Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea magistrale

Development of a smartphone based indoor navigation system for visually impaired people

09-12-2014

Candidato:
Stefano Basso
Matricola IF-626616

Relatore:
Prof.ssa Giada Giorgi

Correlatore:
Ing. Guglielmo Frigo

Anno Accademico 2014-2015

Indice

Introduzione

Lo sviluppo di sistemi di localizzazione e navigazione per pedoni da utilizzare in ambienti interni costituisce una sfida per la comunità scientifica. L'impossibilità di utilizzare tecniche basate su GPS per realizzare tali sistemi, ha concentrato la ricerca sull'utilizzo di tecnologie alternative, quali dispositivi di rete pre-installati nell'ambiente (basati su Wi-fi, Ultra Wide Band, Bluetooth e RFID), sensori inerziali e algoritmi di computer vision. Sebbene tali metodi siano ancora nella loro infanzia, non ci sono dubbi che in futuro tali tecnologie diventeranno diffuse al pari del GPS. Pertanto è importante fin da subito progettare questi sistemi in modo tale che i loro servizi siano usufruibili anche da persone con difficoltà visive.

In questa tesi si sono affrontate le sfide della localizzazione e navigazione in ambienti interni di persone Blind and Visual Impaired (BVI). In particolare, si è realizzato un progetto software modulare per smartphone con sistema operativo Android. Attraverso la lettura di campioni acquisiti dai sensori e processati dall'unità di elaborazione interna dello smartphone, si è realizzato un sistema di *Pedestrian Dead Reckoning (PDR) navigation* che *a*) stima la posizione dell'utente in funzione del numero di passi effettuati e dell'orientazione del suo cammino, *b*) rappresenta il percorso effettuato su una mappa bidimensionale e *c*) salva/carica in forma persistente l'informazione del cammino. L'assenza di apparecchiature esterne, la facilità di installazione del sistema e la famigliarità che un utente BVI già possiede nell'utilizzo dello smartphone sono fattori essenziali che questa applicazione ha considerato affinché sia effettivamente utilizzata da persone con difficoltà visive.

In aggiunta il software è stato progettato ponendo particolare attenzione ai possibili cambiamenti e miglioramenti che dovrà subire in futuro: poiché la ricerca in questo settore è in continua evoluzione, è ragionevole ipotizzare che diverse tecniche ed algoritmi possano essere integrati o andranno a sostituire quelli già presenti. Pertanto è stato necessario realizzare una piattaforma software sufficientemente e facilmente estensibile per agevolare il nuovo codice di sviluppo necessario a soddisfare le esigenze del cliente e/o committente.

La struttura di questo documento è organizzata nel modo seguente: il capitolo 1 espone il contesto scientifico e tecnologico attuale relativo ai sistemi di localizzazione e navigazione per pedoni, il capitolo 2 descrive la progettazione in moduli dell'intera applicazione per Android, i successivi capitoli descrivono nel dettaglio le funzionalità di ogni singolo modulo, il capitolo 9 documenta il comportamento sperimentale del sistema PDR mentre il capitolo 10 discute i possibili sviluppi futuri dell'applicazione, fornendo sia un contributo scientifico che un contributo legato alla piattaforma hardware e software. Infine, nella appendice A si illustrano gli strumenti di sviluppo realizzati e utilizzati.

Introduction

The development of systems for location and navigation of pedestrian in indoor environments is a challenge for the scientific community. Due to the impossibility of using techniques based on GPS to realize such systems, the research is focusing on the use of alternative technologies, such as network devices installed in the environment (based on Wi-Fi, Ultra Wide Band, Bluetooth and RFID), inertial sensors and computer vision algorithms. Although such methods are still in their infancy, in the future there is no doubt that these technologies become widespread like the GPS. Therefore it is important to design these systems in order that visual impaired people can use their services.

In this thesis we have addressed the challenges of localization and navigation in indoor environments for BVI people. In details we have implemented a software modular project for smartphones with Android OS. Through the reading of samples acquired by the sensors and processed by the cpu inside the smartphone, we have created a *PDR* system that *a*) it estimates the user's position as a function of the individuated number of steps and the orientation of its heading, *b*) it represents the path on a two-dimensional map, and *c*) it saves/loads the map in a persistent form. The absence of external equipment, ease of system installation and the familiarity that a BVI user already owns in the use of smartphone are essential factors that this application has considered: the goal is to realize a system that it is effectively used by people with visual impairments.

In addition, the software has been designed with particular attention to possible changes and improvements that it will undergo in the future: as the research in this area is constantly changing, it is reasonable to assume that different techniques and algorithms can be integrated or will replace the existing ones. Therefore it was necessary to create a software platform sufficiently and easily extensible to facilitate the new development of code necessary to meet the customer and/or client's needs.

The structure of this thesis is organized as follows: section 1 exposes the scientific and technological context for pedestrians localization and navigation systems, the chapter 2 describes the design of the modules that realize the Android application, the next chapters describe in detail the functionality of each module, the chapter 9 documents the experimental behavior of the PDR system while chapter 10 discusses the future developments of the application, providing both scientific and hardware/software platform contribution. Finally, in the appendix A, we discuss the development tools made and used during this project.

Capitolo 1

Contesto

Fonte: *edsteinink* website. Ultima visita: 3 dicembre 2014.

1.1 Blind and Visual Impaired people

Ad Agosto 2014, secondo il *World Healt Organization* più di 285 milioni di persone al mondo sono affette da problemi di vista [1]. In accordo con *Internation classification of diseases* [2], tale numero include sia persone con moderate[1] e consistenti[2] difficoltà visive (circa 246 milioni) che persone affette da cecità (circa 39 milioni). Purtroppo, in molti casi, tale handicap condiziona pesantemente la mobilità e l'orientamento di queste persone, causando in loro una perdita di indipendenza e un fattore discriminante. Anche dal punto di vista morale, tale problema necessita di essere affrontato. Sebbene negli ultimi anni i governi internazionali si sono impegnati per contrastare la forma patologica di questo fenomeno, è dovere anche della comunità scientifica del ramo dell'e-

[1]Con accuratezza (o acuità) visiva compresa tra 0.1 e 0.3, secondo la scala *Monoyer*.
[2]Con accuratezza (o acuità) visiva compresa tra 0.05 e 0.1, secondo la scala *Monoyer*.

laborazione dell'informazione svolgere la propria parte, fornendo strumenti che possano essere d'aiuto per guidare gli spostamenti effettuati da un utente Blind and Visual Impaired (BVI).

Innanzitutto è importante distinguere tra due categorie di sistemi di supporto che un utente BVI utilizza per i suoi movimenti [3]:

- Il sistema di supporto primario ha il compito di assicurare la mobilità dell'utente, permettendogli di identificare gli oggetti stazionari e mobili che possono intralciare il suo percorso. Un bastone o un cane guida per non vedenti costituiscono degli esempi di sistema di supporto primario. Si noti che tali sistemi possono essere usati in autonomia per assicurare una navigazione sicura (cioè libera da collisioni) dell'utente.

- Il sistema di supporto secondario ha la funzione di indicare il percorso che l'utente deve percorrere per raggiungere la propria destinazione. Pertanto essi svolgono una funzione di *navigazione* e di *localizzazione* della posizione dell'utilizzatore nell'ambiente in cui esso cammina. Tali strumenti non possono essere utilizzati da soli per garantire una sicura navigazione ma devono essere integrati da un supporto primario: di conseguenza, per questi sistemi non c'è l'obbligo di fornire un'alta accuratezza (ad es. pochi metri potrebbero già soddisfare le esigenze dell'utilizzatore).

Affinché un sistema di supporto secondario sia effettivamente utilizzato da un utente BVI esso non deve essere d'intralcio al supporto primario. Per tale motivo è necessario, fin da subito, progettare nuovi sistemi di navigazione che siano non intrusivi e facili da portare. Poiché è probabile che una delle due mani di un utente BVI sia occupata a mantenere il sistema di supporto primario, è opportuno che tali dispositivi non occupino l'unica mano libera dell'utente, in modo tale da consentire lo svolgimento di altre attività (ad es. proteggersi da una caduta accidentale).

1.2 Contesto scientifico: pedestrian navigation

Lo sviluppo di piattaforme hardware di dimensioni ridotte e a prezzi contenuti ha incrementato l'interesse per la ricerca di algoritmi inerenti alla *localizzazione* e alla *navigazione* di pedoni, non necessariamente affetti da problemi di vista.

I Global Navigation Satellite System (GNSS), quali *GPS*, *GLONASS* o *Galileo*, costituiscono gli strumenti principali attraverso i quali realizzare un sistema di localizzazione in ambienti esterni. Tuttavia, il loro utilizzo in ambienti indoor è reso impossibile dall'attenuazione che il segnale radio riceve. Per tale motivo la ricerca scientifica si è focalizzata su nuove tecnologie ed algoritmi per progettare sistemi alternativi per affrontare e risolvere il problema.

Tra le varie soluzioni proposte, è possibile identificare diversi approcci per localizzare un pedone in un ambiente interno. Tali tecniche si basano su [4]:

- Triangolazione di segnali radio.
- Direct sensing.
- Pattern recognition.
- Pedestrian Dead Reckoning (PDR) navigation.

Triangolazione di segnali radio

In accordo con [5], gli algoritmi che fanno uso di triangolazione di segnali radio (generalmente emessi da una rete Wi-fi nel contesto della localizzazione di pedoni) sfruttano le proprietà geometriche del triangolo per stimare la posizione di un dispositivo. Innanzitutto è bene precisare che tali tecniche richiedono l'utilizzo di due tipologie di apparati: alla prima categoria appartengono un insieme di dispositivi installati in posizioni note dell'ambiente circostante. Alla seconda categoria appartiene il dispositivo che segnala la posizione dell'utente; quest'ultimo deve essere portato o indossato dall'utilizzatore. Attraverso la reciproca comunicazione tra questi apparecchi, il sistema è in grado di stimare la posizione del pedone.

Un prima tecnica di questa famiglia di algoritmi stima la posizione del pedone misurando la sua *distanza* rispetto a molteplici punti di riferimento installati nell'ambiente (ad es. access point Wi-fi): in letteratura, si indica con il termine *lateration* gli algoritmi che si basano su questa idea. La misura di tale distanza può avvenire osservando diverse caratteristiche del segnale emesso tra gli apparecchi: la tecnica Received Signal Strengths (RSS) [6, 7] valuta l'attenuazione di potenza del segnale emesso mentre le tecniche Time Of Arrival (TOA) [7], Time Difference Of Arrival (TDOA) [8, 9] e Round Trip Of Flight (RTOF) [10] fondano i propri algoritmi valutando l'intervallo temporale tra l'invio e la ricezione del segnale. Infatti la distanza dal dispositivo mobile all'apparecchio posizionato in un punto fisso dell'ambiente è proporzionale al tempo di propagazione del segnale.

Una seconda tecnica di localizzazione individua la posizione di un dispositivo determinando la sua angolazione rispetto a molteplici punti di riferimento dell'ambiente: in letteratura tale metodo è noto come *angulation* [11].

Direct sensing

L'insieme di algoritmi che determinano la posizione dell'utente attraverso il rilevamento di speciali identificatori (*tags*) installati nell'ambiente circostante sono indicati con il termine *direct sensing* algorithms. Poiché è nota la posizione di ogni *tags* dell'ambiente, quando il dispositivo in possesso dell'utente rileva uno di questi identificatori allora è possibile localizzare la posizione dello stesso.

Esistono numerose implementazioni di questi algoritmi che utilizzano differenti tecnologie: tra queste si cita l'uso di Radio Frequency Identifier Description (RFID) [12, 13], di Bluetooth beacon [14], di ultrasuoni [15] e di raggi infrarossi [16].

Pattern recognition

Questa famiglia di algoritmi suddividono la propria elaborazione in due parti:

1. La fase *offline* prevede di costruire un'insieme di features (fingerprints), ovvero di caratteristiche discriminanti dell'ambiente circostante.

2. La fase *online* prevede di confrontare la misura attuale dell'ambiente circostante rilevata dal dispositivo indossato dall'utente con l'insieme di features determinato al passo 1. Dal confronto che restituisce il grado di corrispondenza (o confidenza) più elevato è possibile stabilire la posizione del dispositivo.

Affinché i risultati di questo metodo siano soddisfacenti è necessario stabilire accuratamente l'insieme di features discriminanti per l'ambiente circostante.

Si noti che tali algoritmi possono utilizzare fenomeni fisici diversi: ad esempio esistono tecniche che fondano il proprio grado di confidenza sulla base del RSS [17] mentre altri sulla base di algoritmi di computer vision [18], attraverso cui si confronta l'immagine attuale catturata dal dispositivo con le immagini presenti nel dataset acquisito durante la fase offline.

PDR navigation

Gli algoritmi di navigazione PDR determinano la posizione del pedone in funzione dell'ultima posizione nota o stimata. Mentre l'utente cammina, un sistema PDR calcola la sua posizione attraverso i risultati di una piattaforma inerziale in possesso o indossata dal pedone: attraverso le letture di alcuni sensori, come accelerometro e magnetometro, il sistema stima la distanza percorsa e l'orientazione dell'utente rispetto al punto in cui il cammino ha avuto inizio. Spesso, tale piattaforma inerziale è realizzata con l'utilizzo di un Sensors MicroElectroMechanical Systems (MEMS): gran parte dei sistemi di questo tipo [19–25], sfruttano gli Inertial Measurement Unit (IMU) per ricavare un contapassi e una bussola digitale: in questo modo, dopo aver stimato la lunghezza di ogni passo, è possibile implementare una prima forma di navigazione bidimensionale all'interno di una mappa.

Il principale vantaggio di questa categoria di algoritmi consiste nel fatto che l'intero sistema (sensori ed unità di elaborazione) è contenuto nel dispositivo trasportato o indossato dall'utente.

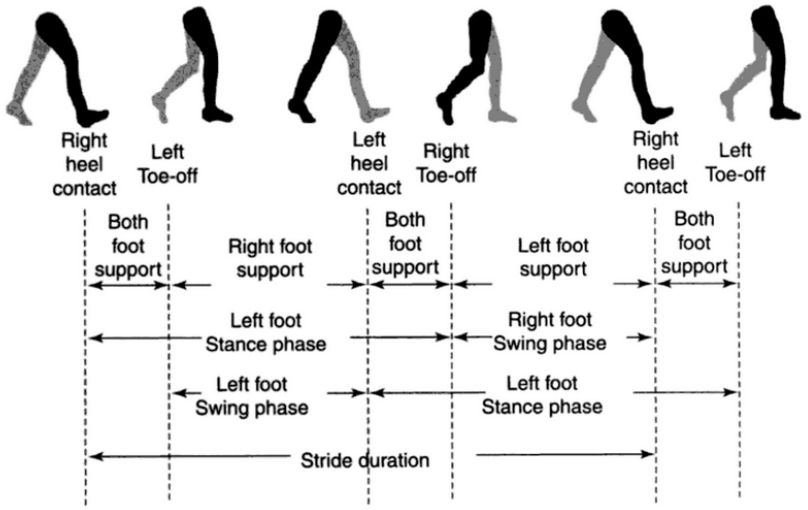


Figura 1.1: Gait cycle. Fonte: Waldemar Karwowski, *International Encyclopedia of Ergonomics and Human Factors (second edition)*, vol. 3, 2006, Taylor & Francis.

In questo modo, è possibile realizzare una piattaforma indipendente dall'ambiente di utilizzo che può fornire risultati in tempo reale.

Poiché il processo di stima è ricorsivo, il risultato della localizzazione tende ad accumulare un notevole errore man mano che il processo è in esecuzione. Di conseguenza è necessario predisporre un meccanismo di correzione per allineare la stima della posizione con un valore corretto. In letteratura varie soluzioni sono state proposte: ad esempio, sistemi di tipo Zero velocity UPdaTe (ZUPT) prevedono di ridurre parte dell'errore accumulato rilevando l'istante in cui la pianta del piede è a contatto con il terreno (in figura 1.1 l'intero intervallo è indicato con il termine *stance phase*) [20, 21, 24]: in questa situazione, un'eventuale sistema installato direttamente nel piede dovrebbe rilevare una velocità pari a $0\ m/s$. Tuttavia, a causa degli errori del MEMS, un valore diverso da 0 può essere indicato. Avendo a disposizione sia il valore teorico che il valore rilevato dal sensore è possibile sviluppare algoritmi per correggere l'errore accumulato dal MEMS. Invece, altri sistemi fanno uso di tecnologie alternative (e in parte già discusse), come segnali radio [26, 27], RFID [28] e/o ad algoritmi di computer vision che correggono ed integrano i risultati della navigazione PDR.

1.3 Contesto relativo alla piattaforma

I metodi esposti nella sezione precedente rientrano tra i sistemi di supporto secondario che persone BVI possono utilizzare. Sebbene i suddetti sistemi di localizzazione e navigazione siano ancora nella fase di iniziale di sviluppo è evidente che tali tecnologie diventeranno un giorno diffuse e utilizzate come i sistemi GNSS. In questa fase è quindi importare progettare tali strumenti in modo tale da consentire il loro utilizzo anche a persone affette da problemi di vista.

Una prima osservazione inerente a tali sistemi consiste nel constatare i differenti fenomeni che essi rilevano: affinché aumenti la probabilità di una localizzazione corretta, è opportuno vincolare (possibilmente) la posizione del dispositivo, al fine di ottenere una misurazione più precisa. Ad esempio, un sistema di tipo ZUPT fornisce risultati più attendibili se installato sopra il piede dell'utente. Viceversa, un dispositivo che faccia uso di algoritmi di computer vision necessita di essere posizionato in prossimità delle spalle, in modo da avere una chiara visione dell'ambiente di fronte all'utente. Invece, la posizione più favorevole per un apparecchio a radio frequenza potrebbe essere in testa poiché è meno probabile che il segnale radio sia attenuato da altre parti

del corpo. Tuttavia, in accordo con quanto osservato nella sezione 1.1, è fondamentale che tali sistemi siano semplici, sia da installare che da utilizzare e che non costituiscano una qualche forma di impedimento al supporto primario della persona BVI.

Un'altra considerazione riguarda l'installazione del sistema: gli algoritmi di triangolazione e direct sensing necessitano di apparecchi da posizionare nell'ambiente circostante. Tale operazione, oltre ad essere invadente e non sempre possibile sulle strutture in cui l'utente BVI effettuerà i suoi spostamenti, fa aumentare i costi di realizzazione dell'intero sistema. Infatti, sebbene un numero sempre maggiore di dispositivi sia stato introdotto per supportate gli spostamenti di tali persone, questo mercato costituisce ancora una nicchia, che spesso ha costi non accessibili da parte di chi realmente ne ha la necessità.

Infine, i metodi che fanno uso della tecnica del pattern matching richiedono una capacità di storage e di calcolo non indifferente per dispositivi compatti che l'utente deve portare con se. Ad esempio, un algoritmo di pattern matching tra immagini può richiedere un dataset iniziale con dimensione di diverse centinaia di MB (a seconda della risoluzione e dal grado di compressione delle immagini) e, come indicato in [29], l'identificazione in tempo reale (o almeno in tempo ragionevole per l'esigenza di un utente BVI) richiede un hardware sufficientemente potente e moderno.

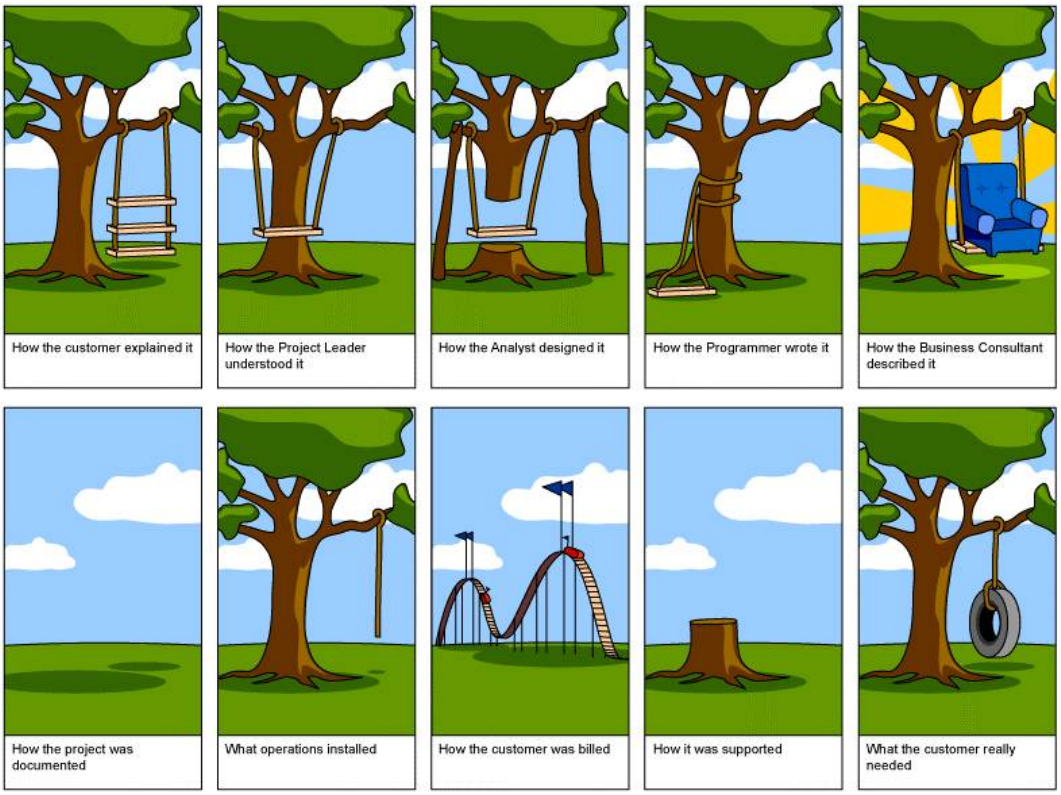
Tuttavia è bene sottolineare che esistono anche apparecchi che si prestano a svolgere un ruolo primario come piattaforma hardware e software per algoritmi di localizzazione e navigazione per persone BVI: lo smartphone è un esempio di tali dispositivi che possiede i sensori e una sufficiente potenza di calcolo per acquisire ed elaborare i dati secondo opportuni algoritmi che guidano e stimano la navigazione di un utente. Un ramo della ricerca scientifica focalizza le proprie energia nella realizzazione (con i dovuti compromessi) degli approcci descritti nella sezione precedente (tra questi si citano [23, 25, 30]). In aggiunta:

- È possibile reperire sul mercato smartphone anche ad un prezzo contenuto (circa 100 €).
- La crescita della loro popolarità fa si che il loro utilizzo da parte di un utente BVI non attragga l'attenzione da parte di altre persone e sia quindi una fonte discriminante.
- In accordo con [27], una parte di soggetti BVI possiede già uno smartphone e quindi ha famigliarità su come si usa il dispositivo.
- Uno smartphone integra gli strumenti adatti (ad es. schermo, motore di vibrazione, cassa audio) per definire un'interfaccia di comunicazione tra il dispositivo e l'utente, in modo tale che quest'ultimo possa comparare l'indicazione ricevuta dall'apparecchio con la percezione che ottiene dal sistema di supporto primario.
- Lo smartphone può essere posto in tasca o in borsa, non occupando le mani dell'utente e non intralciando il sistema di supporto primario.

Nel caso in cui lo smartphone sia il dispositivo utilizzato su cui realizzare l'algoritmo di localizzazione e navigazione, si consideri che, secondo International Data Corporation (IDC) [31], quasi l'85% degli smartphone in commercio nel secondo quadrimestre 2014, utilizzano il sistema operativo Android.

Capitolo 2

Organizzazione del processo software



Fonte: *cartoonshdwallpaper* website. Ultima visita: 3 dicembre 2014.

2.1 Processo software

Il *processo software* è il processo seguito da un'organizzazione per produrre, consegnare ai clienti e far evolvere il prodotto software, dalla nascita dell'idea fino alla consegna e al ritiro del prodotto quando questo è diventato obsoleto [32]. Esso non è una rigida prescrizione per come costruire un'applicazione ma è piuttosto un approccio adattabile che consente al team di sviluppo di

valutare e scegliere l'appropriato lavoro da svolgere; quindi non esiste un processo ideale e più organizzazioni hanno sviluppato i propri processi di sviluppo, sfruttando le capacità dei loro singoli individui e le caratteristiche dei sistemi che devono essere realizzati.

L'*ingegneria del software* è la disciplina *metodologica* che studia i metodi di produzione, le teorie alla base dei metodi e gli strumenti di sviluppo e misura della qualità dei sistemi software; essa è anche una disciplina *empirica*, cioè basata sull'esperienza e sulla storia dei progetti passati [33, 34].

2.1.1 Attività di un processo software

Generalmente un processo software si compone di più attività, cioè di fasi comuni che si applicano indipendentemente dal dominio applicativo, dalla taglia e dalla complessità del progetto o dal grado di rigorosità applicato [35]:

- *Definizione delle specifiche*: determinano i requisiti dell'applicazione costituiti dalle caratteristiche che utente e cliente desiderano che siano presenti nel prodotto software da realizzare. Per caratteristiche del sistema d'interesse si intendono sia gli aspetti *statici* (i dati) che gli aspetti *dinamici* (le operazioni). Nella definizione delle specifiche è necessario descrivere le qualità che devono essere assicurate dall'applicazione, non come queste devono essere progettate e implementate.
- *Progettazione*: è la descrizione della struttura del software da implementare, delle strutture dati, delle interfacce tra i componenti del sistema e degli algoritmi utilizzati.
- *Implementazione*: è il processo che converte una specifica di progetto in un sistema software eseguibile.
- *Validazione*: verifica che la versione eseguibile del sistema sia conforme alle sue specifiche e risponda alle aspettative del cliente.

2.1.2 Soggetti di un processo software

I soggetti di un processo software sono [36]:

- *Cliente* (o committente): è la persona o ente che paga per la fornitura di un prodotto software.
- *Sviluppatore* (o fornitore): è l'individuo o l'organizzazione che produce il software per il cliente.
- *Utente*: è chi interagisce direttamente con il prodotto software e non corrisponde necessariamente al cliente.

2.2 Modello di sviluppo del processo software

Il *modello di sviluppo software* (chiamato anche *modello software*) determina come le attività svolte per lo sviluppo del software sono organizzate e distribuite temporalmente tra loro.

2.2.1 Modello code and fix

Agli inizi dell'informatica, lo sviluppo del software era principalmente un lavoro individuale. Il problema da risolvere era ben noto e non c'era distinzione tra il programmatore e l'utente finale, che spesso era uno scienziato o un ingegnere che sviluppava un'applicazione come supporto alla sua attività principale. Il modello usato in quei giorni è chiamato *code and fix*: il processo di sviluppo non è né formulato precisamente né controllato attentamente. Infatti la produzione

del software consiste nell'iterazione della scrittura del codice e della successiva correzione per migliorarne la funzionalità o per aggiungere nuove caratteristiche.

Purtroppo il modello code and fix è stato causa di molte difficoltà e carenze:

- Dopo una serie di cambiamenti, la struttura del codice diventava disorganizzata rendendo le modifiche successive più difficili da apportare; tuttavia, i problemi di questa natura erano mitigati dal fatto che si trattava di applicazioni piuttosto semplici e sia l'applicazione che il software erano ben conosciuti dall'ingegnere. Purtroppo però, le dimensioni sempre maggiori dei sistemi da sviluppare rendevano difficile la gestione non strutturata della loro complessità.
- L'aggiunta di nuovo personale a un progetto in corsa era estremamente difficile per via della mancanza di documentazione necessaria a capire in maniera corretta l'applicazione.
- Aggiustare il codice diventava difficile in quanto non veniva presa in considerazione (prima di cominciare la stesura del codice) alcuna anticipazione dei cambiamenti che avrebbero potuto diventare necessari. Per lo stesso motivo diventava complesso rimuovere gli errori che richiedevano una ristrutturazione del codice esistente.
- Terminato lo sviluppo del sistema, ci si rendeva conto che spesso il software non soddisfaceva le aspettative dei clienti; il prodotto veniva rifiutato o doveva essere sviluppato nuovamente per ottenere gli obiettivi desiderati. Il risultato conduceva al fatto che il processo di sviluppo non poteva essere predicibile, diventava difficile da controllare, i prodotti venivano completati oltre i tempi stabiliti, spendendo più del budget disponibile e senza soddisfare le aspettative in termini di qualità.

Questi problemi sottolineano la necessità di una fase di pianificazione prima di cominciare a scrivere il codice. Se si segue un processo esplicito, lo sviluppo del software procede in maniera prevedibile e ordinata, riducendo la possibilità di introdurre errori nel prodotto e fornendo un modo per controllare la qualità di quanto si sta producendo.

2.2.2 Modello a cascata

Il modello a *cascata* (talvolta indicato con *ciclo di vita* del software) [37] suggerisce un approccio sequenziale e sistematico dello sviluppo del software che inizia con la definizione dei requisiti specificati dal cliente, prosegue attraverso la progettazione, l'implementazione e la validazione del software, terminando con la sua manutenzione (supporto). Il modello a cascata si basa sull'assunzione che lo sviluppo del software proceda linearmente dall'analisi alla produzione del codice. Nella pratica ciò difficilmente accade ed è necessario prevedere forme disciplinate di cicli di feedback. Esiste, quindi, una sovrapposizione tra fasi adiacenti in cui, entro un certo numero di iterazioni, avviene uno scambio di informazioni e si termina definitivamente la fase precedente, in modo tale da continuare lo sviluppo successivo. Tuttavia è bene notare che l'iterazione tra più fasi è un'operazione costosa, poiché richiede di ingegnerizzare nuovamente del lavoro già svolto ed inevitabilmente conduce alcuni membri del progetto ad aspettare che i tasks da cui il loro lavoro dipende siano terminati.

Il modello ha fornito due importanti contributi alla comprensione dei processi software:

- Il processo di sviluppo del software deve essere soggetto a disciplina, pianificazione e gestione.
- L'implementazione del prodotto dovrebbe essere rimandata fino a quando non sono perfettamente chiari gli obiettivi.

Sebbene il modello a cascata sia abbastanza semplice da seguire, presenta una serie svantaggi:

- La specifica dei requisiti produce un documento scritto che guida e vincola il prodotto da sviluppare. Indipendentemente dalla precisione della descrizione, è molto difficile per un utente anticipare se un sistema costruito secondo le specifiche soddisferà le sue aspettative. L'utente stesso potrebbe non conoscere i requisiti esatti dell'applicazione e gli stessi potrebbero essere in continua evoluzione.

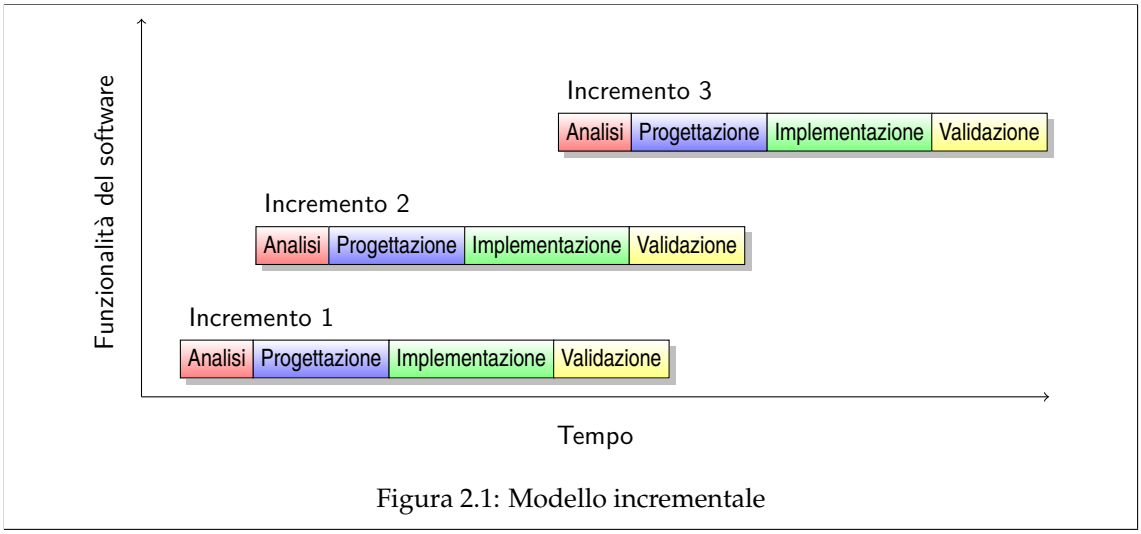


Figura 2.1: Modello incrementale

- Il modello a cascata non sottolinea sufficientemente il bisogno di anticipare possibili cambiamenti: la difficoltà di produrre specifiche dei requisiti complete e corrette è la causa di molti interventi successivi di manutenzione. Se eventuali errori commessi in questa fase non sono subito scoperti, verranno identificati solo dopo che il sistema è stato rilasciato ai clienti. Quindi il modello a cascata non permette una visibilità sufficiente del prodotto in sviluppo fino a quando il prodotto completo non è pienamente implementato.
- Il modello obbliga a standard pesantemente basati sulla produzione di una data documentazione in determinati momenti.

In principio, il modello a cascata dovrebbe essere utilizzato solamente quando i requisiti dell'applicazione sono noti e ben compresi; tuttavia, esso riflette il tipo di processo usato in altri progetti di ingegnerizzazione. Poiché in molte organizzazioni si preferisce l'utilizzo di un unico modello comune per l'intero progetto, l'utilizzo del modello a cascata è frequente.

2.2.3 Modello incrementale

Il modello *incrementale* consiste in un processo di sviluppo in cui alcune porzioni del processo stesso possono essere rimandate in modo da produrre prima possibile un insieme utile di funzionalità [38]. Il sistema è sviluppato come una serie di versioni tali che ognuna di essa aggiunge funzionalità alle precedenti. Ogni versione, chiamata anche incremento, è un'unità software funzionale autocontenuta che esegue una qualche funzione utile al cliente e che viene rilasciata accompagnata da un adeguato materiale di supporto (ad es. la versione 1.4.3 del software o la versione 0.8.4 beta).

Con il modello incrementale le attività del processo sono sovrapposte anziché sequenziali: il processo quindi è costituito da un sottoinsieme di "mini-processi", ognuno caratterizzato dalla proprio sequenza di attività (cioè definizione delle specifiche, progettazione, implementazione e validazione) che si occupano di risolvere problematiche più piccole (figura 2.1). Tale approccio riflette il modo con cui l'uomo risolve i problemi: difficilmente si è in grado di trovare una soluzione completa ed esaustiva del problema in prima analisi ma piuttosto la si ricava tramite successivi raffinamenti, riportando cosa è stato realizzato e come correggere gli errori.

Spesso, il modello incrementale costituisce anche un modello *evolutivo* [39]: le versioni incrementali sono rilasciate man mano che sono sviluppate, si misura il valore aggiunto per il cliente secondo tutte le dimensioni critiche e si aggiusta sia il progetto che gli obiettivi, in base a quanto osservato.

Il modello incrementale introduce molteplici vantaggi:

- Poiché ogni incremento è più semplice dell'intero sistema, è più facile prevedere le risorse necessarie allo sviluppo con un'approssimazione migliore.
- I cambiamenti possono essere gestiti facilmente (anche nella documentazione poichè interessano una porzione limitata del processo) a differenza di quanto avviene per il modello a cascata, in cui si manifestano, come un'attività di post-sviluppo, sotto forma di costosa manutenzione.
- Si può far uso di *prototipi*, cioè di una versione iniziale del sistema software che è utilizzata per dimostrare i concetti, testare nuove soluzioni e mettere in risalto ulteriori problemi. Successivamente il prototipo viene progressivamente trasformato nell'applicazione finale tramite incrementi.
- Il progetto può essere pianificato per limitare i rischi tecnici. Ad esempio, se un sistema richiede l'utilizzo di un nuovo hardware la cui data di consegna è incerta, è possibile pianificare in anticipo gli incrementi in modo tale da posticipare l'uso di quell'hardware il più possibile.

Tuttavia, il modello incrementale presenta anche alcuni svantaggi:

- Il processo è "non visibile", cioè richiede un costante controllo per valutare il suo avanzamento.
- Se il sistema è sviluppato molto rapidamente, il costo per produrre la documentazione che riflette ogni versione del sistema potrebbe superare il costo di sviluppo dello stesso.
- Man mano che più incrementi sono sviluppati, la struttura del sistema tende a degradare, a meno di costose operazioni di ri-ingegnerizzazione. Tale problema accresce notevolmente per ampi progetti in cui differenti team sviluppano differenti parti del sistema. In questo scenario è opportuno definire in anticipo un'architettura del progetto comune per tutte le parti e stabilire le responsabilità dei differenti team di sviluppo su ciascun parte da realizzare.

2.2.4 Scelta del modello software

> *In theory, theory and practice are the same. In practice, they are not.*
>
> *A. Einstein*

> *In theory there is no difference between theory and practice. In practice there is.*
>
> *Y. Berra*

Ricordando che non esiste una rigida prescrizione nel seguire un determinato modello di sviluppo software, per questa tesi si è scelto di adottare una variante del modello di sviluppo incrementale. Nel dettaglio si procederà come segue:

1. Si descrive a grandi linee il problema, individuando i principali obiettivi dell'applicazione.
2. Si definisce un'architettura modulare del software che consenta una successiva definizione delle specifiche, progettazione, implementazione e validazione di ogni singolo *modulo*. Con questo termine si intende un generico fornitore di risorse computazionali o di servizi.
3. Si impongono dei vincoli che dovranno essere soddisfatti dall'implementazione di tutti i moduli.
4. Si stabiliscono le modalità generali di validazione di ogni singolo modulo.
5. Si pianifica l'ordine di sviluppo dei singoli moduli.

