hardware di ogni dispositivo. Infatti, come si può notare in Figura 4.2 dove viene mostrata la distribuzione degli errori assoluti nei vari dispositivi, il Device B è nettamente più impreciso del Device A.

Volendo studiare separatamente le prestazioni dei due algoritmi nei due dispositivi, in Figura 4.3 e in Figura 4.4, si sottolinea ancora più fortemente le differenze tra le precisioni dei due device, distaccati in entrambi i metodi di quasi dieci punti percentuale.

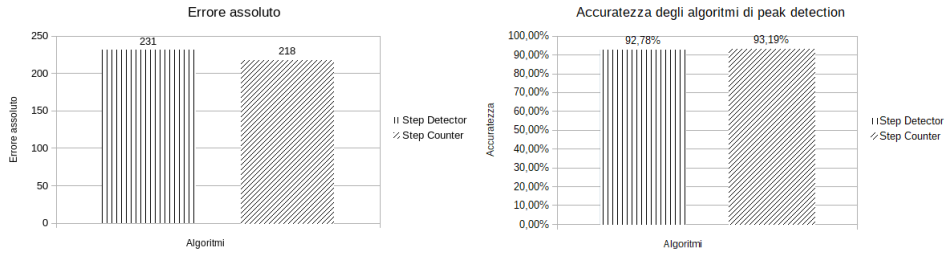

Figura 4.1: Accuratezza degli algoritmi peak detection

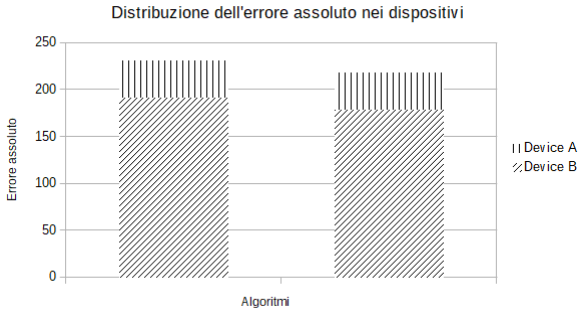

Figura 4.2: Distribuzione dell'errore dell'algoritmo peak detection sui dispositivi

Analizzando ancora più nel dettaglio gli effetti delle differenze hardware dei due dispositivi sui due algoritmi si è calcolato l'errore relativo, attraverso la formula $\frac{|passi\ \ effettuati\ -\ passi\ \ percepiti|}{|passi\ \ effettuati|}$, si evince che il Device A presenta un errore relativo dell'ordine di 10^{-2} mentre il Device B dell'ordine di 10^{-1}.

Stima della lunghezza del passo

I test riguardanti la stima della lunghezza del passo sono stati effettuati in modo analogo a quelli precedenti. Questa volta sono stati effettuati cento passi, sono stati annotati i metri compiuti realmente ad ogni sessione e sono stati confrontati con i metri calcolati dal dispositivo. Ovviamente sono stati testati entrambi i metodi di stima della lunghezza del passo *Dynamic*

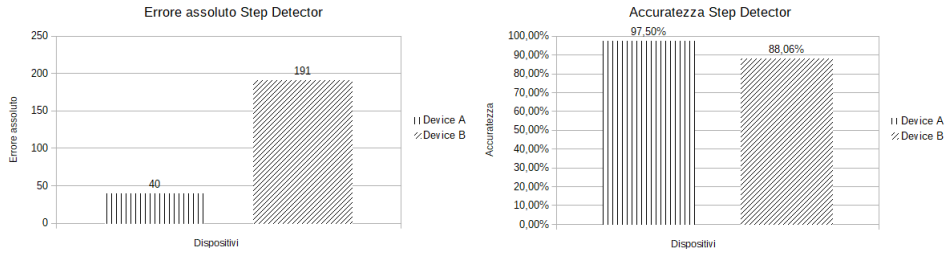

Figura 4.3: Accuratezza dell'algoritmo Step Detector

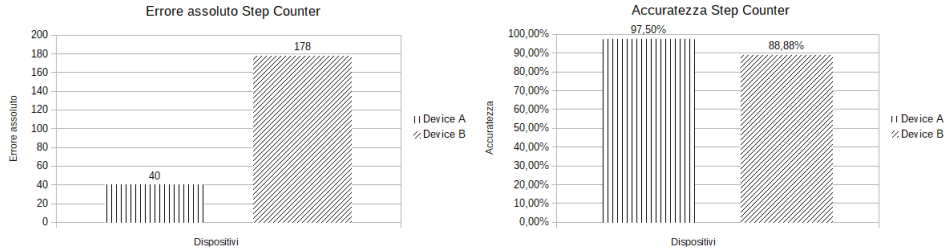

Figura 4.4: Accuratezza dell'algoritmo Step Counter

Step Length e *Static Step Length* su due utenti differenti, con due dispositivi differenti e posizionandoli nelle quattro modalità precedentemente elencate per un totale, anche in questo caso, di *tremiladuecento* passi compiuti. Come in precedenza, data la grande mole di dati raccolti, si mostrano solamente i risultati ottenuti dopo l'analisi dei dati.
Osservando i dati dei test, in particolare l'accuratezza media dei due algoritmi, si nota in Figura 4.5 come la tecnica Static Step Length migliori le prestazioni rispetto alla tecnica Dynamic Step Length. Tuttavia durante le sessioni di test è emerso che tali valori sono fortemente influenzati dalla modalità con la quale sono stati raccolti i dati. Infatti dividendo i risultati in base alle posizioni dei device durante il test (in mano o in borsa) si nota che l'algoritmo Dynamic Step Length è pesantemente penalizzato dalla posizione in borsa perché, come spiegato in precedenza, esso si basa sui dati raccolti dall'accelerometro sull'asse verticale. Tenendo il dispositivo in una posizione casuale all'interno di una borsa esso può acquisire dei dati falsati.

