ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea di Informatica Magistrale

UNA LIBRERIA ANDROID PER ALGORITMI DI LOCALIZZAZIONE INDOOR

Relatore:
Chiar.mo Prof.
Luca Bedogni

Presentata da:
Gianluca Monica

Sessione II
Anno Accademico 2017-18

Elenco delle figure

Introduzione

Se proviamo a riflettere sul numero di volte in cui utilizziamo la tecnologia GPS nella nostra quotidianità ci accorgiamo certamente che si tratta di un numero molto elevato. Il GPS è un sistema di localizzazione e navigazione, basato sullo scambio di informazioni tra un ricevitore e dei satelliti che navigano nell'orbita terrestre, che ci fornisce le coordinate geografiche della posizione in cui ci troviamo. La popolarità dell'utilizzo di questo sensore è stata sostanzialmente incrementata dall'avvento degli smartphone, i quali attraverso applicazioni come Google Maps ci permettono un utilizzo semplice ed efficiente. Lo utilizziamo così tanto perchè è un sistema comodissimo di navigazione, che ci guida da un punto A ad un punto B. Quindi il GPS ci aiuta nella navigazione cosi detta outdoor, per capirci, all'aria aperta. Se invece avessimo bisogno di ottenere informazioni di localizzazione o navigazione all'interno di un edificio il GPS non ci tornerebbe molto utile. Purtroppo è negli ambienti indoor che si manifesta un particolare problema di questa tecnica, ovvero il fatto che non è particolarmente preciso quando il ricevitore si trova all'interno di un edificio. Quindi se vogliamo ottenere informazioni di localizzazione o navigazione, utilizzando il nostro smartphone, all'interno di un cosidetto ambiente indoor ci troviamo costretti a non poter utilizzare la tecnologia GPS. La soluzione a questo problema è rappresentata dall'avvento degli algoritmi di localizzazione indoor. Ne esistono di tipi diversi, che si distinguono per architetture e per tecnologie utilizzate. Possiamo dividerli sommariamente in: algoritmi che utilizzano infrastrutture, che sta a significare che si basano su componenti esterni, questo è il caso dell'utilizzo delle tecnologie Wifi o Bluetooth per esempio, che comunicano con Access Point o Beacon; algoritmi che non utilizzano infrastrutture, vale a dire algoritmi che si servono di tecnologie presenti nei nostri smartphone; algoritmi che incrociano le carattestiche delle due tipologie appena presentate, cosidetti algoritmi ibridi. Come esempio di algorimi che utilizzano infrastrutture spiccano gli algoritmi di Finger Printing, tecniche composte di due fasi, che si basano sulla costruzione di una radio map di determinati valori che rappresenta l'edificio in ci si vuole localizzare. Un'altro approcio molto diffuso è il Pedestrian Dead Reckoning, che utilizza i sensori dell'*accelerometro*, del *giroscopio* e del *magnetometro*. Poi ci sono gli algoritmi ibridi che possono essere di svariati tipi, in quanto incrociano in tante

modalità le tecnologie sopra menzionate. Come si può evincere, esistono molti algorimi di localizzazione indoor diversi, ognuno dei quali riporta metriche di performance differenti che faticano ad essere comparabili tra loro. Inoltre, se volessimo testare differenti tecniche nello stesso ambiente, avremmo bisogno di un'applicazione smartphone per ogni algoritmo. Il che, ovviamente, risulta molto scomodo e poco preciso. Da queste necessità nasce il progetto della mia tesi, il cui obiettivo è la creazione di una libreria, per sistemi operativi Android, che permette di implementare i più svariati algoritmi di localizzazione indoor all'interno della stessa infrastruttura. Quindi, potenzialmente, consentire di implementare ed utilizzare svariati algoritmi di localizzazione indoor con la stessa applicazione Android. Rendendo così possibile una più efficace comparabilità tra le performance degli stessi. Inoltre è stata sviluppata un'applicazione Android che integra la libreria e ne dimostra il funzionamento. La tesi si compone di: il primo capitolo che introduce lo stato dell'arte; il secondo che spiega l'architettura utilizzata; il terzo capitolo illustra l'implementazione della libreria e dell'applicazione di sostegno; il quarto presenta i tre algoritmi implementati che dimostrano l'utilizzabilità e l'affidabilità della libreria; infine il quinto capitolo che riporta delle semplici valutazioni degli algoritmi implementati.

Indice

Capitolo 1

Stato dell'arte

Il mio progetto di tesi è stato lo sviluppo di un software, in particolare una libreria per smartphone Android, che fornisce le seguenti funzionalità:

- la possibilità da parte di un utente di localizzarsi indipendetemente dal contesto (indoor/outdoor) e dalle sottostanti tecnologie utilizzate;
- l'opportunità di implementare molteplici algoritmi di localizzazione indoor svincolandosi dalle tecnologie utilizzabili (come ad esempio il Wifi o l'accelerometro) e dall'architettura dell'algoritmo stesso.

1.1 Il progetto Europeo i-locate

Il progetto di riferimento in questo ambito è i-locate, un progetto Europeo che mira alla sviluppo di un software estendibile che permette la creazione di Location Based Services disinteressandosi delle tecnologie utilizzate e del contesto (indoor/outdoor).

Questo progetto è nato dalle riflessioni di studi che hanno evidenziato che in media si spende il 90% del proprio tempo in ambienti indoor, il più delle volte non familiari. Per questo motivo la facoltà di localizzare persone o oggetti potrebbe incentivare la diffusione di Location-Based Service (LBS), servizi che fanno uso della posizione di un utente.

Questi servizi hanno bisogno di accedere ad informazioni geografiche sia indoor che outdoor. Mentre i dati outdoor sono facilmente reperibili mediante Open Data (OD), in ambienti indoor queste informazioni non sono disponibili. Quindi la possibilità di fornire dati geografici come Open Data anche in ambienti chiusi porterebbe al raggiungimento di benefici sociali ed anche allo sviluppo di numerose attività di business.

1.1.1 Principali obiettivi

Virtual hub Il virtual hub sarà un geo-portale pubblico che permetterà la condivisione di dati riguardanti mappe di spazi indoor (come Open Data). Questi spazi riguarderanno:

- edifici pubblici;
- edifici privati accessibili liberamente al pubblico (es. aeroporti);
- proprietà private (se necessario).

Questo hub è visto come un avanzamento dell'attuale progetto Open Street Map. Open Street Map è un progetto collaborativo finalizzato a creare mappe a contenuto libero del mondo. Il progetto punta ad una raccolta mondiale di dati geografici, con scopo principale la creazione di mappe e cartografie.

Il toolkit open source Permetterà una localizzazione indoor/outdoor interoperabile. Il toolkit stesso sarà rilasciato al pubblico come open source, per facilitare il nascere di nuovi business basati sugli indoor LBS.

In particolare sarà un sistema in grado di riunire diverse tecnologie per fornire una localizzazione indoor/outdoor per persone e asset location con:

- routing multimodale indoor/outdoor;
- asset managment (per il mantenimento delle attività).

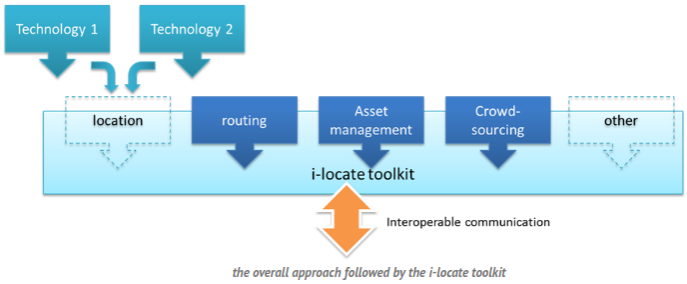


Figura 1.1 Il toolkit

Come mostrato in figura il toolkit sarà sviluppato attraverso un middleware basato sul linguaggio Java. Esso permetterà:

- un'interfaccia di comunicazione aperta volta ad integrare tecnologie differenti;
- scalabilità.

1.1.2 Il middleware

Il punto centrale dell'architettura di i-locate è il middleware. Questo livello fornisce un insieme di servizi al fine di abilitare i LBS. Quindi il focus è l'accessibilità a servizi di localizzazione sia indoor che outdoor.

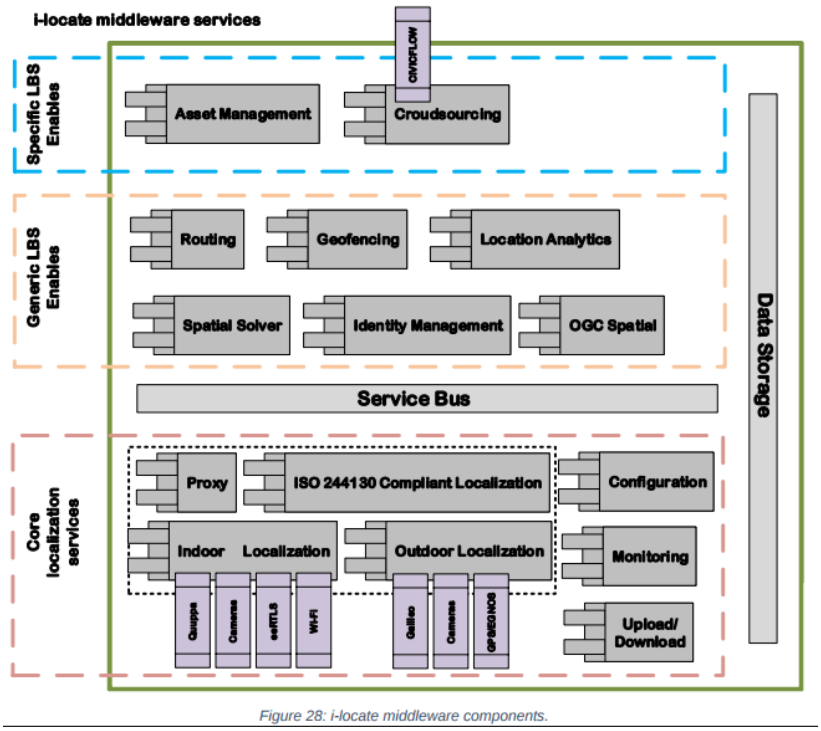


Figura 1.2 Il middleware di i-locate

Primo livello

Fornisce servizi di localizzazione per ambienti:

- **outdoor**, questo componente fornisce informazioni utilizzando un modulo GNSS, oppure GPS se il primo non è disponibile.
- **indoor**, questo componente utilizzerà diverse tecnologie, tra le quali Wifi, 802.15.4 eeRTLS, tecnologia HAIP basata su Bluetooth e un sistema di tracking attraverso fotocamera sviluppato da Trilogis.

Inoltre sono presenti i seguenti moduli:

- **monitoring**, permette il controllo dello stato del sistema (es. batteria rimanente);

- **configuration**, permette la configurazione dei sottosistemi di localizzazione;
- **location resolver**, fornisce un alto livello di astrazione delle tecnologie utilizzate nella piattaforma.

Secondo livello

Fornisce le seguenti funzionalità:

- **routing**, permette di guidare l'utente finale all'interno di ambienti indoor o outdoor;
- **identity manager**, fornisce servizi relativi alla individuazione di persone e oggetti;
- **geofencing**, essenziale per essere avvisati quando un utente entra o lascia un determinato ambiente;
- **location analysis**, utilizzato per compiere analisi e statistiche.

Terzo livello

In questo livello troviamo:

- **asset managment**, fornisce servizi per localizzazione di dispositivi in ambiente indoor;
- **crowdsourcing**, fornisce la connessione a informazioni geografiche crowd-sourcing.

Il mio progetto di tesi prende spunto dai concetti qui sopra enunciati. Nel capitolo successivo farò una breve panoramica dello stato dell'arte attuale per gli algoritmi di localizzazione indoor.

1.2 Tecniche di localizzazione indoor

Il nascere di questi metodi è stato scaturito dagli scarsi risultati che il GPS fornisce in ambienti chiusi. Infatti spesso in queste condizioni non è disponibile o presenta errori non trascurabili.

L'obiettivo è trovare le coordinate di una persona o di un oggetto all'interno di un edificio. A seconda della tipologia di tecnica adottata la posizione può essere classifica principalmente in quattro differenti modi [14]:

- **Posizione fisica** (Physics Location), rappresentata da un punto su una mappa 2D oppure 3D;

- **Posizione simbolica** (Symbolic Location), rappresentata da una certa posizione attraverso il linguaggio comune (es. "Ufficio");
- **Posizione assoluta** (Absolute Location), la posizione è rappresentata attraverso una griglia di riferimento;
- **Posizione relativa** (Relative Location), invece identifica una posizione sulla base della posizione di altri oggetti conosciuti (es. Access Point).

Panoramica degli algoritmi Possiamo distinguere i metodi di localizzazione indoor in base alle tecnologie sulle quali fanno affidamento. Quindi, medodi che utilizzano:

1. la sensoristica presente nei moderni smarphone, detti anche metodi senza infrastruttura. Spiegherò i sistemi di Pedestrian Dead Reckoning, in particolare gli Inertial Navigation System (INS) e Step-and-Heading Systems (SHSs).
2. la tecnologia delle radiofrequenze (es. Wifi, Bluetooth), quindi metodi con infrastruttura. Mi soffermerò sui metodi di triangolazione del segnale ed algoritmi che utilizzano la tecnica Finger Printing.
3. un incrocio tra le sopra citate tecnologie, per questa categoria parlerò in particolare di un metodo che utilizza il barometro combinato con una tecnica di RSS Finger Printing probabilistica.

1.3 Sistemi di localizzazione indoor tramite sensori dello smartphone

In questa categoria troviamo i cosidetti sistemi di Pedestrian Dead Reckoning [11], che utilizzano i seguenti sensori: l' *accelerometro*, utile per misurare l'accelerazione del dispositivo; il *giroscopio*, per captare i movimenti a cui lo smartphone è soggetto; ed il *magnetometro*, che misura l'intensità e la direzione del campo magnetico nel quale ci troviamo. Il Pedestrian Dead Reckoning è un processo in cui la posizione corrente viene ricavata a partire da una posizione precedente, stimando la lunghezza del passo e la direzione della camminata di un utente. Il problema più imponente di queste tecniche è che ad ogni iterazione gli errori si sommano e crescono molto velocemente. Quindi si ha la necessità di un riferimento assoluto per la correzione dell'errore.

Esistono due tipi di sistemi di Pedestrian Dead Reckoning:

1. Inertial Navigation Systems (INS);

2. Step-and-Heading Systems (SHSs).

1.3.1 Riconoscimento del Gait Cycle

Entrambi gli algoritmi sopra citati si basano sul concetto di Gait Cycle, cioè l'alternanza tra la fase di *stance* e la fase di *swing* del piede di un essere umano durante una camminata [11]. La prima è il periodo nel quale il piede è attaccato a terra, mentre l'ultima è la fase in cui il piede sta oscillando e non tocca il terreno. L'obiettivo quindi è stato trovare un metodo affidabile che possa riconoscere il Gait Cycle, sono state sviluppate diverse soluzioni in questo contesto.

Riconoscimento della fase di stance In queste tecniche è necessario che il sensore sia indossato nella scarpe dell'utente. Tipicamente sono basate sulla presenza di una soglia, il sensore riporta valori statici nella fase di stance e conseguentemente i sensori inerziali dovrebbero restituire un mancanza di attività identificabile attraverso una soglia. La soglia può essere utilizzata su numerose metriche, in particolare sui valori dell'accelerometro, della velocità angolare, del magnetometro o su combinazioni di questi. Un'efficiente riconoscimento della fase di stance rende possibile il conteggio dei passi dell'utente. Jimenez ha riportato errori dello 0.1% utilizzando una tecnica basata sulla varianza dell'accelerometro e sulla velocità angolare [12]. Tuttavia, queste tecniche riportano uno svantaggio in particolare, ovvero la presenza obbligatoria di un sensore sulla scarpa dell'utente. Ci sono anche limitazioni nel riconoscimento dello stance in sè, perchè i sensori sulla scarpa subiscono movimenti aggiuntivi associati alla forma della scarpa stessa. In più, la natura empirica delle soglie suggerisce che i punti di inizio e di fine di una fase di stance riconosciuta non rappresentano fedelmente la fase reale del piede umano [17]. Piuttosto rappresentano un sottoinsieme inconsistente del periodo totale di stance.

Riconoscimento di cicli nei sensori Questi algoritmi riconoscono cicli nei valori dei sensori causati dal movimento ripetitivo della camminata. Cercano l'avvenire ripetuto di pattern o determinati eventi e non si basano su sensori da apporre su specifiche parti del corpo. Questi algoritmi includono:

- **Peak Detection**, il movimento del piede viene associato a cambiamenti dell'accelerazione verticale. Queste tipologie sono utilizzate per ricercare potenziali passi. Da notare che ogni impatto scatenato da un piede potrebbe generare molteplici picchi locali, questo può incrementare notevolmente la complessità dell'algoritmo [7].

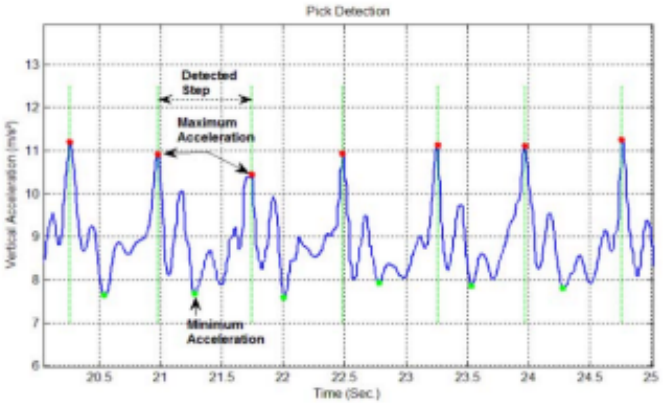


Figura 1.3 Peak detection

- **Zero Crossing**, monitora il valore dell'accelerazione utilizzando essenzialmente una specie di soglia [9].
- **Auto-correlation o template matching**, la natura ciclica della camminata porta alla creazione di periodicità nei valori raccolti dai sensori. Il ciclo può essere estratto cercando il massimo nell'autocorrelazione di una sequenza di dati (es. accelerazione).