6. Si procede allo sviluppo dei singoli moduli (in analogia a quanto avviene nel modello incrementale).
7. Si produce la documentazione dell'applicazione.

Le motivazioni che hanno condotto a questa scelta sono molteplici:

- Natura sperimentale della tesi, che interessa:
  - L'incompletezza dei requisiti dell'applicazione: ad esempio, non è noto a priori l'accuratezza necessaria per la navigazione.
  - La fattibilità dei requisiti dell'applicazione: ad esempio, è in grado l'applicazione di rilevare un singolo passo in tempo reale?
  - L'adattabilità richiesta: un nuovo "incremento" può rispondere in maniera celere alla richiesta del cliente o al sorgere di un nuovo problema non previsto.
- Consente di manipolare anche progetti incompleti in quanto alcuni moduli del sistema possono essere sostituiti da *stub*[1]:
  - Consente di svolgere procedure di validazione sul modulo realizzato.
  - È adatto ad un lavoro di tesi ("prima o poi lo studente dovrà laurearsi…").
- Scomposizione intuitiva del problema poiché è possibile realizzare un modulo per la risoluzione di ogni sotto-problema derivato dalla scomposizione.

2.2.5 Filosofia

Manifesto per Agile Software Development [40]:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> *Individuals and interactions over processes and tools*
> *Working software over comprehensive documentation*
> *Customer collaboration over contract negotiation*
> *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*

Il modello di sviluppo software può avere un approccio plan-drive, agile o una combinazione dei due. Nell'approccio *plan-driven*, tutti gli incrementi sono pianificati in anticipo nella prima fase del processo e i progressi sono misurati direttamente nella realizzazione del piano. Nell'approccio *agile*, solamente i primi incrementi sono pianificati nella prima fase del processo mentre i successivi dipendo dal progresso dell'applicazione stessa o dalle nuove priorità del cliente. Generalmente, è necessario adottare un bilanciamento della componente plan-driven rispetto alla componente agile di un processo.

In questo progetto si è cercato di privilegiare l'approccio agile: ogni aspetto del problema e della soluzione è stato esaminato con il committente (relatore) e di volta in volta si è valutata la bontà del lavoro svolto. Inoltre, si è posta particolare attenzione alla progettazione in vista del cambiamento: si è cercato di anticipare i cambiamenti a cui il software potrà essere sottoposto durante il suo ciclo di vita e, di conseguenza, produrre un progetto software facilmente adattabile a quei cambiamenti.

[1] "*Stubs are computer programs that act as temporary replacement for a called module and give the same output as the actual product or software.*" Fonte: Wikipedia. Ultima visita: 3 dicembre 2014.

2.3 Descrizione del problema

Si vuole progettare un'applicazione Android che faciliti gli spostamenti in ambienti indoor di utenti non vedenti o, più in generale, di persone con difficoltà visive. Una volta installata su smartphone, il software deve monitore il percorso dell'utilizzatore attraverso un contapassi e una bussola digitale, realizzate entrambe con la tecnologia disponibile sul dispositivo Android. Durante questa fase, il software deve essere in grado di costruire una mappa del layout interno di una stanza o di un edificio mentre l'utente cammina da un punto di partenza ad un punto d'arrivo. Tale mappa deve essere salvata su una qualche forma di memoria permanente ed, eventualmente, deve essere realizzato un meccanismo di backup remoto.

Successivamente, è prevista la possibilità, da parte dell'utente, di caricare la mappa corrispondente a delle precise coordinate GPS fornite dallo smartphone ed iniziare una qualche forma di navigazione: in questa fase, l'applicazione deve guidare l'utilizzatore dal punto iniziale al punto d'arrivo della mappa, determinando l'attuale posizione dell'utente attraverso le informazioni del contapassi. L'interfaccia di comunicazione tra utilizzatore e smartphone deve far uso di messaggi acustici e/o sequenze di vibrazioni dello smartphone.

2.3.1 Requisiti software

Di seguito, si individuano i principali requisiti che l'applicazione dovrà soddisfare. L'IEEE definisce requisito software [33]:

1. *A condition or capability needed by a user to solve a problem or achieve an objective.*
2. *A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.*
3. *A documented representation of a condition or capability as in 1 or 2.*

La descrizione del problema e dei requisiti software può presentare problemi di:

- *Ambiguità*: i requisiti sono interpretabili in modo differente dai soggetti (ad es. da cliente e sviluppatore), determinando un certo grado di ambiguità degli stessi. In alcuni casi, il requisito è una semplice descrizione astratta ad alto livello che indica i servizi o i vincoli del sistema, mentre in altre circostanze è una definizione rigorosamente formale delle funzionalità del sistema stesso.
- *Incompletezza*, poiché non includono la descrizione di tutte le caratteristiche richieste.
- *Inconsistenza*: si possono generare conflitti o contraddizioni nella descrizione delle caratteristiche del sistema.
- *Verificabilità*: i requisiti espressi in modo generico dall'utente (ad es. il sistema software deve essere "easy-to-use") possono risultare non quantificabili e difficili da verificare. È quindi necessario esprimere tali requisiti usando una misura determinata che permetta di verificare quantitativamente se il requisito verrà soddisfatto dal sistema software (ad es. ore di formazione dell'utente per l'apprendimento del software).

In questa prima analisi si individuano i principali requisiti software con un alto grado di astrazione e ambiguità: infatti lo scopo è quello di determinare gli obiettivi principali dell'applicazione. Nello sviluppo del software si procederà a determinare con più accuratezza le specifiche di ogni singola parte dell'applicazione.

Requisiti di sistema

L'applicazione dovrà essere compatibile per smartphone con sistema *Android*. Pertanto l'ambiente di sviluppo è vincolato e le risorse hardware a disposizione sono limitate in termini di:

- Potenza di calcolo.

- Capacità della memoria secondaria.
- Ridotto numero di sensori (non espandibile) e con limitate frequenze di campionamento.

Per tali ragioni, l'applicazione da realizzare costituisce di fatto un sistema di navigazione ausiliario (secondario) che non intende sostituire il sistema di navigazione primario, costituito, ad esempio, da un cane-guida addestrato per l'accompagnamento di persone non vedenti.

Requisiti utente

È espressamente indicato che l'utilizzatore dell'applicazione è un utente non vedente o, in termini più generali, un ipovedente. È necessario progettare:

- Una *Graphic User Interface (GUI)* ad hoc per visualizzare lo stato dell'applicazione e che consenta l'inserimento di eventuali parametri dell'algoritmo. Ad esempio, una possibile GUI potrebbe far uso di grandi elementi colorati per rivelare l'esecuzione dell'applicazione e/o di grandi pulsanti per incrementare o decrementare i valori di predeterminati parametri.
- Una *Acoustic User Interface (AUI)* e una *Vibration User Interface (VUI)* di comunicazione che consentano, tramite apposite sequenze di segnali acustici o di vibrazione dello smartphone, di indicare all'utente la direzione verso cui camminare per raggiungere la destinazione. Si noti che, ad differenza di una GUI, tali interfacce consentono all'applicazione di comunicare con l'utente e non viceversa.

Requisiti funzionali

Essi descrivono le funzionalità del software, in termini di servizi che il sistema deve fornire, di come reagisce a specifici tipi di input e di come si comporta in situazioni particolari. Dall'analisi della descrizione dell'applicazione sono emersi requisiti funzionali che interessano concetti diversi:

- *Contapassi e bussola digitale*: hanno lo scopo di misurare rispettivamente la distanza percorsa in funzione del numero di passi e dell'orientazione del cammino rispetto al nord magnetico terreste. Non sono noti a priori l'accuratezza e il tempo di risposta necessari per la determinazione di tali valori.
- *Mappa*: ha la funzione di tener traccia dell'ambiente in cui l'utente ha effettuato i precedenti percorsi. In particolare è necessario assicurare:
  - La *creazione* di una mappa, la quale avviene in accordo con le seguenti direttive:
    1. Le coordinate GPS individuano il punto di partenza all'interno di una mappa inizialmente vuota.
    2. L'utente si sposta, il contapassi calcola la lunghezza del percorso effettuato mentre la bussola digitale determina l'orientazione del cammino.
    3. L'applicazione crea o aggiorna la mappa in funzione dell'indicazione del percorso fornita dal contapassi e dalla bussola digitale. È necessario determinare come quantificare lo spostamento (ad es. in metri o in numero di passi).
    4. La costruzione della mappa termina quando l'utente arriva al suo obiettivo (punto d'arrivo, chiamato anche *goal*) o interrompe il task.
  - Il *salvataggio* e il *caricamento* (tramite coordinate GPS) della mappa.
- *Navigazione*: si occupa di guidare l'utente da un punto ad un altro della mappa. In particolare si procede a:
  - *Calcolare un percorso valido*. Si ipotizza che ad ogni mappa sia associato un goal, che costituisce il punto d'arrivo. La funzione da svolgere consiste nel progettare un algoritmo che determini un percorso libero da collisioni tra il punto di partenza (dato dalle coordinate GPS) e il goal.

  - *Aggiornare il percorso* in funzione della posizione attuale dell'utente. Infatti, potrebbero sorgere delle discrepanze tra le indicazioni fornite dal contapassi e dalla bussola digitale rispetto a cosa effettivamente è stato eseguito dall'utente; in questa situazione è necessario determinare eventuali correzioni nella traiettoria da seguire per raggiungere il goal.

Come descritto nella sezione 1.2, un'applicazione di questo tipo realizza un sistema di navigazione PDR. Di conseguenza è ragionevole ipotizzare che nel seguito del progetto si dovranno affrontare *challanges* legate agli errori di misurazione e alla bassa accuratezza offerta dai sensori (di costo ridotto) presenti negli smartphone, in aggiunta alla variabilità hardware presente nell'ampio mercato di dispositivi Android.

2.4 Considerazioni sulla progettazione

L'obiettivo di questa fase consiste nel determinare i principali blocchi dell'applicazione e le loro interazioni (cioè l'architettura software data dalla struttura organizzativa del sistema) con un alto livello di astrazione; i dettagli del progetto saranno tralasciati allo sviluppo delle singole parti dell'applicazione. In genere, non esiste "una ricetta": la progettazione è un processo creativo, in cui non è spesso possibile procedere in modo sequenziale e predeterminato. Si deve valutare un'idea, proporre una soluzione e rifinirla man mano che nuova informazione è disponibile; inevitabilmente, però, si sarà costretti a rivalutare del lavoro già svolto. Tuttavia anche l'esperienza aiuta nell'esecuzione di questa fase poiché la conoscenza di architetture già testate in sistemi precedenti consente al progettista di intraprendere il progetto in modo rapido e con maggiore fiducia.

2.4.1 Individuazione dei principali moduli dell'applicazione

Si è scelto di adottare l'approccio modulare poiché introduce diversi vantaggi:

- Si adatta al modello incrementale: è possibile progettare e implementare modulo per modulo, facilitando il rilascio incrementale del software.
- *Separazione degli interessi*: la modularizzazione consente di scomporre il problema in sottoproblemi, ognuno dei quali analizzato singolarmente.
- *Information hiding*: l'insieme dei servizi offerti da ogni modulo ai suoi client (utilizzatori) viene detto *interfaccia*. Essi conoscono i suoi servizi solamente attraverso l'interfaccia: l'implementazione risulta essere così nascosta e potrebbe cambiare senza conseguenze per i client (a patto che l'interfaccia rimanga immutata).
- Manutenibilità: risulta più semplice poiché interessa solo il modulo corrente.
- Riusabilità: un modulo può essere riusato in qualsiasi contesto, a patto che i servizi elencati nella sua interfaccia soddisfino le aspettative dei client, indipendentemente dall'implementazione.

Tuttavia è bene sottolineare anche gli svantaggi di un approccio modulare:

- Minor ottimizzazione, poichè un singolo modulo non ha la visione completa di ciò che avviene in altri e potrebbe, ad esempio, svolgere del lavoro già fatto.
- Concetti poco modulari richiedono decisioni globali in prima battuta, le quali spesso non sono ben rilevabili.

I principali moduli individuati derivano direttamente dell'analisi dei requisiti:

- *Interfaccia utente*: ha il compito di ricevere l'input dall'utente e comunicare la direzione degli spostamenti.

- *Contapassi*: utilizza i sensori (accelerometro e magnetometro) per determinare la direzione e il numero di passi effettuati.
- *Navigazione*: determina la successiva direzione che l'utente dovrà intraprendere.
- *Creazione mappa*: costruisce una mappa ricevendo le informazioni dal contapassi.
- *Dati*: salva e carica le mappe dell'applicazione.

2.4.2 Scelta dell'architettura software

Si è scelto di utilizzare un'architettura software a *3 livelli* [41]. In un sistema di questo tipo, le funzionalità sono organizzate in tre livelli "logici" separati, ognuno dei quali dipende solamente dai servizi offerti dai livelli adiacenti.

Si ha quindi:

- Un livello per l'*interfaccia utente*, responsabile della comunicazione tra utente e applicazione.
- Un livello *applicativo* (detto anche *business logic*) che interpreta le richieste dell'utente, determina cosa deve essere fatto ed esegue il servizio richiesto.
- Un livello *dati*, che si occupa della persistenza dei dati dell'applicazione.

In accordo con l'architettura a 3 livelli, si procede ad "assegnare" i singoli moduli a ciascun livello e a determinare quali comunicazioni devono avvenire tra moduli (figura 2.2).

Si noti la presenza del modulo *Core* che ha il compito di garantire la comunicazione tra i moduli e di interfacciare correttamente l'applicazione con il sistema Android (ad es. l'applicazione dovrà rimanere in esecuzione anche nel caso in cui sia attivata la funzione "blocca schermo").

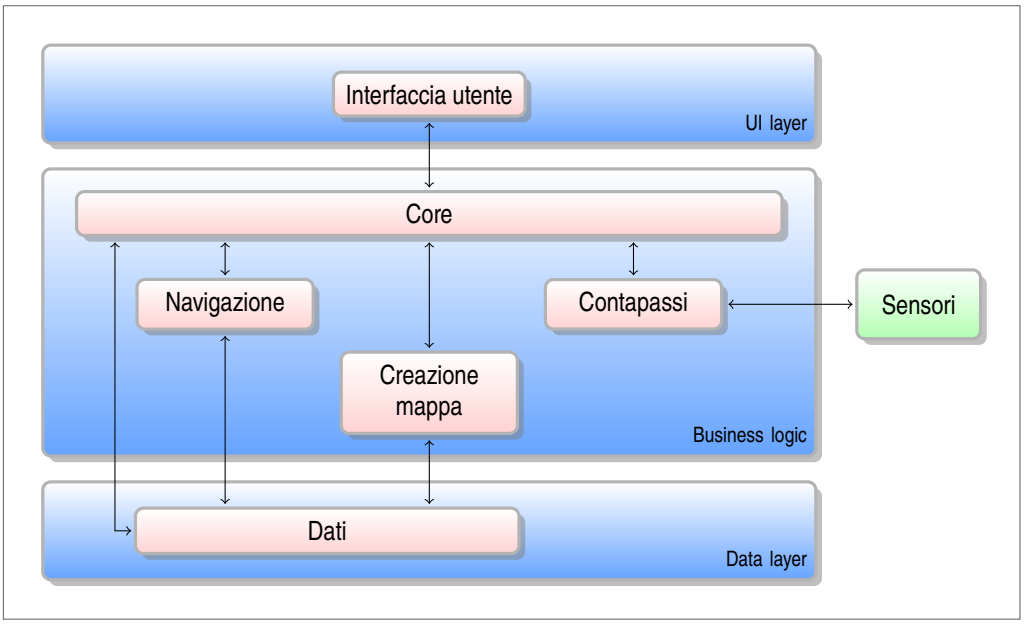


Figura 2.2: Architettura software a tre livelli dell'applicazione. In verde è indicato l'API per accedere ai sensori, la quale fa parte del sistema Android e non di questa applicazione.

Di seguito si descrivono alcune considerazioni generali sulla rappresentazione dell'architettura software:

- Questa rappresentazione è caratterizzata da un alto grado di astrazione e assenza di formalismo. Ha solo lo scopo di individuare:

  - I componenti principali del sistema che saranno poi sviluppati.
  - Quali sono i moduli che "comunicano" tra loro.

- È indicato la presenza di "comunicazione" tra moduli, senza specificare le interfacce e i dati scambiati.
- Ogni modulo può essere scomposto in più componenti minori, ognuno delle quali con una funziona specifica.
- Non è imposto nessun vincolo sullo sviluppo del modulo (ad es. si valuterà in fase di progetto se il modulo *Dati* dovrà far uso di un database o salvare direttamente su file).

La motivazione che ha spinto alla scelta di questa architettura sono:

- La logica dei moduli dell'applicazione ricalca in modo intuitivo la logica dei singoli livelli dell'architettura. Ciò rende più esplicito la separazione d'interessi tra i moduli. Ad esempio, se nel modulo *Navigazione* esiste una funzione che legge una parte di mappa direttamente dalla memoria secondaria allora è opportuno implementare tale funzione nel modulo *Dati*.
- Estensibilità. Se un modulo viene sostituito da un altro con le medesime funzioni ma con implementazione diversa, la restante parte dell'applicazione non dovrà essere modificata. Ad esempio, un modulo *Dati* che salva direttamente su file potrebbe essere sostituito o affiancato da un nuovo modulo *Dati* che faccia uso di un database.
- Distribuzione del sistema. Se un livello diventa troppo oneroso in termini di risorse, può essere distribuito su un altro terminale realizzando un vero e proprio sistema distribuito. Ad esempio, se il livello dati salva un quantitativo di dati troppo elevato per la memoria dello smartphone, si può realizzare un'altra applicazione su pc (il quale è dotato di memoria secondaria con maggiore capacità) che ha il compito di fornire il servizio originale del livello dati dell'applicazione per smartphone.

2.5 Considerazioni sull'implementazione

2.5.1 Framework di sviluppo

L'intero progetto è svolto utilizzando Android Software Development Kit (SDK), il quale fornisce un insieme di Application Programming Interface (API) e tool specifici per lo sviluppo di applicazioni Android utilizzando il linguaggio Java (versione 1.6)[2]. Tra questi si ricorda:

- Android *SDK tools*[3]: fornisce un insieme di strumenti, indipendenti dalla piattaforma (cioè dalla versione di Android su cui è installata l'applicazione da sviluppare), che consentono la creazione e la gestione di emulatori Android, il debug e l'esecuzione di test. *SQLite3*, *Dalvik Debug Monitor Server (ddms)* e *android* sono esempi di software inclusi in Android *SDK tools*.
- Android *SDK builds*[4]: è un insieme di strumenti, dipendenti dalla piattaforma, necessari per la compilazione del codice. Spesso sono integrati all'interno di un Integrated Development Environment (IDE).
- Android *SDK platform*[5]: è un altro insieme di strumenti, dipendenti dalla piattaforma, che consentono di monitorare gli emulatori e i dispositivi Android collegati. Il software *adb* appartiene a questa categoria.
- Una o più *SDK platforms* che consentono di compilare il codice per una specifica versione di Android.

[2] Android supporta un sottoinsieme di Java 1.6 Standard Edition e fornisce un insieme di librerie aggiuntive di terze parti (ad es. `org.apache.http` o `org.json`).
[3] Android SDK tools. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.
[4] Android SDK builds. Fonte: documentazione Android. Link 1 e link 2. Ultima visita: 3 dicembre 2014.
[5] Android SDK platform. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.

- Una o più *Android system image* che consentono di eseguire nell'emulatore (di una specifica versione di Android) il codice sviluppato.
- *SDK manager* che consente di installare nuovi componenti ad Android *SDK* (che di default non è installato con tutti i suoi pacchetti).

Sebbene sia anche possibile sviluppare un'applicazione Android con il Native Development Kit (NDK) in linguaggio C/C++, tale approccio è sconsigliato poiché:

- *"Using native code on Android generally does not result in a noticable performance improvement, but it always increases your app complexity. In general, you should only use the NDK if it is essential to your app—never because you simply prefer to program in C/C++."*[6]
- Mancanza di alcune API (ad es. per `Service` e `ContentProvider`).

Tuttavia, nel caso in cui sia necessario sfruttare particolari istruzioni offerte dall'architettura hardware dello smartphone, come ad esempio le istruzioni Single Instruction, Multiple Data (SIMD), non si esclude la possibilità di realizzare una o più funzioni con Android NDK e successivamente invocarle da Android SDK tramite l'interfaccia Java Native Interface (JNI). Questa tecnica permette di mantenere i vantaggi di Android SDK e di scrivere codice nativo quando necessario.

2.5.2 IDE di sviluppo

La comunità di Android developers consiglia l'utilizzo di uno tra i seguenti Integrated Development Environment (IDE)[7]:

- *Eclipse* con l'Android Developer Tools (ADT) plugin.
- *Android Studio*, sviluppato a partire dall'IDE *IntelliJ*.

Entrambi i prodotti sono "pronti per l'uso" poiché includono la versione più aggiornata di Android SDK: dopo aver installato una versione di Java Development Kit (JDK) maggiore di 1.6 è sufficiente scaricare l'archivio zip dell'IDE scelto, decomprimerlo ed avviare l'eseguibile.

Per questa tesi, si è scelto di utilizzare Android Studio poiché:

- *"...will be the official Android IDE once it's ready."*[8]
- È un ambiente moderno e in rapida crescita: è nato nel maggio 2013 ed ha già ricevuto 8 minor release di aggiornamento[9].

In aggiunta, si è utilizzato un software Version Control System (VCS) per il controllo del versionamento del codice. Nel caso specifico si è utilizzato il tool *Git*[10].

2.5.3 Versione della piattaforma Android

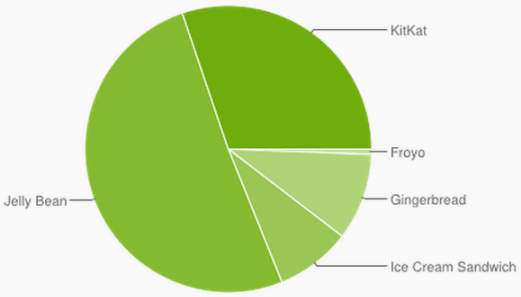
Esistono numerose versione di Android in commercio, ognuna delle quali fornisce una specifica API. È necessario stabilire quale sia minima versione supportata dall'applicazione poiché essa determina il numero di dispositivi compatibili con l'applicazione. La condizione ideale è supportare tutte le versioni di Android, cioè sviluppare l'applicazione con l'API 1.

Per questo progetto, si adotta la seguente strategia:

1. L'implementazione di ogni modulo "si impegna" a utilizzare la minima versione di API; chiaramente questo è solo un "consiglio" e non un vincolo.

[6] Android NDK. Fonte: documentazione Android. Link. Ultima visita: December 3, 2014.
[7] IDE per Android. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.
[8] Android Studio. Fonte: documentazione Android. Link. Ultima visita: December 3, 2014.
[9] Android Studio releases. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.
[10] *Git*. Fonte: *Git* homepage. Link. Ultima visita: 3 dicembre 2014.

Version	Codename	API	Distribution
2.2	Froyo	8	0.6%
2.3.3 - 2.3.7	Gingerbread	10	9.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	8.5%
4.1.x	Jelly Bean	16	22.8%
4.2.x		17	20.8%
4.3		18	7.3%
4.4	KitKat	19	30.2%



Data collected during a 7-day period ending on November 3, 2014.
Any versions with less than 0.1% distribution are not shown.

Figura 2.3: Android market share. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.

2. Al termine dell'implementazione di un singolo modulo, si comunica la versione minima dell'API richiesta. Si impone il vincolo che tale versione debba essere minore o uguale all'API 15.

3. L'applicazione sarà compatibile con la minima versione dell'API tra quelle esposte dai moduli. Nel caso peggiore, l'applicazione sarà compatibile con l'API 15, assicurando una copertura di oltre l'85% dei dispositivi Android (figura 2.3).

2.6 Considerazioni sulla validazione

Questa fase include le attività di:

- *Verifica*, che controlla se il software implementa correttamente la specifica per cui è stato progettato.
- *Validazione*, la quale assicura che l'applicazione soddisfi le necessità del cliente.

Per entrambe le attività si eseguono dei test sperimentali; tuttavia è bene ricordare che questo approccio può essere usato per dimostrare la presenza di malfunzionamenti, non per dimostrare la loro assenza poiché, per ovvie ragioni, non è possibile eseguire test su tutto il dominio di input dell'applicazione.

Si è scelto di eseguire i test per:

- Ogni singolo modulo. Se previsto, si effettuano:
  - *Test di sovraccarico*, in cui il sistema è testato in condizioni di picco (ad es. quando l'applicazione deve manipolare grandi mappe).
  - *Casi d'uso*, che rappresentano una simulazione del comportamento dell'applicazione.
  - *Test di robustezza*, in cui il sistema è sottoposto a condizioni non previste (ad es. quando la navigazione "sconfina" al di fuori della mappa).
  - *Test di prestazioni*, in cui si valuta il tempo d'esecuzione del sistema in particolari condizioni.
- Test complessivo del sistema tramite semplici casi d'uso (ad es. la creazione di una mappa mentre l'utente cammina in una stanza vuota seguendo il perimetro di un rettangolo).

2.6.1 Ambiente di test

I test sono eseguiti con l'utilizzo del framework JUnit[11], già incluso in Android *SDK*. Si è fatto uso di due piattaforme diverse per il test:

- L'emulatore fornito con Android SDK. La configurazione dell'emulatore prevede:
  - Un'immagine Android 4.0 con architettura x86.
  - 512 MB di memoria RAM.
  - Accelerazione grafica abilitata.
  - L'esecuzione dell'applicazione su Dalkin virtual machine.

  L'emulatore è eseguito su un pc Windows 8.1 con processore Intel i5 con l'accelerazione hardware per la virtualizzazione attivata[12].
- Lo smartphone Android Motorola Moto G (1° generazione), con la seguente configurazione:
  - Android 4.4.4 con architettura ARMv7-A.
  - 1 GB di RAM.
  - L'esecuzione dell'applicazione su ART virtual machine.

2.6.2 Ostacoli della validazione

Si sottolinea fin da subito che potrebbe essere non possibile procedere alla validazione di ciascun modulo singolarmente. Ad esempio, per poter funzionare correttamente il modulo *Contapassi* necessita dei dati forniti dai sensori. Con l'emulatore Android utilizzato non è possibile simulare nativamente la generazione di campioni da parte dei sensori. L'implementazione di tale funzione esternamente a tale progetto richiede una mole di lavoro non indifferente. Per tale motivo si è scelto di posticipare la validazione dei moduli *Contapassi* e *Core* al momento in cui entrambi sono stati implementati: in questo modo è possibile validare l'applicazione direttamente nella piattaforma hardware di test.

In aggiunta, anche il grado di dettaglio con cui la validazione è effettuata varia da modulo a modulo. In linea di principio, maggiore è il grado di interdipendenza di un modulo rispetto agli altri, maggiore è il grado di dettaglio con cui la validazione può avvenire. Ad esempio, il modulo *Dati* presenta in modo naturale un bassa interdipendenza con gli altri moduli. A supporto di tale tesi, si evidenzia che il modulo opera solo con i dati forniti, senza preoccuparsi della sua origine: infatti un'eventuale migrazione dalla gestione dell'oggetto *mappa* alla gestione di un altro tipo di oggetto non necessita di una re-ingegnerizzazione di tutto il modulo. Per tale motivo, risulta più semplice progettare un test di validazione che proceda a stimolare e validare i risultati di tutte le funzioni del modulo in modo programmatico, senza che esso sia integrato con il resto dell'applicazione. In questo modo è possibile eseguire test più esaustivi e completi. Viceversa, come indicato in precedenza, la validazione del modulo *Contapassi* non può avvenire in modo isolato poiché necessita del modulo *Core*. Di conseguenza, eventuali interferenze ed errori dei due moduli possono combinarsi tra loro diventando molto difficili da rilevare (ad es. è frequente la situazione in cui il contributo di due errori genera un risultato apparentemente esatto).

Infine si sottolinea che l'accuratezza della validazione di alcuni moduli, intesa come la corrispondenza tra il valore sperimentale ottenuto e il valore teorico atteso, è soggetta ad un diverso grado di affidabilità. Ad esempio, la validazione delle eventuali operazioni di salvataggio e caricamento di una mappa per il modulo *Dati* può avvenire programmaticamente (cioè tramite un apposito programma realizzato su misura) confrontando elemento per elemento se la mappa salvata e successivamente caricata dalla memoria persistente è "equivalente" (cioè contiene lo stesso contenuto informativo) alla mappa originale. In questo caso, poiché il programma di validazione risulta molto semplice e quindi con una ridotta probabilità che esso sia errato, il grado di affidabilità dell'accuratezza della validazione risulta elevato. Viceversa, se la validazione richiede il

[11] *JUnit*. Fonte: *JUnit* homepage. Link. Ultima visita: 3 dicembre 2014.
[12] Richiede l'installazione di Intel®HAXM. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.

controllo del risultato di un'elaborazione complessa, l'eventuale programma o tool esterno adottato per la validazione stessa potrebbe essere, a sua volta, soggetto ad errori. Ad esempio, se il risultato atteso del modulo *Creazione mappa* è calcolato tramite uno strumento esterno (ad es. un foglio elettronico), l'oneroso contributo manuale necessario per questa operazione può essere soggetto ad un numero di eventuali errori maggiore rispetto al programma stesso da validare. In quest'ultimo caso, il grado di affidabilità dell'accuratezza risulta ridotto.

2.7 Pianificazione

L'implementazione del progetto segue il seguente ordine di sviluppo dei moduli:

1. *Dati*.
2. *Creazione mappa*.
3. *Contapassi*.
4. *Core*.
5. *Navigazione*.
6. *Interfaccia utente*.

La scelta di tale ordine dipende dalle attitudini del tesista (specializzazione informatica). Con il procedere del progetto, si valuteranno eventuali priorità e riassegnazioni del carico di lavoro.

Si noti che i moduli *Navigazione* e *Interfaccia utente* compaiono alla fine della pianificazione. Infatti lo sviluppo di tali funzioni presuppone che il modulo *Contapassi* fornisca risultati attendibili onde evitare un continuo disallineamento tra le istruzioni fornite dalla navigazione (che dipendono dalla posizione attuale determinata dal contapassi) e ciò che realmente è stato fatto dall'utente.

Infine si sottolinea l'eventuale rischio nel sviluppare il modulo *Core* nella seconda metà del progetto poiché si potrebbe degradare la struttura dell'applicazione (ad es. se il modulo *Creazione mappa* è sviluppato "male", c'è la tentazione di colmare le sue lacune all'interno del modulo *Core*). Tale fatto dovrà essere uno stimolo per una buona progettazione in modo tale da avere un basso accoppiamento e un'elevata coesione tra moduli. Il termine *accoppiamento* indica la relativa interdipendenza tra moduli mentre il termine *coesione* indica il grado di connessione logica tra gli elementi (dati e funzioni) di un modulo. Essi dovrebbero essere raggruppati per un motivo logico, non in maniera puramente casuale e cooperare tra di loro in modo tale da raggiungere un obiettivo comune: la realizzazione della funzione richiesta per il modulo.

2.8 Documentazione

Questo documento di tesi si propone di descrivere il lavoro svolto. In particolare ogni modulo sviluppato verrà documentato in un apposito capitolo seguendo, in linea di massima, le seguenti direttive:

1. *Definizione delle specifiche*. In analogia all'omonima attività del processo software, si definiscono quali sono le funzioni del modulo accordate con il committente.
2. *Progettazione della soluzione*. In questa sezione si descrive come si intende adempire alle specifiche del modulo: sono illustrate le idee e i modelli su cui si basano le soluzioni proposte. Il grado di dettaglio della documentazione può variare da modulo a modulo: in alcuni casi si farà uso di modelli grafici (ad es. modello *E-R*), in altri di algoritmi scritti in pseudo-codice.
3. *Progettazione software del modulo*. L'obiettivo di tale sezione consiste nel descrivere come sono realizzate a livello software le soluzioni previste per il modulo. In generale, si approfondisce la sezione precedente con un maggiore livello di dettaglio: si documenta le scelte che hanno

portato all'utilizzo di determinate strutture dati, di particolari classi Android e di specifici meccanismi di sincronizzazione tra diverse istanze che realizzano determinate funzioni.

4. *Implementazione*. Tutto il codice è documentato dettagliatamente, anche con commenti in stile *Javadoc*. Tuttavia, in alcune circostanze, è opportuno approfondire alcuni dettagli tecnici per motivare particolari soluzioni implementative adottate.

5. *Validazione*. L'obiettivo di questa porzione di documentazione consiste nel descrivere la modalità con cui il modulo è stato validato sperimentalmente. Come indicato in precedenza, non tutti i moduli sono stati validati singolarmente.

Nonostante i nominativi delle sezioni della documentazione del progetto ricalcano i nomi delle attività del processo software, non è intenzione di questa documentazione descrivere tutte le fase del processo software. L'obiettivo è solo quello di documentare le funzioni, le idee e gli aspetti più contorti dell'implementazione in modo tale da essere comprensibili ad altri sviluppatori che riprenderanno in mano il progetto.

Nella parte finale di questa tesi, si descrive la procedura sperimentale adottata per valutare il comportamento dell'applicazione in modo che l'esperimento possa essere successivamente riprodotto. Il capitolo finale, invece, discute gli sviluppi futuri del progetto sia dal punto di vista scientifico (eventuali nuovi algoritmi) sia dal punto di vista della piattaforma hardware/software.

Capitolo 3

Modulo dati

**"I am not disorganized — I know *exactly* where everything is!
The newer stuff is on top and the older stuff is on the bottom."**

Fonte: *glasbergen* website. Ultima visita: 3 dicembre 2014.

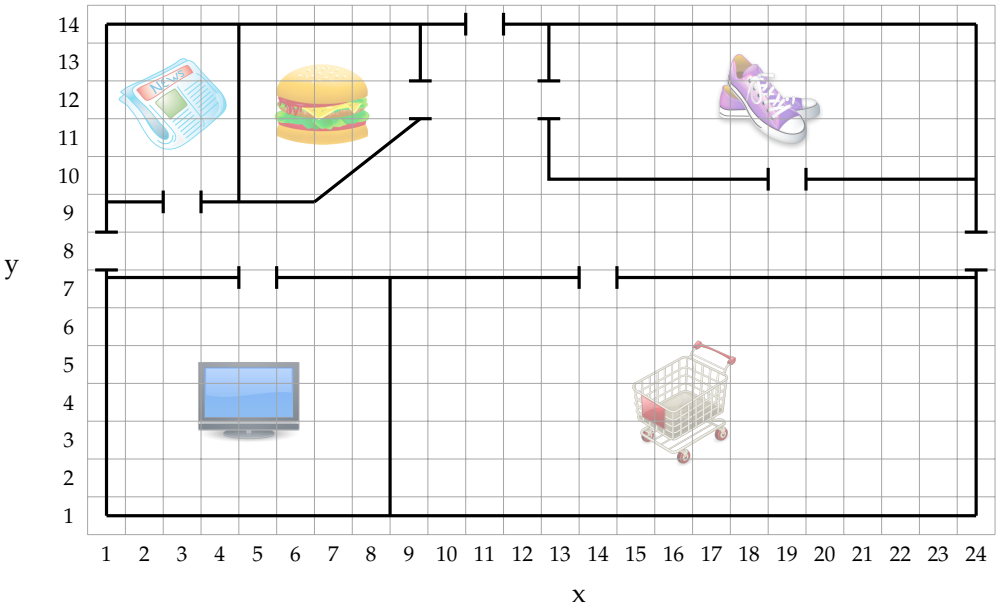
3.1 Definizione delle specifiche

3.1.1 Specifiche dei dati

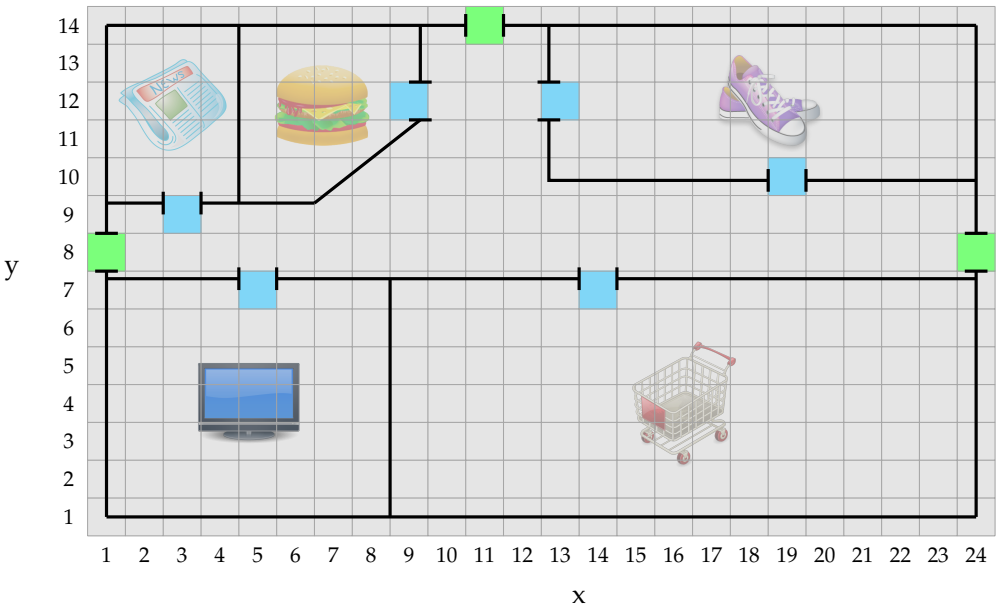
La principale entità dati dell'applicazione è la *Mappa*, cioè una terna $m = (S, E, I)$ in cui:

- S è un insieme di dati organizzati e correlati che costituiscono la rappresentazione della zona di territorio d'interesse.
- E è un insieme di *punti d'accesso* della mappa, chiamati anche *Entry Point Maps (EPMs)*, che indicano da dove iniziare la navigazione.
- I è un insieme di *punti d'interesse* della mappa , chiamati anche *Interesting Points (IPs)*, che indicano i possibili obiettivi della navigazione.