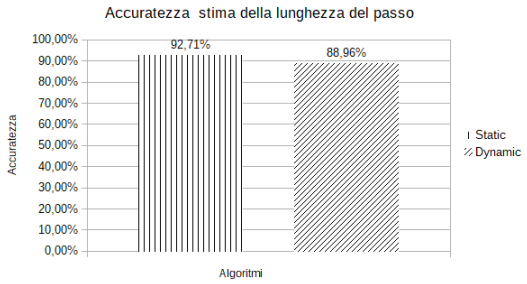

Figura 4.5: Accuratezza degli algoritmi di stima della lunghezza del passo

Non considerando le sessioni con i dispositivi in borsa emerge un ribaltamento delle accuratezze in favore di Dynamic Step Length come possibile vedere in Figura 4.6. Data la natura e lo scopo dell'applicazione sviluppata, è più indicato utilizzare il dispositivo in mano, quindi in questa situazione si considera più accurato l'algoritmo Dynamic Step Length. Come accaduto per l'anali-

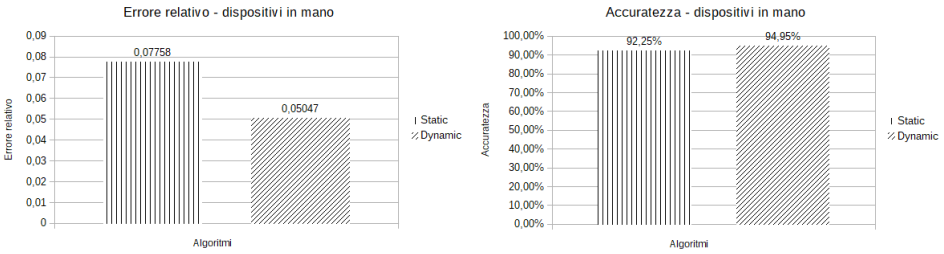

Figura 4.6: Accuratezza degli algoritmi con dispositivo in mano

si precedente si è studiata l'errore assoluto, calcolato attraverso la formula $|metri\ compiuti - metri\ percepiti|$, degli algoritmi nei due dispositivi al fine di verificare se le differenze hardware potessero essere influenti. Come mostrato in Figura 4.7 anche in questo caso le migliori prestazioni vengono registrate in corrispondenza del Device A.

Dati quindi i risultati molto soddisfacenti ottenuti da questi test si può dire che la combinazione **Step Counter & Static Step Length**, seppur di poco, sia migliore rispetto alle altre combinazioni possibili e per questo

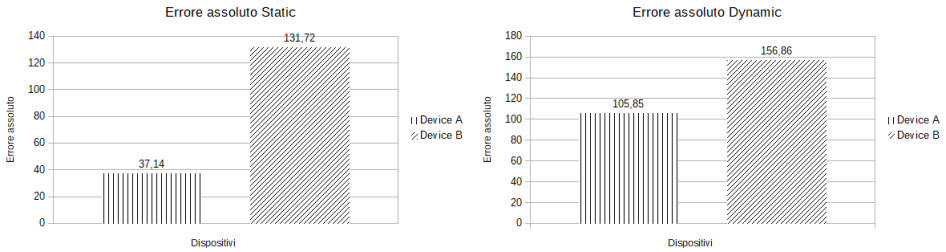

Figura 4.7: Accuratezza dei dispositivi nella stima della lunghezza del passo

motivo nella **NavigationActivity** finale sono stati implementati questi due algoritmi per la localizzazione *PDR*.

4.3 Test della tecnica PDR

4.3.1 Configurazione del test

Questo test è stato effettuato per valutare l'accuratezza del sistema *Pedestrian Dead Reckoning* nella sua interezza, quindi oltre alle tecniche **Step Counter & Static Step Length** si aggiunge la complessità della direzione del passo. I test sono stati effettuati in cinque sessioni ognuna delle quali dallo *User 1*, con i due dispositivi citati precedentemente ripetuti in due ambienti indoor differenti e con differenti percorsi per un totale di 20 test.

4.3.2 Ambiente indoor

Gli ambienti nei quali sono stati effettuati i test sono il *piano terra* (222,8 mq) e il *piano interrato* (452,4 mq) del **DISI** (Dipartimento di Informatica, Scienza e Ingegneria) dell'**Università di Bologna**. Le planimetrie dell'edificio, visibili nella Figura 4.8 (a sinistra il piano terra e a destra il piano interrato) sono state gentilmente concesse dall'Università di Bologna tramite il *prof. Marco Di Felice* e sono state inserite all'interno del client utilizzando le *Android Activity* presentate nei capitoli precedenti.