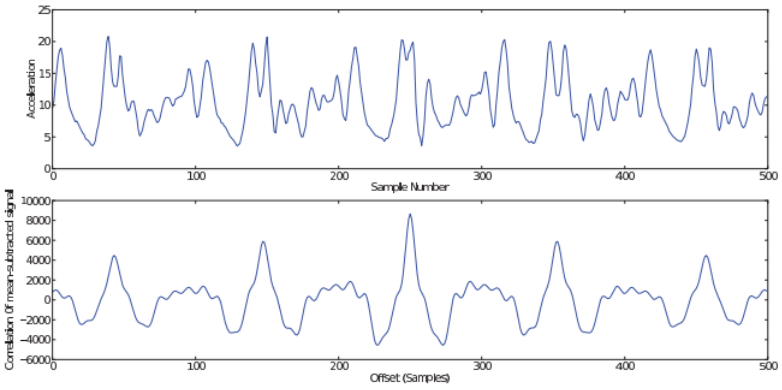


Figura 1.4 Riconoscimento dei cicli tramite autocorrelazione

- **Spectral analysis**, questa tecnica identifica picchi molto forti nelle tipiche frequenze di passi. Viene poi estratto un sottoinsieme di dati (contenente almeno due cicli) dal quale viene calcolata la frequenza dominante che viene assunta come walking frequency, ovvero frequenza della camminata [13].

1.3.2 Inertial Navigation Systems (INS)

Questo metodo fornisce una localizzazione continua basata su tre assi, ovvero posizione, velocità e orientamento. Sfrutta i dati forniti dall'accelerometro e dal giroscopio. 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. Alcuni studi integrano anche l'utilizzo del magnetometro ottenendo risultati migliori. La presenza di "rumore" durante la fase di misuramento può essere migliorata utilizzando delle tecniche di filtering (filtro di Kalman) [11].

1.3.3 Step-and-Heading Systems (SHSs)

Utilizza dei vettori di spostamento per una navigazione in 2D del piano parallelo al terreno. Si basa su tre concetti fondamentali:

- *posizione iniziale*, prerequisito fondamentale per tutte le tecniche di Pedestrian Dead Reckoning;
- *lunghezza del passo*, può essere ricavata applicando SHS in modo ricorsivo partendo dalla posizione iniziale;
- *direzione del passo*, è il punto più delicato di questi metodi poichè ogni persona cammina in modo differente e a velocità diverse. A questo proposito esistono diverse tecniche per risolvere tale problema.

Questi sistemi forniscono una serie di vettori polari composti da $\{step\_length, step\_heading\}$ o più spesso $\{step\_length, step\_heading\_change\}$. Questi vettori possono essere utilizzati in una spazio vettoriale 2D per tenere traccia della posizione. Possono essere stimati da un output di un algoritmo INS, ma possono essere stimati anche senza in modo da evitare un veloce accumularsi di errori (drift).

Calcolo della lunghezza del passo Esistono differenti approcci per risolvere questo problema:

1. assumere la lunghezza come una costante;
2. assumere la lunghezza in base alle frequenze dei passi osservate, utilizzando una delle tecniche precedentemente citate nella sezione 1.3.1;

3. misurazioni dirette di ogni passo. Per esempio, Saarinen utilizza sensori ultrasonici montati nel fronte e nel retro di ogni scarpa [16].

4. alcuni sistemi, per scegliere una lunghezza ottimale, stimano la stessa iterativamente valutando i percorsi prodotti utilizzando lunghezze differenti e planimetrie di edifici;

5. la soluzione per calcolare la lunghezza del passo più accurata è quella proposta da H. Xing e altri [10], X. Wang e altri [4]. 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.

6. un altro metodo efficace è quello di utilizzare una rete neurale BP-ANN (Back Propagation Artificial Neural Network) [10] che tuttavia necessita di una infrastruttura per poter essere messo in atto.

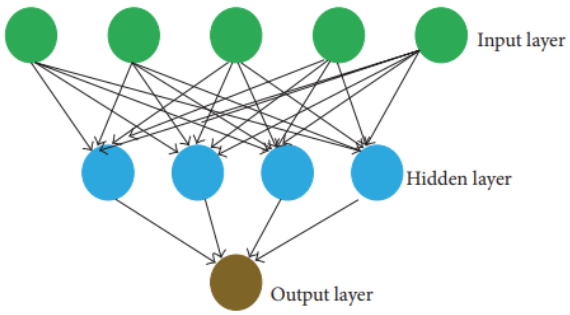


Figura 1.5 Architettura di una Back Propagation Artificial Neural Network

Stima della direzione E' molto simile al caso degli algoritmi INS, per stimare la direzione vengono utilizzati i segnali forniti dal giroscopio. Alcuni sistemi utilizzano il giroscopio posizionato parallelamente al torso umano, assumendo che rimanga stabile durante la camminata. Infine viene utilizzato anche il magnetometro direttamente o insieme al giroscopio.

Stima della posizione di un utente Uno dei metodi che è possibile utilizzare è quello di inserire i dati ottenuti (posizione iniziale, lunghezza del passo e direzione del passo) in

un Filtro di Kalman. In particolare si inseriscono i dati in un sistema di equazioni come il seguente:

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

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 [6].
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].

Riduzione del rumore dei sensori Per ridurre il rumore dei sensori all'interno di un sistema SHS si utilizza la tecnica del **Particle Filters** che è un'approssimazione numerica del filtro di Bayes [16]. Questa tecnica si compone di particelle, ognuna delle quali rappresenta una possibile posizione in 2D. Ad ogni posizione viene assegnato un peso direttamente proporzionale alla probabilità che un utente si trovi in quella posizione. Questa tecnica è iterativa ed è composta da tre passi per ogni iterazione:

- **Update**, ogni particella è posizionata nello scenario 2D basandosi sul modello costruito con i movimenti precedenti (lunghezza del passo precedente, direzione del passo precedente);
- **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);
- **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) 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 $n\Theta$ è il rumore presente nel modello di stima della direzione del passo [2].

1.4 Sistemi di localizzazione basati su tecnologie di radiofrequenze

Questi sistemi si appoggiano a tecnologie molto note come il Wifi, il Bluetooth e la rete dati cellulare. Generalmente un sistema di localizzazione di questo tipo presenta uno o più dispositivi fissi ed uno mobile (smartphone). Nascono per cui le seguenti topologie di localizzazione [5]:

1. **Remote Positioning System**, il nodo mobile (emittente) trasmette il proprio segnale ai nodi fissi, che in seguito lo inoltreranno ad un nodo master il quale provvederà a calcolare la posizione del nodo mobile;

2. **Self Positioning System**, in questo caso i nodi fissi inviano i propri segnali al nodo mobile che in seguito provvederà a calcolare la propria posizione;

3. **Indirect Remote Positioning**, molto simile al Self Positioning System, la principale differenza è che il nodo mobile dopo aver ricevuto i segnali li inoltrerà ad un nodo master che provvederà a calcolare la posizione;

4. **Indirect Self Positioning**, simile al Remote Positioning System, in questo caso le misurazioni effettuate dai nodi fissi sono inviate al nodo mobile che poi calcolerà la propria posizione.

1.4.1 Algoritmi di triangolazione del segnale

Gli algoritmi di triangolazione sfruttano le proprietà del triangolo per stimare la posizione di un oggetto. Hanno due derivazioni: algoritmi di laterazione e angolazione.

Laterazione Questa tecnica stima la posizione di un dispositivo misurando la distanza tra diversi punti di riferimento. Le unità di misura utilizzate possono essere diverse:

- **Time Of Arrival** (TOA) [14], si basa sull'assunzione che la distanza di un dispositivo da un punto di riferimento è direttamente proporzionale al tempo di propagazione del segnale. Il tempo di propagazione viene calcolato assieme alla distanza tra il nodo mobile ed il trasmettitore del segnale. Generalmente si presentano due tipi di problemi: la necessità che trasmettitore e ricevitore siano precisamente sincronizzati e che il timestamp debba essere incluso nel segnale trasmesso al fine di individuare la distanza che il segnale ha percorso.

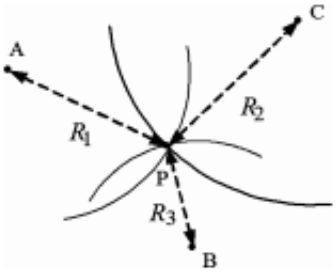


Figura 1.6 Le misurazione TOA devono essere calcolate da almeno tre Reference Point

Data la posizione sconosciuta del nodo 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 dispositivo mobile 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:

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

- **Time Difference Of Arrival** (TDOA) [14], in questo caso l'idea è quella di determinare la posizione relativa del nodo mobile esaminando la differenza dei tempi di arrivo attraverso misurazioni multiple nel tempo. Una locazione 2D può essere stimata dall'intersezione di due o più misurazioni TDOA.

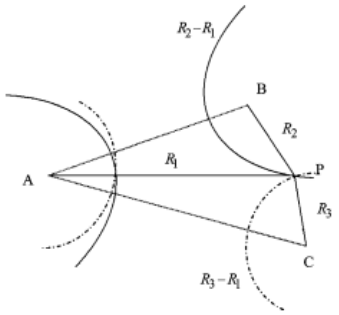


Figura 1.7 Localizzazione basata su TDOA

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]:

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

- **Received Signal Strength** (RSS) [14], stima la distanza del nodo mobile dall'Access Point utilizzando l'attenuazione della forza del segnale emesso da quest'ultimo.

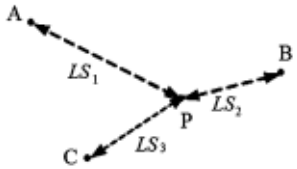


Figura 1.8 Localizzazione basata su RSS

Data la potenza del segnale emesso $p(R_0)$, la potenza del segnale ricevuto $p(R)$ viene calcolata secondo la formula:

$$p(R) = p(R_0) - 10nlog(\frac{R}{R_0}) \quad (1.10)$$

- **Round Trip Time Of Flight** (RTOF), calcola il tempo che un segnale impiega per andare dall'emettitore al ricevitore e tornare indietro. La distanza è calcolabile con l'equazione:

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

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

- **Angolation Of Arrival** (AOA) [14], queste tecniche 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. Come mostrato in figura, i metodi AOA devono usare almeno due Reference Point A, B, e due angoli θ_1, θ_2, per ottenere la posizione 2D del punto P.

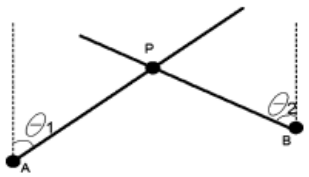


Figura 1.9 Localizzazione basata su AOA

I vantaggi di questo metodo sono:

- ha bisogno solamente di due unità per calcolare una posizione 2D;
- non ha bisogno che i dispositivi siano sincronizzati tra loro.

Mentre gli svantaggi sono:

- lavora molto bene in situazioni di LoS (Line of Sight) ma la precisione diminuisce quando ci sono riflessioni di segnali (multipath). Quindi non va bene per ambienti indoor.
- hardware complesso;
- la precisione diminuisce anche quando il dispositivo target si muove aldilà delle unità.

1.4.2 Algoritmi di posizionamento tramite la tecnica Finger Printing

Questi algoritmi sono i più comuni nel campo della localizzazione indoor (la mia libreria per dimostrazione implementa algoritmi di questo tipo). Si compongono di due fasi [8]:

1. **fase offline**, durante questa fase viene costruita una Radio-Frequency Map dell'edificio in cui ci si vuole localizzare. Per ogni Reference Point vengono campionate le radiofrequenze emesse dai trasmettitori (Access Point). I Reference Point sono semplici postazioni decise a priori. Al termine di questa fase si avrà a disposizione una collezione di radiofrequenze ricevute da ogni Reference Point.

2. **fase online**, in questa fase l'utente non fa altro che registrare una radiofrequenza e confrontarla con quelle salvate precedentemente. Da questi confronti nascerà la posizione stimata dell'utente.

   Per giungere a questa posizione si possono utilizzare diverse metodi. Ecco alcuni esempi di metodi esistenti per il calcolo della posizione del nodo mobile.

Metodi probabilistici Il posizionamento è visto come un problema di classificazione [14]. Supponendo di avere n locazioni candidate $L_1, ..., L_n$, e s, la potenza del segnale rilevata durante la fase online, si calcola la probabilità che il nodo mobile si trovi nella posizione L_i avendo ricevuto la potenza di segnale s.

Viene utilizzata la seguente regola di decisione:

Scegli L_i se

$$P(L_i|s) > P(L_j|s) \tag{1.12}$$

per $i, j = 1, .., n$ con $j \neq i$

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:

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

K-Nearest-Neighbor (KNN) Questo metodo utilizza gli RSS percepiti nella fase online per estrarre K posizioni da quelle raccolte durante la fase offline [14]. Queste K sono le

posizioni considerate più vicine alla rilevazione su cui ci si affida. Successivamente si applica il calcolo della distanza Euclidea per stimare la posizione dell'utente.
A livello matematico, il vettore $\bar{y}$ è confrontato con la radio map dei fingerprint offline. Sia

$$L_K^2 = \{p_1, ..., p_k\}$$

la lista delle coordinate dei calibration point corrispondenti alla lista dei K fingerprint

$$\bar{a}_{1:K} = \{\bar{a}_1, ..., \bar{a}_K\}$$

che soddisfa

$$d(\bar{y} - \bar{a}_i) \leq d(\bar{y}, \bar{a}_j)$$

dove $\bar{a}_i \in \bar{a}_{1:K}$, $\bar{a}_i \notin \bar{a}_{1:K}$ e la funzione $d(\cdot)$ è una misura di distanza scelta.
La seguente formula calcola la posizione stimata.

$$\hat{x} = \frac{1}{K} \sum_{i=1}^{K} p_i \tag{1.14}$$

dove $p_i \in L_{1:K}$.

Reti neurali Le posizioni ottenute durante la fase offline vengono fornite come input per il training di una rete neurale (di solito una rete Multilayer Perceptron con un livello nascosto). Nella fase offline vengono inserite le potenze dei segnali per ogni AP e le corrispondenti coordinate. Nella fase online vengono raccolti gli RSS dal dispositivo mobile e vengono moltiplicati per i pesi ottenuti dalla rete neurale. I risultati di questo procedimento sono passati alla funzione di trasferimento del livello nascosto. Il conseguente output viene moltiplicato per la matrice dei pesi ottenuta dall'allenamento del livello nascosto della rete neurale. Il risultato è un vettore di due elementi che rappresenta la posizione stimata [2] [18].

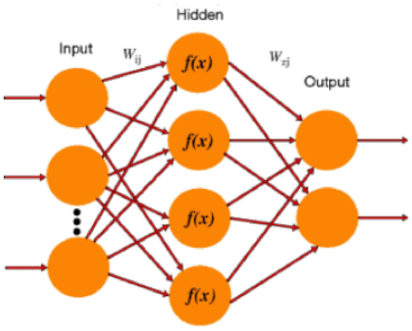


Figura 1.10 Rete neurale Multilayer Perceptron

Smalles M-vertex Polygon (SMP) In questa tecnica, simile a KNN, vengono raccolti gli RSS provenienti dagli AP e vengono scelti i Reference Point considerati più probabili, tenendo conto della potenza del segnale ricevuto per ogni nodo fisso. Dopo di che viene creato un poligono con M vertici scegliendo almeno una base station per ogni RP. La posizione del nodo mobile è data dalla media dei segnali ricevuti prendendo i vertici del poligono con l'area minore. [15].

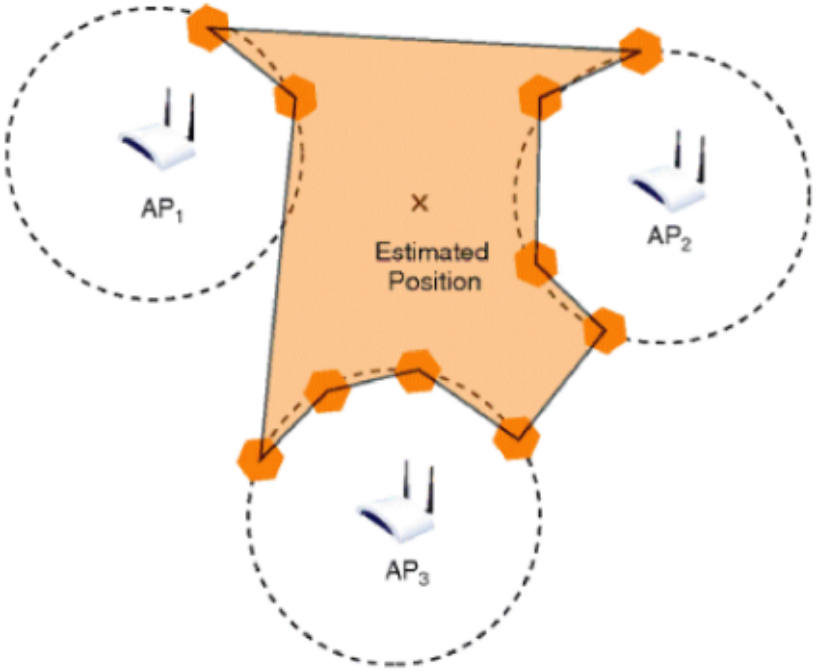


Figura 1.11 I punti evidenziati in arancione sono coloro che captano la stessa potenza di segnale.