Durante la raccolta dei requisiti, è stato accordato con il committente lo sviluppo di una mappa che rappresenta, tramite una matrice, l'eventuale percorso effettuato dall'utente. Le sottosezioni seguenti descrivono le caratteristiche principali di questa tipologia di mappa.



(a) Sovrapposizione della griglia G alla mappa.



Cella $G_{x,y}$ tale che il corrispondente l'elemento $s_{x,y}$ della matrice S_{MMP} è nella forma $s_{x,y} = w$.

Cella $G_{x,y}$ tale che il corrispondente l'elemento $s_{x,y}$ della matrice S_{MMP} è nella forma $s_{x,y} = (w, desc)$. In altre parole $s_{x,y}$ individua un punto d'interesse.

Cella $G_{x,y}$ tale che il corrispondente l'elemento $s_{x,y}$ della matrice S_{MMP} è nella forma $s_{x,y} = (w, desc, gps)$. In altre parole $s_{x,y}$ individua un punto d'accesso.

(b) Semantica delle celle della griglia G.

Figura 3.1: Mappa topografica di un centro commerciale.

Processo di discretizzazione

L'area di territorio da mappare è soggetta ad un "processo di discretizzazione" in celle quadrate (figura 3.1a). Si ottiene così una *griglia* G di celle sovrapposta alla mappa in cui:

- La dimensione "fisica" del lato di ogni cella della griglia è chiamato *passo di discretizzazione* (chiamato anche *discretization step*).
- Ogni cella $G_{x,y}$ della griglia G è individuata da una coppia di coordinate (x, y) in cui $x \in \mathbb{N}$ rappresenta l'indice di colonna, mentre $y \in \mathbb{N}$ rappresenta l'indice di riga.

Mappa Matrice di pesi

Dati i seguenti insiemi:

$$W \subset \mathbb{R}$$

$$D \subset \{\text{Insieme di descrizioni in linguaggio naturale}\}$$

$$GPS \subset \{\text{Insieme di coordinate GPS}^{1}\text{nella forma } (lat, lon)\}$$

Si definisce mappa *Mappa Matrice di Pesi (MMP)* una mappa $m_{MMP} = (S_{MMP}, E_{MMP}, I_{MMP})$ tale che:

- $E_{MMP} \subset (W \times D \times GPS)$
- $I_{MMP} \subset (W \times D)$
- S_{MMP} è una matrice di elementi, chiamata semplicemente *Matrice di Pesi (MP)*, tale che per ogni elemento $s_{x,y}$ di S_{MMP}, la cui posizione è individuata dalla riga y e colonna x[a], vale la seguente condizione:

$$s_{x,y} \in \{w \in W, i \in I_{MMP}, e \in E_{MMP}\} \tag{3.1}$$

  In altre parole significa che un elemento di *MP* può essere:

  - Un numero reale w, chiamato *peso*.
  - Una coppia (w, d) in cui w è il peso e d è una descrizione dell'elemento.
  - Una terna (w, d, gps) in cui w è il peso, d è una descrizione dell'elemento e gps è una coppia di coordinate GPS.

Di seguito si fornisce un esempio di valori degli insiemi D ed GPS per la mappa in figura 3.1b:

$$D = \{\text{Edicola, Fast food, Calzature, Elettrodomestici, Supermercato}\}$$

$$GPS = \{(45.30511969733739, 12.021581679582596),$$
$$(45.30447074407369, 12.020903080701828),$$
$$(45.30369915915992, 12.020908445119858)\}^{b}$$

Più precisamente, l'insieme S che costituisce la rappresentazione del territorio d'interesse, non è costituito solamente dalla matrice S_{MMP} ma anche dal *passo di discretizzazione*. Per brevità, si assume che $S = S_{MMP}$ omettendo il passo di discretizzazione in fase di analisi (ma sarà correttamente rappresentato in fase di sviluppo).

[1] In accordo con lo standard WGS84 (World Geodetic System).

[a] Si noti che tale definizione inverte la semantica degli indici rispetto alla classica definizione matematica, in cui il primo indice individua la riga ed il secondo la colonna di un elemento della matrice. Tale esigenza nasce dal fatto che risulta più intuitivo rappresentare coordinate diverse della griglia con la stessa semantica adottata per rappresentare punti diversi del piano cartesiano xy, in cui x rappresenta il valore lungo l'ascissa e y il valore lungo l'ordinata.

[b] Valori GPS reali relativi alle entrate/uscite del centro commerciale "Piazzagrande" di Piove di Sacco (PD). Fonte: *coordinate-gps.it*. Ultima visita: 3 dicembre 2014.

Celle peso

Ogni elemento $s_{x,y} \in S_{MMP}$ rappresenta l'informazione d'interesse per l'applicazione relativa alla cella $G_{x,y}$ della griglia G (ad es. potrebbe contenere la probabilità che l'utente si trovi nella cella durante il percorso di navigazione della mappa).

Non è specificata la semantica del valore w di ogni elemento della matrice. Infatti è compito del modulo *Creazione mappa* assegnare il peso di ogni elemento in accordo con il proprio algoritmo. A seconda di come la mappa è stata popolata, sarà necessario eseguire un opportuno algoritmo di navigazione. Ad esempio, il modulo *Creazione mappa* potrebbe allocare una matrice S_{MMP} in cui ogni elemento indica il numero di volte in cui l'utente ha attraversato la corrispondente cella della griglia G durante più passaggi di uno stesso percorso di navigazione (cioè da uno stesso punto a ad uno stesso punto b); in questo caso, la mappa rappresenta una specie di "registrazione" del percorso. Successivamente, il modulo *Navigazione* provvederà a leggere direttamente dalla mappa il percorso che collega i due punti. In un altro contesto, invece, il modulo *Creazione mappa* potrebbe allocare una matrice S_{MMP} in cui ogni elemento indica se la corrispondente cella della griglia G può essere attraversata o meno. In questo altro caso, il modulo *Navigazione* provvederà a calcolare esplicitamente un percorso libero da collisioni che collega i due punti.

Sorgente dati

La sorgente dati dell'applicazione è costituita da un insieme **D** di mappe chiamato *dataset*. Inizialmente **D** è composto solo da mappe di tipo *MMP* ma in futuro potrebbe gestire (cioè salvare, caricare e cancellare) più tipologie di mappe diverse (ad es. mappe che utilizzano strutture dati diverse come linked list o matrici non popolate in tutti i loro elementi).

3.1.2 Specifiche delle operazioni

Data una *MMP* $m = (S, E, I)$ sono ammesse le seguenti operazioni:

- *Salva mappa* m su memoria secondaria.
- Dato $gps \in GPS$, *carica mappa* m dalla memoria secondaria. L'utente invoca tale operazione quando necessita di navigare una mappa, cioè determinare l'insieme di spostamenti che consentono, a partire da un punto d'accesso, di raggiungere un punto d'interesse. All'invocazione dell'operazione, l'utilizzatore fornisce in ingresso una coppia di coordinate $gps_2 \in GPS$ da cui vuole iniziare la navigazione. Il modulo *Dati* deve provvedere a caricare dalla sorgente dati e ritornare all'utente la mappa $m = (S, E, I)$ tale che almeno un punto d'accesso $e = (w, d, gps_1) \in E$ "contenga" la coppia di coordinate gps_2, cioè che $gps_1 = gps_2$ sia una condizione valida. Se alla coppia di coordinate $gps_2 \in GPS$ corrispondono più mappe, esse sono tutte restituite all'utente (questa condizione può verificarsi quando uno stesso ambiente è mappato utilizzando algoritmi o parametri diversi).
- Dato $gps \in GPS$, *cancella mappa* m dalla memoria secondaria. L'utente fornisce in ingresso una coppia di coordinate $gps_2 \in GPS$; in modo analogo all'operazione di caricamento, il modulo *Dati* procede a cancellare dal dataset la mappa $m = (S, E, I)$ tale che almeno un punto d'accesso $e = (w, d, gps) \in E$ verifichi la condizione $gps_1 = gps_2$. Se alla coppia di coordinate gps_2 corrispondono più mappe, esse sono tutte cancellate.

Sono previste inoltre alcune operazioni da utilizzare esclusivamente durante lo sviluppo e la validazione del modulo *Contapassi*:

- *Caricamento* e *Cancellazione* di una specifica mappa dato un suo codice identificativo (*id*).
- *Esportazione* di una mappa in un formato elaborabile da altri strumenti software (ad esempio, il formato `csv`). Tale esigenza nasce dal fatto di poter visualizzare, anche graficamente, la mappa costruita.

Per il momento si tralascia lo sviluppo di:

- Una forma di *backup remoto* poiché richiede uno studio preliminare oneroso (ad es. studio di API per l'interfacciamento con servizi come *Dropbox*[2] o *Google Drive*[3]).
- Una funzione efficiente di *modifica* diretta per gli elementi $s_{x,y}$ della matrice già salvati nel dataset: ad esempio la possibilità di "promuovere" un $s_{x,y} = w$ ad un punto d'interesse $s_{x,y} = (w, d)$ senza dover caricare l'intera mappa in memoria centrale, modificarla e salvarla nuovamente nel dataset.
- Una funzione che, dati $gps \in GPS$ e $d \in D$, carichi la mappa $m = (S, E, I)$ tale che $\exists e = (w_e, d_e, gps_e) \in E$ tale che $gps_e = gps$ e $\exists i = (w_i, d_i) \in I$ tale che $d_i = d$. In altre parole, questa funzione carica la mappa m che consente di raggiungere il punto d'interesse con descrizione d a partire dal punto d'accesso con coordinate gps.

3.1.3 Dataset di riferimento

La sorgente dati su cui eseguire i test dell'applicazione è costituita da un dataset $\mathbf{D}$ di 50 *MMP*. Il popolamento delle mappe avviene casualmente, sia sui valori degli elementi della matrice e sia sui parametri delle strutture dati. Tuttavia, ogni mappa contiene un numero di elementi della matrice sufficiente a mappare un'area di 10 m $\times$10 m.

3.1.4 Specifiche di test

Le specifiche di test interessano la particolare specifica di *MMP* realizzata. Poiché la funzione logica del modulo *Dati* è indipendente dalle altre funzioni logiche dell'applicazione è opportuno determinare delle specifiche di test con le quali verificare il corretto funzionamento del modulo.

Ogni mappa soggetta ai test rispetta le condizioni di riferimento della sezione 3.1.3. I test da eseguire sono:

- *Casi d'uso*:
  - Salvataggio di una *MMP* nel dataset e successivo caricamento. La mappa salvata $m_1 = (S_1, E_1, I_1)$ e la mappa caricata $m_2 = (S_2, E_2, I_2)$ devono essere *equivalenti*, cioè devono essere valide tutte le seguenti condizioni:
    1. S_1 e S_2 sono matrici con le stesse dimensioni.
    2. S_1 e S_2 contengono lo stesso numero di celle inizializzate, cioè a cui è stato assegnato un peso w.
    3. Esiste una corrispondenza tra S_1 a S_2 tale che se $s_{1_{x,y}} = w$ allora $s_{2_{x,y}} = w$ e viceversa. In altre parole, se ad una cella della matrice S_1 è assegnato un elemento w, alle stesse coordinate ma nella matrice S_2 deve essere assegnato un elemento w (e viceversa).
    4. Gli insiemi E_1 e E_2 sono equivalenti.
    5. Gli insiemi I_1 e I_2 sono equivalenti.
  - Caricamento e modifica di una *MMP*, attraverso i seguenti passi:
    1. Dato *gps*, carica la corrispondente *MMP* $m_1 = (S_1, E_1, I_1)$.
    2. Modifica, da parte di uno stub del modulo *Creazione Mappa*, di alcuni elementi di S_1 che non appartengono a $E_1 \cup I_1$ in modo tale da creare una nuova mappa $m_2 = (S_2, E_1, I_1)$.
    3. Cancellazione di m_1 dal dataset e successivo salvataggio di m_2 nel dataset.
    4. All'esecuzione di *carica mappa*(*gps*), la mappa m_1 non deve essere restituita.
- *Test di sovraccarico*: salvataggio e successivo caricamento di una *MMP* la cui matrice contiene tutti elementi inizializzati, cioè a cui è stato assegnato un peso w.

[2] *Dropbox*. Fonte: documentazione Dropbox. Link. Ultima visita: 3 dicembre 2014.
[3] *Google Drive*. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.

- *Test di prestazioni*:
  - Caricamento e salvataggio di una *MMP* di grandi dimensioni (almeno una matrice 1000×1000 elementi istanziati).
  - Le operazioni di caricamento, cancellazione e salvataggio di una mappa devono avvenire entro 2 secondi nella piattaforma hardware di test.

3.2 Progettazione della soluzione

La progettazione della soluzione individua come realizzare la specifica della *MMP*, attraverso i seguenti passi:

1. Progettazione concettuale, cioè la costruzione di un modello dati per la realtà da rappresentare.
2. Scelta di un meccanismo di persistenza.

3.2.1 Progettazione concettuale

In questa fase, il progettista deve cercare di rappresentare il contenuto informativo della realtà, senza preoccuparsi né delle modalità con le quali queste informazioni verranno codificate in un sistema reale, né dell'efficienza dei programmi che faranno uso di queste informazioni. Va innanzitutto precisato che spesso non esiste una rappresentazione univoca di un insieme di specifiche, perché le stesse informazioni possono essere rappresentate in modi differenti e non comparabili.

3.2.1.1 Richiami teorici del modello E-R

Il modello *Entità-Relazione (E-R)* consente la rappresentazione concettuale della realtà e dei dati d'interesse ad un alto livello di astrazione [42]. Esso si basa su un insieme di concetti molto vicini alla realtà, quindi facilmente intuibili dai progettisti (e in genere considerati sufficientemente comprensibili e significativi anche per i non-tecnici), ma non implementabili sugli elaboratori. Per la sua intuitività, si sceglie di adottare questo strumento per modellare le *MMPs* dell'applicazione.

Il modello E-R presenta un insieme di costrutti [43]:

- *Entità*: rappresenta la classe di oggetti (fatti, cose, persone, ...) che hanno proprietà comuni ed esistenza autonoma ai fini dell'applicazione di interesse. Un'*occorrenza* di un'entità è un oggetto o istanza della classe che l'entità rappresenta.
- *Associazione* (detto anche *relazione*): rappresenta un legame logico e significativo per l'applicazione di interesse tra due o più entità. Un'occorrenza di associazione è una sequenza ordinata costituita da occorrenze di entità, una per ciascuna delle entità coinvolte nell'associazione.
- *Attributo*: descrive le proprietà elementari delle entità e delle associazioni.
- *Cardinalità delle associazioni*: sono specificate per ciascuna partecipazione di entità a un'associazione e descrivono il numero minimo e massimo di occorrenze dell'associazione a cui una occorrenza dell'entità può partecipare.
- *Identificatore dell'entità* (detto anche *chiave*): è specificato per ciascuna entità e descrive gli attributi del modello che permettono di identificare in maniera univoca le occorrenze delle entità. Talvolta un'entità non ha una chiave ma è identificata componendo una propria chiave *parziale* con la chiave di un'altra entità alla quale è associata mediante un'associazione (tale chiave è detta *chiave esterna*). L'entità priva di chiave propria si chiama entità *debole*, l'entità associata a questa si chiama entità *proprietario* e l'associazione che le lega si dice *identificante*. Affinché l'entità debole possa essere univocamente identificata, essa deve

partecipare all'associazione identificante con vincolo di partecipazione $(1,1)$. In altre parole, per ogni istanza dell'entità debole, deve esistere esattamente una istanza dell'entità proprietario (che la identifica).

- *Generalizzazione*: rappresenta il legame logico tra un'entità E, detta entità genitore, e una o più entità $E_1, \ldots, E_n$, dette entità figlie, di cui E è più generale, nel senso che le comprende come caso particolare. Si dice, in questo caso, che E è *generalizzazione* di $E_1, \ldots, E_n$ e che le entità $E_1, \ldots, E_n$ sono *specializzazioni* dell'entità E.

Nel prossimo paragrafo, si fornisce un esempio di modello E-R analizzando direttamente il modello sviluppato per il modulo *Dati*.

3.2.1.2 Modello E-R per il modulo Dati

Anche nel caso di applicazioni non particolarmente complesse, lo schema E-R che si ottiene può contenere molti concetti correlati in una maniera piuttosto complicata. Ne consegue che la costruzione dello schema finale è, necessariamente, un processo graduale: lo schema concettuale viene progressivamente raffinato e arricchito attraverso una serie di trasformazioni ed eventuali correzioni.

In aggiunta, l'esperienza del progettista aiuta a scovare pattern e criteri di carattere generale per tradurre specifiche informali in concetti del modello E-R. Questi costrutti sono poi utilizzati (sezione 3.3.2.2) per definire schemi che descrivono l'organizzazione e la struttura delle occorrenze dei dati, cioè dei valori assunti dalle mappe al variare del tempo.

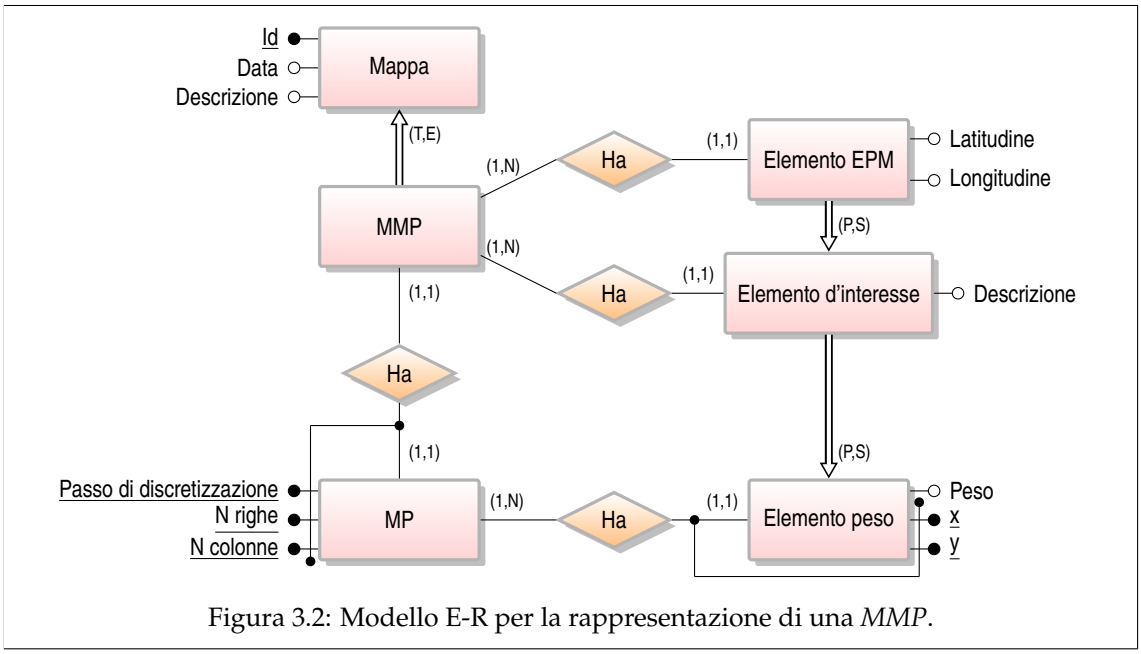


Figura 3.2: Modello E-R per la rappresentazione di una *MMP*.

Il modello E-R sviluppato deriva in modo naturale dai concetti emersi nella definizione delle specifiche dei dati. Esso è rappresentato in figura 3.2 e presenta i costrutti descritti nelle seguenti sottosezioni.

Entità

Le entità rappresentano i tasselli fondamentali del modello e costituiscono i fatti della realtà che si intende rappresentare. Quindi nel contesto dell'applicazione, è intuitivo descrivere la mappa in termini di entità *MP*, *EPM* o *IP*.

Di seguito si fornisce una descrizione delle entità individuate per il modello scelto:

- *Mappa*: descrive le caratteristiche comuni a tutte le mappe attraverso gli attributi:

  - *Id*: è l'identificatore dell'entità.
  - *Data*: indica la data di acquisizione della mappa. Acquisizioni troppo vecchie potrebbero non essere aggiornate (ad es. potrebbero essere stati inseriti nuovi ostacoli, come panchine o vasi).
  - *Descrizione* della mappa.

- *MMP*: è una specializzazione dell'entità *Mappa*. Poiché in questo progetto si sviluppa un unico modello di mappa, la generalizzazione è:
  - *Totale*, cioè non esiste un'occorrenza dell'entità generalizzante che non sia anche occorrenza di una sua specializzazione (in caso contrario, la generalizzazione è *parziale*).
  - *Esclusiva*, cioè un'occorrenza dell'entità generalizzante è occorrenza di al più una sua specializzazione (in caso contrario, la generalizzazione è *sovrapposta*).

  Per il momento non sono previsti particolari attributi discriminanti rispetto alla generalizzazione.

- *MP*: descrive la matrice di una *MMP*. È composta dai seguenti attributi:
  - *Passo di discretizzazione*: è la dimensione in metri del lato di ogni singola cella della mappa originale.
  - *N righe* e *N colonne*: indicano rispettivamente il numero di righe e di colonne della matrice.

  L'esistenza di ogni occorrenza di *MP* dipende dalla *MMP* a cui essa è associata. Pertanto *MP* è un'entità debole, identificata in collaborazione con l'entità proprietario *MMP*. Per il momento si omette di specificare la motivazione dell'identificatore parziale dell'entità debole dato dalla terna *Passo di discretizzazione*, *N righe* e *N colonne*: in seguito, infatti, si osserverà che esso non è più necessario.

- *Elemento peso*: descrive ogni singolo elemento della matrice[4]. A differenza della specifica dei dati, in cui gli elementi $s_{x,y}$ della matrice S_{MMP} appartengono a tre insiemi diversi (si veda la formula 3.1), si è scelto di modellare $s_{x,y}$ attraverso una *gerarchia* di entità. Questo strumento consente di modellare in modo più appropriato la realtà d'interesse: infatti si noti che un punto d'accesso $e = (w_e, d_e, gps) \in E$ "può contenere" l'informazione di un punto d'interesse $i = (w_i, d_i) \in I$ attraverso la coppia (w_e, d_e) tale che $w_e = w_i$ e $d_e = d_i$. Un ulteriore vantaggio di questa soluzione consiste nell'avere una modello della realtà più vicino agli strumenti di programmazione disponibili.

  L'entità *Elemento peso* è descritta dagli attributi:
  - *x*: colonna x-esima della matrice a cui l'occorrenza appartiene.
  - *y*: riga y-esima della matrice a cui l'occorrenza appartiene.
  - *peso* dell'elemento.

  Poiché l'esistenza di ogni occorrenza di *Elemento peso* dipende dall'occorrenza di *MP* a cui essa è associata, *Elemento peso* è un'entità debole. L'identificatore parziale dell'entità debole è dato dalla coppia di attributi *x* e *y*.

- *Elemento d'interesse*: descrive un punto d'interesse. È una specializzazione di *Elemento peso* ed include l'attributo *descrizione*.

- *Elemento EPM*: descrive un punto d'accesso. È una specializzazione di *Elemento d'interesse* ed include gli attributi *Latitudine* e *Longitudine* che costituiscono le coordinate GPS.

[4] Si è scelto il termine *Elemento* anziché *Punto* per distinguere il fatto che, in questo modello dati, ogni punto d'interesse, punto d'accesso e "punto semplice" è un elemento di una matrice.

La generalizzazione tra *Elemento peso* ed *Elemento d'interesse* è parziale: esistono elementi di *MP* che non sono punti d'interesse. Analogamente, la generalizzazione tra *Elemento d'interesse* e *Elemento EPM* è parziale poiché esistono punti d'interesse che non sono punti d'accesso (ad es. l'elemento della matrice che rappresenta l'entrata di un bar posto all'interno di un centro commerciale potrebbe non avere una coordinata GPS poiché è fisicamente difficile acquisire il segnale all'interno di un edificio).

Come già accennato, si noti che, secondo questo modello, tutti i punto d'accesso sono anche punti d'interesse. Questo caratteristica è utile in determinati scenari: ad esempio, nel caso in cui gli elementi della matrice S_{MMP} indicano se le corrispondenti celle della griglia G possono essere attraversate o meno, un'eventuale porta d'ingresso di un edificio potrebbe costituire sia un punto d'accesso, da cui iniziare la navigazione, sia un obiettivo (nel caso in cui essa svolga anche il ruolo di uscita di sicurezza).

Si è scelto di non rappresentare le generalizzazioni *Rappresentazione mappa*, *Punto d'accesso* e *Punto d'interesse* per le rispettive entità *MMP*, *Elemento EPM* e *Elemento d'interesse* per non appesantire (anche graficamente) il modello. Inoltre, poiché non sono previste ulteriori specializzazioni di *Mappa*, esse non contengono attributi significativi.

Associazioni

Nel contesto dell'applicazione, le associazioni assumono spesso il significato del verbo "avere" o "possedere". Intuitivamente, si dirà che una mappa possiede uno o più *EPMs/IPs* e che, in analogia, uno o più *EPMs/IPs* fanno parte di una mappa.

Di seguito, si approfondisce il legame tra le varie entità del modello:

- Associazione *Ha* tra *MMP* e *MP*: un'occorrenza di *MMP* è (obbligatoriamente) in relazione con una occorrenza di *MP*. Infatti l'esistenza di una *MMP* è legata all'esistenza della rispettiva matrice. La cardinalità della partecipazione di *MMP* a questa associazione è $(1, 1)$. Analogamente, un'occorrenza di *MP* è legata ad una sola occorrenza di *MMP*: anche in questo caso, la cardinalità della partecipazione di *MP* a questa associazione è $(1, 1)$.

- Associazione *Ha* tra *MP* e *Elemento peso*: un'occorrenza di *MP* è legata a una o più occorrenze di *Elemento peso*, che costituiscono gli elementi della matrice: infatti non ha senso per l'applicazione inserire matrici vuote. La cardinalità della partecipazione è $(1, N)$. Viceversa, un'occorrenza di *Elemento peso* è legata ad una sola occorrenza di *MP*: infatti un elemento appartiene ad una sola matrice (tuttavia potrebbero esistere celle con i medesimi valori di x, y e *peso* ma appartenenti a matrici diverse). La cardinalità della partecipazione è $(1, 1)$.

- Associazione *Ha* tra *MMP* e *Elemento d'interesse*: un'occorrenza di *MMP* è in relazione con 1 o più occorrenze di *Elemento d'interesse*. Solitamente, per i casi previsti di utilizzo della mappa, si assume sia sempre presente almeno un punto di interesse. Pertanto la cardinalità della partecipazione è $(1, N)$. Tuttavia, in futuro tale vincolo potrà essere rimosso, consentendo anche l'inserimento di mappe senza punti d'interesse. Viceversa, un'occorrenza di *Elemento d'interesse* è legata ad una sola occorrenza di *MMP*: la cardinalità della partecipazione è $(1, 1)$.

- Associazione *Ha* tra *MMP* e *Elemento EPM*: un'occorrenza di *MMP* può essere in relazione con 1 o più occorrenze di *Elemento EPM*: anche in questo caso, non ha senso costruire una mappa senza punti d'accesso. La cardinalità della partecipazione di *MMP* a questa associazione è $(1, N)$. Viceversa, un'occorrenza di *Elemento EPM* è legata ad una sola occorrenza di *MMP*: la cardinalità della partecipazione di *Elemento EPM* a questa associazione è $(1, 1)$.

Osservazioni

- Eventuali acquisizioni di mappe relative alla stessa zona di territorio costituiscono occorrenze delle entità del modello diverse tra loro.

- Come già osservato in precedenza, data una coppia di coordinate GPS, che costituisce "la chiave di ricerca" per la funzione *carica mappa*, più occorrenze di *MMP* potrebbero essere

restituite (ad es. nel caso in cui esistano più mappe relative a piani diversi di uno stesso edificio).

3.2.2 Scelta del meccanismo di persistenza

Dopo aver determinato un modello dati è necessario implementarlo in forma persistente. Un primo approccio consiste nell'implementazione mediante un insieme di classi Java che consentano il salvataggio, il caricamento e la cancellazione delle occorrenze del modello E-R su file. Tuttavia tale metodo risulta oneroso poiché:

- Occorre sviluppare da zero le procedure di inserimento, cancellazione e caricamento di una *MMP*.
- Occorre gestire la consistenza (ad es. se un'occorrenza di *MMP* è eliminata, tutte le occorrenze di *Elemento EPM* e *Elemento d'interesse* ad esse associate non hanno più senso di esistere).
- Può introdurre ridondanza (ad es. potrei non aver implementato nessun meccanismo che individui se due mappe con la stessa matrice sono state inserite).

Un approccio più intelligente consiste nell'utilizzare un DataBase Management System (DBMS), cioè un sistema general-purpose che consente di manipolare una base di dati. Si definisce *base di dati* una collezione di dati correlati e strutturati di interesse per una qualche applicazione reale [44]. In particolare un DBMS consente di:

- *Definire* le strutture, i vincoli e i tipi di dati.
- *Costruire* e *immagazzinare* i dati in qualche supporto di memorizzazione controllato dal DBMS.
- *Manipolare* ed *interrogare* la collezione per recuperare dati specifici, qualunque sia l'applicazione a cui la base di dati è finalizzata.

DBMS in Android

Fin dalla sua nascita, Android supporta nativamente il DBMS *SQLite*[5], cioè integra:

- La libreria software che implementa il core del SQL database engine.
- Un'insieme di classi Java che consentono di interfacciarsi con il core di *SQLite*.

SQLite presenta le seguenti caratteristiche:

- È un DBMS relazionale.
- Supporta le transazioni, cioè una sequenza atomica di operazioni, che può concludersi con un successo o un insuccesso.
- È *serverless*. La maggior parte dei DBMS sono implementati come processo server separato dall'applicazione client; questi ultimi comunicano con il server usando un particolare protocollo (tipicamente TCP/IP) per inviare le richieste e ricevere i risultati. Invece, *SQLite* è eseguito direttamente nel processo client, il quale si occupa di scrivere e leggere i file su disco. Questo approccio ha il vantaggio di non richiedere la configurazione di *SQLite* prima del suo utilizzo. Tuttavia ha i seguenti svantaggi rispetto alla soluzione client-server:
  - Minor protezione dai bug poichè non c'è disaccoppiamento dall'applicazione client.
  - Minor concorrenza.

[5] *SQLite*. Fonte: *SQLite* homepage. Link. Ultima visita: 3 dicembre 2014.

Per lo sviluppo del progetto, si è scelto di utilizzare *SQLite*. Nulla vieta in futuro di far uso anche di un DBMS esterno, il cui server risiede in un altro terminale. Se in futuro l'applicazione riuscirà a raggiungere una stima molto accurata dell'esatto percorso effettuato dall'utente ed a prevedere un processo di normalizzazione che trasforma più acquisizioni di una stessa mappa, effettuata da utenti diversi, in un'unica rappresentazione allora l'utilizzo di un DBMS esterno consentirebbe di condividere le mappe tra più utenti, incrementando la copertura del territorio e la precisione delle mappe. Tuttavia:

- L'applicazione richiederebbe una connessione dati (ad internet) permanente e di buone prestazioni poiché il dataset delle mappe può occupare decine di MB di memoria.
- È necessario prevedere un meccanismo che tuteli la privacy dell'utente (ad es. poter scegliere quali mappe condividere).

Un'altra estensione significativa potrebbe far uso di un database eXtensible Markup Language (XML): un record descritto con tale "linguaggio" agevola l'interoperabilità tra applicazioni diverse.

3.2.2.1 Richiami teorici del modello relazionale

Poichè *SQLite* è un DBMS relazionale, è necessario tradurre il modello E-R sviluppato in un modello relazionale. In questa sezione si forniscono alcune nozioni teoriche su quest'ultimo modello; tali argomenti sono ampiamente trattati in un qualsiasi generico libro di testo riguardante le basi di dati (ad esempio [43]).

Relazione

Un database relazionale si basa sul modello logico di tipo relazionale [45], il quale utilizza il concetto di relazione matematica. Una *relazione matematica* sugli insiemi $D_1, D_2, \ldots, D_n$ (chiamati domini della relazione) è un sottoinsieme di $D_1 \times D_2 \times \cdots \times D_n$. L'insieme di *ennuple* ordinate $(v_1, v_2, \ldots, v_n)$ con $v_1 \in D_1, v_2 \in D_2, \ldots, v_n \in D_n$ costituiscono gli elementi della relazione. Si noti che:

- Non è definito alcun ordinamento fra le ennuple.
- Le ennuple di una relazione sono distinte l'una dall'altra.
- Ciascuna ennupla è, al proprio interno, ordinata: l'i-esimo valore di ciascun componente proviene dall'i-esimo dominio (struttura posizionale).

Tuttavia, la definizione di relazione nel modello relazionale dei dati assume una sfumatura diversa:

- Ad ogni dominio è associato un nome, detto *attributo*, che descrive il "ruolo" giocato dal dominio stesso. Formalmente, si indica con D l'insieme dei domini e con A l'insieme di attributi. Si specifica la corrispondenza fra attributi e domini per mezzo di una funzione $dom : A \to D$, che associa a ciascun attributo $A_i \in A$ un dominio $dom(A_i) \in D$.
- L'ordinamento fra gli attributi è irrilevante: la struttura è non posizionale.
- Si definisce *tupla* una funzione, definita su un sottoinsieme di attributi di A, che associa a ciascun attributo $A_i \in A$ un valore del dominio di $dom(A_i)$.

Si può quindi dare una nuova definizione di relazione (per il modello relazionale dei dati): una relazione su A è un insieme di tuple su A. La differenza fra questa definizione e quella tradizionale di relazione matematica risiede solo nella definizione di tupla: nella relazione matematica si hanno ennuple i cui elementi sono individuati per posizione, mentre nelle tuple della nuova definizione gli elementi sono individuati per mezzo degli attributi, cioè con una tecnica non posizionale.

Generalmente una relazione è rappresentata in forma tabellare in cui:

- Gli attributi possono essere usati come intestazioni delle colonne.
- I valori di ciascuna colonna sono fra loro omogenei, cioè provenienti dallo stesso dominio.
- Le righe sono diverse fra loro.
- L'ordinamento tra le righe è irrilevante.
- L'ordinamento tra le colonne è irrilevante.

In seguito si indica con il termine *tabella* una relazione rappresentata in forma tabellare[6].

In figura 3.3 è rappresentata una relazione, secondo la definizione appena adottata. L'insieme di attributi A è costituito da *{Id, N righe, N colonne, Passo di discretizzazione}* mentre l'insieme di tuple è costituito da tutte le righe (*record*) della tabella.

Modello "basato su valori"

Il modello relazionale è "basato su valori": i riferimenti fra dati in relazioni diverse sono rappresentati per mezzo di valori dei domini che compaiono nelle tuple. Ad esempio, si supponga che i record di una relazione che modella gli *EPMs* debbano riferirsi ai rispettivi record delle *MMPs* a cui gli *EPMs* appartengono. Nel modello relazionale, tale riferimento non è espresso tramite puntatori (dalla relazione per gli *EPMs* alla relazione per le *MMPs*) ma bensì utilizzando lo stesso valore su uno specifico attributo comune ad entrambe le relazioni (*id* della *MMP*).

Si nota che:

- Si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione (e quindi dell'utente).
- La rappresentazione logica dei dati (costituita dai soli valori) non fa alcun riferimento a quella fisica, che può anche cambiare nel tempo: il modello relazionale permette quindi di ottenere l'*indipendenza fisica* dei dati.
- Essendo tutta l'informazione contenuta nei valori, è relativamente semplice trasferire i dati da un contesto a un altro (per esempio se si deve trasferire una base di dati da un computer ad un altro).

Nel modello basato su valori, è conveniente introdurre la seguente notazione: sia t una tupla su A e $A_i \in A$, allora $t[A_i]$ indica il valore di t su A_i (figura 3.3).

MMP

Id	N righe	N colonne	Passo di discretizzazione
12345	50	50	1.5
23456	100	100	1
34567	250	400	2
…	…	…	…

Figura 3.3: Esempio di relazione (secondo il modello relazionale dei dati) espressa in forma tabellare. Se t è la tupla indicata in giallo allora $t[N\ righe] = 100$.