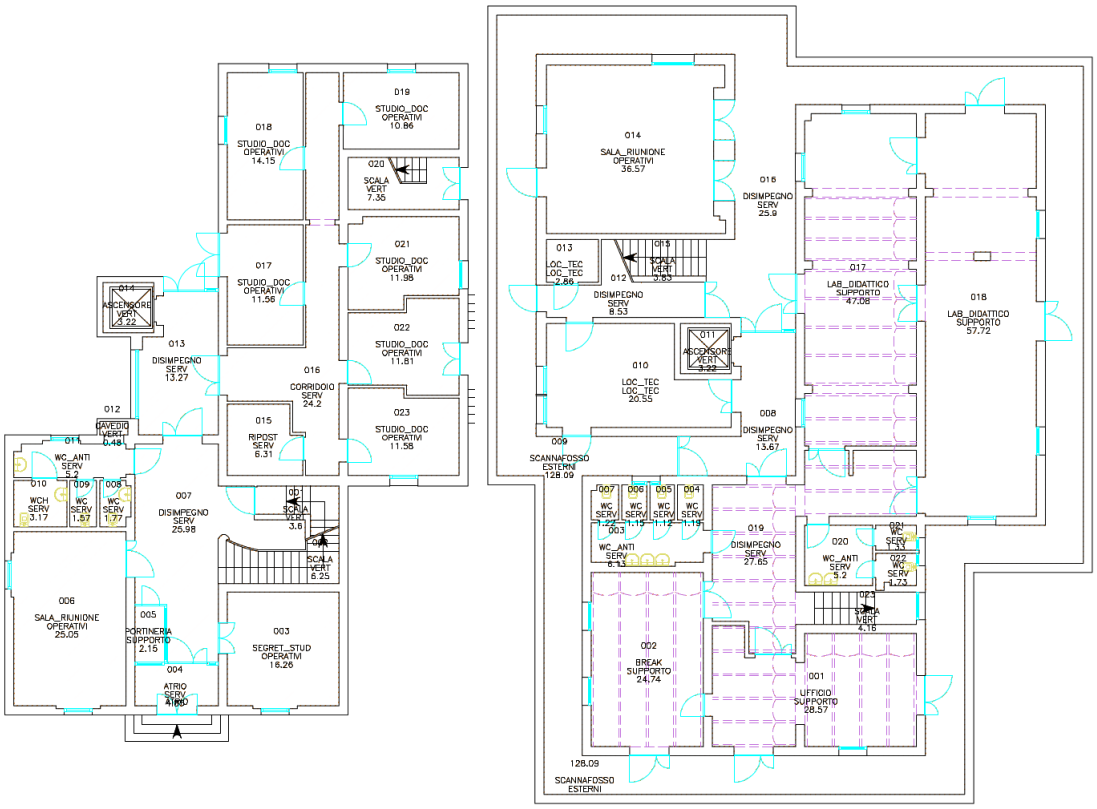

Figura 4.8: Planimetrie degli ambienti indoor

4.3.3 Analisi dei dati della tecnica PDR

In questa sezione si analizzano i dati collezionati durante i test del metodo *PDR* negli scenari indoor descritti. Vengono trattati in modo atomico i due ambienti perché, come si vedrà dai grafici ottenuti, ci sono forti differenze tra i risultati e si cercherà di fornire una motivazione plausibile.

Piano Terra

In Figura 4.9 si possono vedere i risultati degli esperimenti effettuati rispettivamente con il *Device A* (a sinistra) e con il *Device B* (a destra). Il percorso di colore *Rosso* è il percorso che è stato effettivamente compiuto durante i test, gli altri colori sono i percorsi calcolati dai rispettivi dispositivi.

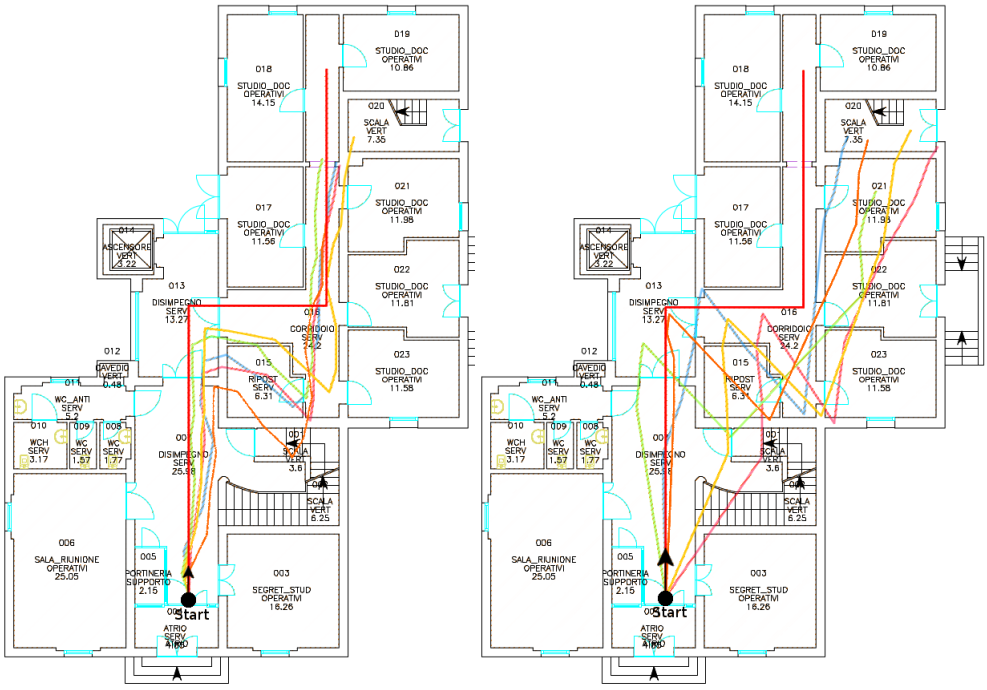

Figura 4.9: Percorsi dei device nel piano terra

Come si evince dalle immagini i risultati di questi test sono stati estremamente soddisfacenti, infatti il *Device A* presenta un errore medio pari a $3,80m$ nella sua posizione finale e, al fronte dei $26m$ percorsi, risulta avere un'accuratezza del $85,38\%$. Per il *Device B* possono essere fatte le stesse considerazioni, infatti presenta un errore pari a $4,38m$ nella sua posizione finale che tradotto porta ad un'accuratezza del $83,15\%$. Le piccole differenze presenti tra l'accuratezza dei due test, notata anche durante i test relativi al riconoscimento del passo e alla stima della sua lunghezza, possono essere riconducibili alle differenze di sensibilità dei sensori hardware nei due dispositivi utilizzati.

Piano interrato

Se i risultati ottenuti dai test nel piano terra possono ritenersi soddisfacenti, non si può dire lo stesso per quelli riguardanti il piano interrato.