1.4.3 Algoritmi di prossimità

Questi algoritmi utilizzano delle antenne a corto raggio (BLE, RFID) nell'ambiente di sperimentazione. Ogni volta che un dispositivo si aggancia ad una delle antenne, si considera che il nodo si trovi molto vicino a quella posizione.

La principale tecnologia utilizzata è Bluetooth Low Energy. Questa tecnica, siccome il raggio del segnale è molto corto, necessita una presenza massicia di antenne. Gli algoritmi di prossimità forniscono una posizione simbolica di un nodo mobile.

1.4.4 Metriche di perfomance

Sono state definite in letteratura specifiche metriche di performance per questa tipologia di algoritmi [14]. In particolare:

- 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.

1.5 Algoritmi di posizionamento ibridi

Infine vi sono gli algoritmi di posizionamento ibridi. Questo tipo di algoritmi nasce dalla necessità di sopperire alle mancanze di accuratezza e precisione degli algoritmi a sensori inerziali, nonchè all'elevato costo degli algoritmi di Finger Printing. Tali sistemi non sono altro che un mix tra gli algoritmi sopra citati.

1.5.1 Tecnica basata su barometro e Wifi

Come esempio di algoritmo ibrido parlerò di questo articolo [1], il quale presenta una tecnica composta di due fasi. La prima fase si occupa di stimare il piano in cui si trova il dispositivo mobile utilizzando le informazioni ottenute dal barometro. Successivamente procede all'attuazione di un algoritmo di Finger Printing probabilistico per stimare la posizione dell'utente.

Stima del piano dell'edificio

Per stimare il piano dell'edificio questa tecnica necessita ovviamente dell'informazione dell'altitudine al quale il dispositivo mobile si trova. Per calcolare correttamente l'altitudine dal piano terreno si ha bisogno del valore della pressione di riferimento P_b e dell'altitudine del piano terreno dal livello del mare. Quest'ultima è accessibile tramite le più recenti coordinate GPS ottenute appena la persona entra nell'edificio. Mentre la prima è più difficile da recuperare, in quanto può cambiare rapidamente a causa di eventi atmosferici. Per risolvere questo problema è stato utilizzato un algoritmo dinamico che calcola la variazione della pressione. Questo algoritmo è capace di riconoscere se la variazione è avvenuta a causa di un cambio dell'altitudine del dispositivo o se è dovuto a fattori esterni. L'idea generale è che quando si nota un cambio improvviso del valore della pressione è perchè l'utente si sta muovendo. Mentre quando si verificano cambiamenti più lenti significa che sono dovuti a cambiamenti naturali e quindi bisogna usarli per correggere il valore P_b. L'algoritmo è composto di due fasi:

1. una fase di riconoscimento, la quale calcola la media μ_i delle ultime N misurazioni del valore della pressione:
$$\mu_i = \frac{\sum_{i=1}^{N} \Lambda_i}{N} \tag{1.15}$$
e la varianza:
$$\sigma_N^2 = \frac{\sum_{i=1}^{N} \Lambda_i^2 - \mu_i^2}{N-1} \tag{1.16}$$
dove Λ_i è l'i-esimo valore della pressione e N è il numero totale di misurazioni. Semplicemente μ_i calcola la media delle ultime N misurazioni e σ_N^2 è la varianza nella stessa finestra temporale. L'idea dietro queste misurazioni è che anche se la pressione cambia durante il giorno, la sua varianza è significamente piccola in un periodo corto. Quindi, se σ_N^2 supera una certa soglia Θ, significa che c'è stato un cambio di altitudine, altrimenti i possibili cambiamente nel livello della pressione sono da attribuire ad altri fenomeni. Formalmente:
$$H = \begin{cases} H_1, & \text{se } \sigma_N^2 > \Theta \\ H_2, & \text{se altrimenti} \end{cases} \tag{1.17}$$

2. ed una di correzione, la quale calcola il piano e corregge il valore P_b con la misurazione aggiornata.

Quindi se $H = H_2$, aggiorniamo P_b con l'ultimo valore disponibile Λ_i. Nel caso in cui $H = H_1$, significa che l'utente si è mosso, quindi viene ricalcolata l'altitudine con la seguente

equazione:

$$P = P_b\Big[\frac{T_b}{T_b - L_b \cdot \omega}\Big]^{\frac{g_0 \cdot M}{R \cdot L_b}} \tag{1.18}$$

dove le variabili menzionate assumono i valori presenti in figura.

Quantity	Description	Value
P_b	Static pressure at the sea level	101325 Pa
T_b	Temperature	288.15 K
L_b	Temperature lapse rate	-0.0065 K/m
ω	Height at which the pressure has to be computed	
g_0	Earth gravity acceleration	9.8 $\frac{m}{s^2}$
M	Molar mass of the Earth's air	0.0289 kg/mol
R	Air gas constant	8.314 $\frac{N \cdot m}{mol \cdot K}$ (see [20])

Figura 1.12 Costanti per il calcolo dell'altitudine

Infine il piano viene calcolato come:

$$f = \frac{\omega_{base} - \omega_{now}}{H} \tag{1.19}$$

dove ω_{base} è l'altezza del piano terra dal livello del mare mentre ω_{now} è l'altitudine attuale calcolata con 1.18. H è l'altezza del piano (nei test effettuati è pari a 3.5 metri).

Algoritmo di Finger Printing probabilistico

Siano L e M rispettivamente il numero di Reference Point e Access Point. Si denota l'l-esimo Reference Point con RP_l e l'm-esimo Access Point con AP_m con $l \in [1,L]$ e $m \in [1,M]$. Ogni RP_l è identificato dalle coordinate $C_l = (x_l, y_l)$. Sia T il numero dei valori RSS, ognuno ottenuto every Δt secondi. Durante la fase offline, è stata costruita una matrice tridimensionale di osservazione $\bar{O}$ composta da L righe e M colonne di T elementi. L'elemento $\bar{o}_{m,l,t}$, con $l \in [1,L]$, $m \in [1,M]$ e $t \in [1,T]$, rappresenta il singolo valore RSS ricevuto dal m-esimo AP, nel l-esimo RP durante la misurazione t-esima. $\mu_{m,l}$ e $\sigma^2_{m,l}$ sono la media e la varianza di tutte le T osservazioni dal m-esimo AP nel l-esimo RP e rappresentano la radio Finger Print di ogni RP. Durante la fase di training questa procedura è iterata per tutti gli RP considerati di un piano specifico per ottenere una radio map. Estendendo l'idea ad una ambiente multipiano,

otteniamo la definizione di FP di un singolo piano f.

$$FP_f = \begin{pmatrix} <\mu_{1,1},\sigma^2_{1,1}> & <\mu_{1,2},\sigma^2_{1,2}> & ... & <\mu_{1,M},\sigma^2_{1,M}> \\ <\mu_{2,1},\sigma^2_{2,1}> & <\mu_{2,2},\sigma^2_{2,2}> & ... & <\mu_{2,M},\sigma^2_{2,M}> \\ \vdots & \vdots & \ddots & \vdots \\ <\mu_{L,1},\sigma^2_{L,1}> & <\mu_{L,2},\sigma^2_{L,2}> & ... & <\mu_{L,M},\sigma^2_{L,M}> \end{pmatrix}$$

dove il generico elemento $FP_l(m,l) =< \mu_{m,l}, \sigma^2_{m,l} >$ è la coppia media-varianza associata al m-esimo AP nel l-esimo RP al piano f, $m \in [1,M]$,$f \in [1,F]$ e $l \in [1,L]$.
Nella fase online invece lo smartphone acquisisce i valori RSS e calcola il vettore di osservazione $o = [o_1,...,o_M]$. Dopo di che si calcola $p(o|l)$ in questo modo:

$$p(o|l) = \prod_{m=1}^{M} \frac{1}{\sqrt{2\pi\sigma^2_{m,l}}} e^{\frac{(o-\mu_{m,l})^2}{2\sigma^2_{m,l}}} \tag{1.20}$$

$\forall l \in [1,L]$.

Ogni valore $p(o|l)$ rappresenta la probabilità del vettore σ di essere "sentito" nel l-esimo RP.
Denotando con L l'insieme degli indici degli RP con cardinalità $|L| = L$ e $l_{max} : \text{argmax}_{l \in L}\{p(o|l)\}$ le coordinate stimate della posizione del dispositivo mobile $\bar{C} = (\bar{x},\bar{y})$ sono calcolate usando il metodo K Weighted Nearest Neighbors (KW-NN) con $K \geq 2$. Questo metodo calcola una media pesata delle coordinate dei K RP con i valori $p(o|l)$ più alti, come da formula:

$$\bar{x} = \frac{\sum_{i=1}^{K} p(o|l)x_i}{\sum_{i=1}^{K} p(o|l)} \; e \; \bar{y} = \frac{\sum_{i=1}^{K} p(o|l)y_i}{\sum_{i=1}^{K} p(o|l)} \tag{1.21}$$

dove

$$(x_i,y_i) \; t.c. \; i : \underset{\forall l \in L^K}{\text{argmax}}\{p(o|l)\} \tag{1.22}$$

e l'insieme $L^K \subset L$ è definito come:

$$L^K = \{L^{K-1} \setminus \{l^{k-1}_{max}\} \; e \; L^1 = L, \; l^1_{max} = l_{max}, \forall k \in [1,K]\} \tag{1.23}$$

Risultati

L'algoritmo evidenzia un'ottima precisione nell'individuazione del piano mediante l'altitudine. Inoltre consente di ottenere un errore di posizionamento minore nonostante siano presenti pochi Access Point. In particolare utilizzando il barometro con 4 Access Point l'errore

di posizionamento è di circa 1.5 metri, lo stesso errore è raggiungibile senza il barometro solamente in presenza di 7 Access Point. Infine nel caso di 5 Access Point l'errore si è mostrato minore di 1.2 metri.

Capitolo 2

Architettura

Come si è fatto notare nel capitolo precedente la varietà di algoritmi per la localizzazione indoor, sia per architettura che per tecnologie utilizzate, è potenzialmente molto ampia. Questa crescita è sicuramente incentivata dalla presenza degli algoritmi cosidetti ibridi che danno luce a numerose alternative. Questi sono i motivi principali per cui si è resa evidente la necessità di avere una infrastruttura che permettesse di implementare qualsiasi tipologia di algoritmo, fornendo notevoli facilitazioni in termini di sviluppo ed utilizzo.

L'infrastruttura da me sviluppata presenta la seguenti funzionalità:

1. possibilità da parte di una generica applicazione di ottenere la posizione del dispositivo, che sia indoor o outdoor;
2. possibilità di implementare qualsiasi tipologia di algoritmo indoor.

A livello astratto la libreria può essere considerata come una *black box* che contiene al proprio interno un numero *n* di algoritmi indoor e che risponde a richieste di localizzazione da parte di un'applicazione.

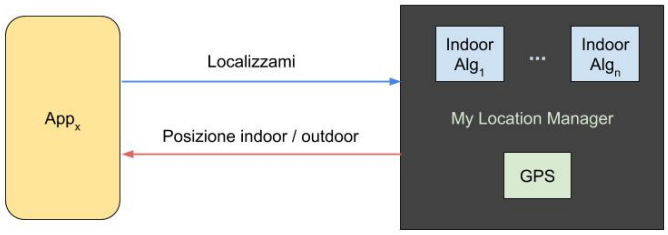


Figura 2.1 Schema astratto dell'architettura

L'architettura della libreria è **stand-alone**, non ha bisogno di alcun software di supporto, eccetto quello di un'applicazione che la integri e la utilizzi. La libreria che ho sviluppato prende il nome di *My Location Manager*.

2.1 Caratteristiche e funzionalità

2.1.1 Localizazzione

La peculiarità del *My Location Manager* è la capacità di riconoscere se il dispositivo si trova in ambiente esterno o interno e di conseguenza restituire la posizione correlata.

Localizzazione Outdoor Nel caso di localizzazione outdoor viene sfruttata la tecnologia GPS. GPS è l'acronimo di Global Positioning System, si tratta di un sistema per il posizionamento globale. Grazie al GPS è possibile localizzare la longitudine e la latitudine di oggetti e persone. Il tutto avviene con i satelliti che stazionano nell'orbita terrestre e permettono di sapere in ogni istante l'esatta ubicazione di un luogo. I satelliti contengono un orologio atomico che calcola al millesimo di secondo il tempo che passa dalla richiesta effettuata dal ricevitore GPS alle risposte ottenute dai satelliti stessi. Il sistema GPS si basa principalmente sugli orologi atomici presenti all'interno dei satelliti solari. Come funziona? Intorno all'orbita terrestre stazionano circa 31 satelliti: in ogni istante inviano la propria posizione alle torri di controllo. Quando un ricevitore GPS viene attivato, riceve le informazioni della posizione dei vari satelliti. Triangolando i dati ricevuti riesce a determinare la propria posizione. La geolocalizzazione è sempre molto precisa in ambienti esterni. Mentre per gli ambienti interni l'errore è molto più alto.

Localizzazione Indoor La libreria capisce di essere in questo caso quando l'accuratezza del segnale GPS supera un certa soglia di errore. Nel capitolo successivo spiegherò nel dettaglio l'algoritmo che discrimina i due casi. Nel caso indoor si utilizza un algoritmo *x* preventivamente scelto. La posizione ritornata assume diverse forme a seconda dell'algoritmo utilizzato.

2.1.2 Estendibilità

Il secondo punto di forza del *My Location Manager* è l'estendibilità. La libreria permette di implementare algoritmi che utilizzano qualsiasi combinazione di tecnologie e architetture. L'architettura dell'infrastruttura è **stand-alone**, ma questo non vieta la possibilità implementare algoritmi con architetture **client-server** o di altro tipo.

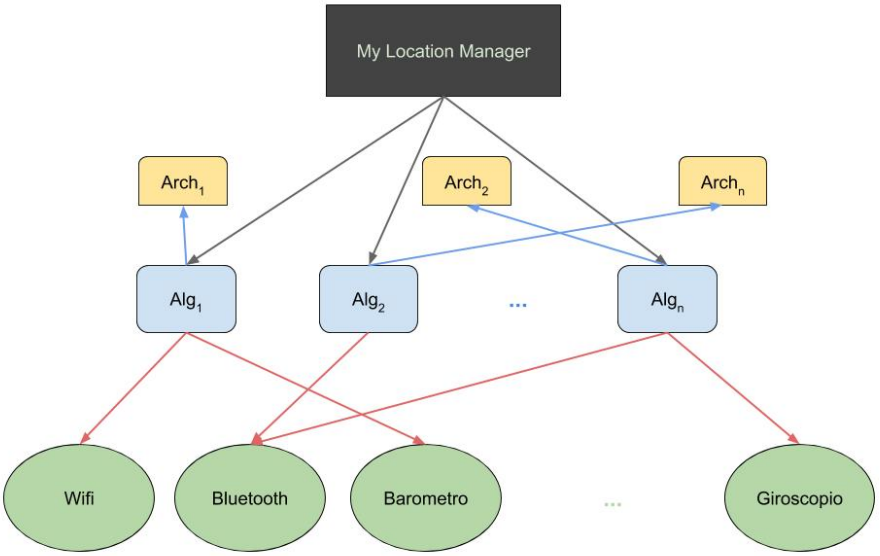


Figura 2.2 Estendibilità

2.2 Scelte di progettazione

2.2.1 Architettura interna della libreria

La libreria si compone essenzialmente di due strumenti:

- il *Location Middleware* che permette il riconoscimento dell'ambiente nel quale lo smartphone si trova. Essenzialmente stima l'ambiente attraverso il confronto dell'accuratezza del segnale GPS con una certa soglia. Successivamente comunica al secondo componente che tipo di localizzazione effettuare;
- il secondo strumento, che da il nome alla libreria, si occupa di ottenere la posizione relativa all'ambiente stimato utilizzando gli algoritmi a sua disposizione.

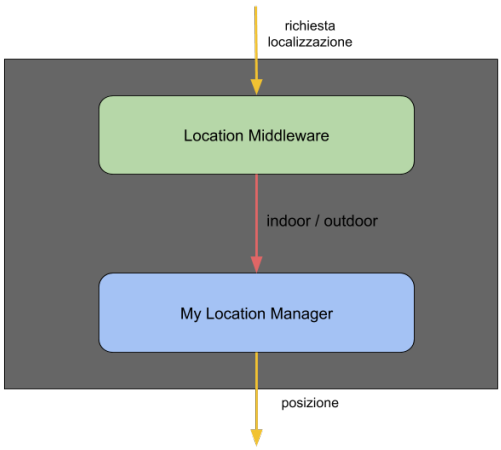


Figura 2.3 Architettura interna della libreria

2.2.2 Modellazione dell'ambiente indoor

L'edificio indoor è stato modellato come l'insieme di diverse informazioni. In particolare:

- il nome;
- la larghezza e l'altezza espresse in 2D;
- le coordinate GPS che identificano l'estremità a Nord Ovest e a Sud Est;
- il numero di piani dell'edificio, i quali a loro volta si compongono di:
  - nome;
  - numero;
  - edificio al quale appartengono.