È possibile intuire che il modello relazionale dell'applicazione sarà costituito da più tabelle, ognuna delle quali popolata da record composti dai valori assunti dalle specifiche occorrenze delle entità e associazioni del modello E-R.

[6]Talvolta, si utilizzerà impropriamente i termini "relazione" e "tabella" come sinonimi.

Schema ed istanze

Uno *schema di relazione* è costituito da un simbolo R, detto nome della relazione, e da un insieme di (nomi di) attributi $X = \{A_1, \ldots, A_n\}$ con $A_i \in A$ (si ricorda che A è l'insieme di attributi). Generalmente è indicato nella forma $R(X)$.

Si definisce *schema della base di dati* **R** un insieme di schemi di relazione con nomi diversi:

$$\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$$

Un'*istanza di relazione* (o semplicemente relazione) su uno schema $R(X)$ è un insieme r di tuple su X.

Un'*istanza della base di dati* su uno schema $\mathbf{R} = \{R_1(X_1), \ldots, R_n(X_n)\}$ è un insieme di relazioni $r = \{r_1, \ldots, r_n\}$ dove ogni r_i, per $1 < i < n$, è una relazione sullo schema $R_i(X_i)$. Spesso con il termine base di dati si intende un'istanza di base di dati.

L'obiettivo delle successive fasi della progettazione (in particolare della progettazione logica, sezione 3.3.2) consiste nel determinare uno schema della basi di dati per l'applicazione d'interesse. In fase di validazione dell'intero software, si procederà a costruire e, in seguito, a valutare la bontà dei risultati ricavati dall'istanza della base di dati.

Valori null

ll modello relazionale impone ai dati una struttura rigida, in cui le informazioni sono rappresentate per mezzo di ennuple ammesse dagli schemi di relazione. Tuttavia i dati disponibili possono non corrispondere esattamente al formato previsto: in tale situazione il modello relazionale ammette l'inserimento del valore nullo (detto semplicemente *null*) il quale denota l'assenza di un valore del dominio. Formalmente, è sufficiente estendere il concetto di tupla: $\forall A_i \in A,\ t[A_i]$ è un valore del dominio $dom(A_i)$ oppure il valore *null*.

Ad esempio, la relazione che modella gli *EPMs* potrebbe prevedere l'attributo "Descrizione". Tuttavia non tutte le istanze *EPMs* rappresentate nel linguaggio di programmazione adottato (Java) potrebbero avere una descrizione: per alcuni il campo di testo potrebbe essere inizializzato alla stringa vuota, altri avere la referenza dell'oggetto a `null`. Per quest'ultimi, emerge il problema di come trattare la persistenza attributo "Descrizione" del record a loro associato.

Di norma si impongono restrizioni sulla presenza di valori *null* poiché essi sono difficilmente confrontabili con i valori del dominio.

Vincoli d'integrità

Le strutture del modello relazionale permettono di organizzare le informazioni di interesse per l'applicazione. In molti casi, però, non è vero che qualsiasi insieme di tuple sullo schema relazionale rappresenta informazioni corrette per l'applicazione. Per tale motivo è stato introdotto il concetto di *vincolo di integrità*, come proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione.

Ogni vincolo può essere visto come un predicato che associa a ogni istanza il valore vero o falso. Se il predicato assume il valore vero si dice che l'istanza soddisfa il vincolo. In generale, a uno schema di base di dati si associa un insieme di vincoli e si considerano corrette (o lecite, o ammissibili) le istanze che soddisfano tutti i vincoli.

È possibile classificare i vincoli a seconda degli elementi di una base di dati che ne sono coinvolti. Si distinguono due categorie:

- Un vincolo è *intra-relazionale* se il suo soddisfacimento è definito rispetto a singole relazioni della base di dati. Ad esempio:
  - Un vincolo di tupla è un vincolo che può essere valutato su ciascuna tupla indipendentemente dalle altre.
  - Un vincolo di dominio è definito con riferimento ai singoli valori.
- Un vincolo è *inter-relazionale* se coinvolge più relazioni.

Ad esempio, nel contesto dell'applicazione, un vincolo d'integrità consiste nell'impedire il salvataggio di una *MP* che non contiene nessun *EPM*. Infatti, non sarebbe possibile interrogare e caricare tale matrice dalla base di dati poiché nessun elemento mantiene il valore delle coordinate GPS.

Vincoli di chiave

Una chiave è un insieme di attributi che identificano univocamente le tuple di una relazione. Più precisamente:

- Un insieme K di attributi è *superchiave* per una relazione r se r non contiene due tuple distinte t_1 e t_2 con $t_1[K] = t_2[K]$.
- K è *chiave* per r se è una superchiave *minimale* (cioè non contiene un'altra superchiave) per r.

L'esistenza delle chiavi garantisce l'accessibilità a ciascun dato della base di dati:

- Ogni singolo valore è univocamente accessibile tramite:
  - Nome della relazione.
  - Valore della chiave.
  - Nome dell'attributo.
- Le chiavi sono lo strumento principale attraverso il quale vengono correlati i dati in relazioni diverse.

Vincoli di integrità referenziale (foreign key)

Vincoli di integrità referenziale (*foreign key*) consentono di correlare informazioni presenti in relazioni diverse attraverso valori comuni, utilizzando la natura "value based" del modello relazionale. Un *vincolo di integrità referenziale* fra un insieme di attributi X di una relazione R_1 e un'altra relazione R_2 impone ai valori su X di ciascuna tupla dell'istanza di R_1 di comparire come valori della chiave (primaria) dell'istanza di R_2. Più formalmente, siano $X = \{a_1, \ldots, a_n\}$ e $K = \{b_1, \ldots, b_n\}$ due insieme ordinati di attributi. Il vincolo è soddisfatto se per ogni tupla t_1 in R_1 su X (senza valori nulli) esiste una tupla t_2 su R_2 con $t_1[a_i] = t_2[b_i]$, $\forall i \mid 1 \leq i \leq n$ (figura 3.4).

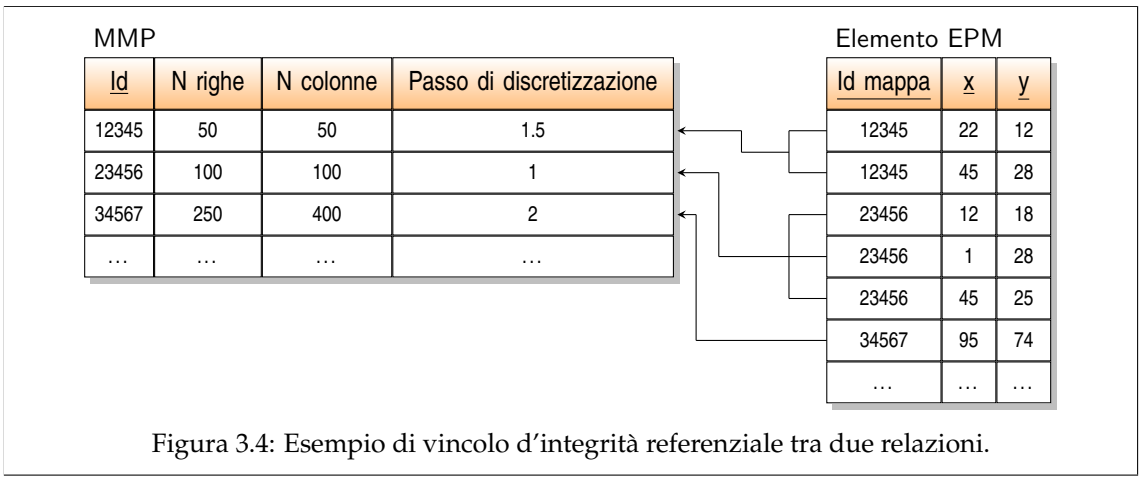


Figura 3.4: Esempio di vincolo d'integrità referenziale tra due relazioni.

Ad esempio, nel contesto dell'applicazione, un vincolo d'integrità referenziale consiste nell'impedire il salvataggio di un *EPM* se non esiste nessuna istanza di *MMP* ad esso associato. In assenza di tale vincolo, nella tabella contenente gli *EPMs* dell'applicazione potrebbero essere contenuti record che non si riferiscono a nessuna *MMP* presente nella base di dati.

3.3 Progettazione software del modulo

La progettazione software del modulo prevede di:

- Tradurre il modello dati in un diagramma delle classi per il paradigma di programmazione *Object-Oriented (O-O)*.
- Tradurre il modello dati in una forma elaborabile per il meccanismo di persistenza adottato.
- Progettare "un meccanismo di mapping" tra il modello dati O-O e il modello dati adottato dalla persistenza (potrebbero non essere uguali). Si sceglie di realizzare tale mappa attraverso il design pattern Data Access Object (DAO).

3.3.1 Diagramma delle classi

Purtroppo il modello E-R non è direttamente elaborabile dal linguaggio di programmazione Java; è necessario quindi tradurlo in una forma diversa, idonea all'elaborazione con il linguaggio di programmazione adottato e preservando, il più possibile, le caratteristiche del modello originale. In questa fase ci si occupa quindi di tradurre il modello della realtà in una forma che consenta l'inserimento di nuove elementi nella mappa, l'aggiornamento dei valori degli elementi della matrice e ci si occuperà in seguito di determinare una forma del modello adatta alla sua persistenza.

Si è scelto di tradurre il modello E-R in figura 3.2 in un diagramma UML delle classi (figura 3.5). Quest'ultima rappresentazione è facilmente implementabile con un linguaggio O-O e, di conseguenza, anche con Java.

La modellazione del diagramma UML segue pari pari il modello E-R. Di seguito, si tralascia di commentare in modo dettagliato il diagramma e ci si focalizza solamente sugli aspetti più rilevanti:

- Ad ogni entità del modello E-R corrisponde una classe o un interfaccia del diagramma delle classi UML. Per entrambi i modelli il medesimo fatto della realtà è modellato da un costrutto con il medesimo nome.
- A differenza del modello E-R, la classe `MMP` possiede gli attributi dell'entità *Mappa*. Infatti, nel linguaggio Java, un'interfaccia non possiede attributi.
- La classe `MP` possiede un attributo `matrice` che referenzia un array bidimensionale che costituisce la matrice della mappa. Esso è costituito da un'array principale (chiamato array "interno") i cui elementi referenziano altri array (chiamati array "esterni"). Al fine di garantire che l'array bidimensionale sia effettivamente una matrice, ogni array "esterno" deve avere lo stesso numero di elementi. Per tale motivo, il metodo `Crea Matrice` si occupa di costruire un'istanza dell'array bidimensionale che sia effettivamente (e dimensionalmente) una matrice.
- `matrice` è costituito da elementi di tipo `Elemento`. Questa interfaccia non include nessun metodo, quindi non impone nessuna restrizione alla sue implementazioni; in questo modo i metodi di `Matrice` possono essere usati con matrici costituite da diverse implementazioni di `Elemento`. Il ruolo dell'attributo `matrice` consiste nel replicare il legame dato dall'associazione *Ha* tra *MP* e *Elemento peso* del modello E-R.
- `Elemento peso` costituisce l'implementazione di `Elemento` sviluppato; essa rappresenta anche la modellazione dell'entità *Elemento peso*. Tuttavia, a differenza di quest'ultima, non sono presenti gli attributi x e y che identificano la posizione dell'elemento nella matrice: essa, infatti, è ricavabile direttamente dalla posizione dell'istanza di `Elemento peso` nell'array bidimensionale `matrice`.
- A differenza di `Elemento peso`, le classi `Elemento d'interesse` e `Elemento EPM` possiedono le coordinate `x` e `y` che consentono di identificare la loro posizione nella matrice. Questa esigenza nasce dai metodi `Add` e `Remove` di `Mappa`: poiché in ingresso è fornito solamente

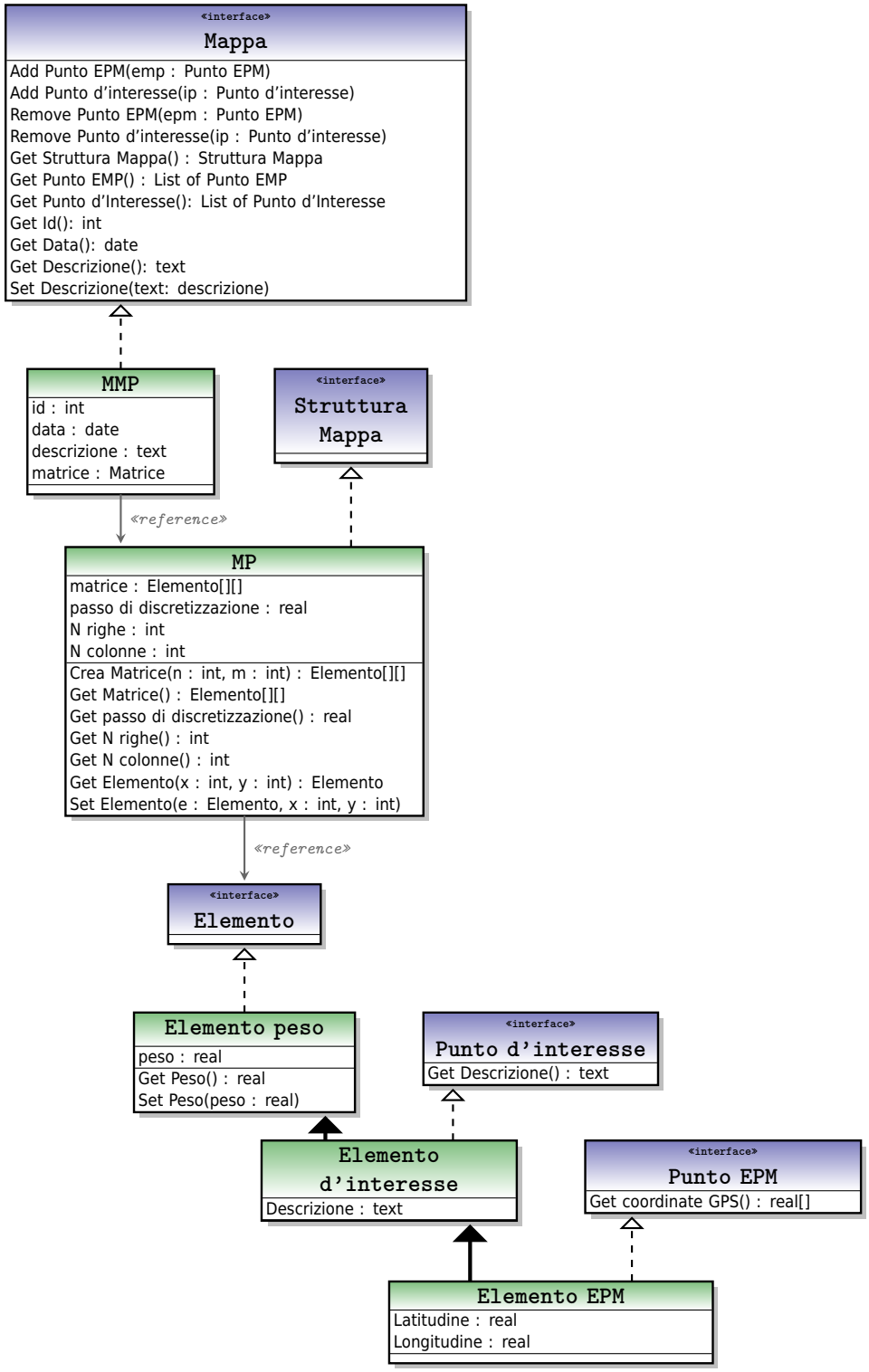


Figura 3.5: Modellazione di una *MMP* attraverso il modello delle classi UML.

l'oggetto d'interesse da (aggiungere o rimuovere) è necessario includere in esso la sua posizione all'interno di `matrice`. Per questo motivo si è scelto di replicare le coordinate `x` e `y` come attributi di `Elemento d'interesse`.

3.3.2 Progettazione logica

3.3.2.1 Ristrutturazione diagramma E-R

Poiché *SQLite* è in grado di manipolare un modello relazionale è necessario tradurre lo schema concettuale della base di dati (cioè il modello E-R) in uno schema logico che sia:

- Equivalente allo schema concettuale di partenza.
- Efficiente rispetto alle operazioni sulla base di dati previste: quindi è necessario valutare il loro carico sulla base di dati (ad es. la frequenza media di esecuzione di un'operazione, la quantità massima di dati richiesta...).
- Aderente alle specifiche dell'applicazione che si intende sviluppare (ad es. facile estensibilità del modello, riutilizzo del codice...).

Durante questa fase è necessario:

1. Ristrutturare lo schema E-R in modo che contenga solo costrutti semplici: entità e associazioni. Non sono ammesse generalizzazioni.
2. Tradurre le entità e le associazioni dello schema E-R in equivalenti schemi logici relazionali.

La ristrutturazione avviene secondo i seguenti passi:

1. Rimozione della generalizzazione tra *Mappa* e *MMP* (figura 3.6). In generale per rimuovere una generalizzazione è possibile procedere in tre modi:
   (a) Rimuovere le entità specializzanti:
      - Le entità figlie sono eliminate e le loro proprietà, attributi ed associazioni sono assegnate all'entità padre.
      - All'entità padre è aggiunto un ulteriore attributo per poter distinguere il tipo delle istanze (attributo discriminante che indica la sottoclasse a cui apparteneva ciascuna tupla).

   (b) Rimuovere le entità generalizzanti: viene rimossa l'entità padre e tutte le sue proprietà: l'identificatore, gli attributi e le associazioni sono ereditate dalle entità figlie.

   (c) Introduzione di relazioni *IS-A*:
      - La generalizzazione è sostituita da associazioni uno-a-uno che legano le entità figlie all'entità padre.
      - Le entità figlie sono identificate esternamente dall'entità padre: poiché la generalizzazione è totale, per ogni istanza delle entità figlie deve esservi un'istanza collegata dell'entità padre.

   Per la ristrutturazione del modello E-R si è scelto di adottare la tecnica del punto 1b. L'eliminazione della generalizzazione tra *Mappa* e *MMP* conduce alla eliminazione di *Mappa*; ciò è possibile poiché la generalizzazione è totale ed esclusiva. *MMP* eredità gli attributi di *Mappa*.

2. Accorpamento delle entità *MMP* e *MP* (figura 3.7). Poiché esiste un'associazione con cardinalità $(1, 1)$ da ambo le parti, si scegli di accorpare le due entità. La nuova entità, chiamata nuovamente *MP*, include gli attributi e le associazioni di entrambe le entità.

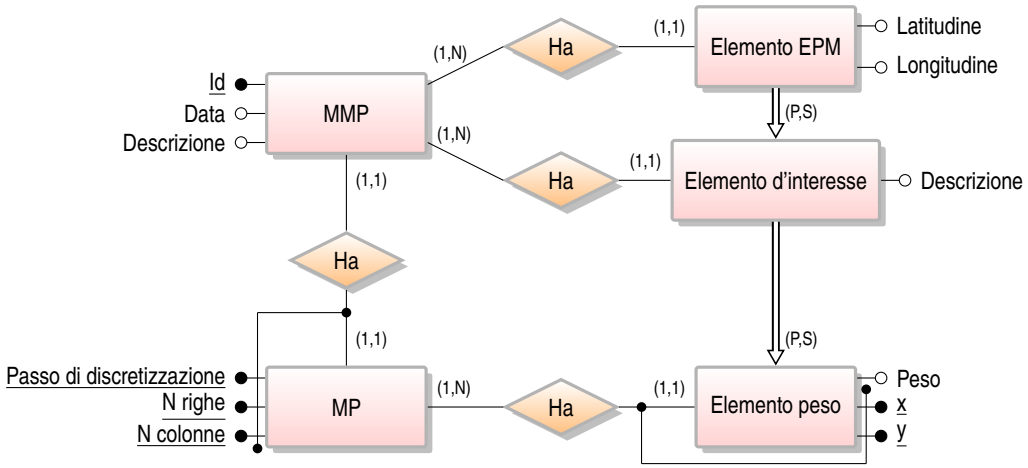


Figura 3.6: Rimozione della generalizzazione tra *Mappa* e *MMP*. Quest'ultima eredità gli attributi di *Mappa*.

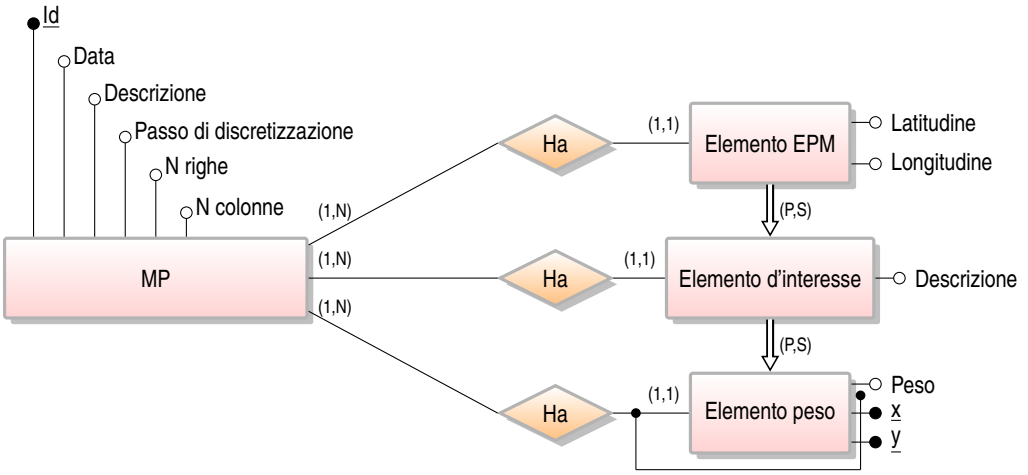


Figura 3.7: Accorpamento delle entità *MMP* e *MP* in una nuova entità chiamata nuovamente *MP*.

3. Conversione della generalizzazione parziale tra *Elemento d'interesse* e *Elemento EPM* in una generalizzazione totale ed esclusiva (figura 3.8). È sempre possibile convertire una generalizzazione parziale in una esclusiva aggiungendo una nuova entità specializzante. Nel caso d'interesse, si introduce l'entità *Elemento Goal* le cui occorrenze appartengono ad *Elemento d'interesse* ma non a *Elemento EPM* (di fatto, gli insiemi di occorrenze di *Elemento Goal* e *Elemento d'interesse* coincidono).

4. L'eliminazione della nuova generalizzazione tra *Elemento d'interesse*, *Elemento EPM* e *Elemento Goal* (figura 3.9). Le entità *Elemento EPM* ed *Elemento Goal* ereditano gli attributi e l'associazione di *Elemento d'interesse*. Si noti che la doppia associazione *Ha* tra MMP e *Elemento EPM* ha la stessa semantica (e la stessa cardinalità): pertanto una di essa viene eliminata.

5. Conversione della generalizzazione parziale tra *Elemento peso*, *Elemento EPM* ed *Elemento Goal* in una generalizzazione totale ed esclusiva (figura 3.10). La nuova entità creata, che contiene le stesse occorrenze di *Elemento peso* che non fanno parte di *Elemento EPM* ed *Elemento Goal*, è chiamata *Elemento peso no EPM/Goal*.

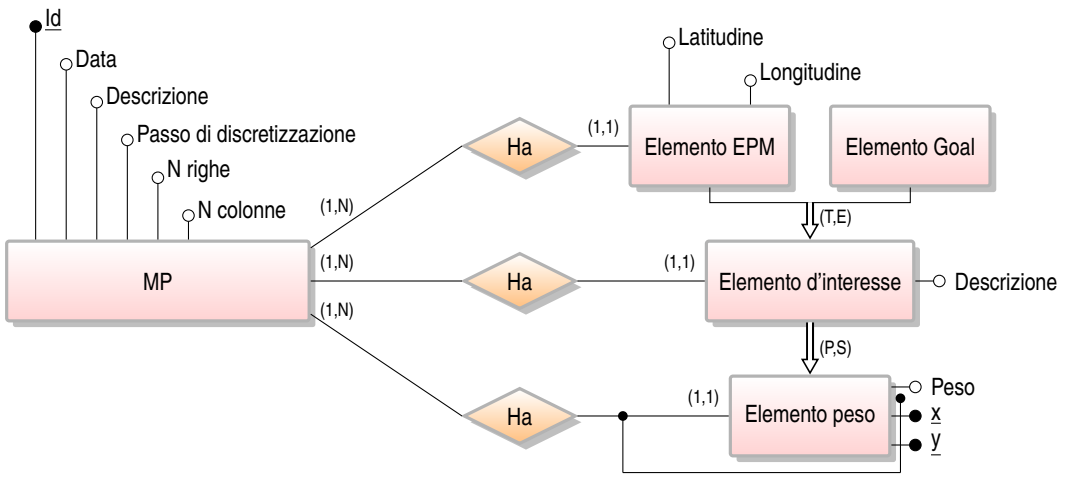


Figura 3.8: Conversione della generalizzazione parziale tra *Elemento d'interesse* e *Elemento EPM* in una generalizzazione totale ed esclusiva.

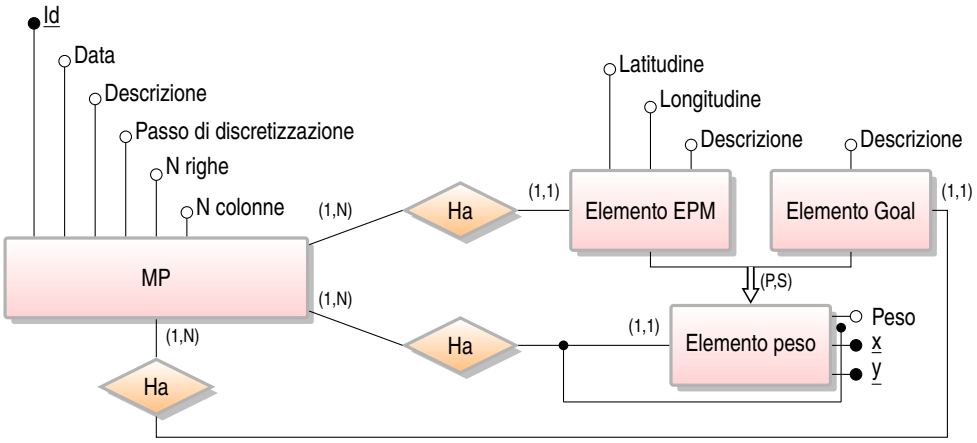


Figura 3.9: L'eliminazione della nuova generalizzazione tra *Elemento d'interesse*, *Elemento EPM* e *Elemento Goal*.

6. L'eliminazione della nuova generalizzazione tra *Elemento peso*, *Elemento EPM*, *Elemento Goal* ed *Elemento peso no EPM/Goal* (figura 3.11). Le entità *Elemento* EPM, *Elemento Goal* e *Elemento peso no EPM/Goal* ereditano gli attributi e le associazioni di *Elemento peso*. Anche in questo caso sono rimosse le associazioni con la stessa semantica.

7. Eliminazione dell'entità *Elemento peso no EPM/Goal*, degli attributi *peso* e aggiunta dell'attributo *Raw* (figura 3.12). Si ipotizza che, quando l'applicazione richiede l'accesso ad una mappa, la sua matrice venga caricata in tutti i suoi elementi (in futuro tale esigenza potrà cambiare, specie se la mappe assumono grandi dimensioni): questo impone che la quantità di memoria centrale dello smartphone sia sufficientemente grande per contenere la mappa. Si decide quindi di:

   - Rimuovere l'entità *Elemento peso no EPM/IP* e tutte le associazioni a cui partecipa.
   - Rimuovere gli attributi *peso* delle entità *Elemento EPM* e *Elemento Goal*.
   - Inserire l'attributo *Raw* nell'entità *MP* che contiene una sequenza binaria che corrisponde alla serializzazione dei valori w della matrice di una *MP*. Il meccanismo di

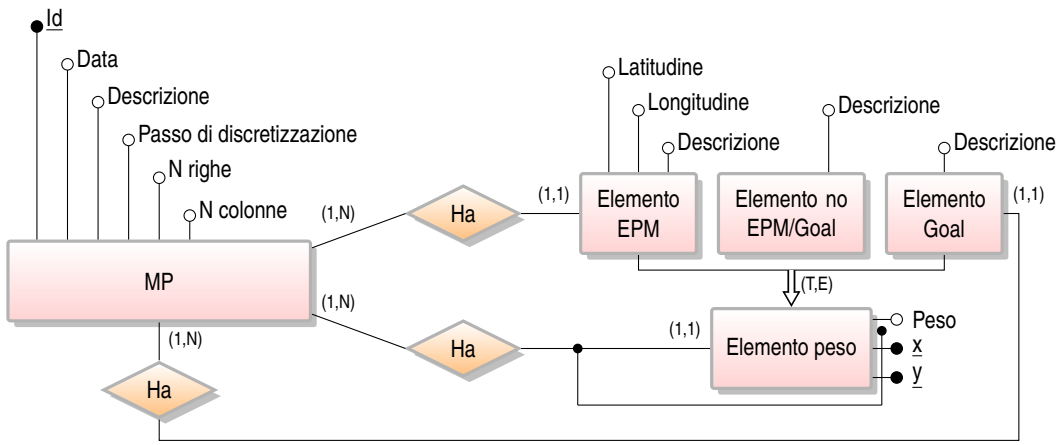


Figura 3.10: Conversione della generalizzazione parziale tra *Elemento peso*, *Elemento EPM* ed *Elemento Goal* in una generalizzazione totale ed esclusiva.

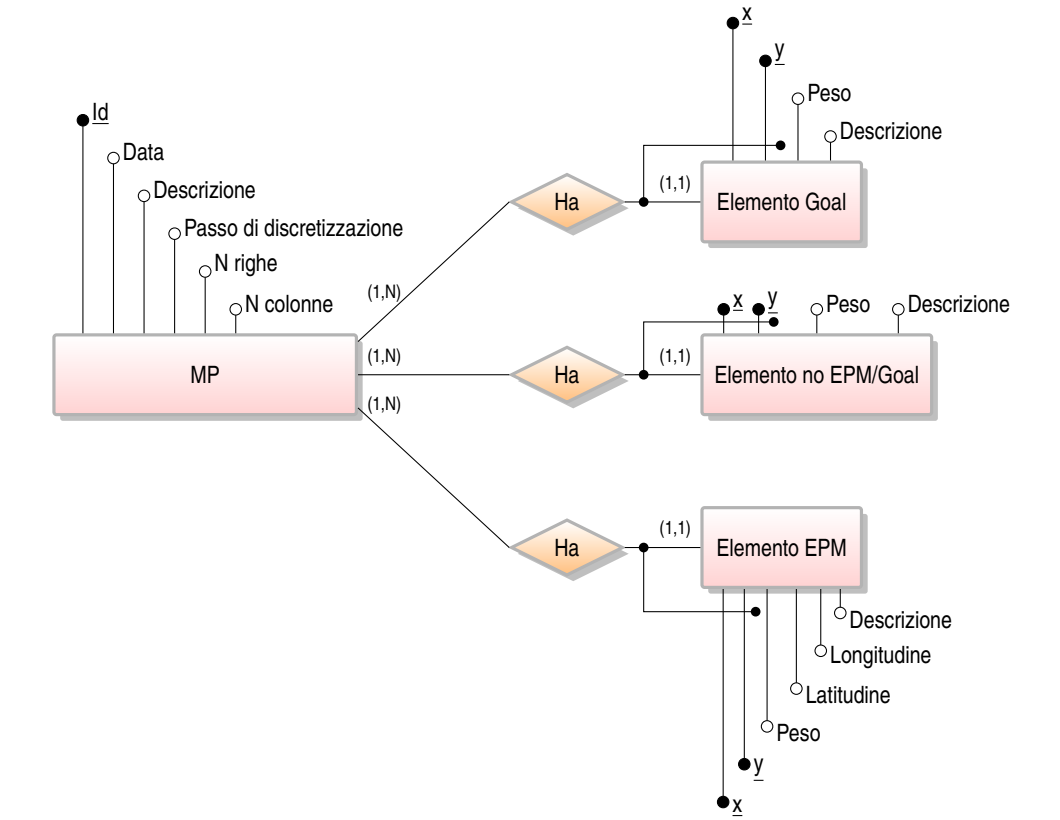


Figura 3.11: L'eliminazione della nuova generalizzazione tra *Elemento peso*, *Elemento EPM*, *Elemento Goal* ed *Elemento peso no EPM/Goal*.

serializzazione e deserializzazione è realizzato esternamente al DBMS. In sintesi si deve provvedere a:

(a) Convertire ordinatamente il peso w di ogni elemento della matrice in un array di bit. In seguito (sezione 3.3.3.5) si specifica come deve avvenire il processo di

serializzazione (ad es. se inserire o meno anche le coordinate x o y dell'elemento).

(b) Salvare l'array di bit nell'attributo *Raw* di *MP*.

(c) Caricare, quando è richiesto, l'attributo *Raw* di *MP*

(d) Deserializzare l'array e allocare la struttura dati corrispondente per rappresentare la matrice (cioè l'array bidimensionale `matrice` della classe `MP`).

Utilizzare un meccanismo di serializzazione e deserializzazione ha il vantaggio di:

- Richiedere minor occupazione di memoria poiché:
  - Il sistema che si adotta per la persistenza può occupare ulteriore memoria oltre a quella effettivamente richiesta dai dati (ad es. DBMS che gestisce un database che contiene 10 record da 10 KB ciascuno richiede generalmente un'occupazione di memoria maggiore di 100 KB).
  - Se una matrice è popolata in quasi tutti i suoi elementi, una serializzazione "intelligente" potrebbe determinare un ordine degli elementi della matrice e serializzare solamente il peso w degli elementi seguendo quell'ordine: se il deserializzatore conosce tale ordine non è necessario salvare le coordinate x e y che individuano la posizione dell'elemento nella matrice.
- Migliore efficienza nell'accesso della memoria. Il salvataggio di un'array anziché di svariate record isolati, impone la sequenzialità dei dati nella memoria secondaria. Poiché le prestazioni delle memorie flash utilizzate nei moderni smartphone variano significativamente a seconda della modalità d'accesso (sequenziale o random) [46], è opportuno limitare il numero di accessi casuali alla memoria.

Tuttavia è bene sottolineare che, adottando il meccanismo di serializzazione e deserializzazione, si perde la possibilità di modellare (a livello dati) un singolo elemento della matrice che non sia un *EPM* o un *IP*.

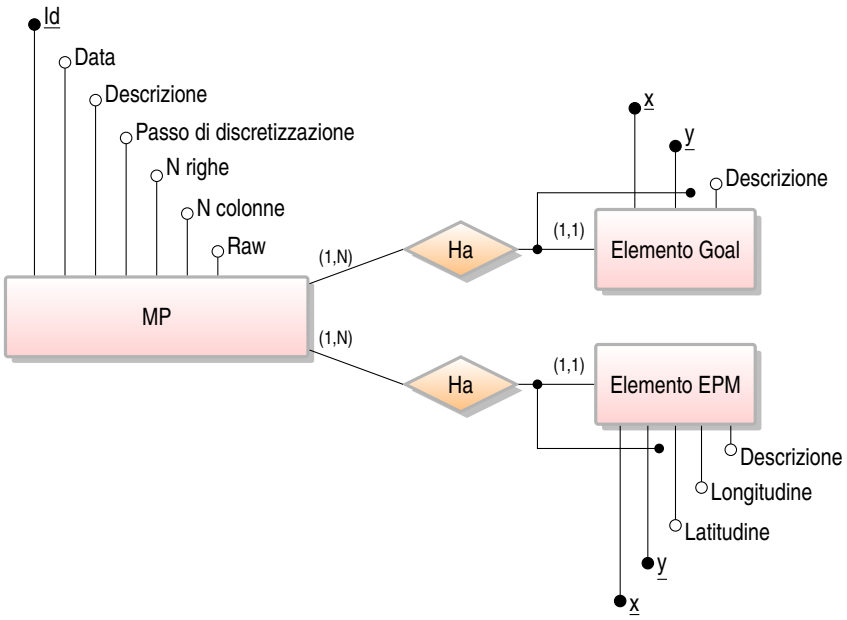


Figura 3.12: Eliminazione dell'entità *Elemento peso no EPM/Goal*, degli attributi *peso* e aggiunta dell'attributo *Raw*.

8. Partizionamento verticale dell'entità *MP* (figura 3.13). Poiché si stima che l'attributo *Raw* abbia un'occupazione di memoria molto elevata rispetto gli altri attributi (ad es. se una MP ha 10000 elementi tali che il peso w di ogni elemento richieda 4 bytes, il campo *Raw* necessita

di almeno 40 KB di memoria rispetto alle poche decine di bytes necessari per salvare tutti gli altri record associati alla *MP*) si preferisce:

- Aggiungere una nuova entità debole *MP Raw*.
- Aggiungere l'associazione *Ha* tra *MP* e *MP Raw*, con cardinalità $(1, 1)$ da ambo i lati.
- Trasferire l'attributo *Raw* da *MP* a *MP Raw*.

Questa soluzione consente di evitare di coinvolgere direttamente il campo *Raw* in eventuali query o join tra relazioni, rallentando di conseguenza le prestazioni del sistema.