Infatti, come possibile vedere in Figura 4.10 rispettivamente eseguiti con il *Device A* (a sinistra) e poi con il *Device B* (a destra), il metodo *PDR* in questa situazione ha riportato risultati poco soddisfacenti. Nelle immagini è possibile vedere come il percorso reale compiuto di colore *rosso* non venga perfettamente seguito dai percorsi calcolati dai dispositivi. Analizzando

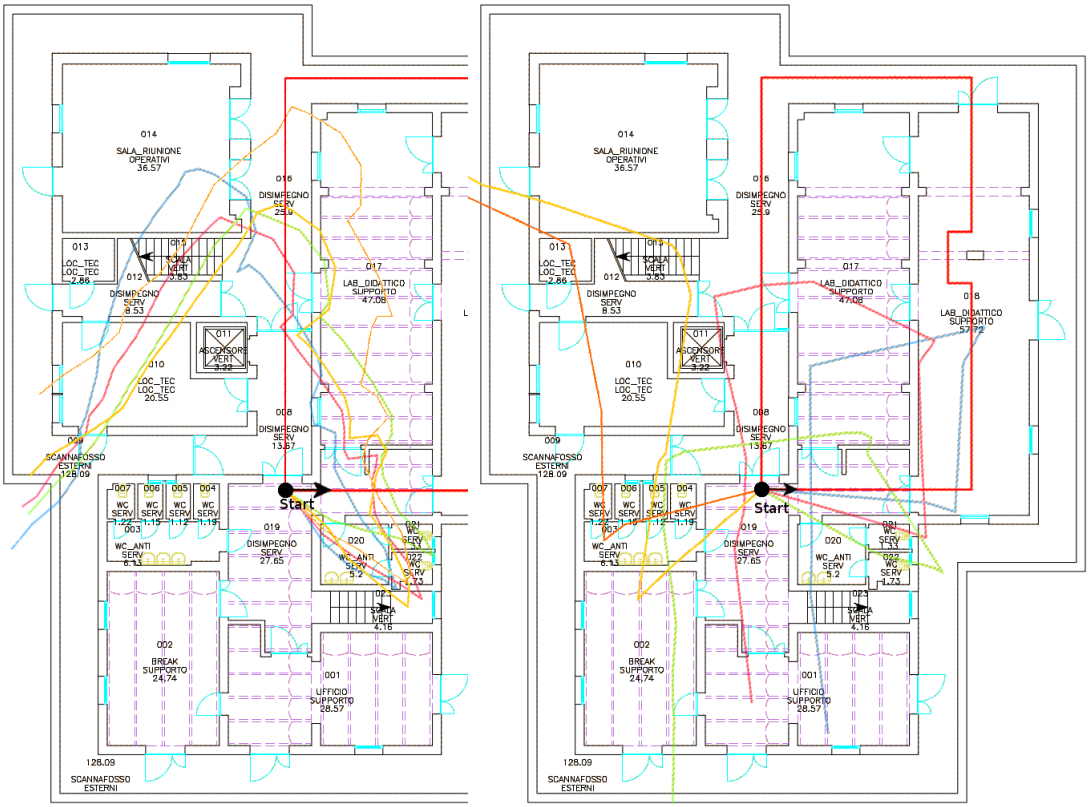

Figura 4.10: Percorsi dei device nel piano interrato

la distanza reale percorsa e quella calcolata dai sensori si evince che l'algoritmo di percezione del passo e quello di stima della lunghezza del passo hanno funzionato correttamente (al netto degli errori già conosciuti con i test precedenti). Infatti calcolando la distanza percorsa realmente, che è pari a *52m*, e confrontandola con le distanze medie calcolate dai due dispositivi, *57,12m* per il *Device A* e *44,19m* per il *Device B* si evince che la causa è sicuramente riconducibile all'errato funzionamento del magnetometro. A

confermare questa ipotesi si sono svolte ricerche più approfondite, effettuate sia con la bussola digitale presente nella **StepsInformationsActivity** che con una bussola reale, è emerso che il *piano interrato* del *DISI* presenta alcune anomalie magnetiche. Una delle cause risiede nel fatto che si sono riscontrati picchi di magnetismo in concomitanza del *Locale tecnico* perché, evidentemente, l'hardware collocato al suo interno provoca sbalzi del campo magnetico. Un altro fattore da prendere in considerazione è il fatto che è presente un'elevata concentrazione di ferro nella struttura che circonda il *piano interrato* e questo può scatenare un effetto **Gabbia di Faraday** all'interno dell'ambiente rendendo non attendibili i valori calcolati dal magnetometro.

Risultati degli esperimenti sulla tecnica PDR

Andando ad analizzare le medie dei percorsi calcolati dai dispositivi nei due ambienti, visibili in Figura 4.11, si possono notare due dati molto significativi. Il primo è che, a parità di dispositivo, gli errori nel *piano interrato* sono più accentuati per i motivi già espressi in questa sezione. L'altro dato che viene ulteriormente confermato da questo grafico è che le differenze hardware dei dispositivi influiscono pesantemente sull'accuratezza delle soluzioni implementate.

4.4 Test degli algoritmi di fusion

4.4.1 Configurazione del test sugli algoritmi di fusion

I test riguardanti gli algoritmi di *fusion* sono stati compiuti nei due stessi ambienti indoor dei test precedenti al fine di valutare eventuali miglioramenti nell'accuratezza del sistema. Come per le sessioni precedenti anche in questo caso sono state provate tutte le combinazioni possibili che comprendono *cinque test*, su *tre algoritmi di localizzazione WiFi-FIngerprinting*, con *cinque algoritmi di fusione*, in *due ambienti indoor* e ripetuti con *due dispositivi* per un totale di 300 test totali. Per applicare la tecnica *WiFi-Fingerprinting*, co-