2.2.3 Vincolo della richiesta di localizzazione generica

E' stato scelto di incaricare la libreria di un delicato compito: capire se lo smartphone, al momento della richiesta, si trova in ambiente esterno o interno ad un edificio. Questa scelta vincola l'utente alla stima operata dalla libreria, in quanto egli non può fare altro che inviare una richiesta generica di localizzazione. Quindi non può scegliere se effettuare una localizzazione indoor oppure outdoor. La libreria restituisce la posizione correlata all'ambiente stimato. Ciò significa che se il My Location Manager stima che lo smartphone sia in ambiente esterno da la possibilità di ottenere solamente la localizzazione GPS e viceversa. Questo fatto è un'arma a doppio taglio, perchè potrebbe condurre a situazioni spiacevoli, in caso di errori

nell'accuratezza del GPS potremmo ottenere un tipo di localizzazione errata. Per esempio, potrebbe succedere che in ambienti indoor la libreria stimi di essere outdoor e quindi fornisca una posizione GPS piuttosto che una posizione indoor. L'utente non ha modo di specificare alla libreria l'ambiente in cui si trova così da risolvere il problema. Dall'altro lato questa scelta dota di intelligenza la libreria.
Da sottolineare che la libreria intesa come spiegata nel capitolo 2.2.1 è soggetta a questo vincolo, mentre lo strumento My Location Manager in sè non lo è. Infatti per evitare questo vincolo basterebbe delegare le richieste di localizzazione direttamente allo strumento My Location Manager, consentendo così la scelta riguardo l'ambiente di localizzazione ma perdendo la feature dell'autoriconoscimento dell'ambiente.

2.3 Integrazione con l'architettura Android

Come accennato in precedenza questa libreria è scritta in Java ed è mirata a dispositivi Android. In questo capitolo introdurrò le caratteristiche principali dell'architettura Android, spiegando a che livello si colloca il mio progetto e come collabora con Android. Infine parlerò delle modalità di integrazione della libreria in un'applicazione preesistente.

2.3.1 Architettura Android

Android è un software open source basato su Linux creato per un ampia gamma di dispositivi mobili. In figura 2.4 vediamo i principali livelli che compongono l'architettura della piattaforma.
Al livello più basso si ha il Linux kernel, l'utilizzo di questo porta a essenziali vantaggi in termini di sicurezza, affidabilità e portabilità. In particolare permette ai produttori di dispositivi mobili di poter sviluppare hardware per un kernel ben conosciuto. Salendo verso l'alto si incontra HAL, ovvero Hardware Abstraction Layer. La sua funzione principale è quella di fornire un interfaccia standard che mette a disposizione le capacità hardware alle Java API di più alto livello. HAL consiste di moduli di più librerie, ciascuna delle quali implementa un'interfaccia per un tipo specifico di componente hardware (es. la fotocamera). Quando il framework API fa una chiamata per accedere ad un componente hardware il sistema carica il modulo relativo.
Al livello superiore si ha Android Runtime, componente scritto per lanciare più macchine virtuali su dispositivi con poca memoria, basato sull'esecuzione di file DEX, formato bytecode similare a quello della Java Virtual Machine (JVM). Allo stesso livello troviamo le librerie C/C++, utili, tra le altre cose, nel fornire metodi di manipolazione grafica 2D o 3D. Al penul-

timo livello salendo verso l'alto c'è il Java API Framework, che mette a disposizione tutte le funzionalità del sistema operativo Android. Queste API sono gli elementi fondamentali che permettono di scrivere applicazioni Android e includono:

- un ricco ed estendibile View System, che permette di costruire la User Interface di un'applicazione;
- il Resource Manager, che permette di accedere a qualsiasi risorsa che non è codice (layout, string, immagini);
- il Notification Manager, che gestisce e mostra le notifiche;
- l'Activity Manager, responsabile della gestione del ciclo di vita di un'applicazione;
- il Content Provider, che permette di accedere ai dati esterni di altre applicazioni.

E' importante notare che gli sviluppatori hanno a disposizione a tutte le API che il sistema Android utilizza. Infine al livello più alto troviamo le applicazioni dei nostri smartphone che parlano con il Java API Framework.

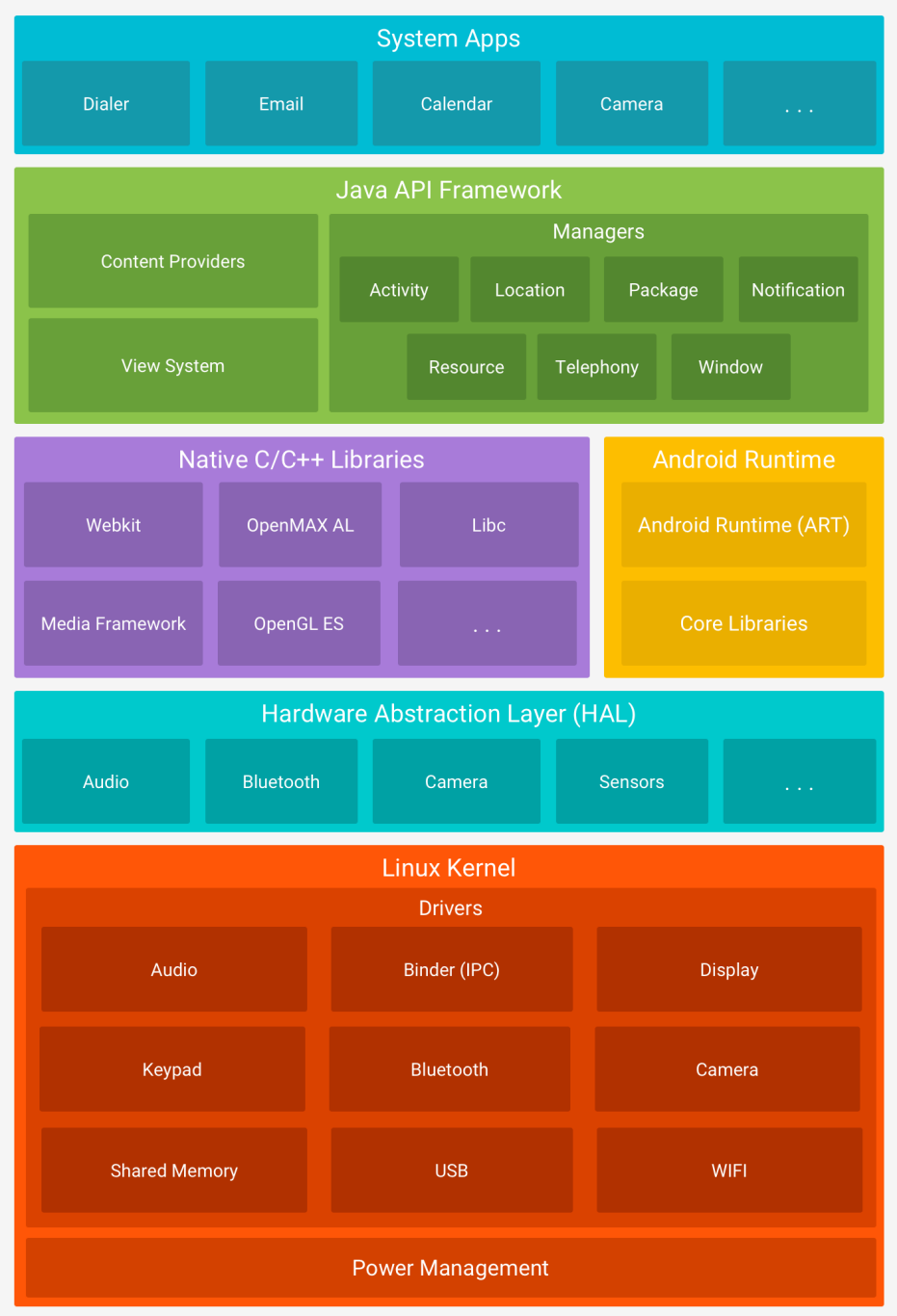


Figura 2.4 Architettura di Android

2.3.2 Il My Location Manager all'interno dell'architettura Android

La libreria si pone, partendo dal basso, tra i penultimi due strati dello stack. Ovvero tra il livello delle applicazioni di sistema ed il livello delle Java API Framework. Il My

Location Manager comunica con quest'ultime per accedere a informazioni e funzionalità del Location Manager e di tutte le classi che curano le interazioni con i sensori necessari alla localizzazione indoor. Poi mette a disposizione delle applicazioni del sistema le funzionalità spiegate precedentemente.

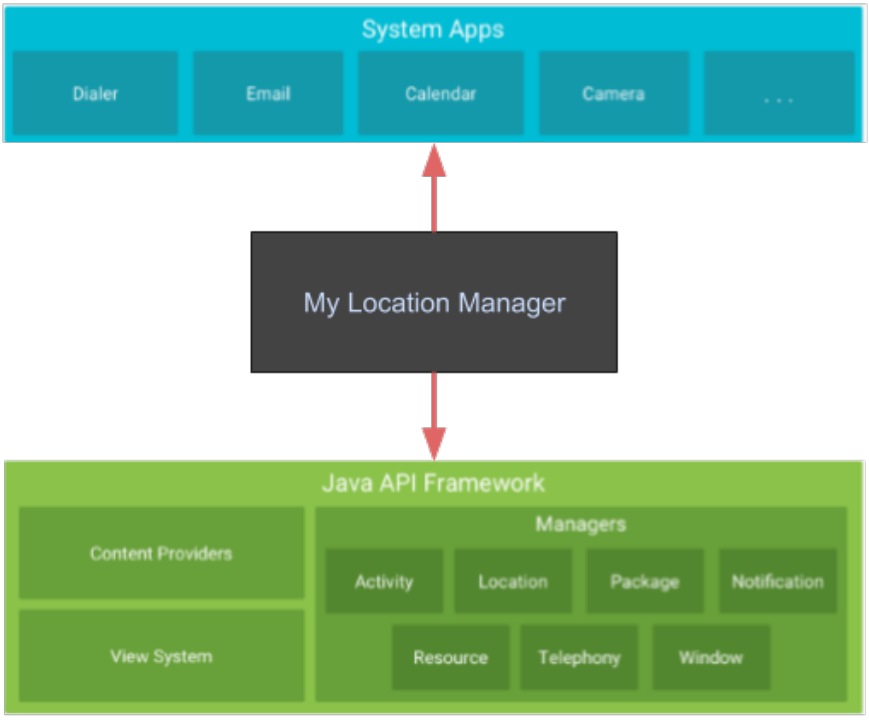


Figura 2.5 My Location Manager all'interno dell'architettura di Android

Sostituibilità Una qualsiasi applicazione che abbia bisogno di ottenere la posizione GPS dello smartphone può farlo semplicemente comunicando con il My Location Manager. In questo senso, di fatto, la mia libreria sostituisce l'utilizzo del Location Manager.

2.3.3 Integrazione in un'applicazione esistente

La libreria è composta di molteplici classi Java che fanno capo a due classi: *Location-Middleware* e *MyLocationManager*. Le quali si occupano di fornire alle applicazioni le funzionalità di localizzazione.

Procedimento di integrazione

Per poter usufruire delle funzionalità della libreria si procede semplicemente con i seguenti passi:

1. si importa nel proprio progetto Android la cartella contenente i file Java;
2. dopo di che si procede all'utilizzo: da un generico file Java basta dichiarare un istanza della classe *LocationMiddleware* e la si usa come una qualsiasi altra classe. Naturalmente devo essere presente almeno un algoritmo indoor implementato, altrimenti questo tipo di localizzazione non è possibile.

A prova di questo è stata sviluppata un'applicazione che si serve della libreria My Location Manager per l'utilizzo di tre algoritmi di Finger Printing che verranno approfonditi nei capitoli successivi. Le activity dell'applicazione utilizzano un istanza della classe LocationMiddleware per richiedere una localizzazione.

2.4 Sviluppi futuri

Ecco alcuni possibili sviluppi futuri:

- la libreria utilizza il Location Manager per recuperare informazioni riguardanti la posizione GPS, si potrebbero utilizzare in aggiunta altre API come il Fused Location Provider Client di Google. Magari fornendo una scelta sulle API che si possono utilizzare;
- la libreria potrebbe, durante l'utilizzo e se specificato, monitorare i movimenti dell'utente di modo da poter captare i cambi di ambiente (da outdoor a indoor e viceversa). In questo caso potrebbe anche stimare in quale edificio l'utente sta entrando (nel caso il determinato edificio sia stato inserito), utilizzando le coordinate GPS relative al perimetro dell'edificio.

Capitolo 3

Implementazione

3.1 Introduzione

In questo capitolo spiegherò nel dettaglio le metodologie di implementazione della libreria My Location Manager e dell'applicazione Android che ne fa uso.

Come accennato nel capitolo 2.2.1 la libreria si compone di due classi Java:

- *LocationMiddleware*, che si occupa di fornire all'applicazione la posizione dello smartphone a seconda dell'ambiente rilevato.
- *MyLocationManager*, che permette l'implementazione di molteplica algoritmi indoor svincolandosi dalle tecnologie e architetture utilizzate.

3.2 Tecnologie

La libreria come l'applicazione è stata sviluppata in Android. Di conseguenza come linguaggio di programmazione è stato utilizzato Java. In particolare per lo sviluppo dell'applicazione è stato utilizzato l'IDE Android Studio alla versione 3.1.3. L'applicazione sviluppata necessita come versione minima del sistema operativo la 6.0 (Marshmallow), mentre la versione di target è la 8.1 (Oreo). Come dispotivo mobile di debug è stato utilizza un nexus 5.

3.2.1 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.

Database

Per lo sviluppo del database è stata utilizzata la libreria Room, introdotta con Android Architecture Components, rappresenta una soluzione "ufficiale" dedicata alla persistenza su Sqlite, caratterizzata da un approccio O/RM.
Tecnicamente, si tratta di un abstraction layer, uno strato software che permette di sfruttare tutte le potenzialità del database senza la preoccupazione di affrontare ogni aspetto nei dettagli.

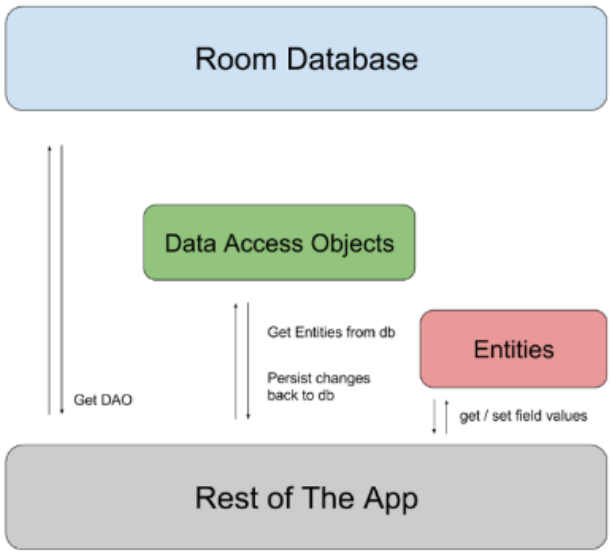


Figura 3.1 Componenti di Room

La libreria Room è composta dai i seguenti componenti:

- **Room database**, rappresenta l'astrazione del database su cui vogliamo lavorare. Verrà creata una classe di questo tipo che costituirà il centro di accesso ai dati: qualsiasi operazione da svolgere passerà di qui;

- **Entity**, ogni classe di questo tipo rappresenta una tabella del database. All'interno delle classi Entity predisporremo tante variabili d'istanza quanti sono i campi previsti dallo schema della tabella, più altri eventuali membri che si renderanno necessari;
- **DAO (Data Access Object)**, un DAO viene utilizzato per incamerare il codice che agirà sui dati, e conterrà i veri e propri comandi per le operazioni CRUD. Tipicamente esistono più DAO in un'applicazione, ma non è necessario ve ne siano tanti quante le Entity. In genere, ogni DAO raccoglie tutte le interazioni che riguardano un sottosistema del software: se stiamo modellando, ad esempio, l'accesso ai dati di una biblioteca, potremo avere un DAO per gestire i dati utente, uno per il catalogo libri, uno per i prestiti e così via.

3.3 Modellazione della libreria

```
public interface LocalizationAlgorithmInterface {

    <T extends View> T build(Class<T> type);

    <T> T locate();

    void checkPermissions();
}
```

Listing 3.1 Interfaccia di un algoritmo di localizzazione

Il primo metodo è necessario nel momento in cui si vuole implementare un algoritmo (solitamente di Finger Printing) che si compone di fasi, quindi di una fase offline e una online. Serve per ottenere un'istanza della View costruita per interagire con l'algoritmo durante la fase offline. Una View non è altro che un componente della User Interface di un'applicazione Android. La funzione *locate* invece ritorna la posizione dell'utente. Mentre *checkPermissions* è utilizzata per richiedere i permessi necessari per un dato algoritmo.
Questa interfaccia è implementata dalle classi *LocationMiddleware* e *MyLocationManager*.

3.3.1 Modellazione del database

In questa sezione spiegherò le entita e le relazioni adottate nello sviluppo del database adoperato dalla libreria. Come precedentemente descritto è stata utilizzata la libreria Room. Il database si compone di 11 entità, che elencherò divise per contesto. I differenti contesti nei quali possiamo collocare le entità per averne una migliore comprensione sono:

- **indoor**, che contiene le entità che modellano l'ambiente indoor e le proprie configurazioni;
- **scan**, cioè le entità che conservano le informazioni riguardanti i risultati degli algoritmi di localizzazione;
- **utils**, entità di sostegno per determinate operazioni;
- **ambiente**, composta dalla fondamentale entità che conserva l'informazione riguardo l'ambiente stimato.