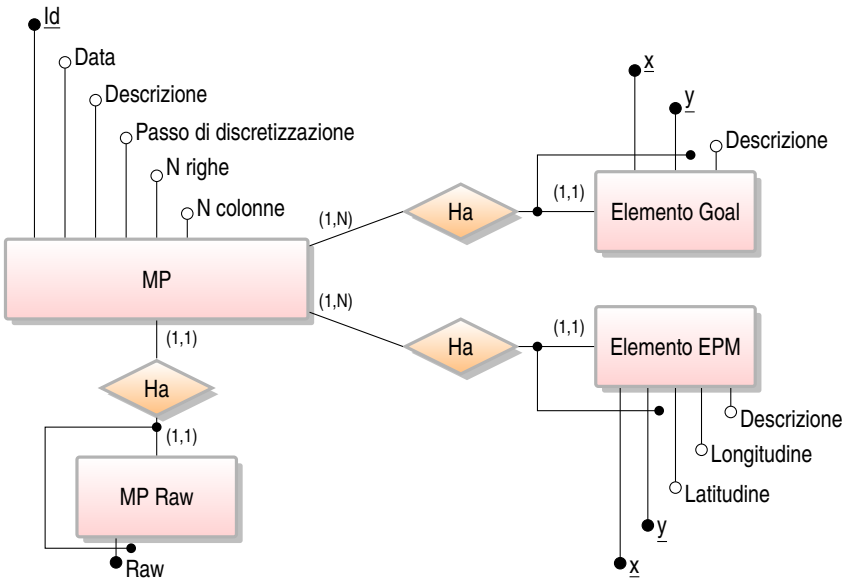


Figura 3.13: Partizionamento verticale dell'entità *MP*.

Lo schema in figura 3.13 costituisce il modello E-R definitivo su cui realizzare la persistenza.

3.3.2.2 Modello logico relazionale

Lo scopo di questa fase consiste nel determinare un "mapping" da modello E-R ristrutturato (figura 3.13) a modello logico relazionale. Si seguono le seguenti regole:

1. Entità: è tradotta in una relazione con lo stesso nome e gli stessi attributi. L'identificatore dell'entità costituisce la chiave della relazione.
2. Relazioni binarie 1 a N: le entità partecipanti sono mappate in equivalenti schemi relazionali in analogia con il punto 1. Alle entità con la partecipazione N nella relazione (cioè *EPM* e *Goal*) è aggiunto l'attributo *Id mappa* che costituisce la chiave esterna.

Il modello relazione prodotto è indicato in figura 3.14. I tipi di dato di ogni attributo sono indicati nella tabella 3.1. Si è scelto:

- Un tipo di dato *real* per *Latitudine* e *Longitudine* in modo tale da consentire al DBMS di selezionare, eventualmente, un intorno di una coordinata GPS fornita in ingresso (il valore fornito dallo smartphone potrebbe differire rispetto all'acquisizione presente nella sorgente dati nonostante si tratti dello stesso punto fisico).
- Un tipo di dato *text* per la *Data* di acquisizione della mappa poiché, per il momento, non si prevedono particolari elaborazioni.

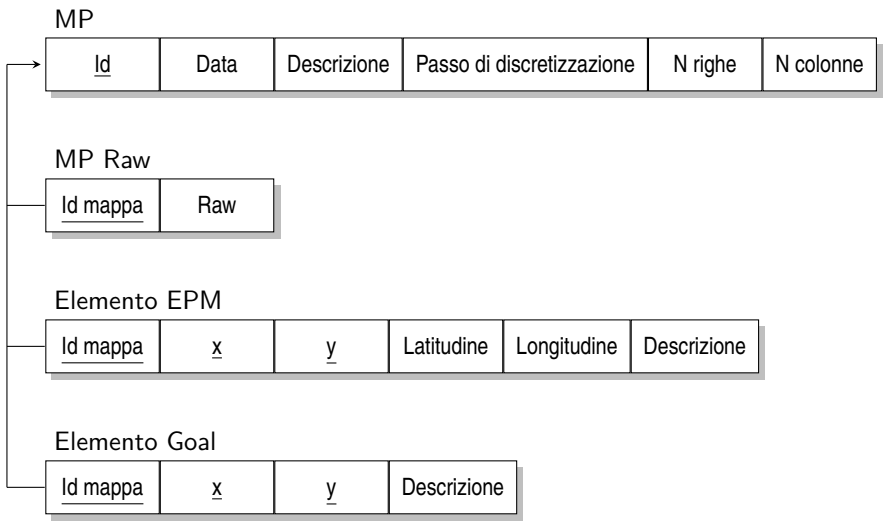


Figura 3.14: Modello relazionale.

Attributo	Tipo di dato
Id	text
Data	text
Descrizione	text
Passo di discretizzazione	int
N righe	int
N colonne	int

(a) MP

Attributo	Tipo di dato
Id mappa	text
Raw	blob

(b) MP Raw

Attributo	Tipo di dato
Id mappa	text
x	int
y	int
Latitudine	real
Longitudine	real
Descrizione	text

(c) Elemento EPM

Attributo	Tipo di dato
Id mappa	text
x	int
y	int
Descrizione	text

(d) Elemento Goal

Tabella 3.1: Tipi di dato degli attributi del modello relazionale.

3.3.3 Data Access Object

Il successivo stadio della progettazione consiste nel consentire l'accesso alle istanze delle classi del diagramma UML in figura 3.5 alle relazioni del modello in figura 3.14 salvate con *SQLite*. Tuttavia è bene sottolineare il seguente aspetto: la persistenza dei dati può essere in futuro implementata attraverso differenti meccanismi, i quali utilizzano diverse API. È bene fin da subito evitare di introdurre codice che si occupa dell'accesso ai dati per un particolare meccanismo di persistenza all'interno delle classi che modellano la mappa (cioè nelle classi rappresentate in figura 3.5) poiché tale dipendenza rende più difficile e tediosa un'eventuale migrazione verso un altro meccanismo di persistenza; infatti, quando la sorgente dati cambia, i componenti dell'applicazione necessitano di essere cambiati per adattarsi ai nuovi tipi di dato della sorgente.

Per tale motivo, si è scelto di utilizzare il design pattern chiamato Data Access Object (DAO) per astrarre ed incapsulare tutti gli accessi alla sorgente dati [47]: esso agisce come un "adattatore" tra il componente software (detto anche client) che necessita di accedere ai dati e la sorgente dati. Generalmente il pattern DAO è implementato attraverso un'interfaccia, che espone le operazioni concesse sui dati soggetti al meccanismo di persistenza, e da una o più implementazioni dell'interfaccia, ognuna delle quali consente l'accesso ad uno specifico meccanismo di persistenza. In questo modo, il client utilizza l'interfaccia esposta dal DAO per accedere alla sorgente dati, indipendentemente dal meccanismo di persistenza adottato dalla sorgente dati. Poiché l'interfaccia esposta dal DAO al client non cambia quando l'implementazione dell'accesso alla sorgente dati sottostante è modificata, questo pattern permette la costruzione di diverse classi DAO che implementano differenti meccanismi di persistenza dei dati.

Nelle figure 3.15 e 3.16 sono rappresentati il diagramma delle classi e il diagramma di sequenza per il pattern DAO. Nel dettaglio si hanno quattro partecipanti:

- Il *Client* è l'oggetto che richiede l'accesso ai dati della sorgente.
- Il *DAO* è l'oggetto che incapsula i dettagli relativi l'accesso alla sorgente dati. Le operazioni di caricamento e salvataggio dei dati sono trasparenti per il client.
- La *sorgente dati*, che può essere un DBMS, un file, un servizio di cloud storage…
- Il *Transfer object class* è l'oggetto che mantiene i dati caricati dalla sorgente dati per opera del *DAO*. Su tale oggetto, il client andrà ad invocare metodi per l'interesse dell'applicazione.

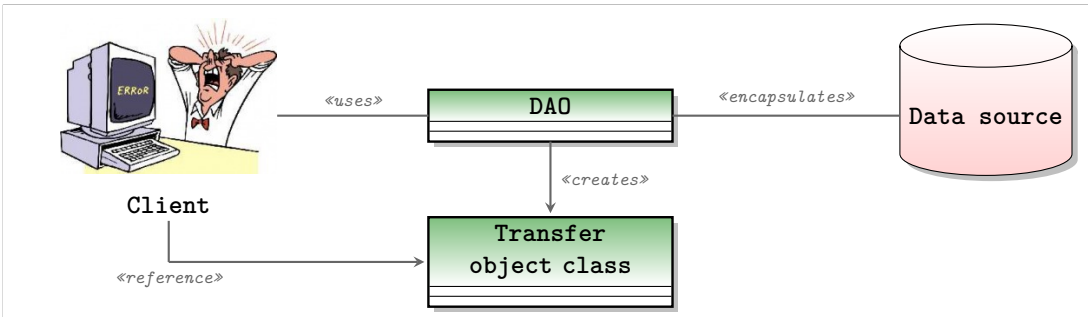


Figura 3.15: Diagramma della classi del pattern DAO.

Si sceglie di sviluppare un DAO per ogni relazione (`DAO MP`, `DAO MP Raw`, `DAO Elemento EPM` e `DAO Elemento Goal`) ed il `DAO MMP` che ha il compito di consentire l'esecuzione delle operazioni espresse nelle specifiche (caricamento, salvataggio e cancellazione di una *MMP* componendo le funzioni degli altri DAO.

3.3.3.1 Estensione con il design pattern abstract factory

Prima di descrivere la progettazione di ogni singolo DAO, si introduce una possibile estensione attraverso il design pattern *abstract factory* [48]. Esso consiste nello sviluppare un'interfaccia che consenta la creazione di famiglie di oggetti con caratteristiche comuni senza dover necessariamente specificare la concreta implementazione. Ad esempio, si supponga di avere uno stesso dataset salvato attraverso meccanismi di persistenza diversi, tra cui *SQLite* e *BaseX* (è un DBMS con supporto per *XML*). Inoltre si supponga di aver implementato i relativi DAO per accedervi. L'*abstract factory* definisce quale famiglia di DAO devono essere istanziati (ad es. `DAO leggi`) ma non quale concreta implementazione del DAO deve essere restituita (ad es. `DAO leggi SQLite` o `DAO leggi BaseX`): è compito del client richiedere una specifica "factory" (classe che crea altri oggetti, cioè DAO nel contesto dell'applicazione) oppure del sistema, il quale fornisce l'implementazione disponibile per una data richiesta. Si chiarisce il design pattern *abstract factory* e la sua interazione con il pattern DAO attraverso le figure 3.17 e 3.18.

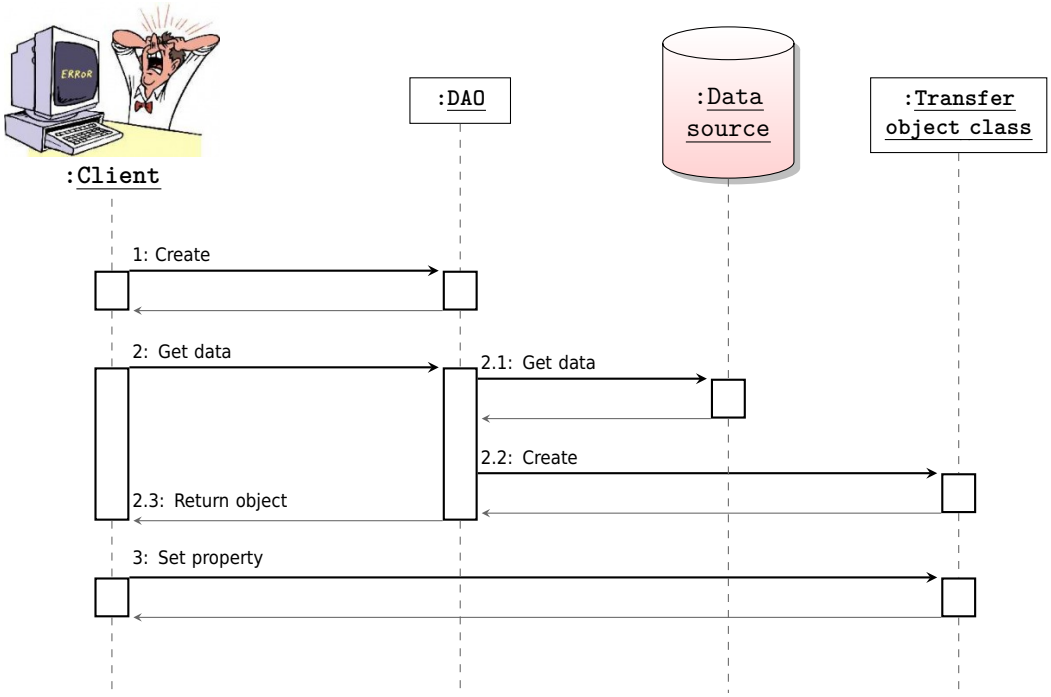


Figura 3.16: Diagramma delle sequenze semplificato che illustra l'interazione tra i partecipanti del pattern DAO.

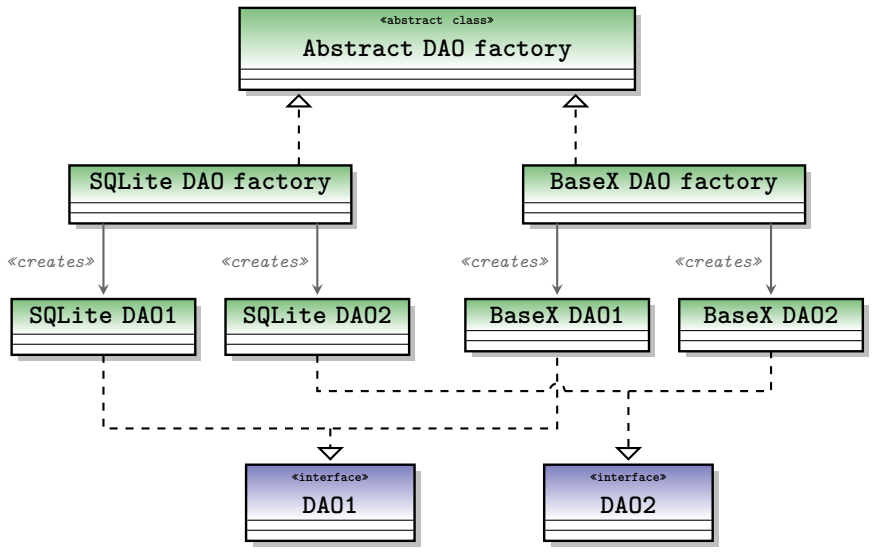


Figura 3.17: Diagramma delle classi del design patter abstract factory DAO.

3.3.3.2 Salvataggio di una mappa

Il salvataggio di una *MMP* avviene mediante il `DAO MMP`. Esso ha il compito di allocare le istanze per gli altri quattro DAO ed invocare, in sequenza, i metodi per inserire un nuovo record nelle relazioni. Il diagramma delle sequenze per questa operazione è rappresentato in figura 3.19; è interessante notare che tale diagramma evidenzia come il client debba utilizzare il DAO per salvare una mappa.

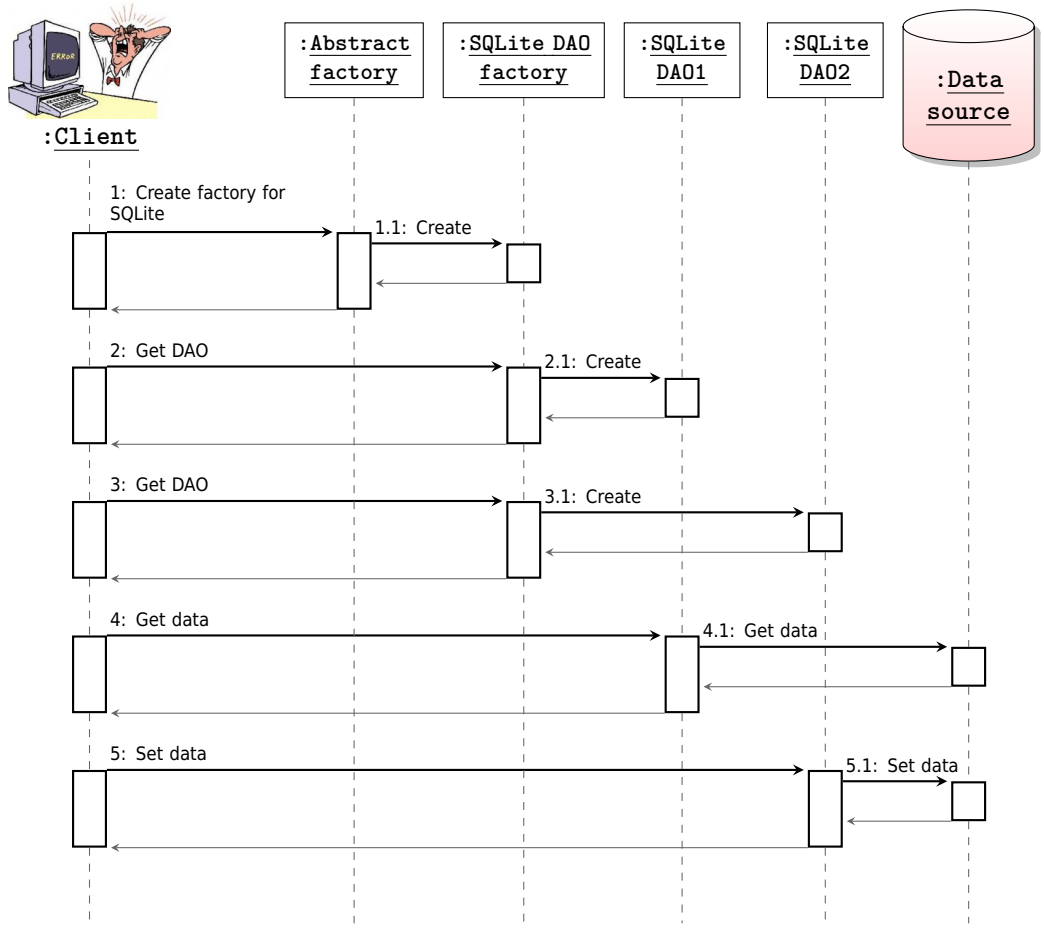


Figura 3.18: Diagramma delle sequenze semplificato del design pattern abstract factory DAO.

`DAO MP`, `DAO MP Raw`, `DAO Elemento EPM` e `DAO Elemento Goal` hanno il compito di salvare un nuovo record nella tabella di loro competenza seguendo la seguente sequenza:

1. Estrazione degli attributi d'interesse (dalle istanze delle classi in figura 3.5).
2. Creazione di un nuovo record.
3. Inserimento nella tabella appropriata.

Lo pseudocodice dei rispettivi DAO relativo all'inserimento di record nelle tabelle è indicato negli algoritmi 3.1, 3.2, 3.3 e 3.4.

Si ricorda che l'inserimento di nuovi dati nel database non deve violare i vincoli d'integrità. Data la tupla t inserita in *MP Raw*, sia $t[id\,mappa] = k$ il valore dell'identificatore della mappa. Per non violare il vincolo d'integrità referenziale, è necessario che esista nella tabella *MP* un record r tale che $r[id\ mappa] = k$ affinché l'inserimento vada a buon fine. In caso contrario, *SQLite* lancerà un'eccezione. Considerazioni analoghe sono valide per la chiave esterna $id\ mappa$ delle relazioni *Elemento EPM* e *Elemento Goal*.

3.3.3.3 Caricamento di record mediante DAO

Il caricamento di una *MMP* data la coppia di coordinate $gps = (lat, lon)$ avviene mediante il `DAO MMP`. Indicativamente, è necessario eseguire le operazioni esposte nell'algoritmo 3.5. Si noti

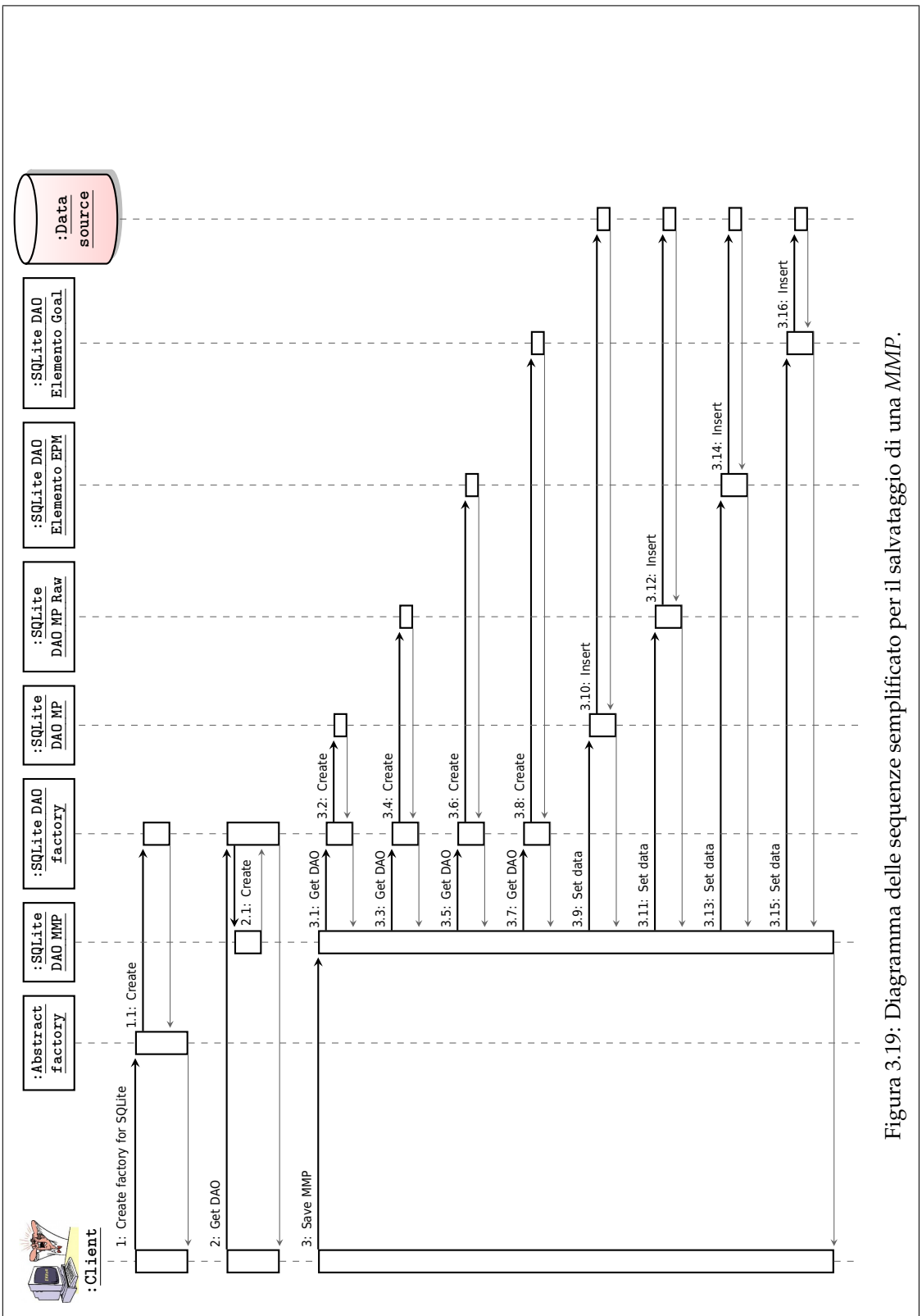


Figura 3.19: Diagramma delle sequenze semplificato per il salvataggio di una *MMP*.

Algoritmo 3.1 Salva in *MP*.

Input: *Id*, *data*, *descrizione* `matrice`
Output: Inserisce un nuovo record in *MP*

1: **function** SALVA IN MP(*Id*, *data*, *descrizione*, `matrice`)
2: *Passo di discretizzazione* ← GET PASSO DI DISCRETIZZAZIONE(`matrice`)
3: *N righe* ← GET N RIGHE(`matrice`)
4: *N colonne* ← GET N COLONNE(`matrice`)
5: *r* ← CREA RECORD PER MP(*Id*, *data*, *descrizione*, *Passo di discretizzazione*, *N righe*, *N colonne*)
6: INSERISCI IN MP(*r*)

Algoritmo 3.2 Salva in *MP Raw*.

Input: *Id mappa*, *mappa*
Output: Inserisce un nuovo record in *MP Raw*

1: **function** SALVA IN MP RAW(*Id mappa*, *mappa*)
2: *m* ← GET MATRICE(*mappa*)
3: *array* ← SERIALIZZA(*m*)
4: *r* ← CREA RECORD PER MP RAW(*Id*, *array*)
5: INSERISCI IN MP RAW(*r*)

Algoritmo 3.3 Salva in *Elemento EPM*.

Input: *Id mappa*, `elemento EPM`
Output: Inserisce un nuovo record in *Elemento EPM*

1: **function** SALVA IN ELEMENTO EPM(*Id mappa*, `elemento EPM`)
2: *x* ← GET X(`elemento EPM`)
3: *y* ← GET Y(`elemento EPM`)
4: *Latitudine* ← GET LATITUDINE(`elemento EPM`)
5: *Longitudine* ← GET LONGITUDINE(`elemento EPM`)
6: *Descrizione* ← GET DESCRIZIONE(`elemento EPM`)
7: *r* ← CREA RECORD PER ELEMENTO EPM (*Id mappa*, *x*, *y*, *Latitudine*, *Longitudine*, *Descrizione*)
8: INSERISCI IN ELEMENTO EPM(*r*)

Algoritmo 3.4 Salva in *Elemento Goal*.

Input: *Id mappa*, `elemento d'interesse`
Output: Inserisce un nuovo record in *Elemento Goal*

1: **function** SALVA IN *Elemento Goal*(*Id mappa*, `elemento d'interesse`)
2: *x* ← GET X(`elemento d'interesse`)
3: *y* ← GET Y(`elemento d'interesse`)
4: *Descrizione* ← GET DESCRIZIONE(`elemento d'interesse`)
5: *r* ← CREA RECORD PER ELEMENTO GOAL(*Id mappa*, *x*, *y*, *Descrizione*)
6: INSERISCI IN ELEMENTO GOAL(*r*)

che possono esistere più record di *Elemento EPM* restituiti dall'istruzione numero 2. Infatti possono esistere più mappe che rappresentano una zona di territorio individuata da una stessa coppia di coordinate GPS.

Algoritmo 3.5 Carica *MMP*.

Input: *Latitudine, Longitudine*
Output: Un insieme M di istanze di `MMP` con i dati caricati dal database.

1: **function** CARICA MMP(*lat, lon*)
2: $r \leftarrow$ records di *Elemento EPM* tali che per ogni $r_i \in r$ si ha che
r_i[*Latitudine, Longitudine*] = (*lat, lon*)
3: **for all** $r_i \in r[i]$ **do**
4: $id\ m \leftarrow r_i$[*Id mappa*] ▷ Estrae da r_i il valore dell'identificatore *Id mappa* della mappa.
5: *MP* ←CARICA DA MP(*Id m*)
6: *MP Raw* ←CARICA DA MP RAW(*Id m*)
7: *EPMs* ←CARICA DA ELEMENTO EPM(*Id m*)
8: *Goals* ←CARICA DA ELEMENTO GOAL(*Id m*)
9: $M \leftarrow M \cup$ CREA NUOVA MMP(*MP, MP raw, EPMs, Goals*)
10: **return** M

Si valuta l'impatto di due possibili implementazioni dell'algoritmo 3.5:

1. Caricamento "manuale": si eseguono in sequenza le istruzioni dell'algoritmo. Ciò richiede l'implementazione di una funzione "carica" per `DAO MP`, `DAO MP Raw`, `DAO Elemento EPM` e `DAO Elemento Goal` che interroghi le rispettive tabelle ed estragga i dati d'interesse.

2. Caricamento tramite DBMS: a differenza del punto precedente, è possibile delegare al DBMS l'operazione di ricerca ed estrazione dei record d'interesse. In questo caso, tramite l'utilizzo del linguaggio dichiarativo d'interrogazione SQL, è necessario indicare solo i "dati" da estrarre e non "come" questi debbano essere ottenuti.

I due approcci offrono vantaggi e svantaggi diversi:

- Righe di codice: l'approccio 2 richiede la scrittura di un minor numero di righe di codice poiché è necessario specificare solo "quali" dati sono necessari ma non "come" ottenerli.

- "Pulizia" dei record estratti. I record estratti dalle singole interrogazioni delle tabelle dell'approccio 1 contengono solo i dati della tabella a cui si riferiscono; non è necessaria nessuna elaborazione. Viceversa, i record restituiti dall'approccio 2 necessitano di una post elaborazione per estrarre i contenuti d'interesse.

- Concorrenza. Sebbene sia improbabile l'uso concorrente del database per opera di quest'applicazione, un'eventuale concorrenza di accessi con l'approccio 2 andrà regolamentata utilizzando gli strumenti del DBMS. Viceversa, con l'approccio 1, è necessario scrivere del nuovo codice di controllo.

```
1  SELECT MP.Id,
2  MP.Data,
3  MP.Descrizione,
4  MP.Passo di discretizzazione,
5  MP.N righe,
6  MP.N colonne,
7  Elemento EPM.x AS x EPM,
8  Elemento EPM.y AS y EPM,
9  Elemento EPM.Latitudine,
10 Elemento EPM.Longitudine,
11 Elemento EPM.Descrizione AS Descrizione EPM,
12 Elemento Goal.x AS x Goal,
13 Elemento Goal.y AS y Goal,
14 Elemento Goal.Descrizione AS Descrizione Goal
15 FROM MP, Elemento EPM, Elemento Goal
16 WHERE MP.Id IN (
17   SELECT Elemento EPM.Id mappa
18   FROM Elemento EPM
19   WHERE Elemento EPM.Latitude = ? AND Elemento EPM.Longitude = ?)
20 ORDER BY MP.id, Elemento EPM.x, Elemento EPM.y ASC;
```

Listato 3.1: Bozza di codice *SQL* che interroga le tabelle *MP*, *Elemento EPM* ed *Elemento Goal* per estrarre i dati di interesse per l'operazione carica *MMP*. La tabella *MP Raw* non è coinvolta.

Si è scelto di sviluppare un'implementazione intermedia tra i due approcci:

1. In accordo con l'approccio 2, si scrive un query per ottenere tutti i dati d'interesse ad eccezione dell'attributo *Raw* di *MP Raw* (listato 3.1). Essa è composta da due query annidate: l'interrogazione interna (dalla riga 17 alla riga 19) estrae gli *Id mappa* d'interesse da *Elemento EPM* (riga 2 dell'algoritmo 3.5) mentre la query esterna estrae tutti i record inerenti alle mappe tali che il valore del loro *Id* è uguale al valore di *Id mappa* estratto dalla query interna. Il carattere "?" è sostituito da un valore numerico reale in fase di esecuzione della query.

   Dato il dataset in figura 3.20, l'esecuzione di questa query fornisce il result set in figura 3.21; si noti la presenza di dati ripetuti. È necessario quindi una scansione ad hoc per estrarre i dati di interesse.

2. In accordo con l'approccio 1, per ogni *Id mappa* ricavato dal punto precedente si interroga la tabella *MP Raw* per recuperare il valore del campo *Raw*.

La motivazione di questa scelta consiste nel determinare un compromesso tra la quantità di codice da scrivere e la velocità d'esecuzione dell'interrogazione. Poiché, come già notato, la dimensione in bytes del valore del campo *Raw* può essere molto grande in confronto a tutti gli altri valori degli altri attributi, si è preferito non includere il suo accesso direttamente nella query principale, per non appesantire il result set restituito. Il riassunto dell'operazione di *carica MMP* è esposto in figura 3.22.

3.3.3.4 Eliminazione di record mediante DAO

Anche la cancellazione di una *MMP* data la coppia di coordinate $gps = (lat, lon)$ avviene mediante il `DAO MMP`, in analogia con l'operazione di caricamento (algoritmo 3.6 e figura 3.24).

L'istruzione 5 è responsabile della cancellazione di un record dalla relazione *MP*: tuttavia tale operazione elimina l'intera mappa. Infatti, il vincolo d'integrità referenziale sulle chiavi esterne, assicura che, quando un record r di *MP* è eliminato, anche i record di *MP Raw*, *Elemento* EPM e *Elemento Goal* che hanno una referenza ad r siano eliminati (figura 3.23).

MP

Id	Data	Descrizione	Passo di discretizzazione	N righe	N colonne
1	05-09-2014	Mappa A	1	100	100
2	05-09-2014	Mappa B	0.5	250	500
3	02-09-2014	Mappa C	0.5	500	500
…	…	…	…	…	…

Elemento EPM

Id mappa	x	y	Latitudine	Longitudine	Descrizione
1	5	91	45.222123	11.336908	Entrata nord
1	80	12	45.222088	11.336208	Entrata sud
2	244	49	34.441112	-116.668542	Scalinata
3	11	221	36.668721	137.668650	Entrata ovest
3	126	418	36.668734	137.668664	Entrata nord
…	…	…	…	…	…

Elemento Goal

Id mappa	x	y	Descrizione
1	44	18	Bar
2	114	74	Servizi igenici
2	78	350	Auditorium
2	110	191	Accettazione
3	221	358	Bar
3	28	441	Informazioni
3	48	55	Bancomat
3	150	185	Supermarket
3	255	300	Edicola
…	…	…	…

MP Raw

Id mappa	Raw
1	1001…
2	1101…
3	0001…
…	…

Figura 3.20: Dataset di esempio.

Algoritmo 3.6 Elimina *MMP*.

Input: *Latitudine, Longitudine*
Output: L'eliminazione di un *MMP* dal database.

1: **function** ELIMINA *MMP*(*lat*, *lon*)
2: $r \leftarrow$ record r di *Elemento EPM* tali per ogni $r_i \in r$ si ha che $r_i[Latitudine, Longitudine] = (lat, lon)$.
3: **for all** $r_i \in r[i]$ **do**
4: *Id* $m \leftarrow r_i[Id\ mappa]$ ▷ Estrae da r_i il valore dell'identificatore *Id mappa* della mappa.
5: ELIMINA DA MP(*Id* m)

Id	Data	Descr. mappa	Passo di discr.	N righe	N colonne	x EPM	y EPM	Latitudine	Longitudine	Descr. EPM	x Goal	y Goal	Descr. Goal
3	02-09-2014	Mappa C	0.5	500	500	11	221	36.668721	137.668650	Entrata ovest	221	358	Bar
3	02-09-2014	Mappa C	0.5	500	500	11	221	36.668721	137.668650	Entrata ovest	28	441	Informazioni
3	02-09-2014	Mappa C	0.5	500	500	11	221	36.668721	137.668650	Entrata ovest	48	55	Bancomat
3	02-09-2014	Mappa C	0.5	500	500	11	221	36.668721	137.668650	Entrata ovest	150	185	Supermarket
3	02-09-2014	Mappa C	0.5	500	500	11	221	36.668721	137.668650	Entrata ovest	255	300	Edicola
3	02-09-2014	Mappa C	0.5	500	500	126	418	36.668734	137.668664	Entrata nord	221	358	Bar
3	02-09-2014	Mappa C	0.5	500	500	126	418	36.668734	137.668664	Entrata nord	28	441	Informazioni
3	02-09-2014	Mappa C	0.5	500	500	126	418	36.668734	137.668664	Entrata nord	48	55	Bancomat
3	02-09-2014	Mappa C	0.5	500	500	126	418	36.668734	137.668664	Entrata nord	150	185	Supermarket
3	02-09-2014	Mappa C	0.5	500	500	126	418	36.668734	137.668664	Entrata nord	255	300	Edicola

Figura 3.21: Result set dell'esecuzione della query del listato 3.1. Sfruttando l'ordinamento imposto negli attributi *Id*, *x EPM* e *y EPM* (riga 20 della query), si può scandire il result set per estrarre solo i dati rilevanti (gli elementi evidenziati in giallo); le ripetizioni di dati già estratti sono ignorate.

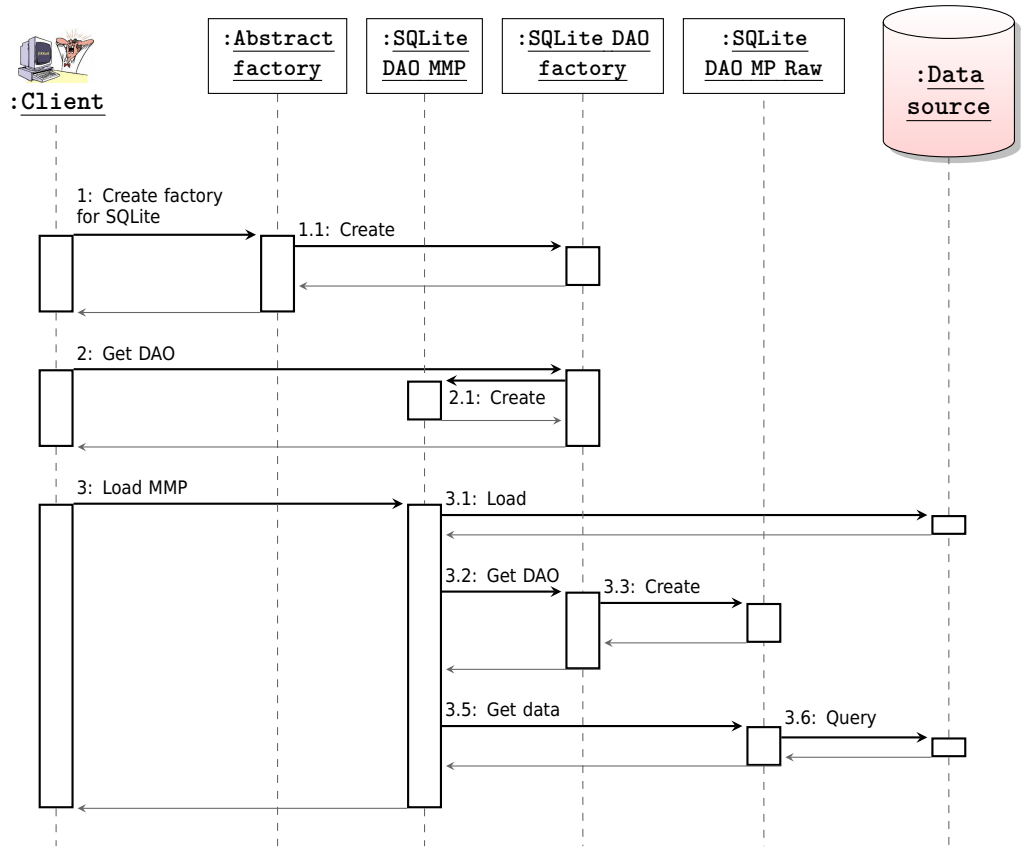


Figura 3.22: Diagramma delle sequenze semplificato del caricamento di una *MMP*.