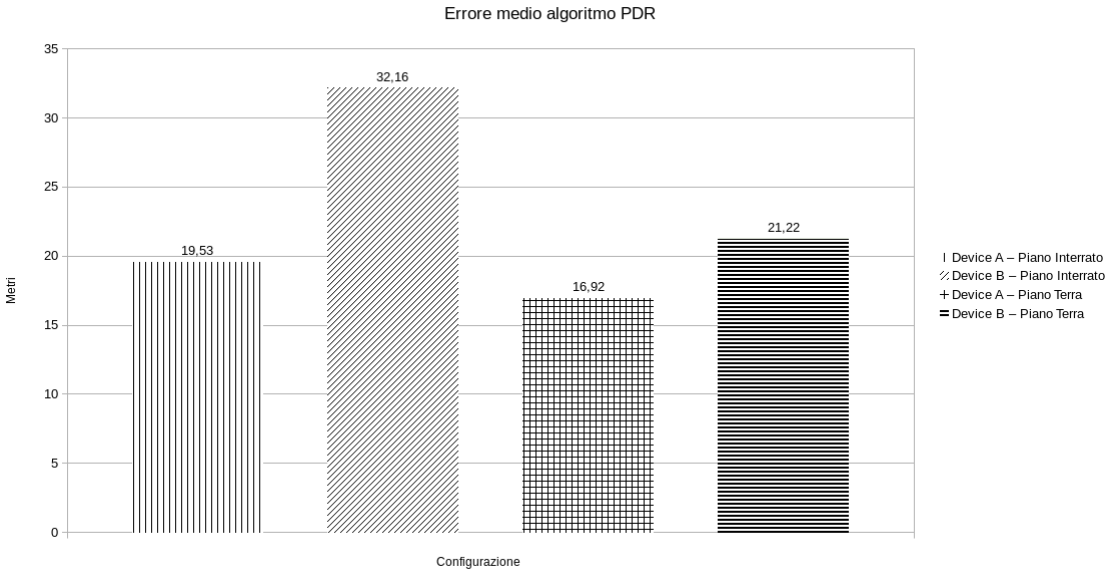

Figura 4.11: Errori nel metodo PDR

me da prassi, è stato opportuno effettuare la mappatura iniziale dell'ambiente scansionando gli *Access Point* da alcuni *Reference Point*. Per motivi pratici gli RP sono stati scelti sul percorso effettuato e sono mostrati nell'ultima sezione di questo capitolo quando verranno mostrati i cammini effettuati.

4.4.2 Analisi dei dati

In questa sezione viene esposta l'analisi dei dati e vengono inizialmente valutati i risultati ottenuti confrontando gli algoritmi implementati per la localizzazione tramite *WiFi-Fingerprinting*, successivamente vengono confrontate le accuratezze tra i gli algoritmi di *fusing* ed in fine vengono analizzate le differenze con il metodo *PDR*.

WiFi-Fingerprinting

I risultati ottenuti dai test sugli algoritmi implementati per il metodo *WiFi-Fingerprinting*, *Name Based, Rss Based* e *Name+Rss Based*, hanno

fatto emergere un'accuratezza elevata da parte di questi algoritmi infatti essi presentano un errore medio di circa $3m$. Traducendo l'errore medio degli algoritmi in *accuratezza* si può confermare che i tre algoritmi siano equiparabili, dal momento che tutti superano il 90%. In Figura 4.12 sono mostrati i risultati.

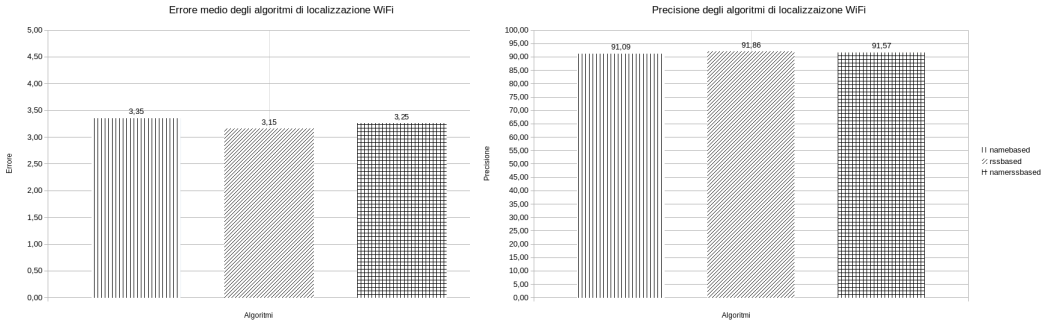

Figura 4.12: Errore e accuratezza algoritmi WiFi-Fingerprinting

Algoritmi di fusion

Come più volte anticipato precedentemente, sono stati sviluppati e valutati *cinque* algoritmi di *fusion*: **Only WiFi**, **Midpoint**, **Midpoint+Threshold**, **Proporzionale** e **Proporzionale2**. L'analisi dei dati provenienti da questi test ha portato alla luce un evidente miglioramento dell'accuratezza media della localizzazione. Come possibile vedere in Figura 4.13 l'errore medio calcolato in metri risulta essere inferiore di oltre il 60% migliorando l'accuratezza della localizzazione di circa il 15% in 3/4 degli algoritmi sviluppati. Dal grafico si sono volutamente omessi i risultati riguardanti l'algoritmo *OnlyWi-Fi* perché avendo effettuato l'analisi dell'accuratezza in concomitanza delle scansioni WiFi si sono ottenuti dei risultati non perfettamente attendibili.

Suddividendo l'analisi dei dati degli algoritmi di fusion per i due ambienti indoor emergono ancora in maniera più nitida i miglioramenti che ha portato la localizzazione tramite WiFi quando le condizioni del campo magnetico sono corrotte. Come possibile vedere in Figura 4.14 nel piano interrato l'errore