Indoor Le entità di questo tipo servono per descrivere l'ambiente indoor nel quale gli algoritmi sono utilizzati. Sono le seguenti:

- **Building**, che contiene le informazioni dell'edificio. Quindi nome, altezza, larghezza, coordinate dei punti estremi rispettivamente a Nord Over e Sud Est. La chiave primaria è l'*id* auto incrementale e la chiave unica è l'attributo nome.
- **Floor**, contenente le informazioni riguardanti un determinato piano. Quindi nome, numero e id dell'edificio a cui appartiene. La chiave primaria è l'*id*, mentre la chiave unica è la coppia *(idBuilding,numero)*.
- **Algorithm**, questa entità contiene il nome dell'algoritmo e la colonna *phases* che indica se si tratta di un algoritmo a due fasi oppure no. Se $phases = 1$ significa che siamo di fronte ad un algoritmo a due fasi, 0 altrimenti. La chiave primaria è sempre l'*id* e la chiave unica è il nome.
- **Config**, che contiene il nome dell'algoritmo a cui la configurazione è associata, il nome del parametro ed il relativo valore. Questa entità modella la possibilità di definire specifici parametri di configurazione per un determinato algoritmo. Come chiave primaria ha l'*id*, mentre la chiave unica è la tripla *(idAlgoritmo,nome del parametro, valore)*.

Scan Bisogna precisare che, siccome ho sviluppato tre algoritmi di tipo Finger Printing, le seguenti entità riporteranno attributi relativi a questa specifica tipologia di algoritmo. Le entità di questa tipologia sono:

- **ScanSummary**, modella il generico scan di un algoritmo. Contiene l'id del *building* associato, l'id del $floor$ associato, l'id del *algorithm*, l'id della $config$ associata, l'id della rete Wifi utilizzata (che assume valore pari a -1 se non necessario) e la colonna

type che indica se si tratta di uno scan offline oppure online. La chiave primaria è l'*id* mentre la chiave unica è la tupla *(idEdificio,idPiano,idAlgoritmo,idConfig,type)*.

- **OfflineScan**, che contiene l'id dello scan associato, l'id del quadratino indicante il Reference Point, l'id dell'AP wifi, il valore collezionato, il timestamp e le informazioni di latitudine e longitudine GPS. La chiave primaria è l'*id* mentre la chiave unica è la tripla *(idScan,idCella,idWifiAP)*.
- **OnlineScan**, composta dall' id dello scan associato, dal Reference Point stimato dall'algoritmo, dal Reference Point attuale (dal quale l'utente ha lanciato l'algoritmo), il timestamp e le informazioni GPS. La chiave primaria è l'*id* mentre la chiave unica è l'informazione del timestamp.

Utils In questo contesto sono presenti entità che hanno svolto funzioni di appoggio e conservazione di informazioni riguardo la rete wifi.

- **wifiNetwork**, composta dal ssid e mac della rete wifi. La chiave primaria è l'*id* e la chiave unica è l'ssid.
- **wifiAP**, che contiene l'id del ssid (nome della rete) e del bssid della rete (identificatore dell' AP). La chiave primaria è l'*id* mentre la chiave unica è la coppia *(idSsid,bssid)*.
- **liveMeasurements**, utilizzata per monitorare valori di sensori utilizzati negli algoritmi. Contiene l'id dell'algoritmo a cui fa riferimento, l'id dell'AP wifi (se necessario), il nome ed il valore del sensore monitorato, in più ha l'informazione del numero, nel caso si abbia bisogni di più registrazioni. La chiave primaria è l'*id* mentre la chiave unica è la tupla *(idAlgorithm,idAP,name,numero)*.

Ambiente Ovvero l'entità **locInfo** che contiene una colonna **indoorLoc** la quale indica se la libreria ha stimato un ambiente indoor o outdoor. **indoorLoc** è uguale a 1 nel caso di ambiente indoor, altrimenti è uguale a 0.

3.3.2 Parametri di configurazione indoor

Le entità specificate nel contesto indoor compongono i parametri di configurazione che le classi *LocationMiddleware* e *MyLocationManager* utilizzano. Nel codice Java sono rappresentati da un ArrayList di oggetti tipo *IndoorParams*.

```
public class IndoorParams implements Serializable {
    private IndoorParamName name;
    private Object paramObject;
}
```

Listing 3.2 Classe che modella il generico parametro indoor

Dove *IndoorParamName* è la seguente.

```
public enum IndoorParamName {
    BUILDING,
    FLOOR,
    ALGORITHM,
    SIZE,
    CONFIG
}
```

Listing 3.3 Enum IndoorParamName

3.4 Location Middleware

I principali compiti di questa classe sono:

- stimare l'ambiente;
- comunicare alla classe *MyLocationManager* l'algoritmo da utilizzare con le informazioni di configurazione appena spiegate.

La classe *LocationMiddleware* implementa la classe Android *LocationListener* e contiene tra gli altri un campo del tipo *MyLocationManager*.

Costruttore Nel costruttore si provvede a registrare il listener del *LocationManager* in modo da poter reperire aggiornamenti sulla posizione.

Metodi I metodi più importanti sono:

- *init()*, viene invocata all'interno della funzione *OnLocationChanged*, ovvero quando si ottiene la posizione GPS del dispositivo. E' colei che utilizza l'algoritmo spiegato nella sezione 3.4.1. Dopo aver stimato l'ambiente inserisce tale informazione in DB nell'entita *locInfo*.

- *istantiate()*, la quale si preoccupa di recuperare dal DB l'informazione appena citata e in base ad essa istanziare la classe *MyLocationManager*. Indicando un algoritmo indoor opportunamente scelte dall'utente nel caso di ambiente interno, oppure specificando GPS nel caso contrario.
- *build()*, che chiama semplicemente la funzione *build* dell'istanza del *MyLocationManager*;
- *locate()*, che invoca la *locate* del *MyLocationManager*.

3.4.1 Algoritmo di stima dell'ambiente

L'algoritmo di stima dell'ambiente è quel metodo che riconosce se l'utente che sta usando l'applicazione si trova all'interno di un edificio oppure all'esterno. L'algoritmo si basa sull'utilizzo della classe *LocationManager* delle Java API Framework, la quale permette di ottenere la posizione GPS dello smartphone modellata attraverso la classe *Location*. Questa fornisce, oltre la latitudine e la longitudine relativa alla posizione GPS, l'informazione relativa all'accuratezza della posizione, espressa in metri. Questa informazione viene confrontata con un valore di soglia e in seguito viene stimato l'ambiente di appartenenza dell'utente.

Sia gps_{acc} l'informazione di accuracy ottenuta dalla classe *Location* di Android e sia Θ un valore di soglia, anch'esso espresso in metri, con $\Theta = 20$. Più gps_{acc} è bassa più la posizione ritornata è precisa. L'algoritmo è il seguente:

$$a = \begin{cases} indoor, & \text{se } gps_{acc} > \Theta \\ outdoor, & \text{se altrimenti} \end{cases} \tag{3.1}$$

dove a è l'ambiente stimato.

Questo algoritmo viene eseguito ogni volta che l'applicazione viene lanciata.Per tutto il tempo di vita rimanente dell'applicazione l'ambiente stimato rimane invariato. Questo è possibile perchè l'informazione riguardo l'ambiente viene inserita in database nella tabella *indoorLoc* solamente al primo aggiornamento della posizione.

3.4.2 Comunicazione con la classe My Location Manager

Come mostrato in figura 3.2 la classe *LocationMiddleware* effettua i seguenti passi prima di istanziare il My Location Manager:

1. per prima cosa, dopo aver stimato l'ambiente, provvede ad inserire in DB l'informazione. Precisamente utilizza l'algoritmo mostrato qui sopra ed inserisce il risultato nella tabella *indoorLoc* del database.

2. dopodichè passa alla procedura di istanziazione che non fa altro che recupare l'informazione dell'ambiente dal database;

3. infine procede all'instanziazione della classe *MyLocationManager*, passando le relative configurazioni indoor *IndoorParams*. Queste ultime sono passate dalla activity che sta utilizzando il middleware. Nel caso indoor specificherà un determinato algoritmo indoor previamente impostato, nel caso contrario sarà utilizzata di default la classe nativa *LocationManager* per ottenere una localizzazione GPS.

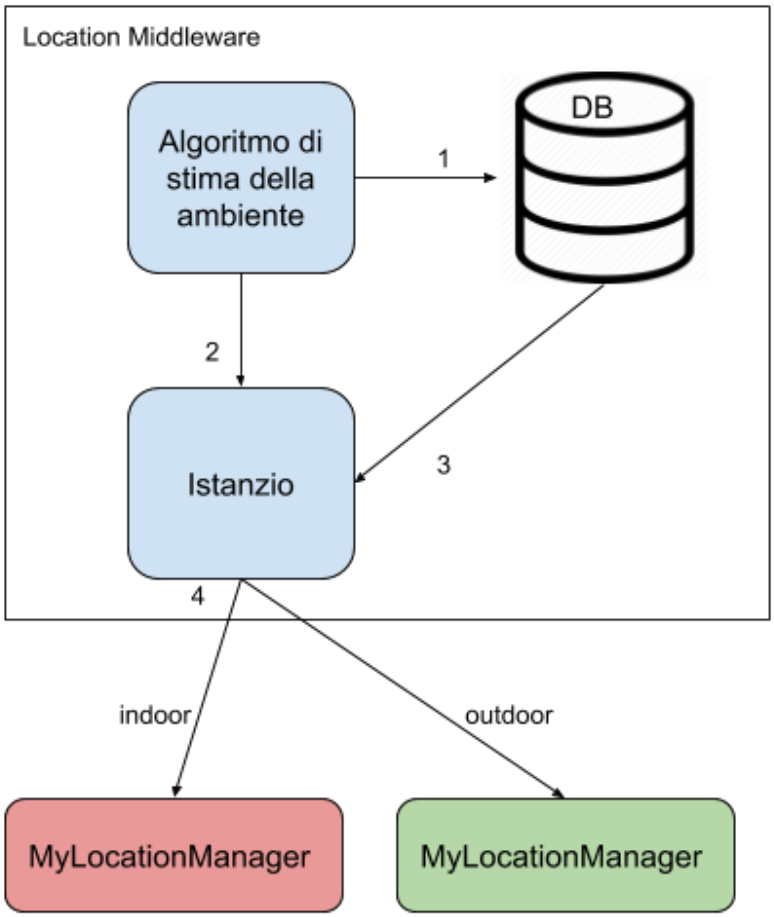


Figura 3.2 Istanziazione del My Location Manager

3.5 My Location Manager

Ora veniamo alla parte fondamentale della libreria che permette di estendere molteplici algoritmi e conseguentemente di scegliere di utilizzare uno di questi. Come il *LocationMiddleware* la classe *MyLocationManager* estende l'interfaccia di cui si è parlato precedentemente 3.3.

3.5.1 Struttura

Per ottenere la fondamentale feature di estendibilità già precendemente spiegata si è scelto di strutturare la classe come esposto in figura. Le fondamentali entità che entrano in gioco sono:

- l'interfaccia *LocalizationAlgorithmInterface*, la quale definisce il comportamento del *LocationMiddleware*, del *MyLocationManager* e di tutti gli algoritmi che si vogliono implementare.
- la classe *LocationManager* che supporta la *MyLocationManager* nel recupero della posizione GPS;
- infine la classe *MyLocationManager*, la quale istanzia un determinato algoritmo scelto.

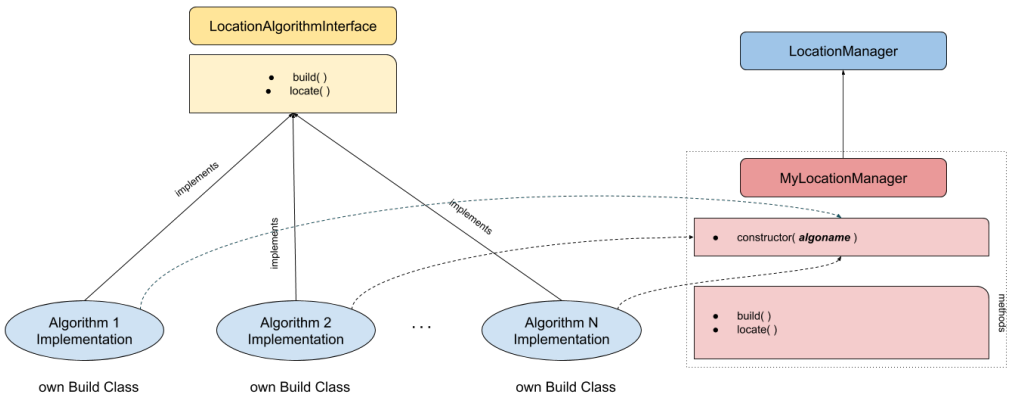


Figura 3.3 Struttura del My Location Manager

L'idea è che ogni qual volta che si vuole implementare un algoritmo indoor si procede alla creazione di una classe Java che estende l'interfaccia 3.3. L'enorme potenzialità di questo approccio è che tale classe è indipendente. Infatti può comporsi di qualsiasi classe e utilizzare qualsiasi sensore o tecnologia.

La classe *MyLocationManager* contiene al suo interno un attributo di tipo *LocalizationAlgorithmInterface*, questo sarà istanziato con l'algoritmo che corrisponde al nome *algoName* indicato. Come abbiamo visto prima è il *LocationMiddleware* che si occupa di istanziare questa classe, quindi nel caso di ambiente esterno si impegna a fornire come *algoName* GPS. Nel caso contrario fornirà il nome di uno specifico algoritmo indoor.

Costruttore Semplicemente all'interno del costruttore è presente uno statement *switch* che, a seconda del nome dell'algoritmo indicato, istanzia in modo diverso il *MyLocationManager*. I casi del sopra citato *switch* possono essere estesi per tutti gli algoritmi che si vogliono implementare, in questo preciso punto si ottiene la nozione di estendibilità. Nella mia applicazione sono presenti i seguenti casi:

- *MAGNETIC_FP*;
- *WIFI_RSS_FP*;
- *WIFI_BAR_RSS_FP*.

Build e Locate Queste funzioni seguono il percorso mostrato qui sotto in figura.

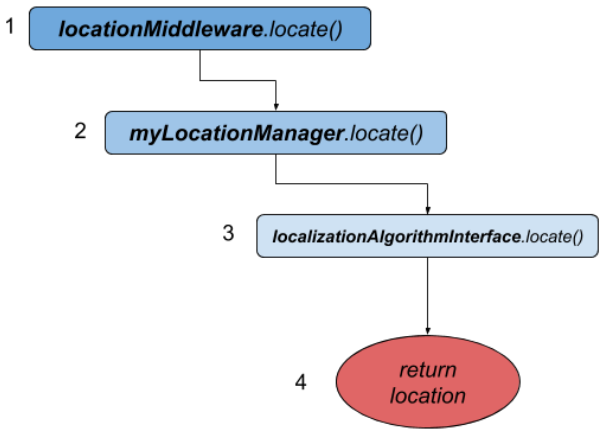


Figura 3.4 Schema della serie di chiamate operate dalla libreria

In figura 3.4 è mostrata la catena di chiamate che la libreria esegue prima di ritornare la posizione. La prima chiamata è effettuata attraverso la classe *LocationMiddleware*, questa parte direttamente da un'activity dell'applicazione. A sua volta viene chiamata la stessa funzione della classe *MyLocationManager*, che a sua volta chiama la *locate* della

LocalizationALgorithmInterface. Da notare che questa classe viene istanziata all'interno del costruttore del My Location Manager con la classe dell'algoritmo scelto. Quindi di fatto si sta chiamando la *locate* dell'algoritmo indoor scelto, la quale ritornerà prima o poi la posizione. Prima o poi perchè anch'essa può comporsi di altre classi. Nel caso dei tre algoritmi indoor di Finger Printing che ho implementato ogni classe contiene al suo interno una classe che si occupa della fase offline ed una che si occupa della classe online.

3.5.2 Gestione dei permessi

Un altro campo molto importante è quello di tipo *MyPermissionsManager*, classe che si occupa di richiedere i permessi necessari per un algoritmo *x*. Quando al costruttore del *MyLocationManager* viene passata l'informazione riguardo l'algoritmo lui provvede ad inoltrarla alla sopra citata classe insieme ad una lista di permessi necessari per quell'algoritmo. In questo modo nel momento dell'istanziazione di un determinato algoritmo verrano richiesti all'utente i permessi necessari. Inoltre verrà richiesto l'accensione di determinate tecnologie (es. wifi, GPS).

3.5.3 Gestione dei parametri indoor

I parametri indoor seguono il flusso delle chiamate della figura 3.4. Quindi semplicemente vengono passati dal *Location Middleware* fino alla classe dell'algoritmo indoor selezionato dall'utente, passando per *MyLocationManager* e *LocalizationAlgorithmInterface*. Anche in questo caso la catena di passaggio parte da un'activity dell'applicazione.

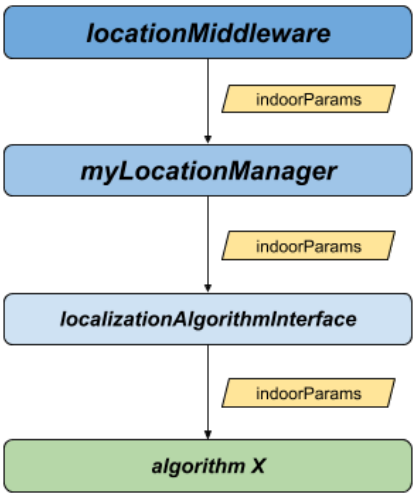


Figura 3.5 Flusso di passaggio dei parametri di configurazione

3.6 L'applicazione Android

E' stata sviluppata un'applicazione Android che fa uso della libreria My Location Manager. Lo scopo dell'applicazione è dare la possibilità all'utente di ottenere la propria posizione (indoor o outdoor che sia) sfruttando la libreria. Nella casistica di ambiente interno l'utente è in grado di modellare a proprio piacimento i parametri di configurazione citati prima.