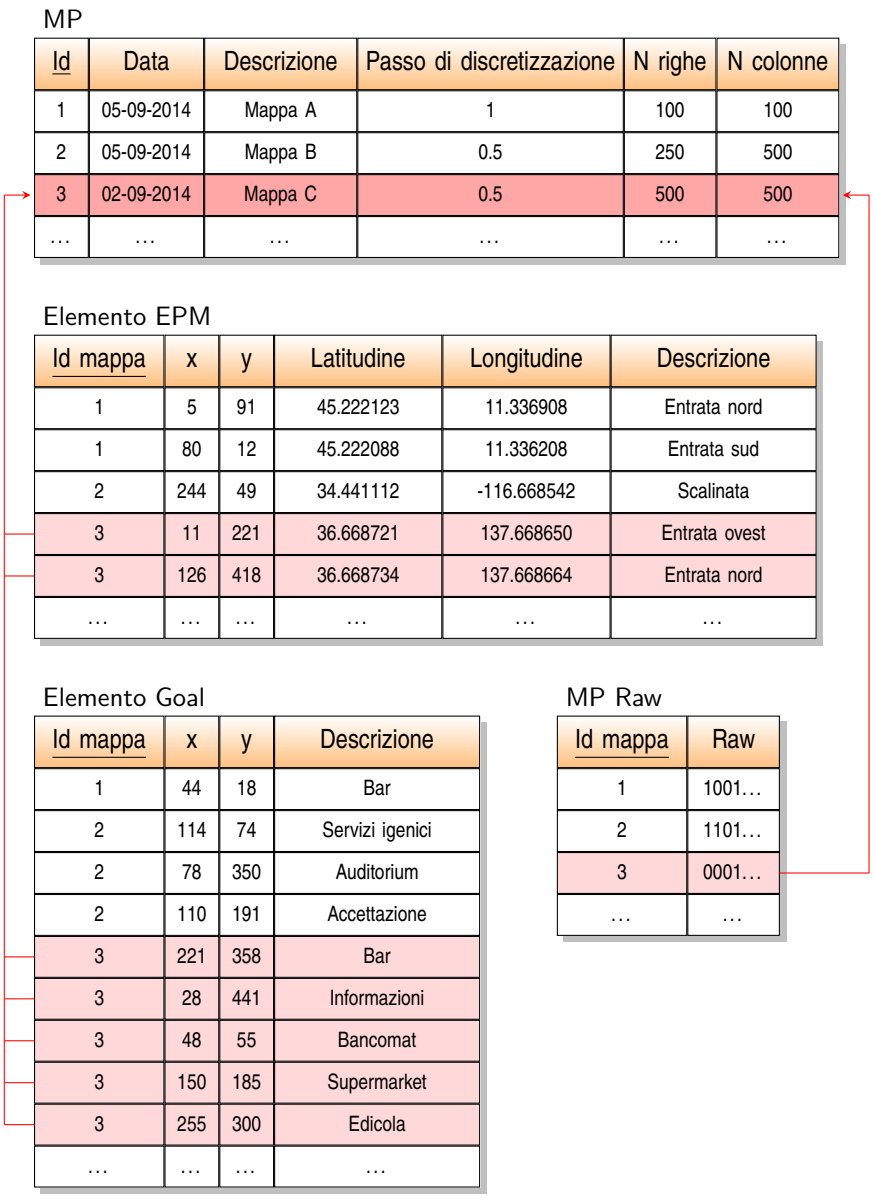


Figura 3.23: Eliminazione di una *MMP*. Grazie al vincolo di integrità referenziale è possibile impostare nel DBMS l'opzione che consente l'eliminazione a cascata di tutti i record (indicati in rosa) che referenziano il record principale eliminato (indicato in rosso).

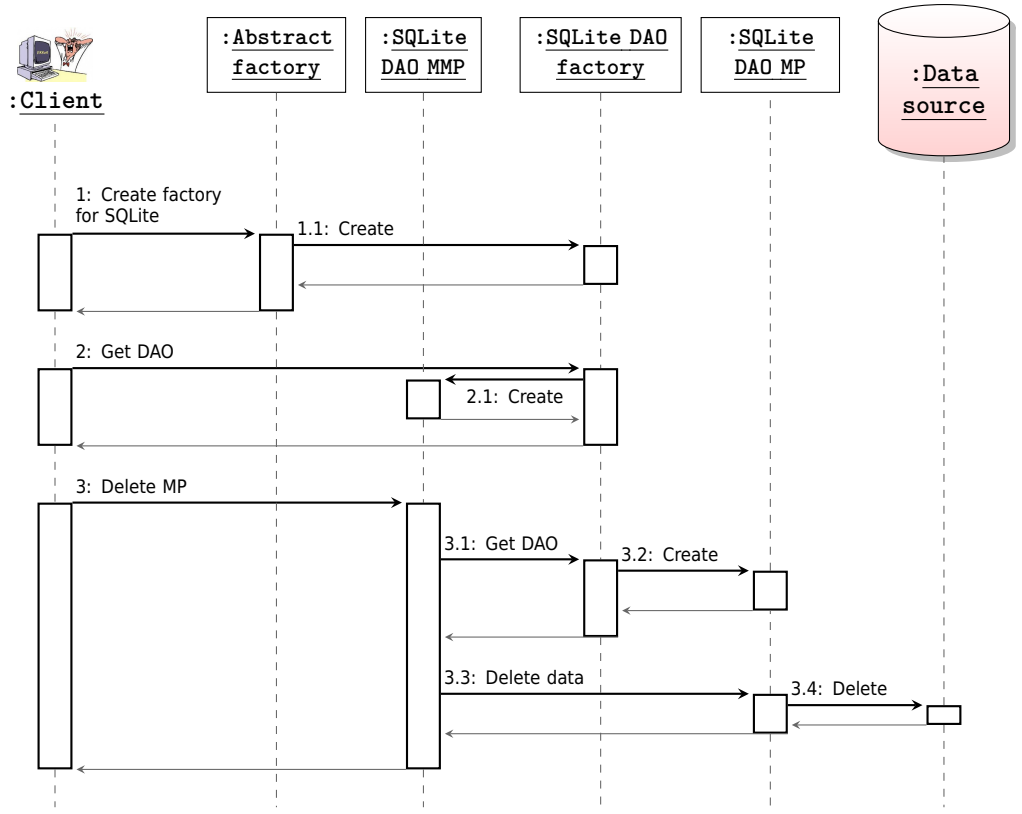


Figura 3.24: Diagramma delle sequenze semplificato dell'eliminazione di una *MMP*.

3.3.3.5 Meccanismo di serializzazione

L'ultimo passo della progettazione consiste nell'elaborare un meccanismo di serializzazione e deserializzazione dell'oggetto `matrice` della classe `Matrice`. L'operazione di serializzazione (algoritmo 3.7) riceve in input un array bidimensionale di `Elemento peso` e produce in uscita un array di byte; viceversa, l'operazione di deserializzazione (algoritmo 3.8) riceve in input un array di byte e produce in uscita un array bidimensionale di `Elemento peso`.

Si è scelto di serializzare solamente gli elementi inizializzati dell'array bidimesionale `matrice`. Questa soluzione ha il vantaggio di serializzare pochi elementi per matrici "sparse", cioè per matrici che hanno pochi elementi inizializzati. Tuttavia, richiede il salvataggio esplicito delle coordinate `x` e `y` di ogni elemento che indicano la sua posizione nell'array bidimensionale `matrice`.

Algoritmo 3.7 Serializza `matrice` di `Elemento peso`.

Input: `matrice` di `Elemento peso`
Output: Array di byte.

1: **function** SERIALIZZA `MATRICE` DI `ELEMENTO PESO`(`matrice` di `Elemento peso`)
2: *bytes*[] è l'array di output
3: **for all** $e \in$ `matrice` di `Elemento peso` tale che e è inizializzato **do**
4: $x \leftarrow$ GET X(e)
5: $x_{bin} \leftarrow$ CONVERTI IN NOTAZIONE BINARIA(x)
6: AGGIUNGI IN CODA A BYTES(x_{bin})
7: $y \leftarrow$ GET Y(e)
8: $y_{bin} \leftarrow$ CONVERTI IN NOTAZIONE BINARIA(y)
9: AGGIUNGI IN CODA A BYTES(y_{bin})
10: *peso* $\leftarrow$ GET PESO(e)
11: $peso_{bin} \leftarrow$ CONVERTI IN NOTAZIONE BINARIA(*peso*)
12: AGGIUNGI IN CODA A BYTES($peso_{bin}$)
13: **return** *bytes*[]

Algoritmo 3.8 Deserializza un array di byte in una `matrice` di `Elemento peso`.

Input: *input[]*
Output: `matrice` di `Elemento peso`

1: **function** DESERIALIZZA UN ARRAY DI BYTE IN UNA `MATRICE` DI `ELEMENTO PESO`(*input[]*)
2: `matrice[] []` è l'array bidimensionale di `Elemento peso` di output
3: $n \leftarrow$ NUMERO DI ELEMENTI(*input*[])
4: $i \leftarrow 1$
5: **while** $i \leq n$ **do**
6: $x \leftarrow$ LEGGI X(*input*[])
7: $y \leftarrow$ LEGGI Y(*input*[])
8: *peso* $\leftarrow$ LEGGI PESO(*input*[])
9: `matrice[x][y]` $\leftarrow$ CREA NUOVO `ELEMENTO PESO`(*peso*)
10: $i \leftarrow i + 1$
11: **return** `matrice[] []`

3.4 Implementazione

3.4.1 Ordine degli elementi della matrice di una MP

In questa sezione si vuol far notare che gli indici con cui si accede ad un elemento dell'array bidimensionale `matrice` della classe `Matrice` non corrispondono alla posizione che uno sviluppatore potrebbe attendersi. In questo testo, per accedere all'elemento alla riga y e alla colonna x, si è usata la notazione `matrice[x][y]`. Un'eventuale implementazione che segua pari pari questa notazione porterebbe ad avere una matrice i cui elementi adiacenti in memoria non corrispondono alle celle adiacenti nella griglia. In generale, questa circostanza non crea inconvenienti purché si utilizzi sempre la stessa metodologia di accesso (cioè sempre nella forma `matrice[x][y]`).

Nell'eventualità che nasca l'esigenza di mantenere anche in memoria un ordinamento degli elementi coerente con l'ordinamento delle celle della griglia (ad esempio, se è necessario svolgere qualche forma di elaborazione sui dati della matrice, avere gli elementi ordinati agevola il ragionamento), si consiglia l'utilizzo dei metodi `get Elemento` e `set Elemento` di `Matrice` che provvedono a:

- Invertire l'ordine di utilizzo degli indici x e y (cioè l'accesso alla matrice avviene nella forma `matrice[y][x]`).
- Eseguire un'operazione di offset sull'indice di riga. In questo modo, la riga a della griglia diventa la riga $nRows - 1 - a$ dell'array bidimensionale.

Il consiglio (come tra l'altro è stato fatto per il resto del progetto) è di utilizzare i suddetti metodi in modo da facilitare il successivo sviluppo di nuovo codice che utilizzi gli elementi di `matrice`.

3.4.2 Serializzazione e deserializzazione: classe `MatrixWeightSerDeser`

Il meccanismo di serializzazione di ogni elemento di `matrice` potrebbe utilizzare la classe `ByteBuffer`, i cui metodi consentono agevolmente la serializzazione e deserializzazione dei tipi di dato di Java. Tuttavia, la continua allocazione di memoria e l'esecuzione del processo di *garbage collector* (figura 3.25) rallentano sensibilmente l'esecuzione dell'applicazione. Per tale motivo si è scelto di operare "a basso livello", tramite operazioni di *shift* e *cast* sui singoli byte.

3.4.3 Salvataggio del database su memoria secondaria

La memoria secondaria (persistente) di Android è suddivisa in memoria interna e memoria esterna. Generalmente, la memoria interna è tipo flash mentre la memoria esterna è costituita da una scheda esterna (ad esempio, una micro SD card). Nel caso in cui il dispositivo non sia provvisto di uno slot per la lettura e scrittura di schede, la memoria esterna è costituita da una partizione della memoria flash.

Di default, Android riserva uno spazio di memoria interna per ogni applicazione installata. Tale memoria è privata e non accessibile dalle altre applicazioni. Durante lo sviluppo del progetto è nata l'esigenza di esportare il database in modo da analizzare i dati delle mappe con altri strumenti; è stato quindi necessario salvare il database su memoria esterna. Poiché l'accesso al database avviene mediante una sottoclasse di `SQLiteOpenHelper`, l'unico modo per poter modificare la destinazione del database consiste nel fornire un'istanza di `Context` opportunamente modificata. Tale funzione è svolta dalla classe `DbContextForPath` che provvede a ridefinire adeguatamente il metodo `getDatabasePath` e ritornare un *path* relativo alla memoria secondaria.

Tuttavia, i test esposti nella prossima sezione sono stati eseguiti nella memoria interna, al fine di garantire la piena compatibilità anche con la versione dell'emulatore utilizzata.

```
10-01 07:40:43.340    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 514K, 7% free 9677K/10311K, paused 1ms+0ms
10-01 07:40:43.360    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 348K, 6% free 9713K/10311K, paused 1ms+0ms
10-01 07:40:43.390    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 348K, 6% free 9749K/10311K, paused 0ms+1ms
10-01 07:40:43.420    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 402K, 6% free 9790K/10311K, paused 1ms+0ms
10-01 07:40:43.450    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 365K, 5% free 9827K/10311K, paused 0ms+1ms
10-01 07:40:43.480    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 389K, 5% free 9867K/10375K, paused 0ms+0ms
10-01 07:40:43.510    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 352K, 5% free 9903K/10375K, paused 0ms+2ms
10-01 07:40:43.577    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 378K, 5% free 9942K/10439K, paused 0ms+0ms
10-01 07:40:43.639    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 400K, 5% free 9983K/10503K, paused 0ms+1ms
10-01 07:40:43.670    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 437K, 6% free 9947K/10503K, paused 0ms+1ms
10-01 07:40:43.700    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 887K, 10% free 9497K/10503K, paused 0ms+2ms
10-01 07:40:43.730    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 267K, 9% free 9617K/10503K, paused 0ms+1ms
10-01 07:40:43.740    1453-1466/? D/dalvikvm:  GC_FOR_ALLOC freed 400K, 9% free 9569K/10503K, paused 3ms
10-01 07:40:43.759    1453-1458/? D/dalvikvm:  GC_CONCURRENT freed 251K, 7% free 9782K/10503K, paused 0ms+0ms
```

Figura 3.25: Estratto del file di log in cui si nota l'intervento del *garbage collector* nell'implementazione del meccanismo di serializzazione e deserializzazione che utilizza i metodi della classe `ByteBuffer`.

3.5 Test

Come previsto dalle specifiche, i test sono stati eseguiti sia sull'emulatore che sullo smartphone. Seguono alcune considerazioni:

- Casi d'uso: eseguiti correttamente per i due scenari previsti.
- Test di sovraccarico: eseguiti come variante dei casi d'uso.
- Test di prestazione: tutte le operazioni sulle mappe del dataset di riferimento sono eseguite entro i 2 secondi preventivati in fase di analisi. Invece il caricamento di una mappa di grandi dimensioni fallisce. La causa sembra essere l'impossibilità di caricare un attributo di tipo `BLOB` maggiore di 1 MB[7]. Una possibile soluzione consiste nel salvare su file l'attributo *Raw* ed inserire sul database solamente il path per poter recuperare il file. Un'altra alternativa consiste nell'utilizzare librerie d'accesso esterno, come *sqlite4java*[8].

Dal punto di vista teorico, tutte le operazioni richiedono, per la loro esecuzione, un tempo proporzionale alla taglia del dati su cui operano:

- L'operazione di salvataggio di una *MMP* scandisce una sola volta tutti gli elementi della matrice per poi convertirli in una rappresentazione binaria.
- Nell'operazione di caricamento, non è noto come avviene il reperimento dei dati da parte di *SQLite*. Tuttavia si può affermare che il result set restituito dalla query del listato 3.1 è scandito una sola volta.
- Anche l'eliminazione di una *MMP* richiede ad *SQLite* di scandire al più una volta l'intero database.

3.5.1 Esigenze della validazione dell'applicazione

Durante la validazione dell'applicazione è nata l'esigenza di disaccoppiare temporalmente la determinazione della posizione dell'utente dalle effettive operazioni per costruire una mappa. In questo modo, una volta acquisite le misure per la navigazione PDR è possibile valutare la

[7] Nella documentazione ufficiale di Android, non vi è traccia di questo limite. Tuttavia, *SQLite* prevede questo vincolo, il quale è configurato in fase di compilazione. In rete è possibile trovare diverse discussioni in cui gli utenti si lamentano di questo inconveniente (ad esempio qui o qui). Una possibile spiegazione sulla scelta del valore di 1 MB adottato da Android dipende dalle prestazioni offerte da *SQLite*: secondo la documentazione ufficiale, già con attributi di tipo `BLOB` che occupano più di 200 KB conviene (in termini di tempo d'esecuzione) il salvataggio esterno al DBMS.

[8] *sqlite4java*. Fonte: *sqlite4java* homepage. Link. Ultima visita: 3 dicembre 2014

bontà della posizione stimata in funzione di più mappe, caratterizzate da dimensioni e passo di discretizzazione diversi.

Per questo motivo, si è scelto di inserire un'altra tabella nel database, il cui schema relazionale è indicato in figura 3.26.

Strides data

Id coppia	Id mappa	N strides	Azimuth	Tag

Figura 3.26: Schema della relazione *Strides data*.

Si noti che la relazione *Strides data* non ha nessun vincolo d'integrità con le altre relazione della base di dati.

3.6 Considerazioni e sviluppi futuri

Il modulo *Dati* realizzato si presenta come uno strumento sufficientemente flessibile su cui gli altri moduli possono appoggiarsi. L'utilizzo del design pattern DAO consente di estendere il livello dati senza intaccare il modello dati utilizzato nella programmazione O-O (figura 3.5). In questo modo, il codice funzionante già scritto che utilizza tale modello non dovrà essere modificato. Una possibile estensione è indicata in figura 3.27.

Infine l'applicazione potrebbe essere estesa con altre funzionalità utili. Ad esempio, si supponga di avere più tipologie di mappa. L'utente, che necessita di interrogare il dataset per recuperare una mappa date le coordinate GPS, potrebbe non essere a conoscenza di quale tipologia di mappa sono presenti nel dataset per quelle coordinate. Di conseguenza non è in grado di richiedere un DAO adeguato. Uno sviluppo efficiente potrebbe sollevare l'utente da questo onere ed occuparsi di recuperare, se presenti, le varie tipologie di mappe.

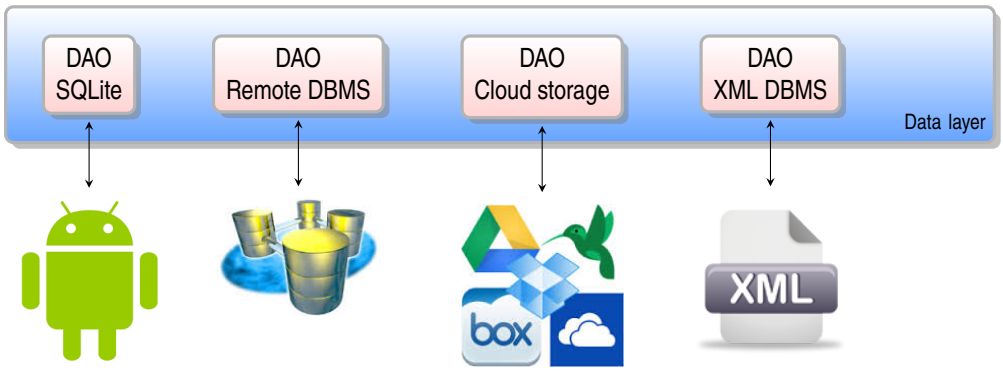
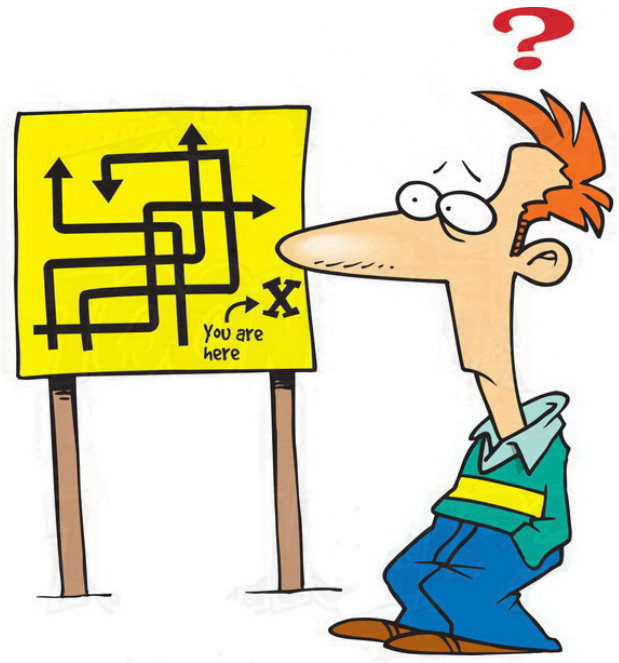


Figura 3.27: Esempio di possibile estensione del livello dati dell'applicazione.

Capitolo 4

Modulo Creazione mappa



Fonte: *clipartof* website. Ultima visita: 3 dicembre 2014.

4.1 Definizione delle specifiche

Il modulo *Creazione mappa* deve popolare una *MMP* gestita (cioè salvata e caricata) dal modulo *Dati* attraverso i dati forniti dal modulo *Contapassi*. Nel dettaglio, si è accordato con il committente lo sviluppo di queste specifiche:

- La mappa da costruire costituisce una "registrazione" del percorso effettuato dall'utente da un punto iniziale ad un punto finale. Per far ciò, si sovrappone un piano cartesiano alla griglia della mappa, come indicato in figura 4.1; si noti che l'origine del piano coincide con il bordo inferiore sinistro della cella in posizione $(1, 1)$. Lo scopo del piano cartesiano consiste nel rappresentare il percorso in metri effettuato dall'utente.
- Il percorso reale dell'utente è approssimato con una sequenza ordinata di segmenti, in cui due segmenti consecutivi condividono un estremo (figura 4.2). La generazione di tale ap-

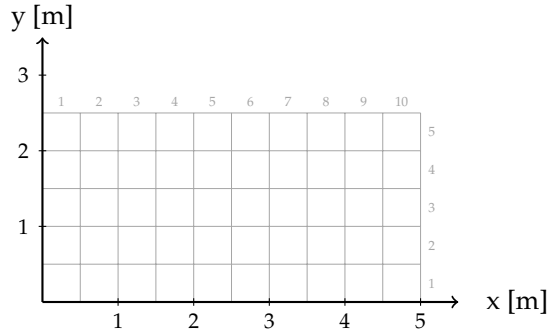


Figura 4.1: In nero è indicato il piano cartesiano sovrapposto alla griglia (rappresentata in grigio). Il passo di discretizzazione della griglia è 0.5 m.

prossimazione è affidata al modulo *Contapassi*. Quest'ultimo fornisce al modulo *Creazione mappa* una coppia di punti *(start, end)*: *start* individua il punto nel piano cartesiano da cui ha inizio il segmento (chiamato anche spostamento) mentre *end* individua il punto in cui il segmento termina. Si noti che:

  - I punti *start* e *end* sono punti definiti in $\mathbb{R}^+ \times \mathbb{R}^+$ e indicano le coordinate in metri degli estremi del segmento.
  - Si sceglie di sviluppare un approccio "push" in cui il modulo *Contapassi* fornisce i dati quando sono disponibili.

- Dato un segmento, il modulo *Creazione mappa* deve selezionare quali elementi della matrice aggiornare, cioè assegnare o modificare il peso dell'elemento. Si sceglie di aggiornare tutti gli elementi della matrice che corrispondono alle celle della griglia attraversate dal segmento (geometrico) che unisce i punti *start* e *end* del piano cartesiano.

- Il campo *peso* di un elemento della matrice di una *MMP* tiene traccia della "frequenza" con cui l'utente ha attraversato la relativa cella. Per il momento tale valore non rappresenta una vera e propria frequenza (intesa come numero di occorrenze) ma rappresenta una semplice quantità che è incrementata ogni volta che la cella della griglia associata è attraversata dal segmento. L'idea di base consiste nel fatto che, dopo aver registrato più percorsi ripetuti da uno stesso *start* ad uno stesso *end*, gli elementi della matrice con il peso maggiore costituiscano i tasselli (o parte di essi) del percorso "più probabile" effettuato dall'utente. Tale valore sarà utilizzato dal modulo di *Navigazione* per scegliere un percorso più idoneo da far effettuare all'utente.

- Come si nota nella figura 4.2, l'aggiornamento delle celle attraversate da un segmento potrebbe aggiornare elementi le cui celle corrispondenti non sono state direttamente attraversate dall'utente. Per mitigare questo effetto si sceglie di aggiornare, secondo un opportuno fattore di scala, anche gli elementi della matrice "adiacenti" agli elementi corrispondenti alle celle attraversate dal segmento di una porzione del percorso.

- Data la cella c alle coordinate(i, j), con $i \geq 1$ e $j \geq 1$, si dice che *c contiene* il punto $p = (x, y)$ se $i - 1 \leq x < i$ e $j - 1 \leq y < j$. Utilizzando questa definizione, si impone che la cella che *contiene* lo *start* del primo segmento costituisca un *punto d'accesso* mentre la cella che *contiene* l'*end* dell'ultimo segmento costituisca un *punto d'interesse*. Nel seguito del progetto, si tralascia la generazione di una coppia di coordinate GPS reale di ogni *punto d'accesso*.

- I punti cardinali di una mappa seguono la convenzione illustrata in figura 4.2.

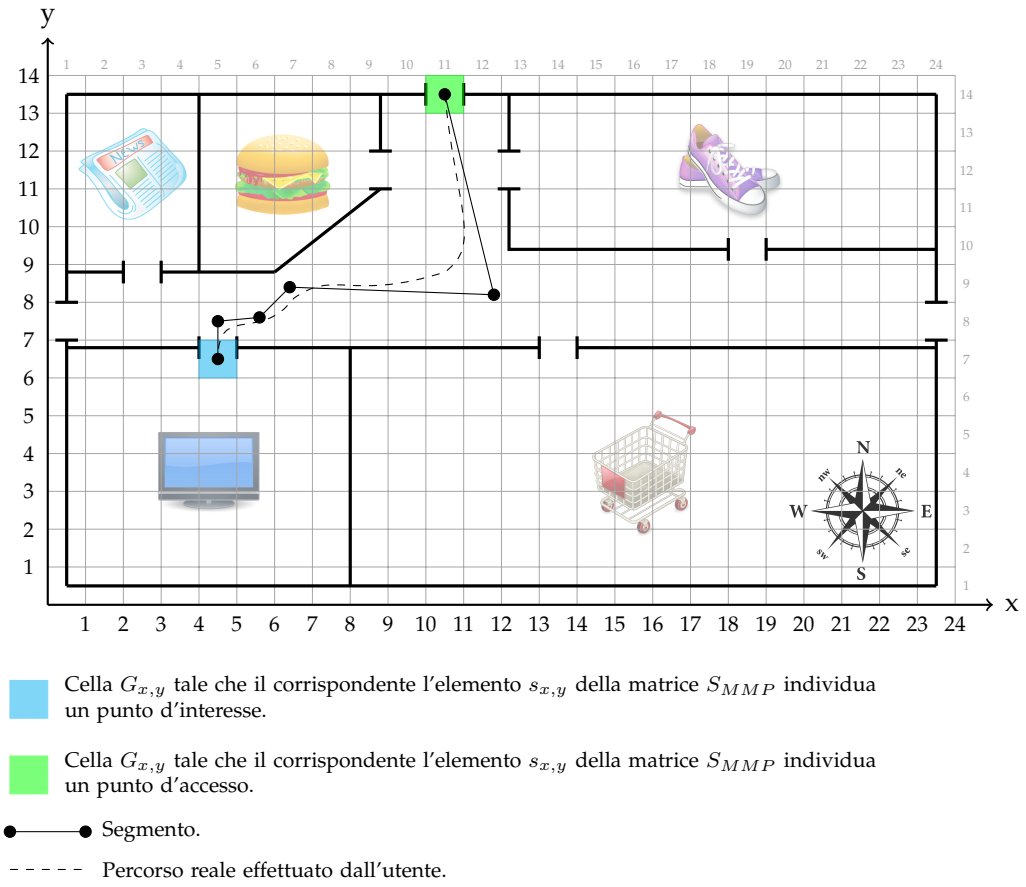


Figura 4.2: Esempio di cammino effettuato dall'utente e dei segmenti individuati dal modulo *Contapassi*.

4.2 Progettazione della soluzione

Si sceglie di sviluppare le funzioni del modulo seguendo l'algoritmo 4.1, il quale dovrà essere eseguito per ogni porzione (segmento) di cammino individuata dalla coppia di punti *start* e *end*.

Di seguito si analizza la progettazione delle singole funzioni dell'algoritmo.

4.2.1 Funzione "Get cella"

Get cella si occupa della conversione tra un punto del piano cartesiano e le coordinate della cella della griglia che lo contiene. Si adotta la seguente simbologia:

- $c_i.x$ e $c_i.y$ sono i valori delle coordinate x e y della cella c_i.
- Δd il passo di discretizzazione della *MMP* m.

Si stabilisce quindi la seguente convenzione:

$$c_{start}.x = \left\lfloor \frac{start.x}{\Delta d} \right\rfloor$$

$$c_{start}.y = \left\lfloor \frac{start.y}{\Delta d} \right\rfloor$$

$$c_{end}.x = \left\lfloor \frac{end.x}{\Delta d} \right\rfloor$$
$$c_{end}.y = \left\lfloor \frac{end.y}{\Delta d} \right\rfloor$$

Algoritmo 4.1 Aggiorna mappa.

Input: $start, end, m$ ▷ m è una *MMP*
Output: Aggiorna la *MP* m

1: **function** AGGIORNA MAPPA($start, end, m$)
2: $\Delta d \leftarrow$ GET PASSO DI DISCRETIZZAZIONE(m)
3: $c_{start} \leftarrow$ GET CELLA($start, \Delta d$)
4: $c_{end} \leftarrow$ GET CELLA($end, \Delta d$)
5: ANALIZZA I CASI BASE(c_{start}, c_{end})
6: $eq \leftarrow$ CALCOLA IL SEGMENTO START-END($start, end$)
7: $P \leftarrow$ DETERMINA UN PUNTO DEL PIANO PER OGNI CELLA ATTRAVERSATA DAL SEGMENTO START-END($start, end, eq, \Delta d$)
8: **for** $p \in P$ **do**
9: AGGIORNA ELEMENTI DELLA MATRICE(p)

4.2.2 Funzione "Analizza casi base"

Come suggerisce il nome, lo scopo di questa funzione consiste nell'analizzare i casi particolari di aggiornamento della mappa. Prima di procedere è opportuno osservare che, una volta ricavata la sequenza $c_{start}, \ldots, c_i, \ldots, c_{end}$ ordinata di celle da aggiornare, è necessario specificare se il primo elemento c_{start} della sequenza debba essere aggiornato o meno. Infatti, nel caso di aggiornamento in successione di due segmenti (ad esempio, $\overline{AB}$ e $\overline{BC}$), bisogna prestare attenzione a non aggiornare due volte una stessa cella (riprendendo l'esempio, la cella che contiene B costituisce la c_{end} della sequenza associata ad $\overline{AB}$ e la c_{start} della sequenza associata ad $\overline{BC}$). In generale, la sequenza di celle da aggiornare non include la cella che contiene lo *start*, ad eccezione del primo segmento. Nel seguito di questo capitolo, verrà omesso l'aggiornamento della prima cella.

Di seguito si analizzano i casi base dell'operazione di aggiornamento delle celle:

- Se i punti *start* e *end* sono *contenuti* in una stessa cella della griglia, cioè $c_{start} = c_{end}$ allora non si aggiorna nessuna cella.
- Se c_{end} è un *vicino* di c_{start} secondo la definizione di *vicinato di Moore*[1] (figura 4.3), cioè se $c_{end} \in M_{start} = \{(x, y) : |x - c_{start}.x| \leq 1, |y - c_{start}.y| \leq 1\}$, allora si aggiorna solamente l'elemento della matrice corrispondente a c_{end}.
- Se c_{end} è allineato orizzontalmente o verticalmente a c_{start} si procede come segue:
  - Se $c_{start}.y = c_{end}.y$ allora si aggiornano gli elementi della matrice che corrispondono a tutte le celle c_i tali che:

$$c_i = \begin{cases} c_{start}.y < c_i.y \leq c_{end}.y & \text{se } c_{start}.y < c_{end}.y \\ c_{end}.y \leq c_i.y < c_{start}.y & \text{se } c_{end}.y < c_{start}.y \end{cases}$$

  - Se $c_{start}.x = c_{end}.x$ allora si aggiornano gli elementi della matrice che corrispondono a tutte le celle c_i tali che:

$$c_i = \begin{cases} c_{start}.x < c_i.x \leq c_{end}.x & \text{se } c_{start}.x < c_{end}.x \\ c_{end}.x \leq c_i.x < c_{start}.x & \text{se } c_{end}.x < c_{start}.x \end{cases}$$

[1] Vicinato di Moore. Fonte: *cell-auto.com*. Link.

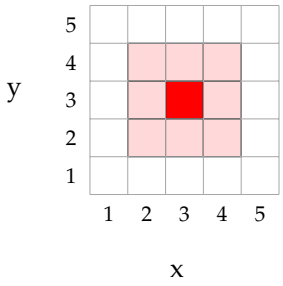


Figura 4.3: In rosa sono indicate le celle vicine alla cella in rosso secondo la definizione di vicinato di Moore.

4.2.3 Funzione "Calcola il segmento start-end"

Utilizzando le equazioni 4.1 e 4.2 è possibile calcolare la retta del piano cartesiano passante per i punti *start* e *end*.

$$m = \frac{end.y - start.y}{end.x - start.x} \tag{4.1}$$

$$y = mx + q \tag{4.2}$$

Per ridurre i possibili casi da analizzare in seguito, si impone che $start.x \leq end.x$. Se questa condizione non è soddisfatta, è sufficiente "scambiare" i valori delle rispettive coordinate dei punti *start* e *end* (cioè $start.x \leftrightarrows end.x$ e $start.y \leftrightarrows end.y$): questa operazione non influenza lo sviluppo poiché esiste un'unica retta passante per due punti.

4.2.4 Funzione "Determina un punto del piano per ogni cella attraversata dal segmento start-end"

L'obiettivo di tale funzione consiste nel determinare una sequenza di punti P tali che ogni $p \in P$ sia un punto del piano cartesiano *contenuto* (secondo la definizione della sezione 4.1) in una cella della griglia attraversata dal segmento tra *start* e *end*. Inoltre, è opportuno che non esistano più $p \in P$ che siano *contenuti* nella stessa cella poiché essi determinerebbero un insensato aggiornamento multiplo dello stesso elemento della matrice.

L'algoritmo 4.2 espone la soluzione adottata per determinare la sequenza P. L'idea principale consiste nel determinare un punto p, per ogni cella attraversata dal segmento che unisce *start* e *end*, in modo tale che una delle due componenti di p (cioè il valore di x o il valore di y) è calcolata in modo indipendente (righe di pseudocodice 6 e 11 dell'algoritmo 4.2) mentre l'altra è calcolata in accordo con l'equazione della retta passante per *start* e *end* (righe di pseudocodice 7 e 12). In altre parole, con questa strategia, si impone il valore ad una componente per ogni punto p_i (ad esempio $p_1.x = 2.5$, $p_2.x = 3.5$, $p_3.x = 4.5$) e si calcola il corrisponde valore dell'altra in accordo con l'equazione della retta. Ovviamente è necessario scegliere con accuratezza i valori della componente indipendente, in modo tale da:

- Non generare punti p che non possono essere *contenuti* da nessun cella attraversata dal segmento tra *start* e *end*.
- Non generare più punti p *contenuti* in una stessa cella.

Si è scelto di calcolare in modo indipendente la componente in cui la retta "cresce" più velocemente, valutando il valore del coefficiente angolare m della retta. In questo modo è possibile determinare un maggior numero di elementi della matrice da aggiornare (figure 4.4a e 4.4b).