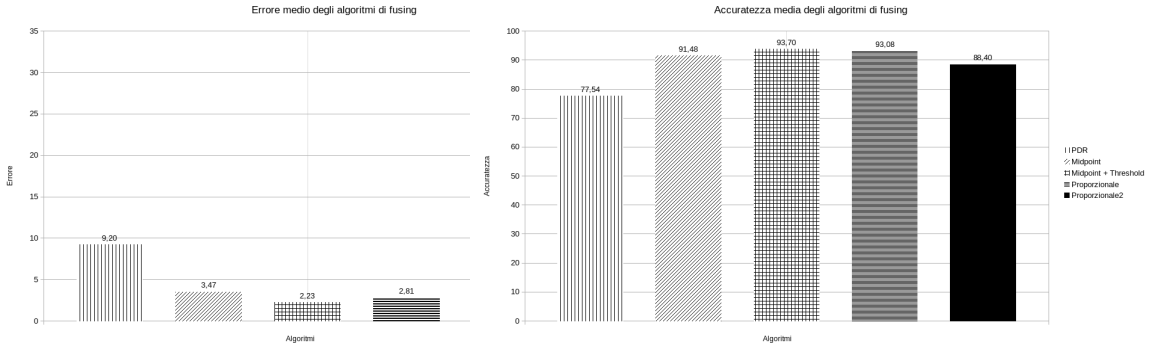

Figura 4.13: Errore e accuratezza algoritmi di fusion

medio è stato drasticamente abbattuto. In Figura 4.15 si possono notare dei

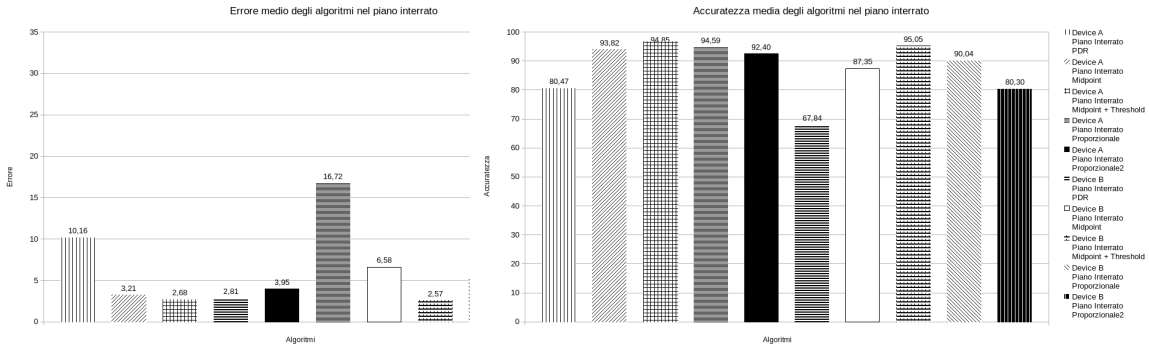

Figura 4.14: Algoritmi a confronto nel piano interrato

lievi miglioramenti che la fusione delle localizzazioni ha portato nel piano terra.

4.4.3 In conclusione

In conclusione si può dire che confrontando i dati ottenuti dai test sugli algoritmi di *fusion* e quelli con il metodo *PDR* c'è un miglioramento nell'accuratezza media rispettivamente dell'$8,56\%$ con l'algoritmo *Midpoint*, dell'$11,19\%$ con l'algoritmo *Midpoint+threshiold*, del $10,45\%$ con l'algoritmo *Proporzionale* e del $4,90\%$ con l'algoritmo *Proporzionale2*. Per quanto

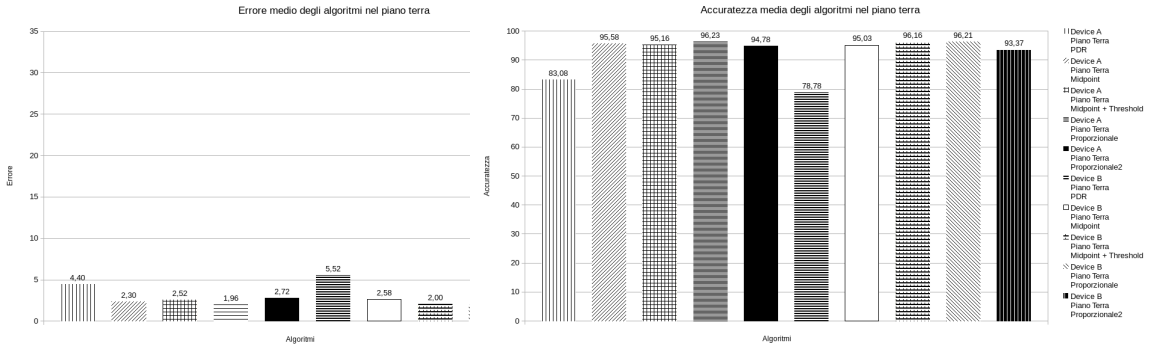

Figura 4.15: Algoritmi a confronto nel piano terra

riguarda il problema riscontrato con il *piano interrato* dove la quantità di ferro presente nell'edificio procurava degli errori di calcolo del magnetometro si può dire che siano stati risolti grazie all'introduzione della tecnica *WiFi-Fingerprinting*. Al fine di mostrare l'effettiva efficacia di questi algoritmi vengono di seguito inserite alcune figure che rappresentano i percorsi migliori calcolati durante le sessioni di test. All'interno delle figure sono stati posizionati dei simboli per specificare i RP. Le figure mettono in risalto le differenze *hardware* dei dispositivi durante l'acquisizione dei dati provenienti dai sensori inerziali e dai sensori WiFi.