3.6.1 Funzionalità

L'applicazione fornisce le seguenti funzionalità:

- scelta dei parametri indoor, vale a dire l'edificio, il piano e l'algoritmo da utilizzare. Più eventuali parametri aggiuntivi legati all'algoritmo. Negli algoritmi di Finger Printing utilizzati si ha la possibilità di scegliere la grandezza delle celle che rappresentano i Reference Point.

- possibilità di ottenere la propria posizione. Quest'ultima dipende, come già accennato, dall'ambiente che la libreria stima. Nel caso outdoor verra restituita e mostrata la posizione GPS sulla mappa. Nel caso contrario verrà utilizzato l'algoritmo indoor selezionato. Siccome nel mio lavoro ho implementato tre algoritmi di Finger Printing si accederà alle schermate che permettono di eseguire gli scan offline e online. Da sottolineare il fatto che l'implementazione di altri algoritmi potrebbe portate a differenti necessita dal punto di vista grafico, questo sarà a carico di coloro che svilupperanno gli algoritmi.

3.6.2 Le activities

L'applicazione è composta di 5 activity.

- **MainActivity**, presenta una schermata di gestione delle impostazioni da utilizzare. Permette di accedere a tutte le altre activity riportate.

- **InsertBuildingActivity**, dalla quale è possibile inserire in DB tutte le informazioni relative ad un nuovo edificio;

- **InsertFloorActivity**, permette di inserire nuovi piano per l'edificio che si sta inserendo;

- **ScanActivity**, da a disposizione strumenti di User Interface per svolgere la fase di training degli algoritmi implementati.

- **LocateActivity**, permette di lanciare la fase online degli algoritmi e ottenere la posizione indoor.

MainActivity Questa activity è composta di cinque fragment:

- **BuildingFragment**, che permette la scelta dell'edificio oppure l'inserimento di uno nuovo nel quale effettuare la localizzazione indoor;
- **FloorFragment**, che permette la scelta del piano;
- **AlgorithmFragment**, per scegliere l'algoritmo indoor;
- **ParamFragment**, che permette di scegliere particolari opzioni per l'algoritmo scelto, la peculiarità sta nel fatto che ogni algoritmo carica il proprio Fragment;
- **ButtonFragment**, che permette di accedere alle activity di scan offline e online.

Nelle figure sottostanti vediamo la schermata principale nel caso outdoor e indoor. In alto è riportata la posizione geografica dell'utente con indicazione dell'ambiente stimato.

Nella schermata indoor è possibile accedere alla activity di scan solo dopo aver immesso tutte le informazioni, fino a quel momento il bottone rimane disattivato. Il bottone locate, che fa accedere all'activity di scansione online, rimane disattivato se per le configurazioni scelte non è presente in DB una scansione offline. Nella figura 3.7 è mostrata l'activity visualizzabile quando l'ambiente stimato è indoor. Nel caso outdoor vengono disabilitate tutte le scelte e rimane cliccabile solo il bottone locate. In questo caso al click del bottone locate verrà caricato un fragment nel quale è possibile visualizzare la propria posizione su una mappa (fornita da Google Maps API).

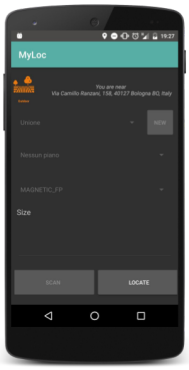


Figura 3.6 Outdoor

Figura 3.7 Indoor

ScanActivity Questa activity è responsabile della fase online degli algoritmi implementati.

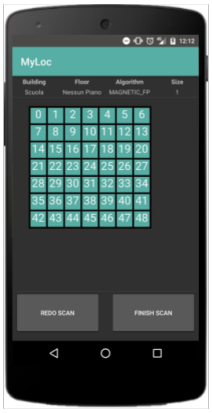


Figura 3.8 Schermata di scan offline

Da questa activity è possibile visualizzare le precedenti configurazioni scelte (in alto) e visualizzare la mappa dell'edificio nel quale si vuole effettuare lo scan. Lo scan inizia semplicemente al primo evento touch percepito su un quadratino e continua a piacimento dell'utente. L'utente decide di terminare lo scan cliccando sul bottone relativo. Inoltre,se vuole, può rifare lo scan.

LocateActivity Anche in questa activity è possibile vedere le configurazioni precedentemente scelte. L'utente inizia lo scan online semplicemente immettendo la propria posizione attuale, dopo di che l'algoritmo stimerà continuamente una posizione con tempo prestabilito tra uno scan e l'altro.

Figura 3.9 Schermata di scan online
In figura l'activity che si presenta appena dopo l'apertura. Nella mappa sono mostrati i quadratini che compongono l'edificio scannerizzato in fase offline dall'utente.

Quando l'utente inizia la localizzazione il quadratino relativo alla posizione attuale verrà colorato in blu, mentre quello stimato dall'algoritmo sarà rosso (nel caso in cui la stima sia sbagliata). Altrimenti, in caso di successo, il quadratino diventa verde.

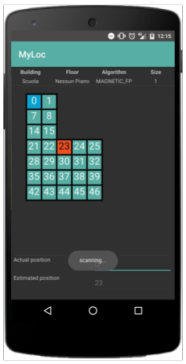


Figura 3.10 Errore dell'algoritmo

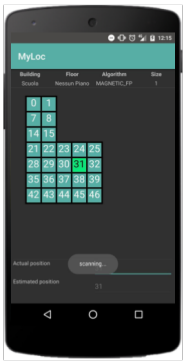


Figura 3.11 Successo dell'algoritmo

InsertBuildingActivity Questa activity permette di inserire un nuovo edificio, inserendo nome, altezza, larghezza e le coordinate dei punti estremi a Nord Est e Sud Ovest direttamente cliccando sulla mappa presente. Quest'ultima risiede in un Fragment che sfrutta le Google Maps API.

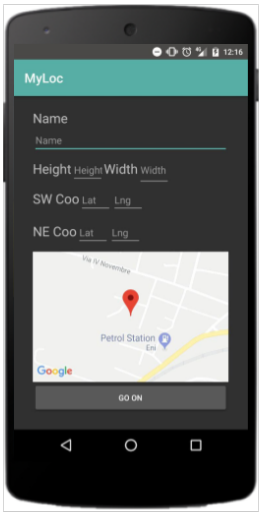


Figura 3.12 Schermata di inserimento di un nuovo edificio

InsertFloorActivity Questa activity permette di inserire un nuovo piano per l'edificio che si sta inserendo specificando il nome ed il numero.

Figura 3.13 Schermata di inserimento di un nuovo piano

3.6.3 Comunicazione tra i Fragment

I Fragment della MainActivity comunicando tra di loro sfruttando le funzioni *OnFragmentInteractionListener* e passandosi un ArrayList di *IndoorParams*.

3.6.4 Utilizzo della libreria My Location Manager

Nei capitoli precedenti abbiamo parlato della classe *LocationMiddleware* e della classe *MyLocationManager* spiegandone le caratteristiche e le modalità di comunicazione. Ma come viene utilizzata la libreria all'interno dell'applicazione? Innanzi tutto l'app si serve della classe *LocationMiddleware* per comunicare con la libreria utilizzando le funzioni *istantiate*, *build* e *locate*. In ogni activity sono svolte le seguenti operazioni:

- **MainActivity**
  1. la classe viene istanziata all'interno della funzione *OnCreate*();
  2. nella stessa funzione viene recuperata attraverso il *LocationMiddleware*, precisamente con la funzione *isIndoorLoc*(), l'informazione dell'ambiente che risiede in DB. Questo a solo scopo di impostare la UI a seconda dell'informazione dell'ambiente.
- **ScanActivity**, viene ottenuta l'istanza della classe *LocationMiddleware* dichiarata in precedenza, insieme ai parametri di configurazione *indoorParams*, dopo di che:

1. viene chiamata la funzione *istantiate*() che in questo caso può solamente istanziare l'algoritmo indoor richiesto;
2. viene chiama la funzione *build*() che permette di effettuare lo scan offline.

- **LocateActivity**, anche in questo viene ottenuta l'istanza del *LocationMiddleware* con i parametri di configurazione, dopo di che:

  1. viene chiamata la funzione *locate*() che a seconda dell'ambiente permetterà le due localizzazioni differenti.

Util per il passaggio dell'istanza LocationMiddleware Si è scelto di implementare una classe, denominata *MyApp*, per gestire il passaggio dell'istanza della classe *LocationMiddleware* in tutta l'applicazione. Questa classe estende la classe *Application* di Android, ed è in grado di essere utilizzata in qualsiasi activity dell'applicazione. La si è utilizzata, oltre che per il motivo appena citato, anche per ottenere il contesto dell'applicazione.

Una volta istanziato il middleware nella main activity viene salvato nella classe *MyApp* attraverso una funzione di set e viene recuperato dalle altre activity attraverso una funzione get.

Capitolo 4

Algoritmi

In questo capitolo parlerò degli algoritmi di Finger Printing che ho implementato nella libreria My Location Manager. Precisamente sono:

- algoritmo di Magnetic Finger Printing;
- algoritmo di Wifi Finger Printing;
- algoritmo di Wifi Finger Printing più barometro.

Tutti e tre si basano sul concetto di Finger Printing deterministico. Spiegherò la teoria su cui si basa questo approccio e la relativa implementazione nella libreria. Dopo di che parlerò in dettaglio delle implementazioni dei tre algoritmi citati sopra.

4.1 Algoritmo di Finger Printing Deterministico

4.1.1 Wifi-based

Fase offline

Questa fase si occupa di costruire la radio map dividendo l'area di interesse in celle. Queste celle sono dette Reference Point. I valori RSS trasmessi dai presenti AP sono collezionati nei Reference Point per un certo periodo di tempo.
L'i-esimo elemento della mappa ha la forma:

$$M_i = \left(B_i, \{\vec{a_{i,j}} | j \in N_i\}, \Theta_i\right) \tag{4.1}$$

con $i = 1, ..., M$.
Dove B_i è l'i-esimo Reference Point. Il vettore $\vec{a_{i,j}}$ contiene i valori RSS percepiti dal j-esimo

AP ed il suo k-esimo elemento è denotato come $a_{i,j}^{k}$. Il parametro Θ_i contiene ogni altra informazione che può essere utile in fase di localizzazione. Potrebbe essere per esempio l'orientazione dello smartphone.
L'i-esimo Finger Print è denotato come R_i, mentre l'insieme di tutti i Finger Printing come $R = \{R_1, ..., R_M\}$. Quindi l'i-esimo elemento della radio map è $M_i = (B_i, R_i)$.

Fase online

L'algoritmo implementato si basa sulla minimizzazione della distanza Euclidea delle misurazioni RSS considerando tutti gli AP recepiti. Più formalmente la posizione stimata è scelta come il Reference Point x_j che è più vicino, cioè:

$$\hat{x} = \underset{x_j}{\operatorname{argmin}} \Big(\sum_{i=1}^{n} \big(r_i - \rho_i(x_j) \big)^2 \Big) \tag{4.2}$$

dove r_i è il valore RSS misurato dall'i-esimo AP durante la fase di localizzazione online, $\rho_i(x_j)$ è il valore RSS recepito dall'i-esimo AP nel j-esimo Reference Point.

4.1.2 Basato su campo magnetico

L'algoritmo di minimizzazione della distanza Euclidea utilizzato per la tecnica di Finger Printing mediante campo magnetico è leggermente diverso. Questo perchè il valore ricevuto è unico a differenza del caso Wifi dove è necessario ascoltare tutti gli AP. Di seguito sono mostrate le variazioni alle formule per le rispettive fasi.

Fase offline In questo caso l'i-esimo elemento della radio map ha la forma seguente:

$$M_i = \Big(B_i, \vec{a}_i, \Theta_i \Big) \tag{4.3}$$

con $i = 1, ..., M$.
Dove tutte le variabili tramite il vettore a corrispondo alle entità spiegate nel caso Wifi. In questo caso $\vec{a}_i$ è un singolo valore del campo magnetico.

Fase online Nella fase online la formula diventa la seguente:

$$\hat{x} = \underset{x_j}{\operatorname{argmin}} \Big(\big(r_i - \rho_i(x_j) \big)^2 \Big) \tag{4.4}$$

4.1.3 Implementazione all'interno della libreria

Prima di inserire nel database le informazioni relative agli scan offline e online la libreria provvede a popolare la tabella *ScanSummary*. In particolare fornisce i dati mostrati in tabella, nel caso in cui l'algoritmo non utilizza il Wifi il corrispodente campo sarà impostato uguale a -1. La colonna *type* indica se lo scan che si sta effettuando è online oppure offline.

idBuilding	idFloor	idAlgorithm	idConfig	idWifiNetwork	type

Inoltre nella colonna $idConfig$ viene specificata la configurazione da utilizzare per quello scan. Nel caso dei tre algoritmi implementati la configurazione riguarda la grandezza delle celle che compongono la mappa.

Fase offline

Durante la fase offline l'utente mediante l'applicazione popola la radio map, che verrà salvata in DB nella tabella *OfflineScan*, inserendo righe della forma mostrata in tabella.

idScan	idCell	idWifiAP	value	timestamp	latitude	longitude

L'$idScan$ contiene l'informazione dello scan associato a quella rilevazione. L'$idCell$ rappresenta lo specifico Reference Point nel quale si è ottenuto il valore. L'$idWifiAP$ contiene l'id dell'AP Wifi, è uguale a -1 se l'algoritmo non utilizza la tecnologia Wifi. La colonna *value* contiene il valore recepito in quel dato RP, potrebbe trattarsi di un valore RSS, di un valore di campo magnetico o di un valore di qualche altro sensore. La colonna $timestamp$ contiene le informazioni della data e dell'ora in cui la registrazione è avvenuta. Le colonne $latitude$ e $longitude$ rappresentano una posizione GPS, precisamente la posizione in cui la registrazione è avvenuta.

Formalmente, riferendosi alla definizione 4.1 di elemento di una radio map le variabili in questione sono state interpretate in questo modo:

	B_i	$a_{i,j}^k$		θ_i		
idScan	idCell	idWifiAP	value	timestamp	latitude	longitude

$idCell$ si riferisce all'i-esimo Reference Point. La coppia $(idWifiAP, value)$ modella la posizione $a_{i,j}^k$ del vettore $\vec{a_{i,j}}$ che contiene i valori RSS ricevuti dal j-esimo AP ricevuti dal i-esimo RP. Infine Θ_i rappresenta le informazioni aggiuntive $(timestamp, latitude, longitude)$.

Fase online

La fase online è realizzata da una classe Java che esegue l'operazione indicata in figura 4.2. Sono state create due classi, una per la versione Wifi dell'algoritmo e l'altra per la versione basata sul campo magnetico. Le quali vengono istanziate con una determinata radio map rappresentata da un ArrayList di oggetti OfflineScan e con i valori RSS live percepiti dagli AP presenti, nel caso Wifi, oppure con il valore live del campo magnetico.
Dopo avere ottenuto la posizione stimata dall'algoritmo l'applicazione procede ad inserire il risultato nella tabella *OnlineScan*.

idScan	idEstimatedPos	idActualPos	timestamp	latitude	longitude

In tabella *idEstimatedPos* si riferisce a $\hat{x}$ della figura 4.2. *idActualPos* si riferisce alla posizione reale dell'utente. Entrambe si riferiscono a Reference Point, quindi corrispondono all'id di una determinata cella.

4.2 Creazione e gestione della UI

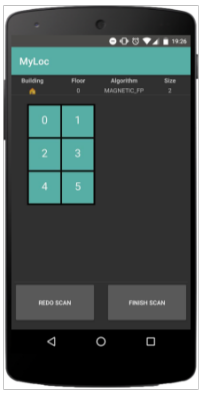
Tutti gli algoritmi implementati si servono della classe *MapView* per portare a termine le due fasi. Questa classe disegna la mappa dell'edificio nella UI.

4.2.1 MapView

La MapView viene utilizzata sia in fase offline che in fase online. In fase offline riceve i parametri di configurazione *IndoorParams* necessari per la costruzione corretta della mappa dell'edifico. Mentre in fase online riceve anche le indicazioni riguardo la posizione stimata, la posizione reale dell'utente e le informazioni riguardo le celle scannerizzate durante la fase offline. Sfrutta la classe *Canvas* che permette di disegnare all'interno di un activity attraverso l'utilizzo della funzione *onDraw*.
In particolare i parametri di cui fa uso sono:

- la *height* dell'edificio;
- la *width* dell'edificio;
- la *config*, nella quale è specificata *cellSize*, cioè l'informazione della grandezza in metri delle celle in cui si suddivide l'area dell'edificio. Questa informazione è impostata nella schermata principale dell'applicazione. In questo modo l'utente è in grado di effettuare gli scan variando l'area dei Reference Point.

- la *cella* che rappresenta la posizione stimata dall'algoritmo;
- la *cella* che rappresenta la posizione reale dell'utente.
- la lista di $OfflineScan$ utile per disegnare la mappa in fase online.



In figura si vede un esempio di disegno della mappa. L'immagine a sinistra rappresenta un edificio 6x4 metri, con grandezza della cella pari a 1 metro. Mentre l'altra rappresenta lo stesso edificio con grandezza della cella pari a 2.