Per il calcolo della componente indipendente si è scelto di generare punti che risultano "centrati" nella cella corrispondente lungo la direzione della componente indipendente. Ad esempio,

Algoritmo 4.2 Determina un punto del piano per ogni cella attraversata dal segmento start-end.

Input: $start, end, eq, \Delta d$
Output: P ={punti del piano cartesiano tali che ogni punto è *contenuto* in una cella della griglia attraversata dal segmento tra $start$ e end}

1: **function** DETERMINA UN PUNTO DEL PIANO PER OGNI CELLA ATTRAVERSATA DAL SEGMENTO START-END($c_{start}, c_{end}, eq, \Delta d$)
2: $m \leftarrow$ GET M(*eq*) ▷ Ottiene il coefficiente angolare della retta
3: $q \leftarrow$ GET Q(*eq*) ▷ Ottiene l'intercetta della retta
4: **if** $m \geq 1$ **then** ▷ Scansione lungo l'asse y
5: **for** $c_{start}.y < i < c_{end}.y$ **do** ▷ i assume solo valori naturali
6: $y \leftarrow i\,\Delta d + \frac{\Delta d}{2}$
7: $x \leftarrow \frac{y-q}{m}$
8: $P \leftarrow P \cup \{(x, y)\}$
9: **else** ▷ Scansione lungo l'asse x
10: **for** $c_{start}.x < i < c_{end}.x$ **do** ▷ i assume solo valori naturali
11: $x \leftarrow i\,\Delta d + \frac{\Delta d}{2}$
12: $y \leftarrow mx + q$
13: $P \leftarrow P \cup \{(x, y)\}$
14: $P \leftarrow P \cup \{end\}$
15: **return** P

dato $c_{start}.x = 1.95$, $c_{end}.x = 5.08$ e $\Delta d = 0.5$ m, se la componente indipendente è x si è scelto di generare celle il cui valore della componente x vale $2.25, 2.75, 3.25, 3.75, 4.25$ e 4.75 rispettivamente. Questa strategia assicura che non siano generati più punti che appartengono ad una stessa cella della griglia.

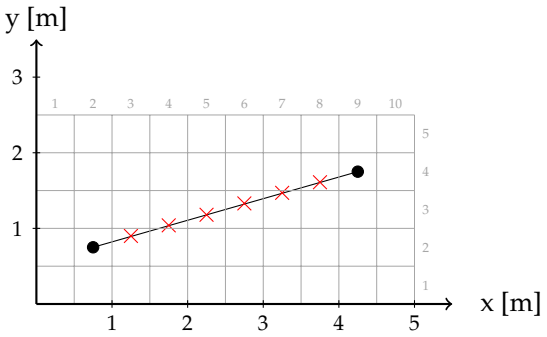
Tuttavia si osservi che tale metodo non genera un punto per ogni cella attraversata dal segmento tra *start* e *end*. Ad esempio, in figura 4.4a, alla cella (4,2) non è associato nessun punto, sebbene sia attraversata dal segmento. Nonostante questo algoritmo non adempia completamente alla specifica, si è scelto comunque di utilizzarlo.

4.2.5 Funzione "Aggiorna elementi della matrice"

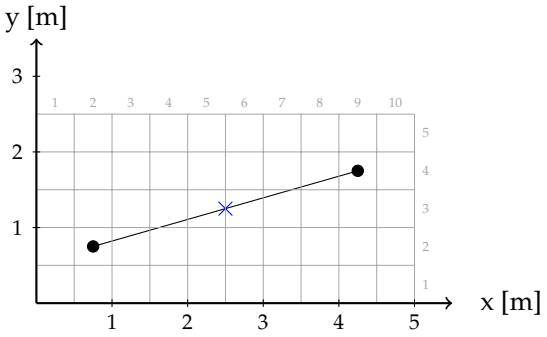
Per ogni $p \in P$ è necessario scegliere quali elementi della matrice di una *MMP* aggiornare. Infatti è ragionevole aggiornare non solo l'elemento corrispondente ad una precisa cella ma anche gli elementi "vicini" in modo da mitigare eventuali errori commessi dal modulo *Contapassi*.

Si sceglie di sviluppare la strategia indicata nell'algoritmo 4.3. Si noti che possono verificarsi due casi:

- Se la cardinalità di P è minore o uguale a 2 significa che le rispettive celle corrispondenti ai punti $p \in P$ sono confinanti secondo il vicinato di Moore (righe di pseudocodice 4 e 5); in questo caso non si procede ad aggiornare le celle adiacenti.
- Se la cardinalità di P è maggiore a 2, dato un punto $p \in P$ si può verificare uno dei seguenti casi:
  - p è in prossimità di un bordo (figura 4.5a) della cella in cui è *contenuto*. Se il punto $p \in P$ è sufficientemente vicino (entro una soglia ϵ) al bordo (perimetro) della cella allora si aggiorna, secondo un opportuno fattore di scala, anche la cella adiacente. Poiché la coordinata indipendente di p è generata in modo tale da essere "centrata" nella cella, esiste solo una cella adiacente lungo l'asse corrispondente alla coordinata dipendente.
  - p è "centrato" nella cella (figura 4.5b) in cui è *contenuto*. Se il punto $p \in P$ non è sufficientemente vicino (entro una soglia ϵ) al bordo della cella allora si aggiorna, secondo un altro opportuno fattore di scala, anche le due celle adiacenti lungo l'asse a cui corrisponde la coordinata dipendente.



(a) x è calcolato in modo indipendente, con passo 0.5 m. Si ottengono i valori 1.25, 1.75, 2.25, 2.75, 3.25 e 3.75.



(b) y è calcolato in modo indipendente, con passo 0.5 m. Si ottiene solo il valore 1.25.

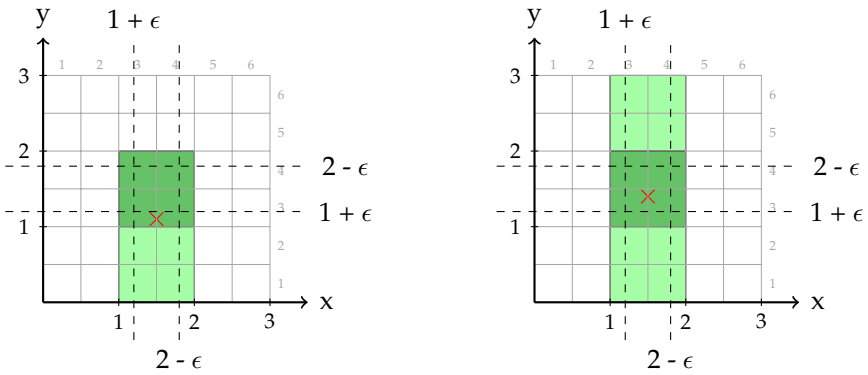
Figura 4.4: Determinazione delle celle attraversate dal segmento che unisce *start* e *stop*. Il coefficiente angolare della retta passante per *start* e *stop* è minore di 1.

Infine, si noti che l'algoritmo 4.4 provvede a creare una nuova istanza di un elemento peso della matrice se esso non è inizializzato.

Algoritmo 4.3 Aggiorna elementi della matrice.

Input: P ={punti del piano cartesiano in cui ogni punto è contenuto in una cella della griglia attraversata dal segmento tra $start$ e end}, map è una *MMP*, m coefficiente angolare della retta
Output: Aggiorna m

1: **function** AGGIORNA ELEMENTI DELLA MATRICE(P, map, m)
2: $w_1, w_{2m}, w_{2s}, w_{3m}, w_{3s}$ sono pesi
3: **if** $|P| \leq 2$ **then** ▷ Aggiornamento delle singole celle e non dei vicini
4: **for** $p \in P$ **do**
5: AGGIORNA ELEMENTO(p, w_1, map)
6: **if** $m \geq 1$ **then** ▷ Scansione lungo l'asse y
7: **for** $p \in P$ **do**
8: $c_p \leftarrow$ GET CELLA($p, \Delta d$)
9: $e_r \leftarrow (c_p.x + 1)\,\Delta d$ ▷ Valore in m del "confine destro" di c_p
10: $e_l \leftarrow c_p.x\,\Delta d$ ▷ Valore in m del "confine sinistro" di c_p
11: **if** $p.x > e_r - \epsilon$ **then** ▷ p vicino al "confine destro" di c_p
12: $c_r \leftarrow (c_p.x + 1, c_p.y)$
13: AGGIORNA ELEMENTO(c_p, w_{2m}, map)
14: AGGIORNA ELEMENTO(c_r, w_{2s}, map)
15: **if** $p.x < e_l + \epsilon$ **then** ▷ p vicino al "confine sinistro" di c_p
16: $c_l \leftarrow (c_p.x - 1, c_p.y)$
17: AGGIORNA ELEMENTO(c_p, w_{2m}, map)
18: AGGIORNA ELEMENTO(c_l, w_{2s}, map)
19: **else** ▷ p "centrato" in c_p
20: $c_r \leftarrow (c_p.x + 1, c_p.y)$
21: $c_l \leftarrow (c_p.x - 1, c_p.y)$
22: AGGIORNA ELEMENTO(c_p, w_{3m}, map)
23: AGGIORNA ELEMENTO(c_r, w_{3s}, map)
24: AGGIORNA ELEMENTO(c_l, w_{3s}, map)
25: **else**
26: **for** $p \in P$ **do** ▷ Scansione lungo l'asse x
27: $c_p \leftarrow$ GET CELLA($p, \Delta d$)
28: $e_a \leftarrow (c_p.y + 1)\,\Delta d$ ▷ Valore in m del "confine superiore" di c_p
29: $e_b \leftarrow c_p.y\,\Delta d$ ▷ Valore in m del "confine inferiore" di c_p
30: **if** $p.y > e_a - \epsilon$ **then** ▷ p vicino al "confine superiore" di c_p
31: $c_a \leftarrow (c_p.x, c_p.y + 1)$
32: AGGIORNA ELEMENTO(c_p, w_{2m}, map)
33: AGGIORNA ELEMENTO(c_a, w_{2s}, map)
34: **if** $p.y < e_b + \epsilon$ **then** ▷ p vicino al "confine inferiore" di c_p
35: $c_b \leftarrow (c_p.x, c_p.y - 1)$
36: AGGIORNA ELEMENTO(c_p, w_{2m}, map)
37: AGGIORNA ELEMENTO(c_b, w_{2s}, map)
38: **else** ▷ p "centrato" in c_p
39: $c_a \leftarrow (c_p.x, c_p.y + 1)$
40: $c_b \leftarrow (c_p.x, c_p.y - 1)$
41: AGGIORNA ELEMENTO(c_p, w_{3m}, map)
42: AGGIORNA ELEMENTO(c_a, w_{3s}, map)
43: AGGIORNA ELEMENTO(c_b, w_{3s}, map)



(a) Aggiornamento di due elementi.

(b) Aggiornamento di tre elementi.

Figura 4.5: Esempio di aggiornamento di un punto p (indicato con il simbolo $\times$) in cui x è la componente calcolata in modo indipendente.

Algoritmo 4.4 Aggiorna elemento.

Input: m è una *MP*, c è un cella che corrisponde ad un elemento di m, w è un peso
Output: Aggiorna un elemento di m

```
1: function AGGIORNA ELEMENTO(m, c, w)
2:     if m[c.x][c.y] è inizializzato then
3:         m[c.x][c.y].w ← m[c.x][c.y].w + w
4:     else
5:         m[c.x][c.y] ← NEW ELEMENTO PESO(c, w)
```

4.3 Progettazione software del modulo

La modellazione del modulo *Creazione mappa* tramite classi Java è molto semplice. Sono state definite due interfacce:

- `Selettore di celle`
- `Aggiornamento celle`

Si è sviluppato un implementazione di ogni interfaccia:

- `Selettore lineare di celle` è l'implementazione `Selettore di celle`. Si occupa di implementare le funzioni *"Analizza i casi base"*, *"Calcola il segmento start-end"* e *"Determina un punto del piano per ogni cella attraversata dal segmento start-end"*.
- `Aggiornamento celle vicine` è l'implementazione `Aggiornamento celle` e si occupa di implementare la funzione *"Aggiorna elementi della matrice"*.

4.4 Test

La fase di validazione del modulo consiste dei seguenti passi:

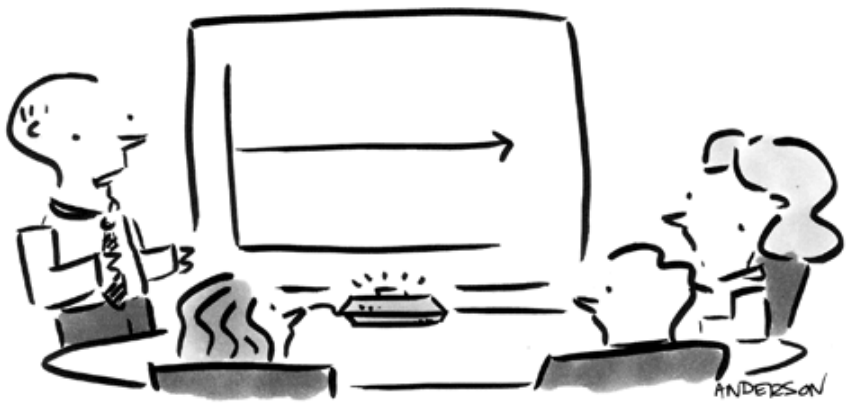
1. Creazione di un'istanza di `Selettore lineare di celle` e `Aggiornamento celle vicine`.
2. Generazione di una sequenza fittizia di coppie *(start, end)*.
3. Stimolazione delle istanze create con la sequenza di coppie *(start, end)*.

4. Verifica dell'aggiornamento complessivo della mappa con il risultato ottenuto con un altro tool (foglio elettronico Excel).

Come si può intuire, questa procedura di validazione non è automatica, richiede un lavoro manuale particolarmente oneroso e aumenta considerevolmente la possibilità di commettere errori di validazione (dovuti ad errori effettuati durante la generazione del foglio elettronico Excel).

Capitolo 5

Modulo Contapassi



"After closer investigation, it's become clear that we need to enter more than one value."

Fonte: *andertoons* website. Ultima visita: 3 dicembre 2014.

Le funzioni del modulo *Contapassi* sono utilizzate sia dal modulo *Creazione mappa* che dal modulo *Navigazione*. La descrizione presente in questo capitolo si occupa solo di documentare il comportamento del modulo durante l'interazione con il modulo *Creazione mappa*. Non verranno trattate le funzioni necessarie ad assicurare il corretto funzionamento del modulo *Navigazione*.

5.1 Definizione delle specifiche

Una delle principali funzioni del modulo *Contapassi* consiste nel generare la coppia di punti *(start, end)* che il modulo *Creazione mappa* dovrà utilizzare. Per svolgere tale funzione si stabilisce lo sviluppo dei seguenti componenti, ognuno dei quali dovrà adempire a precise specifiche:

- Un componente, chiamato *Strides counter*, deve contare il numero di strides effettuati da un utente che cammina. In accordo con [49], si definisce *stride* la successione di due passi effettuati dell'utente, cioè le sequenze {passo sinistro, passo destro} o {passo destro, passo sinistro}. Per lo scopo del progetto, non sono ammessi strides nella forma {passo sinistro, passo sinistro} o {passo destro, passo destro}.
- Un componente chiamato *Orientation checker* deve calcolare l'orientazione del cammino dell'utente. Nel contesto del progetto, con il termine *orientazione* (in seguito chiamato anche

azimuth per adeguarsi alla terminologia adottata nella documentazione Android) si intende l'angolo, definito sul piano costituito dal pavimento su cui l'utente cammina, tra il nord magnetico terrestre e la direzione iniziale dello spostamento effettuato dall'utente.

- Un componente chiamato *Walk converter* ha il compito di:
  1. Ricevere in ingresso una coppia *($n_{strides}$, azimuth)* che rappresenta una porzione del cammino effettuato dall'utente ed approssimata dai componenti *Strides counter* e *Orientation checker*.
  2. Convertire la porzione del cammino nella coppia di punti *(start, end)* elaborabile dal modulo *Creazione mappa*.

5.1.1 Strides counter

Lo *Strides counter* ha la funzione di:

- Analizzare le informazioni ricevute dai sensori e determinare se l'utente è fermo o in movimento: se quest'ultimo è fermo, allora lo smartphone è nello stato chiamato *static*. Viceversa, se l'utente è in movimento, cioè subisce una variazione di accelerazione "rilevante" entro un certo intervallo di tempo, allora è nello stato chiamato *walk*.

  Quando il dispositivo è nello stato di *walk*, si suppone che l'utente stia effettivamente camminando. Pertanto, nel corso di questo progetto, si esclude la possibilità che l'utilizzatore lasci attiva l'applicazione anche quando svolge altre attività. Ad esempio, se l'utente è seduto in un autobus in movimento, il dispositivo potrebbe rilevare correttamente lo stato di *walk* (poiché il dispositivo subisce una variazione di accelerazione) ma in realtà l'utente non sta camminando.

- Quando l'utilizzatore cammina, cioè il dispositivo è nello stato di *walk*, lo *Strides counter* deve individuare e contare il numero di strides effettuati. Poiché il componente *Walk converter* necessita del numero di passi calcolato dallo *Strides counter* prima di procedere, si impone che il conteggio sia arrestato ogni qualvolta si verifichi la transizione dallo stato *walk* allo stato *static*. Al verificarsi di questo evento, lo *Strides counter* deve:
  - Fornire il numero di strides al *Walk converter*.
  - Inizializzare a 0 il conteggio degli strides in modo da essere pronto per la successiva porzione del cammino.

5.1.2 Orientation checker

L'*Orientation checker* ha il compito di calcolare l'azimuth iniziale del cammino effettuato dall'utente. Si impone che:

- L'azimuth deve essere determinato quando l'utente è fermo, cioè il dispositivo si trova nello stato *static*.
- Se l'azimuth è pari a 0 radianti allora la direzione del cammino punta verso il nord magnetico, se vale $\frac{\pi}{2}$ punta verso est, se vale $-\frac{\pi}{2}$ punta verso ovest e se vale $\pm\pi$ allora punta verso sud.
- Eventuali cambi del valore di azimuth effettuati dall'utente mentre cammina non sono rilevati. Pertanto, al fine di garantire il corretto funzionamento di questa prima implementazione del modulo *Contapassi*, l'utente dovrà mantenere traiettorie il più possibile rettilinee in modo tale da diminuire l'errore di approssimazione associato al dal segmento tra *start* e *end* (come indicato in figura 4.2).

Il valore dell'azimuth del cammino effettuato dall'utente deve essere comunicato al *Walk converter*.

5.1.3 Walk converter

Data la coppia *($n_{strides}$, azimuth)* per determinare la coppia di punti *(start, end)* è necessario, oltre all'utilizzo di funzioni trigonometriche, convertire il numero di strides nella lunghezza del cammino effettuata. Nella realizzazione di questo componente, si è accordato di associare a ciascun stride una lunghezza fissa. Sebbene quest'ultima dipende da molti fattori (frequenza del cammino, altezza ed età dell'utente...) si sceglie comunque di effettuare una conversione statica poiché l'eventuale errore commesso rientra nella tolleranza ammessa per un sistema di navigazione secondaria che questo progetto intende realizzare.

5.1.4 Strumenti di sviluppo

È stato accordato con il committente lo sviluppo di un insieme di strumenti esterni all'applicazione per poter analizzare e studiare con più accuratezza la sequenza di valori generati dai sensori dello smartphone durante il cammino effettuato dall'utente. Nel dettaglio si è accordato lo sviluppo di:

- Un'applicazione Android (chiamata *Sensor data carrier*) che:
  - Acquisisce i valori dai sensori dello smartphone.
  - Invia in tempo reale i dati acquisiti dai sensori tramite rete Wi-fi.
- Un'applicazione LabVIEW su sistema operativo Windows (chiamata *Acquire sensor data*) che riceve e salva su disco i dati dei sensori inviati dall'applicazione *Sensor data carrier*.
- Un'altra applicazione LabVIEW che:
  - Carica da disco i dati dei sensori salvati precedentemente.
  - Elabora e sperimenta le possibili strategie da utilizzare per l'implementazione dell'algoritmo dello *Strides counter* e dell'*Orientation checker*.

La documentazione relativa a questo progetto è presentata nell'appendice A.

5.2 Progettazione della soluzione

5.2.1 Posizione dello smartphone

Affinché l'applicazione operi in maniera corretta è necessario posizionare lo smartphone:

- In una delle due tasche laterali (cioè lungo i fianchi) dei pantaloni.
- Con orientazione simile a quella indicata in figura 5.1:
  - L'asse y dello smartphone (figura 5.2a) è allineato all'asse z del sistema di riferimento *Terra* (figura 5.2b) ma con verso opposto.
  - L'asse x dello smartphone è allineato al vettore spostamento dell'utente ma con verso opposto.

Vincolare la posizione dello smartphone permette di semplificare notevolmente il problema. Inoltre consente di raggiungere una maggiore accuratezza nell'acquisizione ripetuta dei valori dell'azimuth compiuti dall'*Orientation checker*.

Figura 5.1: Posizione del telefono affinché l'applicazione operi in modo corretto. Gli assi indicati si riferiscono al sistema di riferimento *Smartphone* originale.

5.2.2 Sensori disponibili in Android

Android mette a disposizione numerose API per interfacciarsi con un vasto insieme di sensori presenti nei vari dispositivi. Tuttavia, nella maggior parte degli smartphone, solamente un numero ridotto di sensori è disponibile. Generalmente, i modelli di fascia bassa del mercato, includono i seguenti sensori: accelerometro, magnetometro, sensore di prossimità e, in alcuni modelli, anche il sensore di luminosità.

È ragionevole ipotizzare che una persona con difficoltà visive non sia propensa ad acquistare costosi smartphone: quest'ultimi, infatti, oltre a possedere un ampio parco di sensori, includono ulteriori caratteristiche avanzate che l'utente potrebbe non riuscire a sfruttare. Ad esempio, a causa del suo handicap fisico, una persona con difficoltà visive difficilmente potrà riuscire a godere della migliore esperienza d'uso che uno schermo ad alta risoluzione, installato in questi dispositivi, può offrire.

Per tale motivo si preferisce sviluppare un'applicazione che faccia uso solamente dell'accelerometro e del magnetometro, in modo tale da raccogliere un bacino di utenza più vasto.

In aggiunta, è opportuno sottolineare che:

- Dalle API Android non è possibile definire una frequenza di campionamento per un sensore. È tuttavia possibile definire "un suggerimento" sul valore del ritardo tra un campione e l'altro[1].
- Non è possibile acquisire due campioni relativi a due sensori diversi allo stesso istante di tempo.

[1] "The delay that you specify is only a suggested delay. The Android system and other applications can alter this delay. As a best practice, you should specify the largest delay that you can because the system typically uses a smaller delay than the one you specify". Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.

Nel seguito del progetto si descrivono gli algoritmi utilizzati supponendo di prelevare i dati alla massima frequenza di campionamento consentita dai sensori (pari a circa 90 Hz sia per l'accelerometro che per il magnetometro).

5.2.3 Strides counter

5.2.3.1 Definizione di stride e step

Nel proseguo di questo documento, si utilizzerà una definizione di stride leggermente sfumata, a seguito della seguente osservazione: poiché l'applicazione prevede che lo smartphone sia inserito in una tasca dell'utente, è ragionevole ipotizzare che la sequenza di due passi che costituisce uno stride sia percepita come una sequenza di due picchi d'accelerazione. Poiché si impone di utilizzare solamente le tasche laterali dei pantaloni, se lo smartphone è inserito nella tasca destra allora esso percepirà il passo destro (cioè quello che prevede il movimento in avanti della gamba destra) con maggiore intensità rispetto al passo sinistro. Pertanto si definisce *stride* il passo effettuato con la stessa gamba nella cui tasca dei pantaloni si trova lo smartphone e *step* il passo effettuato con l'altra gamba. Con questa convenzione, una successione alternata di passi destri e sinistri si traduce in una successione alternata di strides e steps (o steps e strides).

5.2.3.2 Tracce d'acquisizione

Attraverso *Sensor data carrier* e l'applicativo LabView *Acquire sensor data*, sono state registrate un insieme di tracce relative ai valori dell'accelerazione percepita dallo smartphome mentre l'utente cammina. Infatti, al fine di realizzare lo *Strides counter*, è ragionevole utilizzare i valori dell'accelerometro, come osservato nella sezione 5.2.3.1.

In figura 5.3 sono rappresentate due tracce acquisite con due smartphone diversi. Seguono alcune considerazioni:

- Entrambe le tracce visualizzano i valori acquisiti dall'accelerometro relativi allo stesso cammino, in accordo con il sistema di riferimento dei dispositivi indicato in figura 5.2a: infatti, entrambi gli smartphone sono stati inseriti nella stessa tasca, sovrapposti uno sopra l'altro.
- In seguito si indicherà con il termine *vettore accelerazione* A la terna (a_x, a_y, a_z) in cui la componente a_i indica l'accelerazione lungo l'asse i-esimo.
- Si noti che, nonostante entrambe le tracce presentino lo stesso andamento, i valori assoluti percepiti dai sensori risultano diversi.

5.2.3.3 Analisi delle tracce

Osservando le tracce in figura 5.3 si può notare la presenza di picchi ad intervalli regolari. Ad esempio, nella traccia in figura 5.3a, in un breve intorno di $t_1 = 6400$ ms, $t_2 = 7500$ ms, $t_3 = 8700$ ms, $t_4 = 9800$ ms e $t_5 = 11000$ ms si osserva un aumento del valore assoluto di ognuno delle tre componenti dell'accelerazione percepita dallo smartphone. Tale fenomeno risulta più visibile se è calcolata la norma 2 (o norma euclidea indicata nella formula 5.1) del vettore accelerazione, come è rappresentato in figura 5.4a.

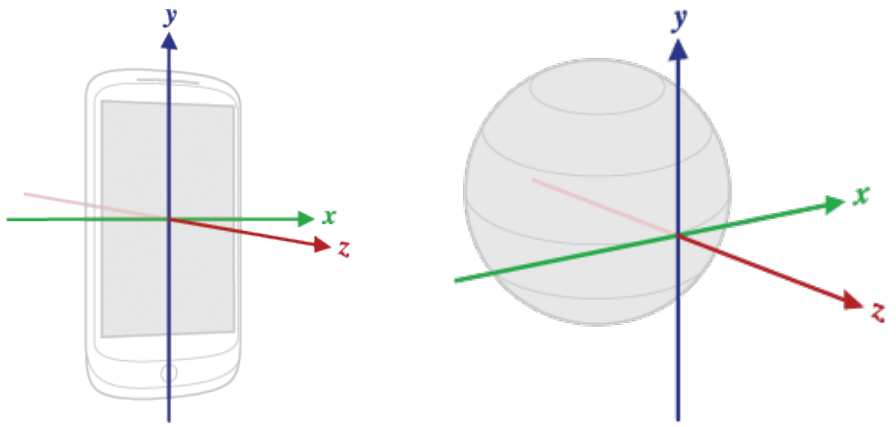
$$\|A\| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{5.1}$$

In quest'ultima traccia si può osservare:

- La corrispondenza tra i 5 picchi emersi e 5 strides effettivamente effettuati dall'utente.
- L'intervallo temporale che separa due picchi è il medesimo che separa due strides (poco più di un secondo).

Pertanto, una strategia per rilevare gli strides effettuati dall'utente consiste nel rilevare i picchi della funzione $\|A\|$ che superano una certa soglia.

Un'altra osservazione interessante riguarda i picchi di $\|A\|$ associati allo step. Si noti che:



(a) Sistema di riferimento dello *Smartphone*.

(b) Sistema di riferimento *Terra*. Gli assi x e y sono tangente al terreno nella locazione in cui si trova lo smartphone.

Figura 5.2: Sistemi di riferimento. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.

- Nella traccia in figura 5.4a, tali picchi non sono ben distinti.
- Nella traccia in figura 5.4b (eseguita sempre dallo stesso utente) essi sono ben visibili. Tuttavia all'ultimo strides, costituito dal passo con cui l'utente si ferma, corrisponde un picco di intensità minore rispetto ai precedenti.

Fin da questa prima disamina emerge l'elevata variabilità delle tracce generate da uno stesso utente, condizionate anche dalla frequenza della camminata e dalle calzature adottate.

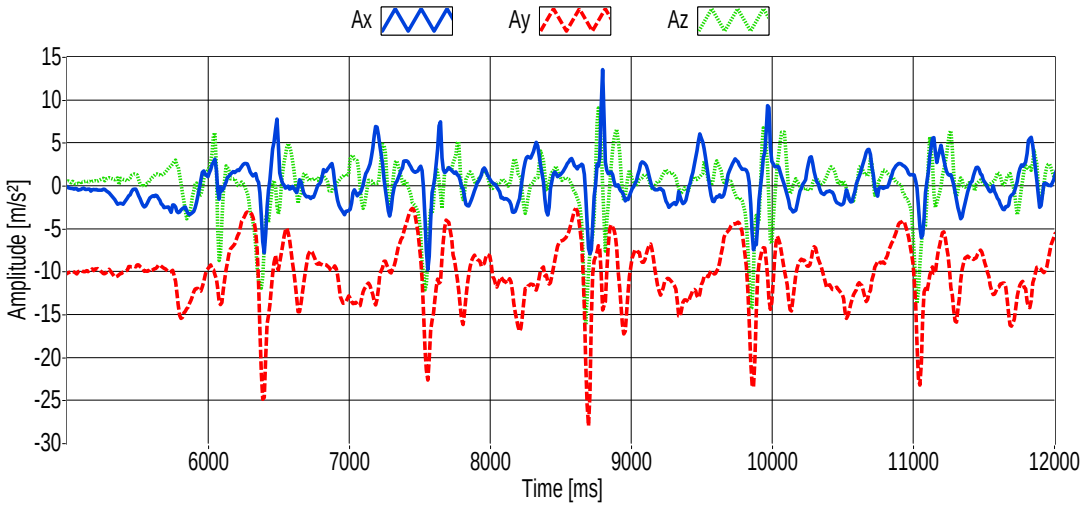
5.2.3.4 Algoritmo

Osservando le tracce 5.4a e 5.4b si costruisce un algoritmo per la rilevazione degli strides confrontando il valore corrente di $\|A\|$ con una soglia calcolata in funzione della media e della deviazione standard (campionaria) su una finestra mobile degli ultimi n campioni (algoritmo 5.1). In letteratura, idee analoghe sono descritte in [19, 25, 30].

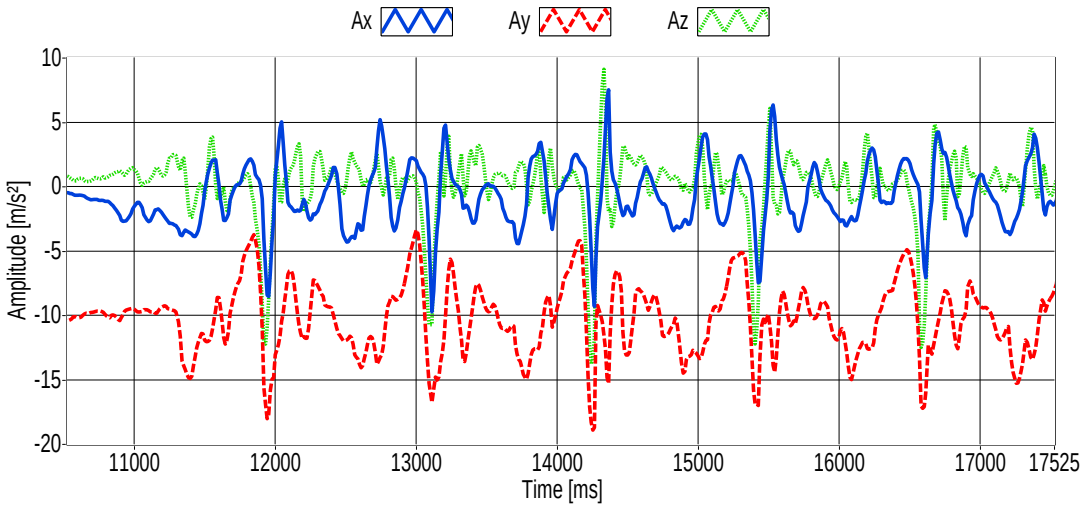
Seguono alcune considerazioni:

- Sia A_{peak}=$(A_1, \ldots, A_k)$ la sequenza di k valori consecutivi di $\|A\|$ tali che $\forall A_i \in A_{peak}$, $A_i > soglia$. L'algoritmo non rileva i picchi di accelerazione, ma individua solo il primo elemento della finestra di accelerazione di valore in modulo superiore alla soglia prefissata.
- Nel caso in cui l'utente effettui uno stride, più valori di $\|A\|$ consecutivi superano la relativa soglia: tuttavia tali valori sono associati sempre allo stesso stride. Affinché l'algoritmo non rilevi più volte lo stesso evento, si è fatto uso di un flag chiamato $stride\ detected$. Tale variabile assume il valore:
  - $TRUE$ quando è in corso l'identificazione di uno stride.
  - $FALSE$ altrimenti.

  Ovviamente, alla prima iterazione dell'algoritmo, $stride\ detected$ vale $FALSE$. Qualora una sequenza A_{peak} inerente ad uno stride costituisca l'input dell'algoritmo, il confronto $A_1 > soglia$ ritorna esito positivo. Di conseguenza l'algoritmo imposta il flag $stride\ detected$ a $TRUE$. Alla successiva iterazione, anche il confronto $A_2 > soglia$ fornisce esito positivo ma il valore $TRUE$ del flag $stride\ detected$ impedisce che l'algoritmo rilevi nuovamente lo stesso stride (riga di codice 13). Appena il valore attuale di $\|A\|$ è inferiore o uguale alla



(a) Motorola Moto G. File: `Tracce d'acquisizione/Stefano/1MA.csv`

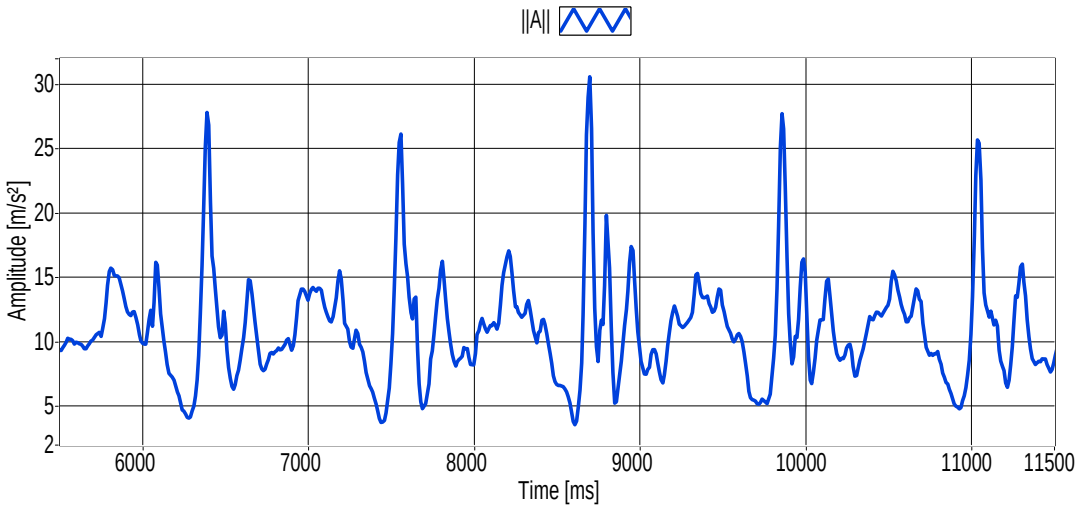


(b) Samsung S3. File: `Tracce d'acquisizione/Stefano/1SA.csv`

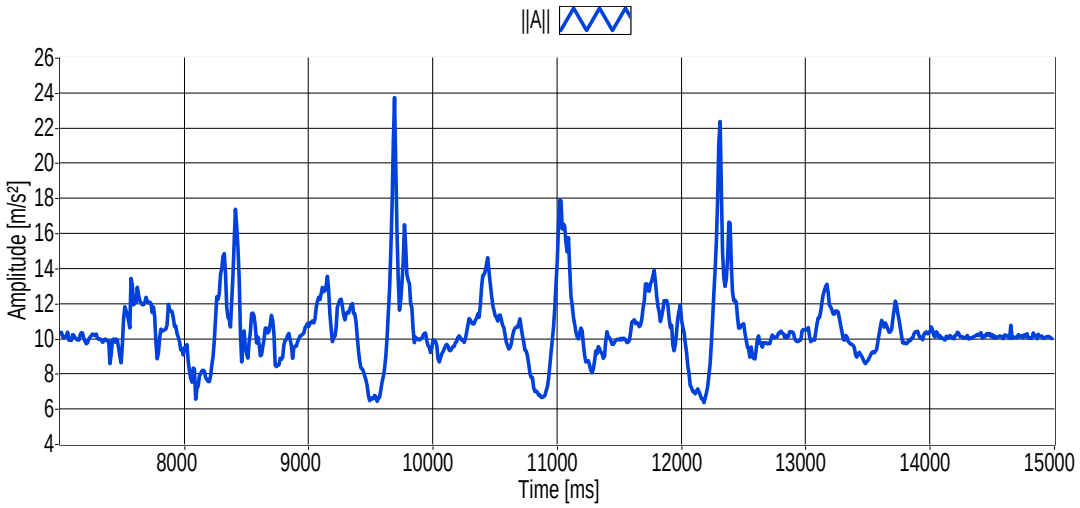
Figura 5.3: Tracce costituite dai valori acquisiti dall'accelerometro. Lo stesso evento (cioè il cammino), composto da 5 strides, è acquisito con due terminali diversi. Entrambi gli smartphone sono stati posizionali all'interno della stessa tasca, uno sovrapposto all'altro. L'asse delle ascisse indica un intervallo temporale diverso poiché l'applicazione *Sensor data carrier* non è stata avviata allo stesso istante per entrambi i dispositivi.

soglia, l'algoritmo imposta $stride\ detected$ a $FALSE$. Questa condizione segnala la fine del periodo di identificazione dello stride precedente: l'algoritmo è quindi pronto per un nuovo rilevamento.