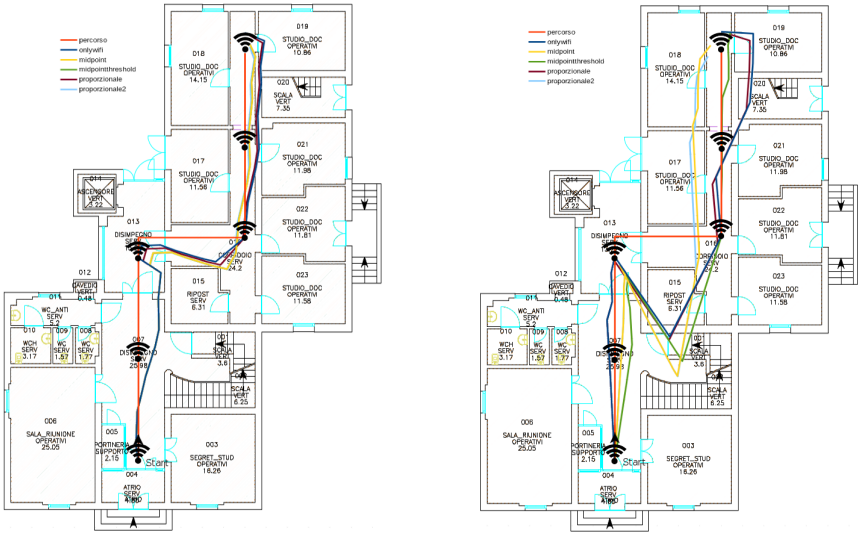

Figura 4.16: Algoritmi di fusion nei cammini nel piano terra

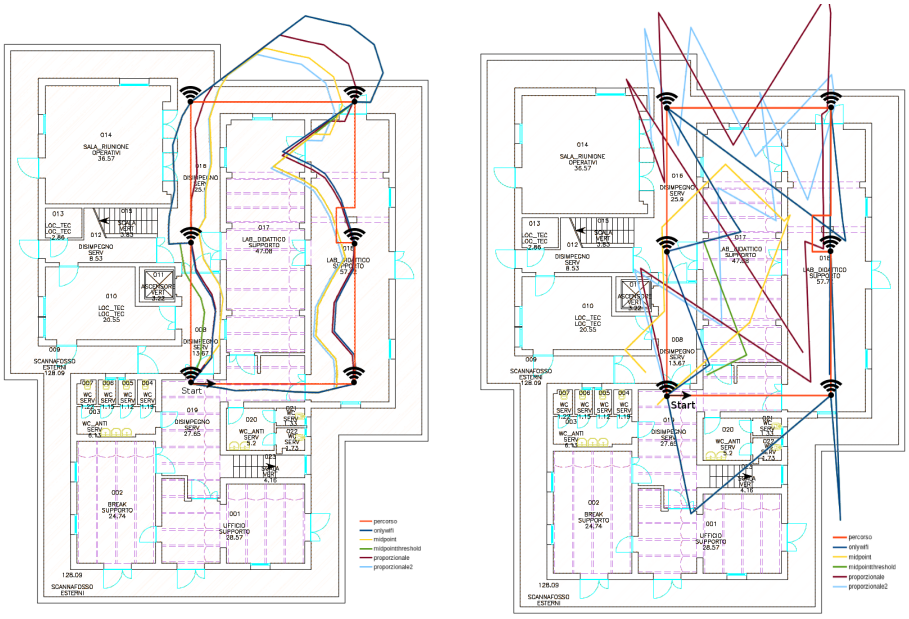

Figura 4.17: Algoritmi di fusion nei cammini nel piano interrato

Conclusioni

In questa tesi è stato presentato **StepApp**, un sistema di localizzazione indoor basato sulla fusione di dati provenienti dai sensori di uno smartphone e dalle radiofrequenze emesse dagli Access Point presenti all'interno di uno scenario. Lo scopo prefissato è quello di realizzare un sistema a basso costo, scalabile e di migliorare l'accuratezza dei sistemi presi singolarmente. Inizialmente è stato presentato lo stato dell'arte delle tecnologie prese in considerazione, successivamente si sono descritte le scelte progettuali e implementative che hanno portato alla realizzazione del sistema ed in fine sono stati mostrati i risultati dei test.
Il primo requisito è sicuramente stato soddisfatto grazie da una buona progettazione del sistema, infatti l'utilizzo di un semplice elaboratore come server e di un tradizionale smartphone Android come client collegati alla stessa rete locale permette di mantenere i costi di realizzazione veramente bassi senza la necessità di un hardware specifico. La scalabilità del sistema è anche dovuta alla buona progettazione, infatti tutte le informazioni necessarie sono facilmente reperibili e l'unica fase che richiederebbe un tempo di elaborazione elevato per essere portata a termine è la fase *offline* del metodo *WiFi-Fingerprinting*. Come si evince dall'analisi dei dati il sistema migliora l'accuratezza dei sistemi singoli superando in quasi tutti i casi il 93%.
In conclusione si può dire che tutti gli obiettivi prefissati per questa tesi sono stati raggiunti fornendo un sistema decisamente accurato, dal basso costo e scalabile. Le tecniche di *peak detection*, gli algoritmi di stima della lunghezza del passo, l'algoritmo di stima della direzione del passo, gli algoritmi

di localizzazione tramite radiofrequenze e gli algoritmi di *fusion* sono stati adeguatamente testati e valutati in modo sperimentale su degli scenari reali. Data la modularità del sistema si apre ad una serie di sviluppi futuri. In primis sarebbe utile implementare un sistema di riconoscimento del piano dell'edificio utilizzando il barometro, altro sensore contenuto negli smartphone. Un altro metodo per migliorare ulteriormente l'accuratezza sarebbe quello di aggiungere un'altra tecnica di localizzazione indoor utilizzando algoritmi di prossimità con l'ausilio di dispositivi *Bluetooth Low Energy*. Una funzione ulteriormente implementabile sarebbe quella di aggiungere la geolocalizzazione satellitare in modo da consentire di spostarsi tra i vari edifici di una stessa mappa caricando automaticamente le impostazioni delle mappe indoor. In fine un'utilissima funzione che può essere aggiunta al sistema è quella della navigazione, infatti sarebbe utile farsi guidare dal dispositivo come avviene per i tradizionali navigatori.

Bibliografia

[1] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080, 2007.

[2] R. Harle. A survey of indoor inertial positioning systems for pedestrians. *IEEE Communications Surveys and Tutorials*, 15(3):1281–1293, 2013.

[3] H. Xing, J. Li, B. Hou, Y. Zhang, and M. Guo. Pedestrian stride length estimation from imu measurements and ann based algorithm. *Journal of Sensors*, 2017, 2017.

[4] L. Fang, P. J. Antsaklis, L. A. Montestruque, M. B. McMickell, M. Lemmon, Y. Sun, H. Fang, I. Koutroulis, M. Haenggi, M. Xie, et al. Design of a wireless assisted pedestrian dead reckoning system-the navmote experience. *IEEE transactions on Instrumentation and Measurement*, 54(6):2342–2358, 2005.

[5] H. Ying, C. Silex, A. Schnitzer, S. Leonhardt, and M. Schiek. Automatic step detection in the accelerometer signal. In *4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007)*, pages 80–85. Springer, 2007.