Gestione dei Finger Print nella UI Gli eventi touch sulla mappa sono gestiti dalla funzione *OnTouchListener*.

```
public <T extends View> T build(Class<T> type){
        mV = new MapView(MyApp.getContext(),null, null,
           indoorParams, null);
        mV.setOnTouchListener(new View.OnTouchListener() {
            // gestione evento touch sulla mappa nella UI
        });
        return type.cast(mV);
}
```

Listing 4.1 Gestione degli eventi touch sulla mappa

Nella figura sottostante è presente l'immagine che raffigura un inserimento di un valore per la costruzione della radio map. Nel caso illustrato, sarà inserita nella tabella OfflineScan un riga con $idCell = 24$.

Figura 4.1 Touch sulla mappa

4.3 Magnetic Finger Printing

4.3.1 Struttura dell'algoritmo

Questo algoritmo è stato implementato nella libreria servendosi di tre classi. Una classe padre e due classi figlie. La classe padre prende il nome di *MagneticFieldAlgorithm*, mentre le figlie sono *MagneticOfflineManager* e *MagneticOnlineManager*. Le ultime sono figlie nel senso che sono degli attributi della prima. Questa struttura è stata utilizzata in tutti gli algoritmi perchè risultata ottima in quanto semplifica la gestione delle due fasi.

4.3.2 La classe padre

Questa classe implementa l'interfaccia *LocalizationAlgorithmInterface* presentata in 3.3.

Campi Contiene i seguenti tre campi:

- un'oggetto di tipo *MagneticOfflineManager*;
- un'oggetto di tipo *MagneticOnlineManager*;
- un'ArrayList di *IndoorParams* che modellano i parametri di configurazione indoor.

Metodi Come metodi si limita ad'avere quelli specificati dall'interfaccia, tra cui *build* e *locate*, che istanziano la rispettive classi e chiamano le rispettive funzioni.

4.3.3 La classe per la fase offline

Si tratta della classe *MagneticOfflineManager*, la quale viene istanziata all'interno delle funzione *build*, in seguito lei stessa esegue la propria implementazione della *build*.

SensorEventListener *MagneticOfflineManager* implementa la classe nativa di Android *SensorEventListener*, la quale è utilizzata per ricevere notifiche dal *SensorManager* quando ci sono nuovi dati della tipologia richiesta. La ricezione dei dati è gestita nella funzione *OnSensorChanged*.

Il campo magnetico M è calcolato come:

$$M = \sqrt{magX^2 + magY^2 + magZ^2} \tag{4.5}$$

dove le tre variabili in questione rappresentano il campo magnetico per il relativo asse ottenute dalla funzione qui sotto.

```
@Override
public void onSensorChanged(SensorEvent event) {
    float magX = event.values[0];
    float magY = event.values[1];
    float magZ = event.values[2];
    liveMagnitude = Math.sqrt((magX * magX) + (magY *
       magY) + (magZ * magZ));
    // do stuffs with liveMagnitude
}
```

Listing 4.2 Gestione ricevimento del campo magnetico

La funzione build Questa funzione è responsabile della creazione della mappa nella UI che rappresenta l'edificio selezionato e cura tutte le interazioni con essa. Come mostrato nella sezione precedente al touch dell'utente sulla mappa viene gestito l'ottenimento del campo magnetico e il relativo inserimento dell'offlinescan.

In figura si evidenzia la logica dell'applicazione durante la fase offline:

- ad un tempo x avviene il collezionamento del valore del campo magnetico all'interno della tabella *LiveMeasurements*;

- ad un tempo $x_1 > x$ il valore viene recepito dall'applicazione ed inserito nella tabella *OfflineScan*, insieme a tutti i valori indicati nella tabella.

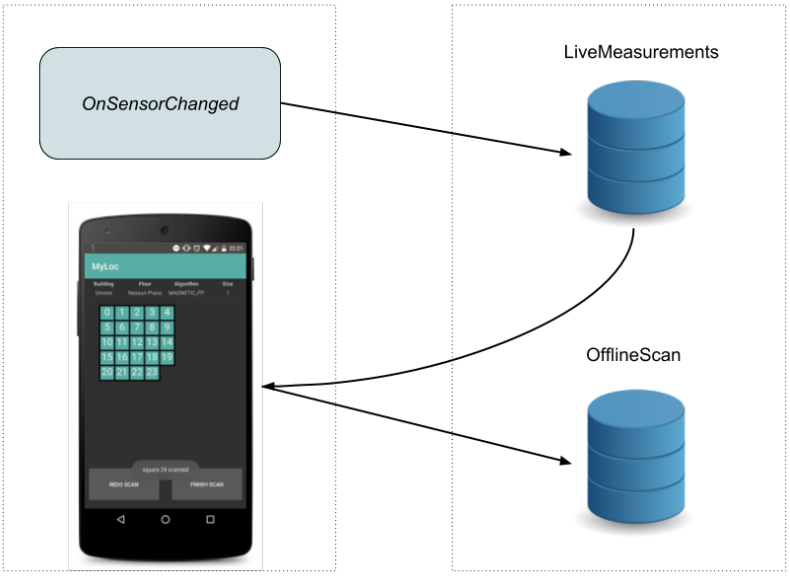


Figura 4.2 Fase offline

4.3.4 La classe per la fase online

Questa classe come la precedente implementa la classe *SensorEventListener* e tutta la logica è sviluppata nella funzione *locate*. La funzione *OnSensorChanged* si occupa di registrare un aggiornamento per il campo magnetico. Dopo averlo ottenuto si preoccupa di inserirlo in DB, precisamente nella tabella *LiveMeasuremets*.

Quando viene invocata la funzione *locate* essa si preoccuperà di recuperare da DB il valore del campo magnetico e lo passerà insieme alle informazioni sullo scan offline alla classe responsabile dell'algoritmo di minimizzazione della distanza Euclidea come illustrato in figura. Dopo avere ottenuto la posizione stimata, questa viene inserita in database nella tabella *OnlineScan*.

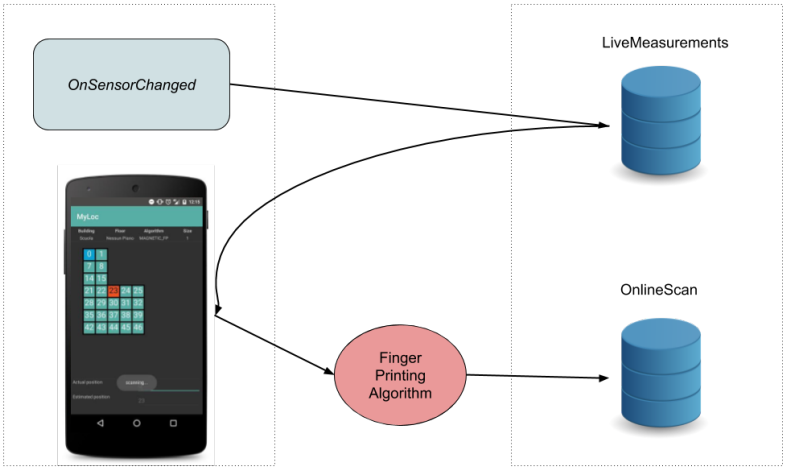


Figura 4.3 Fase online

Anche in questo caso è importante notare che l'inserimento in *LiveMeasurements* e la chiamata della funzione *locate* avvengono in due istanti diversi, seppur molto vicini tra loro di modo che le informazioni siano coerenti.

4.4 Wifi Finger Printing

4.4.1 Struttura dell'algoritmo

Anche questo algoritmo si serve di una classe padre: *WifiAlgorithm*, la quale ha come attributi le classi *WifiOfflineManager* e *WifiOnlineManager*. Inoltre si serve di un'altra classe che prende il nome di *WifiScanReceiver*, la quale è responsabile degli scan wifi necessari per recuperare i valori RSS dei differenti AP.

4.4.2 La classe padre

Anche questa classe implementa l'interfaccia *LocalizationAlgorithmInterface* presente in 3.3.

Campi Ha come attributi:

- un'oggetto di tipo *WifiOfflineManager*;
- un'oggetto di tipo *WifiOnlineManager*;
- un'oggetto di tipo *WifiScanReceiver*;

- la lista di oggetti di tipo *IndoorParams*.

4.4.3 La classe WifiScanReceiver

Questa classe estende la classe nativa di Android *BroadcastReceiver*. Viene utilizzata per intercettare eventi del tipo *SCAN_RESULT_AVAILABLE_ACTION*, quindi per registrare i valori RSS. In particolare quando succede un evento di questo tipo la classe *WifiScanReceiver* provvede ad inserire le informazioni nella tabella LiveMeasurements.

idAlgorithm	idAP	name	number	value

In questo modo mette a dispozione dell'applicazione il reperimento continuo di questi valori, che avviene in fase online. Questa classe si serve della classe *AsyncTask* per lanciare un thread background che ogni qualvolta che intercetta quel tipo di evento provvede ad aggiornare le informazioni nel database.

```
if (intent.getAction()
    .equals(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION)) {
    // manage wifi infos
}
```

Listing 4.3 Gestione ricevimento dei valori RSS

4.4.4 La classe per la fase offline

WifiOfflineManager al suo interno si dota di un *BroadcastReceiver*, il quale viene azionato ogni qual volta l'utente clicca sulla mappa per effettuare lo scan di una cella. In questo modo sono inserite nella tabella OfflineScan le informazioni relative ai valori RSS. In questo caso la logica di utilizzo del BroadcastReceiver è diversa dalla fase online, perchè si ha bisogno del valore proprio nell'istante in cui è eseguita l'azione touch sullo schermo.

Build La logica qui sopra è implementata dentro la funzione *build* ed è mostrata nella figura sottostante. Al termine dello scan il *BroadcastReceiver* viene deregistrato.

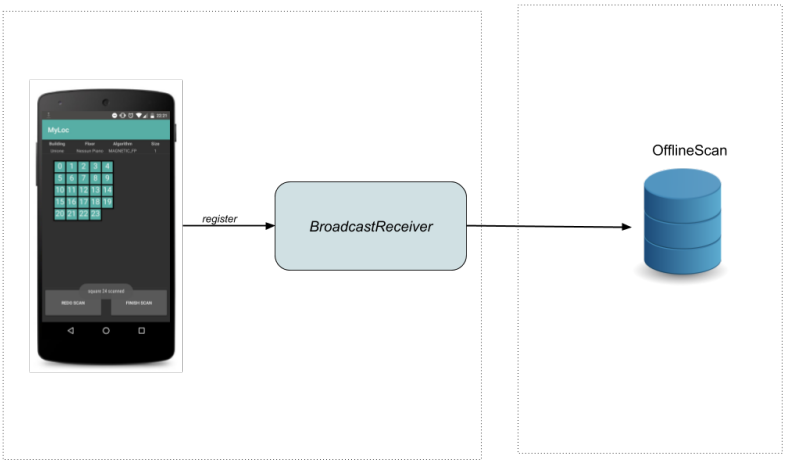


Figura 4.4 Fase offline

Gestione della rete Wifi e degli AP Questa classe individua, nel momento in cui viene istanziata, se lo smartphone è connesso al Wifi e provvede ad inserire nella tabella *WifiNetwork* tutte le reti utilizzate. Inoltre si preoccupa, per ogni rete, di salvare i BSSID degli Access Point nella tabella *WifiAP*.

4.4.5 La classe per la fase online

Questa classe fa uso della classe *WifiScanReceiver*. Al suo interno definisce solamente la funzione *locate*, la quale quando viene chiamata procede a recuperare dalla tabella *LiveMeasurements* i valori RSS live per ogni AP. Dopo di che, insieme alle informazioni sullo scan offline, li fornisce alla classe che si occupa di stimare la posizione.

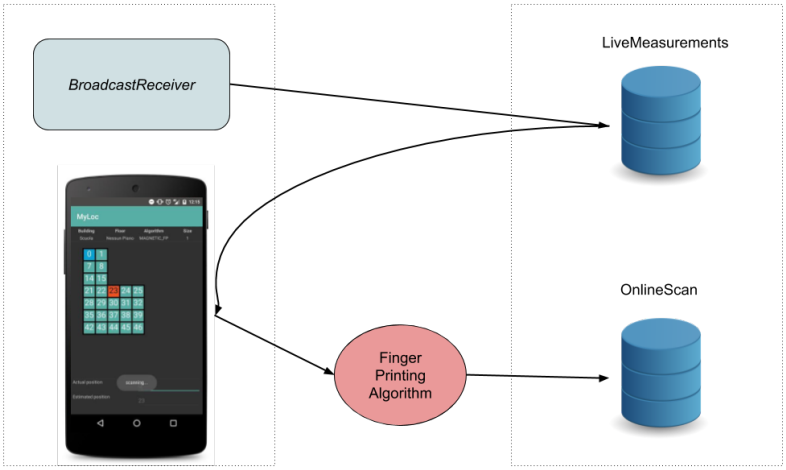


Figura 4.5 Fase online

In figura è mostrata la logica di funzionamento della classe online. In questo caso il BroadcastReceiver agisce in un momento precedente (con un tempo ragionevolmente basso, di modo che i valori siano affidabili) a quello della chiamata della funzione *locate*. In particolare siccome la stima della posizione è continua non appena si inserisce la propria posizione, il BroadcastReceiver aggiorna continuamente la tabella *LiveMeasurements* finchè non si termina l'algoritmo.

4.5 Wifi Finger Printing + Barometro

4.5.1 Struttura dell'algoritmo

Questa classe implementa l'algoritmo che da il titolo a questo paragrafo. Come per gli altri casi sono state utilizzate due classi rispettivamente per la fase offline ed online, le quali fanno capo alla clase *WifiBarAlgorithm*.

Inoltre sono presenti le seguenti classi:

- la classe *PressureListener*, che si occuppa di collezionare i valori della pressione in un certo periodo di tempo;
- la classe *SeaAltitudeListener*, che si occupa di recepire il valore dell'altezza sopra il livello del mare nella posizione dello smartphone.

4.5.2 La classe per la fase offline

Questa classe è praticamente identica a quella utilizzata nel caso dell'algoritmo Wifi RSS. Anche in questo ambito, durante la fase offline si procede ad attivare il BroadcastReceiver per ottene i valor RSS che servono per comporre gli scan offline che saranno inseriti in db.

4.5.3 Il SeaAltitudeListener

E' stata creata una classe che estende il *LocationListener* che è incaricata di salvare in database l'informazione inerente all'altitudine rispetto al livello del mare. Questo dato è stato reperito all'interno della funzione *OnLocationChanged*, la quale ritorna l'attuale posizione GPS dello smartphone con tante informazioni correlate, tra cui l'altezza dal livello del mare espressa in metri. Questa informazione è salvata nella tabella *LiveMeasurements*.

idAlgorithm	idAP	name	number	value
X	-1	seaAltitude	1	$value_1$

```
@Override
public void onLocationChanged(Location location) {
    // getting sea altitude from GPS location
}
```

Listing 4.4 Gestione ricevimento altezza dal livello del mare

4.5.4 Il PressureListener

Per modellare il comportamento specificato nella sezione 1.5.1 si è creata una classe dal nome *PressureListener* che implementa la classe nativa di Android *SensorEventListener*.

```
@Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() ==
           Sensor.TYPE_PRESSURE) {
        // saving data in db
        }
}
```

Listing 4.5 Collezionamento dei valori di pressione

Tale classe nella funzione indicata qui sopra si occupa di salvare nella tabella *LiveMeasurements* le informazioni riguardo ai dati di pressione ricevuti. In particolare vengono continuamente aggiornati dieci valori. Questo procedimento entra in azione quando viene istanziata la classe per la fase online, la quale poi reperirà i dati per procedere alla stima del piano.

idAlgorithm	idAP	name	number	value
X	-1	pressure	1	$value_1$
X	-1	pressure	2	$value_2$
...	...	...	...	...
X	-1	pressure	10	$value_{10}$

Nella tabella *LiveMeasurements* viene anche inserita l'informazione riguardo l'altezza calcolata utilizzando la formula 1.18, dove P è il valore della pressione ricevuto dal listener.

4.5.5 La classe per la fase online

Stima del piano La stima del piano segue i punti illustrati in 1.5.1. Quindi come prima cosa vengono recuperati dalla tabella *LiveMeasurements* i dati che rispondono al nome "pressure". Da questi sono calcolate la media e la varianza, rispettivamente in figure 1.15 e 1.16. Quindi rispettando la formula 4.6:

$$H = \begin{cases} H_1, & \text{se } \sigma_N^2 > \Theta \\ H_2, & \text{se altrimenti} \end{cases} \tag{4.6}$$

si ricalcolerà l'altitudine nel caso la varianza σ_N^2 superi una certa soglia altrimenti viene utilizzata l'altitudine precedentemente inserita nella tabella *LiveMeasurements*. Dopo di che si arriva finalmente al calcolo del piano utilizzando la formula 4.7:

$$f = \frac{\omega_{base} - \omega_{now}}{H} \tag{4.7}$$

dove ω_{base} è l'altezza del piano terra dal livello del mare, questa informazione è stata recuperata utilizzando la sopra citata *SeaAltitudeListener*. Mentre ω_{now} è l'altitude attuale calcolata con 1.18. H è l'altezza del piano (nei test effettuati è pari a 5 metri).

Localizzazione tramite FP deterministico Naturalmente prima di procede alla fase online devono essere scannerizzati tutti i piani dell'edificio in questione. L'informazione del piano viene poi utilizzata per pescare dal database la radio map associata a quel piano. Precisamente

si cerca nella tabella *ScanSummary* quello scan relativo al piano stimato, dopo di chè, nel caso positivo, viene utilizzato l'algoritmo di FP deterministico su gli scan offline trovati.