[6] H. Weinberg. Using the adxl202 in pedometer and personal navigation applications. *Analog Devices AN-602 application note*, 2(2):1–6, 2002.

[7] N. Wang, E. Ambikairajah, S. J. Redmond, B. G. Celler, and N. H. Lovell. Classification of walking patterns on inclined surfaces from accelerometry data. In *2009 16th International Conference on Digital Signal Processing*, pages 1–4, July 2009.

[8] W. Elloumi, A. Latoui, R. Canals, A. Chetouani, and S. Treuillet. Indoor pedestrian localization with a smartphone: A comparison of inertial and vision-based methods. *IEEE Sensors Journal*, 16(13):5376–5388, 2016.

[9] J. W. Kim, H. J. Jang, D. H. Hwang, and C. Park. A step, stride and heading determination for the pedestrian navigation system. *Positioning*, 1(08):0, 2004.

[10] J. E. Bertram and A. Ruina. Multiple walking speed–frequency relations are predicted by constrained optimization. *Journal of theoretical Biology*, 209(4):445–453, 2001.

[11] S. Yang and Q. Li. Ambulatory walking speed estimation under different step lengths and frequencies. In *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*, pages 658–663. IEEE, 2010.

[12] J. Saarinen, J. Suomela, S. Heikkila, M. Elomaa, and A. Halme. Personal navigation system. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 1, pages 212–217. IEEE, 2004.

[13] X. Wang, M. Jiang, Z. Guo, N. Hu, Z. Sun, and J. Liu. An indoor positioning method for smartphones using landmarks and pdr. *Sensors*, 16(12):2135, 2016.

[14] J. Hightower and G. Borriello. Particle filters for location estimation in ubiquitous computing: A case study. In *International conference on ubiquitous computing*, pages 88–106. Springer, 2004.

[15] C. Drane, M. Macnaughtan, and C. Scott. Positioning gsm telephones. *IEEE Communications magazine*, 36(4):46–54, 1998.

[16] N. Kumar, A. Ahmad, and D. Prasad. Survey of downlink control channel resource allocation techniques in lte. In *India Conference (INDICON), 2015 Annual IEEE*, pages 1–5. IEEE, 2015.

[17] M. Di Felice, C. Bocanegra, and K. R. Chowdhury. Wi-lo: Wireless indoor localization through multi-source radio fingerprinting. *To appear on the 10th International Conference on COMmunication Systems NETworkS (IEEE COMNSNETS 2018)*, 2018.

[18] L. Chen, K. Yang, and X. Wang. Robust cooperative wi-fi fingerprint-based indoor localization. *IEEE Internet of Things Journal*, 3(6):1406–1417, 2016.

[19] Y. Kim, Y. Chon, and H. Cha. Mobile crowdsensing framework for a large-scale wi-fi fingerprinting system. *IEEE Pervasive Computing*, 15(3):58–67, July 2016.

[20] C. Wu, Z. Yang, C. Xiao, C. Yang, Y. Liu, and M. Liu. Static power of mobile devices: Self-updating radio maps for wireless indoor localization. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 2497–2505. IEEE, 2015.

[21] D. Mascharka and E. Manley. Lips: Learning based indoor positioning system using mobile phone-based sensors. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual*, pages 968–971. IEEE, 2016.

[22] E. S. Lohan, J. Talvitie, and G. Granados. Data fusion approaches for wifi fingerprinting. In *Localization and GNSS (ICL-GNSS), 2016 International Conference on*, pages 1–6. IEEE, 2016.

[23] M. Brunato and R. Battiti. Statistical learning theory for location fingerprinting in wireless lans. *Computer Networks*, 47(6):825–845, 2005.

[24] C. Wu, L. Fu, and F. Lian. Wlan location determination in e-home via support vector classification. In *Networking, sensing and control, 2004 IEEE international conference on*, volume 2, pages 1026–1031. IEEE, 2004.

[25] P. Prasithsangaree, P. Krishnamurthy, and P. Chrysanthis. On indoor position location with wireless lans. In *Personal, Indoor and Mobile Radio Communications, 2002. The 13th IEEE International Symposium on*, volume 2, pages 720–724. IEEE, 2002.

[26] R. Faragher and R. Harle. Location fingerprinting with bluetooth low energy beacons. *IEEE journal on Selected Areas in Communications*, 33(11):2418–2428, 2015.

[27] Q. Lu, X. Liao, S. Xu, and W. Zhu. A hybrid indoor positioning algorithm based on wifi fingerprinting and pedestrian dead reckoning. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*, pages 1–6. IEEE, 2016.

[28] L. Mainetti, L. Patrono, and I. Sergi. A survey on indoor positioning systems. In *Software, Telecommunications and Computer Networks (SoftCOM), 2014 22nd International Conference on*, pages 111–120. IEEE, 2014.

[29] Z. Ma, Y. Qiao, B. Lee, and E. Fallon. Experimental evaluation of mobile phone sensors. 2013.

Ringraziamenti

Vorrei ringraziare il Prof. Marco Di Felice, relatore di questa tesi, per la professionalità e la costanza con le quali mi ha seguito e aiutato durante la stesura.

Un ringraziamento speciale alla mia famiglia, in particolare a mia madre e mio padre: è grazie al loro sostegno e al loro incoraggiamento se oggi sono riuscito a raggiungere questo traguardo.

Un ringraziamento agli amici di una vita, lontani negli occhi ma vicini nel cuore, e ai miei compagni di corso, che ogni giorno hanno condiviso con me gioie, sacrifici e successi. L'affetto e il sostegno che mi hanno dimostrato rendono questo traguardo ancora più prezioso.

Un ringraziamento ai miei coinquilini, che grazie alla loro amicizia quotidiana mi hanno sempre fatto sentire a casa non facendomi mai mancare l'affetto di cui avevo bisogno.

Un grazie speciale anche a Silvia, certezza e colonna portante di questa avventura, fonte inesauribile di conoscenza, di supporto e di aiuto.