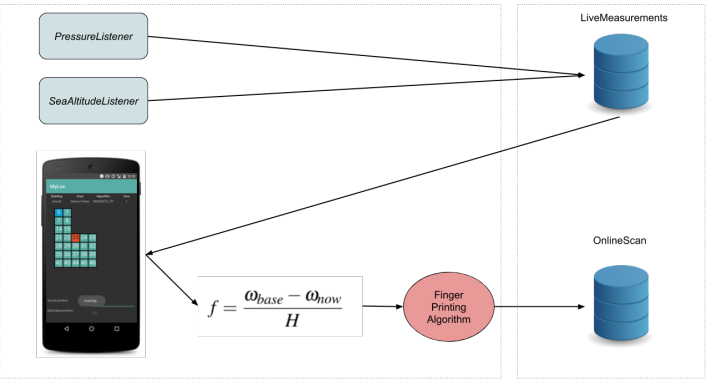


Figura 4.6 Fase online

In questo caso l'activity mostrerà, in caso positivo, l'informazione del piano stimato (in alto) e il relativo RP corrispondente alla posizione stimata dall'algoritmo di FP deterministico.

Capitolo 5

Performance Evaluation

Nei capitoli precedenti ho parlato dell'architettura della libreria, della sua implementazione e della integrazione dei vari algoritmi. In questo capitolo illustrerò, nonostante l'obiettivo della mia tesi sia stato la creazione della libreria, i test effettuati sugli algoritmi di Finger Printing basati su Wifi, campo magnetico e sull'algorimo ibrido Wifi più barometro. Sottolineo che gli algoritmi sono stati implementati a scopo di mostrare e validare il corretto funzionamento della libreria e non a scopo di ricerca sugli stessi. In particolare, per ognuno, evidenzierò metriche di valutazione e i problemi riscontrati.

Per quanto riguarda il terzo algoritmo, che è una variazione dell'articolo [1] in quanto utilizza un algoritmo di FP deterministico, saranno discussi solo i test sulla stima del piano. Questo perchè i test sull'algoritmo di Finger Printing saranno discussi per l'opportuno algoritmo Wifi.

Metriche di valutazione Per gli algoritmi basati su Wifi e campo magnetico si è scelto di utilizzare le seguenti metriche di valutazione:

- *accuracy*, calcolata come la media della distanza Euclidea tra la posizione stimata e quella reale;
- la *percentuale di successo* degli algoritmi sul totale di tentativi effettuati.

Ambiente di testing Gli ambienti utilizzati in fase di testing sono stati: una sala studio dell'Università di Bologna, per i test su gli algoritmi basati su Wifi e campo magnetico; il dipartimento di Matematica dell'Università di Bologna, per i test sull'algoritmo Wifi più barometro, il quale possieede 8 piani. Per il primo edificio sono state scelte lunghezza e larghezza pari a 8 metri. Sono stati svolti, per gli algoritmi di Finger Printing, due test:

- uno con grandezza della cella pari a 1 metro;
- l'altro con grandezza pari a 2 metri.

L'ambiente considerato è dotato di 4 Access Point Wifi.

Entità dei test (Algoritmi basati su Wifi e campo magnetico) Per ogni algoritmo e per ogni tipologia di test, per almeno 5 Reference Point diversi, sono stati eseguiti almeno 5 tentativi di localizzazione, per un totale di circa 100 localizzazioni totali. Le quali sono equamente distribuite tra le due tipologie di algoritmo, e rispettivamente tra le due grandezze delle celle scelte.

5.1 Algoritmo di Magnetic Finger Printing

Questa sezione è dedicati agli esperimenti riguardanti l'algoritmo di Finger Printing basato su campo magnetico. Spiegherò perchè, purtroppo, i risultati non sono stati ottimi e illustrero i relativi grafici.

5.1.1 Problemi riscontrati

Si sono presentati in particolare due tipologie di problemi, qui sotto elencate.

Omogeneità dei valori L'errore più comune incontrato è stato quello in cui si hanno aree anche molto distanti tra loro con valori di campo magnetico molto simili. Durante i test è capitato spesso che la posizione stimata, all'interno della porzione di edificio presa in considerazione, fosse molto distante da quella reale. Questo perchè la presenza di valori omogenei tra loro ha fatto si che l'algoritmo di minimizzazione della distanza Euclidea stimasse quelle posizione come simili. Questo problema è anche noto come *Finger Print Small Dimension Problem* [3], si presenta quando la radio map collezionata è composta di Finger Print di dimensione pari a 1, in questo caso Finger Print composte solamente dal valore del campo magnetico. Esso porta ad una importante inaffidabilità dei Finger Print e alla costruzione di una radio map di valori ambigui.

Imprecisione delle coordinate GPS Per effettuare la valutazione dei test si sono utilizzate le coordinate geografiche salvate rispettivamente in fase offline ed online. Questo per permettere il calcolo della distanza Euclidea tra le coordinate della posizione stimata e quelle della posizione reale, in modo da calcolare l'errore di localizzazione. Il noto problema dei

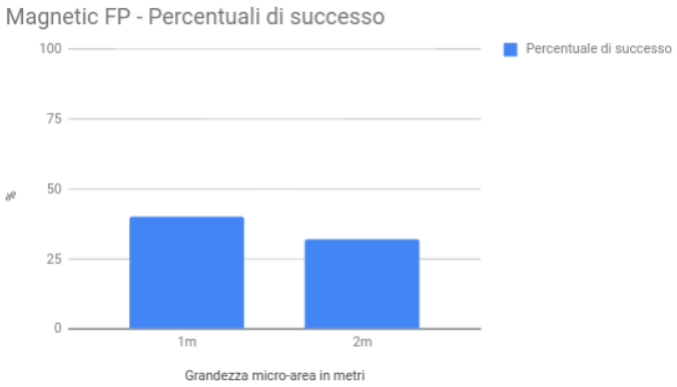
GPS in ambienti indoor ha fatto si che le coordinate GPS siano risultate molto imprecise, in particolare è successo che per scansioni di Reference Point diversi le coordinate della posizioni fossero molto simili se non uguali. Questo ha portato a dei risultati dei test non attendibili (presenti in figura qui sotto).

5.1.2 Accuracy

In figura, nell'asse y osserviamo i metri di errore stimati dalla distanza Euclidea tra posizione stimata e posizione reale. Mentre nell'asse x osserviamo i risultati ottenuti per le due configurazione di grandezza del quadratino utilizzate.

Le accuracy calcolate sono risultate incoerenti per i motivi che ho elencato nel paragrafo precedente. In particolare i valori molto bassi ottenuti, significherebbero che l'algoritmo ottiene degli eccellenti risultati, in realtà sono dovuti alla poca precisione del GPS in ambienti indoor, che ha portato all'ottenimento di coordinate sempre molto simili tra loro. Praticamente il calcolo della distanza Euclidea tra le due posizioni ha coinvolto valori praticamente uguali.

5.1.3 Percentuali di successo



Nell'asse y sono rappresentate le percentuali di successo, mentre nell'asse x i valori ottenuti per ciascuna tipologia di test. L'algoritmo di Finger Printing basato su campo magnetico ha riportato una percentuale di successo pari al 40 % con la grandezza della cella pari a 1 metro, mentre nell'altro caso la percentuale è del 32 % circa. Le scarse prestazioni di questo algoritmo sono causate dal problema della *Finger Print Small Dimension Problem*, di cui si è discusso in precedenza.

5.2 Algoritmi di Wifi Finger Printing

Qui di seguito espongo i principali problemi che possono verificarsi in approcci di questo tipo.

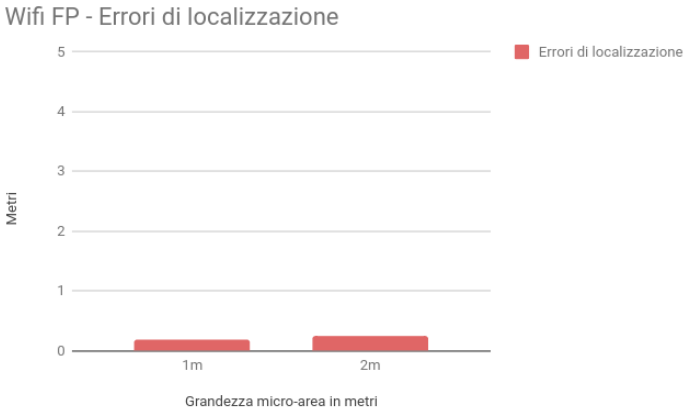
5.2.1 Problemi riscontrati

Missed AP Problem Il più comune problema che si presenta in algoritmi di questo tipo è il cosidetto "Missed AP Problem". Questo problema si presenta quando micro-aree, tra loro vicine, riportano registrazioni di valori RSS di Access Point differenti. Di conseguenza una delle due aree percepirà meno AP (o più). Ovviamente questo problema, per la natura della minimizzazione della distanza Euclidea, porta ad un incremento degli errori in fase di localizzazione.

Numero di AP Una variabile molto importante che decide le sorti di questo algoritmo è il numero di AP accessibili, che è direttamente proporzionale alle percentuali di successo dell'algoritmo. Ovvero, meno AP ci sono, meno i risultati sono accurati, e viceversa.

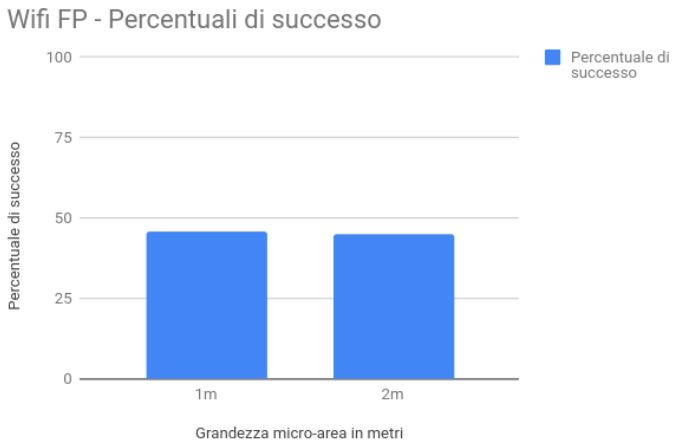
Imprecisione delle coordinate GPS Anche per questo algoritmo valgono le considerazioni fatte precedentemente riguardo all'imprecisione del GPS.

5.2.2 Accuracy



Anche nel caso Wifi si sono riscontrate le problematiche legate alla precisione del GPS già discusse in precedenza che hanno portato all'ottenimento di valori incoerenti.

5.2.3 Percentuali di successo



Le percentuali di successo ottenute sono del 46 e 45 % rispettivamente per grandezza di 1 e 2 metri. I bassi risultati ottenuti possono essere stati causati dal problema dell'access point mancante. E anche in questo caso si può parlare di Finger Printing poco affidabili, in quanto sono di dimensione 1, perchè contengono solamente l'informazione del valore RSS.

Conseguentemente la radio map create risulta potenzialmente ambigua, non come nel caso del campo magnetico, perchè in questo caso i 4 AP presenti aiutano a raggiungere una stima migliore della posizione dello smartphone.

5.3 Algoritmi Wifi più barometro - Stima del piano

Come precedentemente descritto per questo algoritmo è stata valutata solamente la parte inerente alla stima del piano. In quanto l'algoritmo di Finger Printing deterministico applicato dopo aver individuato il piano è lo stesso presentato nel paragrafo precedente. Nel capitolo precedente si è parlato delle variabili che entrano in gioco in questa tecnica, cioè valore della pressione e altitudine rispetto il livello del mare. Il reperimento della pressione atmosferica, attraverso l'utilizzo del barometro, non ha riportato problemi ed è il valore è risultato affidabile.

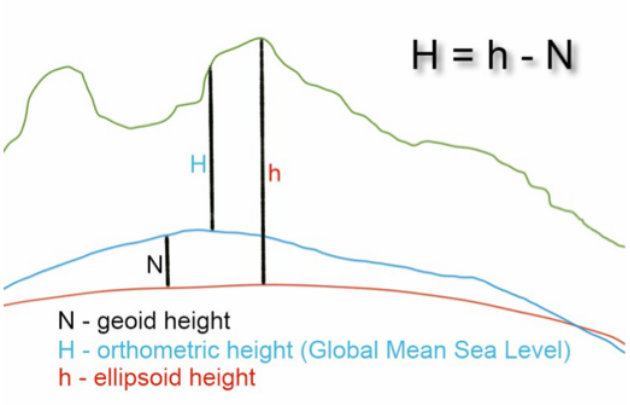
5.3.1 Problemi riscontrati

L'altitudine è stata recuperata dall'oggetto *Location* di Android, il quale fornisce l'altitudine sopra il livello del mare, come indicato da documentazione. Il reperimento dell'altitudine, tramite l'utilizzo del GPS, è risultato inaffidabile causando problemi nell'algoritmo di stima. Si è presentato in particolare il problema indicato qui sotto.

WGS84 Reference Ellipsoid Problem Quando lo smartphone recupera l'altitudine sopra al livello del mare, in realtà sta facendo una stima, utilizzando un modello geodetico. L'altitudine ritornata dall'oggetto di Android rappresenta l'altitudine in metri rispetto l'ellissoide WGS84, che è una approssimazione della superficie terrestre. Questo è un problema legato alla tecnologia GPS. Come accenanto prima questo problema ha alterato i risultati della formula della stima del piano 4.7.

Work Around Nel mio lavoro di tesi non è stato studiato un modo per sorpassare questo problema. Naturalmente esistono diverse soluzioni a questo problema, tra le quali:

- calcolare l'altezza del ricevitore come la differenza tra l'altitudine sopra il riferimento dell'elissoide e l'altitudine del geoide in quel punto sulla terra.



- calcolare l'altitudine come media delle ultime *n* altitudini ricevute, di modo da avere un valore più affidabile.
- oppure ottenere l'altezza di una data coordinata GPS chiamando un servizio web affidabile.

5.4 Sintesi

In questo capitolo si sono descritti i test ed i problemi incontrati per gli algoritmi sviluppati. Più che l'affidabilità delle tecniche si è dimostrato che la libreria, cuore della mia tesi, è in grado di implementare diverse tipologie di algoritmo. In quanto il mio lavoro si è concentrato più sull'integrazione degli algoritmi che sulla loro accurattezza rimando all'articolo [1] per consultare i risultati affidabili dell'algoritmo Wifi più barometro.

Conclusioni

Il lavoro di questa tesi è stato la progettazione ed implementazione di una libreria, per sistemi operativi Android, con gli obiettivi seguenti: possibilità di soddisfare richieste di localizzazione (indoor o outdoor) da parte di un'applicazione; possibilità di implementare, ed utilizzare, molteplici algoritmi indoor svincolandosi dalle possibili architetture e tecnologie sottostanti. Si è parlato dello stato dell'arte, quindi del progetto i-locate come riferimento per questo lavoro, e delle differenti categorie di algoritmi indoor presenti in letteratura. Si è quindi poi illustrata l'architettura del sistema presentato soffermandosi sui possibili problemi e sviluppi futuri. Successivamente si è argomentato riguardo l'implementazione della libreria e dell'applicazione Android che la integra. Infine si sono descritti gli algoritmi implementati nella libreria e i relativi test. Si è dimostrato che sono stati raggiunti gli obiettivi prefissati aprendo quindi a possibili sviluppi futuri, come per esempio: il tracking live della posizione dell'utente, parte della libreria, di modo da captare eventuali cambi di ambiente; il possibile utilizzo della libreria per confrontare gli esiti di ogni algoritmo, di modo da avere una metrica comune che possa valutare l'affidabilità di ogni algoritmo.

Bibliografia

[1] Bisio, I., Sciarrone, A., Bedogni, L., and Bononi, L. (2018). Wifi meets barometer: Smartphone-based 3d indoor positioning method. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6.

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

[3] Davidson, P. and Piché, R. (2017). A survey of selected indoor positioning methods for smartphones. *IEEE Communications Surveys Tutorials*, 19(2):1347–1370.

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

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

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

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

[8] Felice, M. D., Bocanegra, C., and Chowdhury, K. R. (2018). Wi-lo: Wireless indoor localization through multi-source radio fingerprinting. In *2018 10th International Conference on Communication Systems Networks (COMSNETS)*, pages 305–311.

[9] Goyal, P., Ribeiro, V. J., Saran, H., and Kumar, A. (2011). Strap-down pedestrian dead-reckoning system. In *2011 International Conference on Indoor Positioning and Indoor Navigation*, pages 1–7.

[10] Haifeng Xing, Jinglong Li, Bo Hou, Yongjian Zhang, and Meifeng Guo (2017). *Pedestrian Stride Length Estimation from IMU Measurements and ANN Based Algorithm*. Journal of Sensors.

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

[12] Jimenez, A. R., Seco, F., Prieto, C., and Guevara, J. (2009). A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu. In *2009 IEEE International Symposium on Intelligent Signal Processing*, pages 37–42.

[13] LEVI, R. (1996). Dead reckoning navigational system using accelerometer to measure foot impacts. *U.S. Patent Number 5,583,776*.

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

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

[16] Saarinen, J., Suomela, J., Heikkila, S., Elomaa, M., and Halme, A. (2004). Personal navigation system. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 1, pages 212–217 vol.1.

[17] Won Kim, J., Jin Jang, H., Hwang, D.-H., and Park, C. (2004). A step, stride and heading determination for the pedestrian navigation system. *Journal of Global Positioning Systems*, 3:273–279.

[18] Wu, C.-L., Fu, L.-C., and Lian, F.-L. (2004). Wlan location determination in e-home via support vector classification. In *IEEE International Conference on Networking, Sensing and Control, 2004*, volume 2, pages 1026–1031 Vol.2.