- La figura 5.5 mette in risalto l'andamento della soglia in funzione della durata n della finestra mobile. Sperimentalmente $n = 100$ è un valore ragionevole che consente di non identificare gran parte degli steps (come avviene per $n = 50$) e di non tralasciare i picchi associati agli strides.
- In generale, il coefficiente k dovrebbe assumere un valore in funzione della durata della finestra mobile. Sperimentale si è scelto $k \in [1.5, 3.5]$.



(a) Relativo alla traccia 5.3a composta da 5 strides. Lo step che intervalla due strides non è facilmente rilevabile. File: `Tracce d'acquisizione/Stefano/1MA.csv`



(b) Relativo ad una traccia composta da 5 strides. Lo step che intervalla due strides è facilmente rilevabile. Tuttavia all'ultimo stride, costituito dal passo con cui l'utente si ferma, corrisponde un picco di intensità minore rispetto ai precedenti. File: `Tracce d'acquisizione/Stefano/3MA.csv`

Figura 5.4: Norma 2 (o norma euclidea) del vettore accelerazione.

- Nella figura 5.6a si può osservare il ridotto numero di elementi che costituiscono la sequenza A_{peak}. Per tale motivo è opportuno non scegliere frequenze di campionamento troppo piccole. La sezione 5.3.1 discute tale problematica.
- Nella figura 5.6a si può osservare la presenza di due picchi in corrispondenza del primo step effettuato dall'utente. Tale circostanza si verifica a causa dell'elevata durata della finestra mobile: è necessario quindi determinare un adeguato compromesso sulla sua dimensione.

L'algoritmo 5.2 propone una serie di miglioramenti all'algoritmo principale:

- I calcoli per identificare uno stride sono eseguiti solo se l'utente è in movimento, cioè se la deviazione standard supera una soglia σ_{static} (riga 12).

Algoritmo 5.1 Strides counter.

Input: n, k
Output: $n_{strides}$

```
1:  function STRIDES COUNTER(n, k)
2:      valori ← insieme contenete gli ultimi n valori della norma 2 calcolata sui campioni
                 dell'accelerometro
3:      stride detected ← FALSE          ▷ Flag, vale TRUE quando è in corso l'identificazione di
                                           uno stride
4:      while in funzione do
5:          a ← GET ACCELERAZIONE( )
6:          a_norm ← GET NORMA(a)
7:          RIMUOVI IL VALORE PIÙ VECCHIO(valori)
8:          valori ← valori ∪ a_norm
9:          μ ← CALCOLA MEDIA(valori)
10:         σ ← CALCOLA DEVIAZIONE STANDARD CAMPIONARIA(valori)
11:         soglia ← μ + kσ
12:         if a_norm > soglia then
13:             if stride detected = FALSE then
14:                 n_strides ← n_strides + 1
15:                 stride detected ← TRUE
16:         else
17:             stride detected ← FALSE
18: return n_strides
```

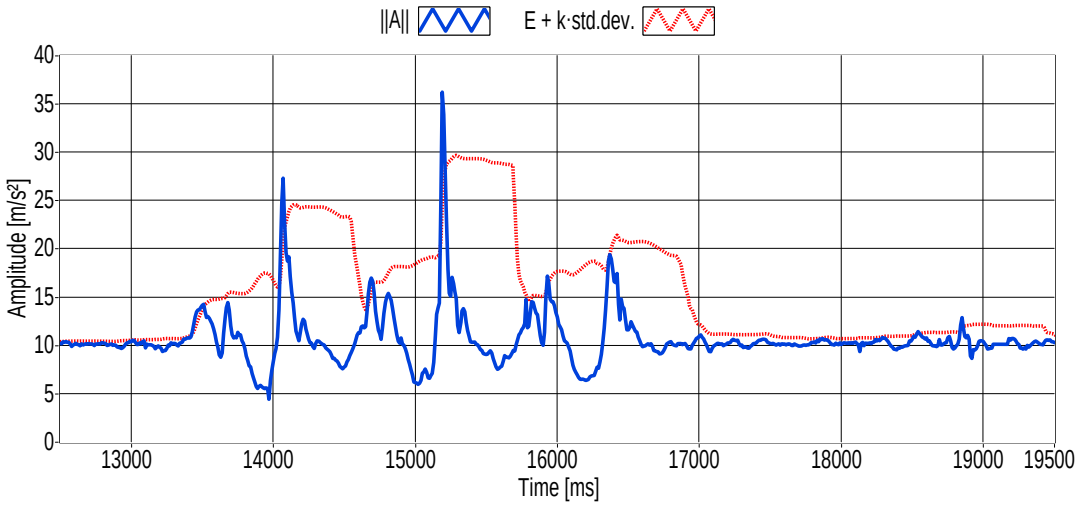
- Se il dispositivo è nello stato *static*, si sceglie di ridurre (se possibile) la dimensione della finestra mobile (righe 13 e 14); il valore minimo che la durata può assumere è n_{min}. Viceversa, se il dispositivo è nello stato *walk*, si sceglie (se possibile) di incrementare la dimensione della finestra (righe 32 e 33); il valore massimo che la durata può assumere è n_{max}.
- Per mitigare il problema evidenziato in figura 5.6b, si sceglie di utilizzare due diversi fattori k che moltiplicano la deviazione standard: $k_{transitorio}$ e $k_{operativo}$. La scelta di quale valore utilizzare (riga 16) dipende dall'intensità della deviazione standard; la soglia σ_k, determinata sperimentalmente, svolge tale funzione. L'idea consiste nell'utilizzare un soglia più alta (grazie a $k_{transitorio}$, in cui $k_{transitorio} > k_{operativo}$) quando il dispositivo esce dallo stato di *static* e la deviazione standard non ha ancora raggiunto un valore sufficientemente elevato per generare una soglia adeguata.
- Si impone che se la soglia non raggiunga un valore minimo (riga 21), essa sia rimpiazzata da $soglia_{min}$. Tale esigenza nasce dal fatto di poter permettere l'inserimento e l'estrazione dello smartphone senza che vengano individuati strides inesistenti. Tale soglia è determinata sperimentalmente.
- Si sceglie di non consentire il rilevamento di strides troppo ravvicinati temporalmente tra loro (riga 26). Tale esigenza nasce dal fatto che, in certe circostanze, il (primo) picco associato ad uno stride è seguito, entro un breve intervallo (circa 200 ms) da un altro picco (figura 5.7). Alternativamente a questa soluzione, è possibile filtrare il segnale per eliminare le suddette fluttuazioni ad alta frequenza.

La figura 5.8 illustra il risultato dell'algoritmo 5.2.

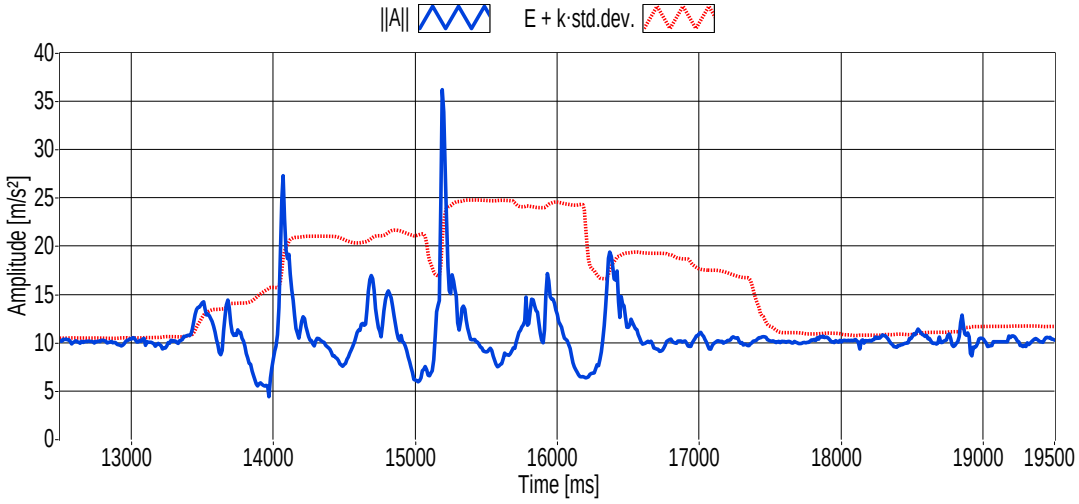
Algoritmo 5.2 Strides counter migliorato.

Input: $n_{min}, n_{max}, k_{transitorio}, k_{operativo}, \sigma_k, \sigma_{static}, soglia_{min}, soglia_{\Delta T}$
Output: $n_{strides}$

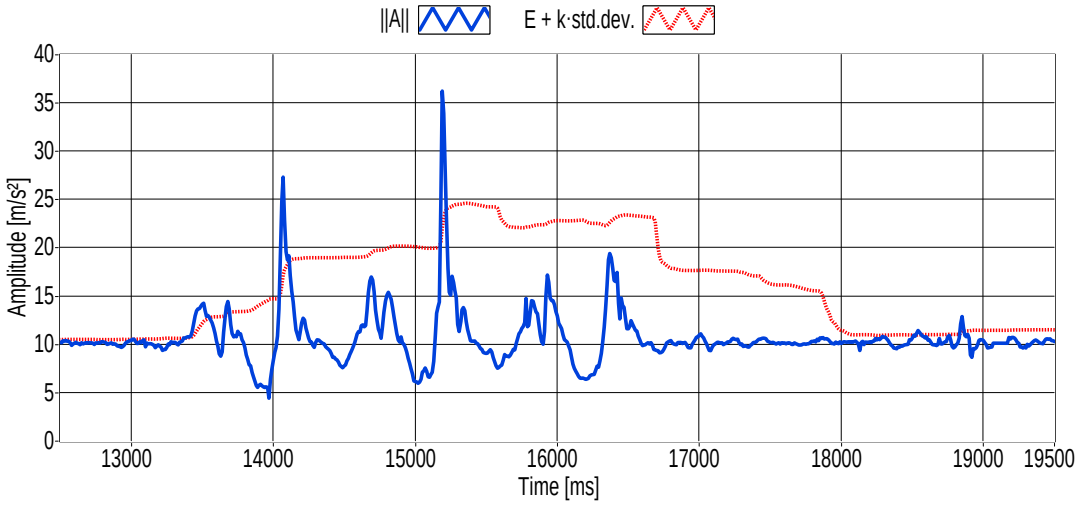
1: **function** STRIDES COUNTER($n_{min}, n_{max}, k_{transitorio}, k_{operativo}, \sigma_k, \sigma_{static}, soglia_{min}, soglia_{\Delta T}$)
2: $valori \leftarrow$ insieme contenete gli ultimi n valori della norma 2 calcolata sui campioni dell'accelerometro
3: $stride\ detected \leftarrow FALSE$ ▷ Flag, vale $TRUE$ quando è in corso l'identificazione di uno stride
4: $Ultimo\ timestamp \leftarrow 0$ ▷ Mantiene il timestamp dell'ultimo stride individuato
5: **while** in funzione **do**
6: $a \leftarrow$ GET ACCELERAZIONE( )
7: $a_{norm} \leftarrow$ GET NORMA(a)
8: RIMUOVI IL VALORE PIÙ VECCHIO($valori$)
9: $valori \leftarrow valori \cup a_{norm}$
10: $\mu \leftarrow$ CALCOLA MEDIA($valori$)
11: $\sigma \leftarrow$ CALCOLA DEVIAZIONE STANDARD CAMPIONARIA($valori$)
12: **if** $\sigma < \sigma_{static}$ **then**
13: **if** $|valori| > n_{min}$ **then**
14: RIMUOVI IL VALORE PIÙ VECCHIO($valori$)
15: **else**
16: **if** $\sigma > \sigma_k$ **then**
17: $k \leftarrow k_{transitorio}$
18: **else**
19: $k \leftarrow k_{operativo}$
20: $soglia \leftarrow \mu + k\sigma$
21: **if** $soglia < soglia_{min}$ **then**
22: $soglia \leftarrow soglia_{min}$
23: **if** $a_{norm} > soglia$ **then**
24: **if** $stride\ detected = FALSE$ **then**
25: $\Delta T \leftarrow$ GET TIMESTAMP(a_{norm}) - $Ultimo\ timestamp$
26: **if** $\Delta T > soglia_{\Delta T}$ **then**
27: $n_{strides} \leftarrow n_{strides} + 1$
28: $Ultimo\ timestamp \leftarrow$ GET TIMESTAMP(a_{norm})
29: $stride\ detected \leftarrow TRUE$
30: **else**
31: $stride\ detected \leftarrow FALSE$
32: **if** $|valori| < n_{max}$ **then**
33: RIPRISTINA IL VALORE PIÙ VECCHIO($valori$)
34: **return** $n_{strides}$



(a) $n = 50$.

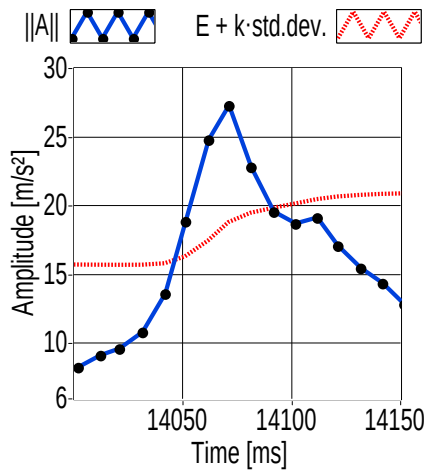


(b) $n = 100$.

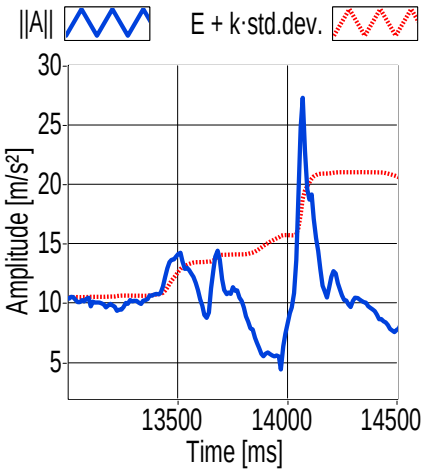


(c) $n = 150$.

Figura 5.5: Soglia in funzione della taglia n della finestra mobile. Il coefficiente moltiplicativo k della deviazione standard vale 2.5. File: `Tracce d'acquisizione/Stefano/4MA.csv`



(a) Un numero limitato di campioni superano la soglia.



(b) Il primo passo viene rilevato (2 volte) anche se è uno step poiché la soglia calcolata è bassa.

Figura 5.6: Focus della traccia 5.5b (con i medesimi parametri d'esecuzione). File: `Tracce d'acquisizione/Stefano/4MA.csv`.

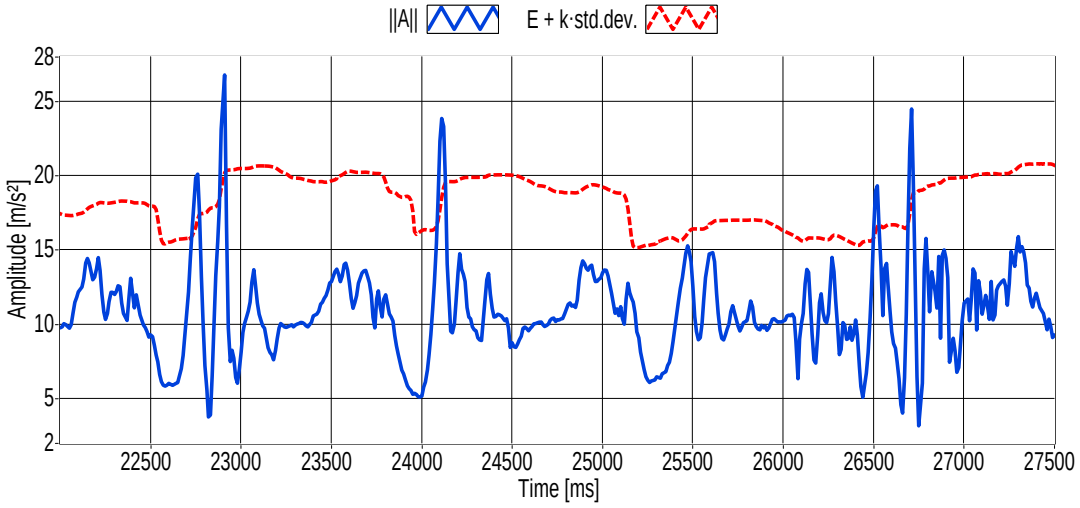
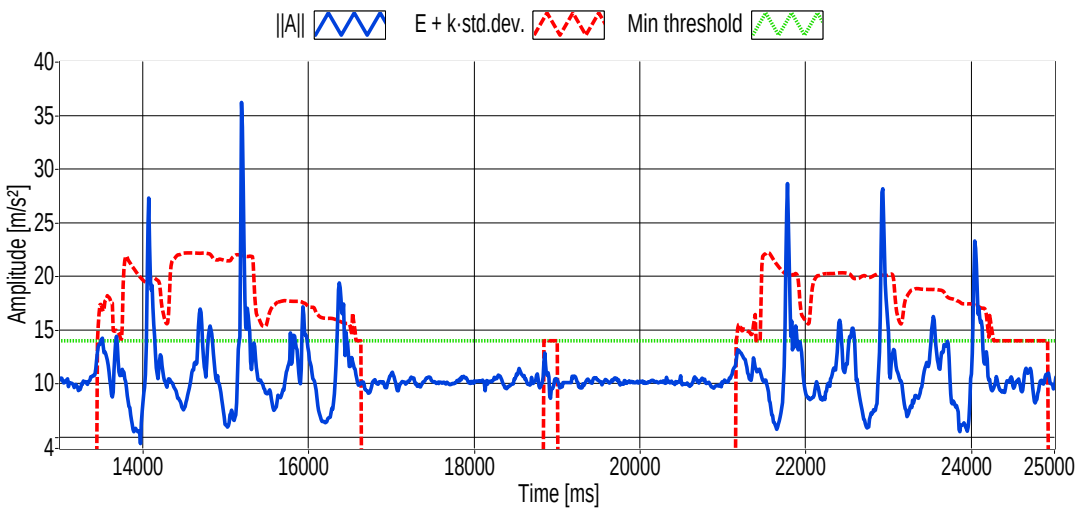
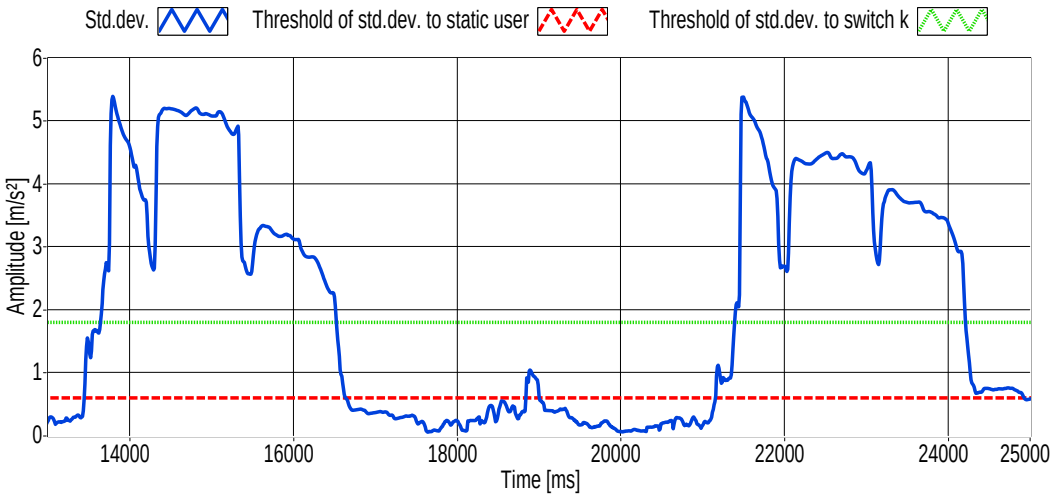


Figura 5.7: $n = 100, k = 2.5$. Si noti la successione ravvicinata (entro 200 ms) di due picchi di intensità in corrispondenza degli strides a $t_1 = 22750$ ms e $t_2 = 26700$ ms. File: `Tracce d'acquisizione/Stefano/1MA.csv`



(a) $n_{min} = 15$, $n_{max} = 100$, $k_{transitorio} = 3.5$, $k_{operativo} = 2$, $soglia_{min} = 14\ m/s^2$, $soglia_{\Delta T} = 700\ ms$. La soglia non è calcola se il dispositivo è nello stato di *static*.



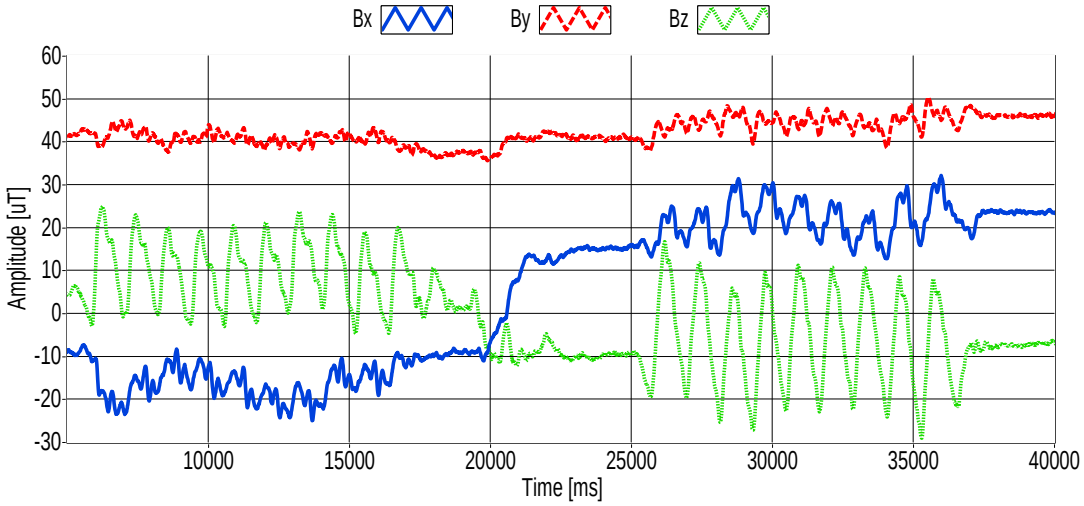
(b) Andamento della deviazione standard. $\sigma_{static} = 0.6\ m/s^2$, $\sigma_k = 1.8\ m/s^2$.

Figura 5.8: Traccia generata con l'algoritmo *Strides counter migliorato*. File: `Tracce d'acquisizione/Stefano/4MA.csv`

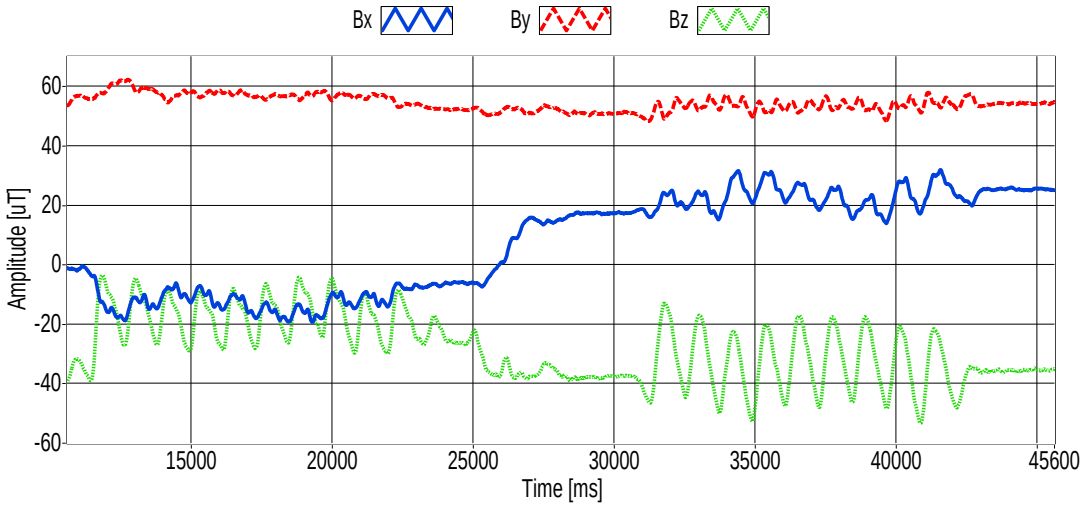
5.2.4 Orientation checker

5.2.4.1 Tracce d'acquisizione

In figura 5.9 sono rappresentate le tracce d'acquisizione dei valori del magnetometro. Come si può osservare, l'accelerazione variabile a cui è sottoposto il dispositivo quando l'utente cammina perturba i valori acquisiti dal magnetometro. Per tale motivo, si è scelto di semplificare il problema calcolando l'azimuth quando l'utente è fermo.



(a) Motorola Moto G. File: `Tracce d'acquisizione/Stefano/1MM.csv`.



(b) Samsung S3. File: `Tracce d'acquisizione/Stefano/1SM.csv`.

Figura 5.9: Tracce costituite dai valori acquisiti dal magnetometro. Il cammino è lo stesso delle acquisizioni in figura 5.3. Tuttavia le tracce si estendono per una durata maggiore: infatti il cammino rettilineo composto da 5 strides è seguito da una rotazione di 180° (individuabile all'istante precedente alle porzioni stazionarie delle tracce) e da un altro cammino rettilineo composto da 4 strides.

5.2.4.2 Calcolo dell'azimuth

Anziché sviluppare da zero l'algoritmo per la determinazione dell'azimuth, si è scelto di utilizzare le funzioni offerte da Android (`getRotationMatrix` e `getOrientation` della classe `SensorManager`) che utilizzano le misure acquisite dall'accelerometro e dal magnetometro.

Il problema da risolvere consiste nel determinare l'orientazione del sistema di riferimento *Smartphone* (indicato con S) rispetto al sistema di rifermento *Terra* (indicato con T). Al fine di semplificare la spiegazione, si supponga di mantenere lo smartphone appoggiato ad un piano parallelo al piano xy del sistema di riferimento T (ad es. posizionando il dispositivo su un tavolo con lo schermo rivolto verso il cielo). Poiché entrambi i sistemi di riferimento hanno la stessa origine, è necessario:

- Determinare la matrice di rotazione R_S^T che consente di esprimere un generico punto del sistema di riferimento S in funzione del sistema di riferimento T.
- Data la matrice R_S^T, calcolare l'azimuth, cioè l'angolo di rotazione attorno all'asse z del sistema di riferimento T (cioè l'angolo compreso tra la il vettore spostamento dell'utente e il nord magnetico terreste).

Per descrivere la matrice di rotazione R_S^T si procede in più passi:

1. Si determinano tre versori associati al sistema di riferimento T espressi secondo il sistema di riferimento S, in accordo con l'algoritmo 5.3. Questa base ortonormale è determinata combinando (tramite l'operazione di prodotto vettoriale, modulo e norma di un vettore) i valori di accelerazione e campo magnetico acquisiti dai sensori dello smartphone. I tre versori E, N e G del sistema T espressi in S puntano rispettivamente a est, a nord e al centro della *Terra*.

2. In accordo con [50], si descrive la giacitura del sistema T rispetto ad S (cioè l'orientamento dei versori di T calcolati al punto precedente, cioè E, N e G) tramite la matrice di rotazione:

$$R_T^S = \begin{bmatrix} e_x & n_x & g_x \\ e_y & n_y & g_y \\ e_z & n_z & g_z \end{bmatrix}$$

3. Per l'interesse dell'applicazione, è necessario disporre della matrice di rotazione R_S^T, cioè quella che rappresenta il sistema S rispetto a T. Infatti, si è interessati alla posizione dello smartphone rispetto al nord magnetico terrestre. Poiché la matrice di rotazione è ortogonale, l'inversa corrisponde alla trasposta. Quindi:

$$R_S^T = (R_T^S)^T = \begin{bmatrix} e_x & e_y & e_z \\ n_x & n_y & n_z \\ g_x & g_y & g_z \end{bmatrix}$$

Data la matrice R_S^T, attraverso le formule di Eulero è possibili calcolare il valore dell'azimuth come segue:

$$azimuth = arctg2(e_y, n_y)$$

In questo documento non si approfondisce la matematica da cui deriva tale formula. È possibile reperire buoni riferimenti in [51, 52] o alternativamente documentarsi sulla fattorizzazione della matrice $R_z R_y R_x$.

A completare l'esamina, si ricorda che le precedenti formule si riferiscono ad un'orientazione dello smartphone diversa rispetto a quella in cui il dispositivo si troverà effettivamente quando il programma è in esecuzione (figura 5.1). A tale scopo è necessario mappare il sistema di

Algoritmo 5.3 Calcolo versori del sistema di riferimento Terra.

Input: $A = [a_x, a_y, a_z]$ è il vettore accelerazione, $B = [b_x, b_y, b_z]$ è il vettore campo magnetico. Entrambi i vettori sono composti dai valori grezzi percepiti dai sensori.

Output: $E = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}$, $N = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$, $G = \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix}$

1: **function** GETROTATIONMATRIX(A, B)
2: $A_x = \begin{bmatrix} a_x \\ 0 \\ 0 \end{bmatrix}$, $A_y = \begin{bmatrix} 0 \\ a_y \\ 0 \end{bmatrix}$, $A_z = \begin{bmatrix} 0 \\ 0 \\ a_z \end{bmatrix}$,
3:
4: $B_x = \begin{bmatrix} b_x \\ 0 \\ 0 \end{bmatrix}$, $B_y = \begin{bmatrix} 0 \\ b_y \\ 0 \end{bmatrix}$, $B_z = \begin{bmatrix} 0 \\ 0 \\ b_z \end{bmatrix}$,
5: $E = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}$, $N = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$, $G = \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix}$,
6: $E_x \leftarrow B_y \times A_z - B_z \times A_y$
7: $E_y \leftarrow B_z \times A_x - B_x \times A_z$
8: $E_z \leftarrow B_x \times A_y - B_y \times A_x$
9: $e_x \leftarrow |E_x|$
10: $e_y \leftarrow |E_y|$
11: $e_z \leftarrow |E_z|$
12: $E \leftarrow \frac{E}{\|E\|}$
13: $g_x = |A_x|$
14: $g_y = |A_y|$
15: $g_z = |A_z|$
16: $G \leftarrow \frac{G}{\|G\|}$
17: $n_x \leftarrow |A_y \times E_z - A_z \times E_y|$
18: $n_y \leftarrow |A_z \times E_x - A_x \times E_z|$
19: $n_z \leftarrow |A_x \times E_y - A_y \times E_x|$
20: $N \leftarrow \frac{N}{\|N\|}$
21: **return** $E = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}$, $N = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$, $G = \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix}$

riferimento *Smartphone* su un nuovo sistema di riferimento attraverso la seguente mappa:

$$\begin{aligned} x &\leftarrow -y_{old} \\ y &\leftarrow -z_{old} \\ z &\leftarrow x_{old} \end{aligned} \tag{5.2}$$

La nuova matrice di rotazione R_S^T diventa:

$$R_S^T = \begin{bmatrix} e_z & -e_x & -e_y \\ n_z & -n_x & -n_y \\ g_z & -g_x & -g_y \end{bmatrix}$$

In questo modo l'azimuth è l'angolo nel piano xy del sistema di riferimento T tra il nord magnetico e l'opposto dell'asse x originale del dispositivo (cioè il "lato corto") . Il suo valore vale:

$$azimuth = arctg2(e_x, n_x)$$

La funzione $arctg2(y, x)$ è definita $\forall (y, x) \in \mathbb{R}^2$ con $(y, x) \neq (0, 0)$. Si noti che, grazie all'accelerazione dovuta alla forza di gravità e percepita dallo smartphone, la coppia (e_x, n_x) non può mai assumete il valore $(0, 0)$.

5.2.4.3 Algoritmo

L'azimuth è calcolato solo quando il dispositivo è nello stato di *static*; quando avviene la transazione da *static* a *walk*, l'*Orientation checker* fornisce l'azimuth del segmento di cammino che l'utente sta effettuando. Si sceglie di restituire l'azimuth come valore della media di una finestra mobile degli ultimi n valori calcolati nello stato di *static*.

Sperimentalmente, si sceglie di mantenere la durata della finestra mobile piccola (circa 20 campioni che ad una frequenza di campionamento di circa 90 Hz corrispondono a circa 4 decimi di secondo). Ovviamente, al fine di ottenere un valore coerente con l'effettiva orientazione di partenza dell'utente, si consiglia di attendere pochi decimi di secondo (senza ruotare) prima di iniziare il cammino.

5.2.4.4 Cambio del sistema di riferimento smartphone

Utilizzando l'algoritmo proposto, l'azimuth assume il valore 0 quando lo smartphone è orientato verso nord (cioè quando l'asse $-x$ del sistema S punta al nord magnetico), $-\frac{\pi}{2}$ quando punta a ovest, $\frac{\pi}{2}$ quando punta a est e assume un valore in un intorno $I = I_1 \cup I_2$ quando punta a sud, in cui $I_1 = (-\pi + \epsilon, -\pi)$ e $I_2 = (\pi - \epsilon, \pi)$. Se l'utente sta procedendo verso sud (circa), la finestra mobile contiene valori appartenenti all'intervallo non continuo $I_1 \cup I_2$. La media calcolata dalla finestra potrebbe restituire un valore non coerente con l'orientazione dell'utente. Ad esempio, se $n = 4$ e la finestra contiene i valori (espressi in gradi) $a_1 = 175, a_2 = -175, a_3 = 179, a_4 = -177$, l'azimuth restituito è 2, che rappresenta un'orientazione verso nord e quindi errata.

Per risolvere tale problema si sceglie di adottare due sistemi di riferimento *Smartphone* (figura 5.10):

- Il sistema di riferimento *principale* utilizza la mappa 5.2.
- Il sistema di riferimento *secondario* utilizza la seguente mappa:

$$\begin{aligned} x &\leftarrow y_{old} \\ y &\leftarrow -z_{old} \\ z &\leftarrow x_{old} \end{aligned} \tag{5.3}$$

  Con quest'ultimo sistema di riferimento, l'azimuth è definito come l'angolo nel piano xy di T tra il sud e l'asse x (originale) dello smartphone. In questo modo l'azimuth assume il valore 0 quando il dispositivo è orientato verso sud, $-\frac{\pi}{2}$ quando punta a est, $-\frac{\pi}{2}$ quando punta a ovest e assume un valore in un intorno $(-\pi + \epsilon, -\pi) \cup (\pi - \epsilon, \pi)$ quando punta a nord.

Si definiscono due soglie: $s_1 = -\frac{5}{6}\pi$ e $s_2 = \frac{5}{6}\pi$. Quando l'applicazione è avviata, l'*Orientation checker* utilizza il sistema di riferimento *principale*. Se durante il calcolo dell'azimuth, esso assume un valore compreso in $(-\pi, s_1) \cup (s_2, \pi)$, si procede a:

1. Selezionare il sistema di riferimento *secondario*.
2. Convertire tutti i valori dell'azimuth salvati nella finestra mobile nel nuovo sistema di riferimento. Per far ciò si utilizza la seguente formula:

$$azimuth_{s_2} = \begin{cases} azimuth_{s_1} - \pi & \text{if } azimuth_{s_2} \geq 0 \\ azimuth_{s_1} + \pi & \text{if } azimuth_{s_2} < 0 \end{cases} \tag{5.4}$$

L'idea di questa procedura consiste nel cambiare sistema di riferimento quando l'utente assume un'orientazione in prossimità del sud (come ben evidenziato nella figura 5.10).

Per simmetria, anche quando si utilizza il sistema di riferimento secondario, se l'azimuth assume un valore compreso in $(-\pi, s_1)$ o (s_2, π), il quale indica che l'utente sta avanzando verso nord, si procede a selezionare nuovamente il sistema di riferimento principale e a convertire tutti i valori della finestra mobile in accordo con l'equazione 5.4.

Si noti che, se l'utente sta procedendo con orientazione simile ad una delle due soglie (ad esempio, $azimuth_{s_1} = \frac{5}{6}\pi$), quando avviene lo switch verso l'altro sistema di riferimento, il nuovo azimuth non è in prossimità di una delle due soglie (riprendendo l'esempio, $azimuth_{s_2} = -\frac{\pi}{6}$); in questo modo non si verificano continui cambi di sistema di riferimento entro brevi intervalli di tempo.

Affinché il valore dell'azimuth restituito alla funzione chiamante sia coerente con tutti i precedenti valori restituiti, si sceglie di convertire ciascun azimuth da restituire come valore espresso nel sistema di riferimento principale.

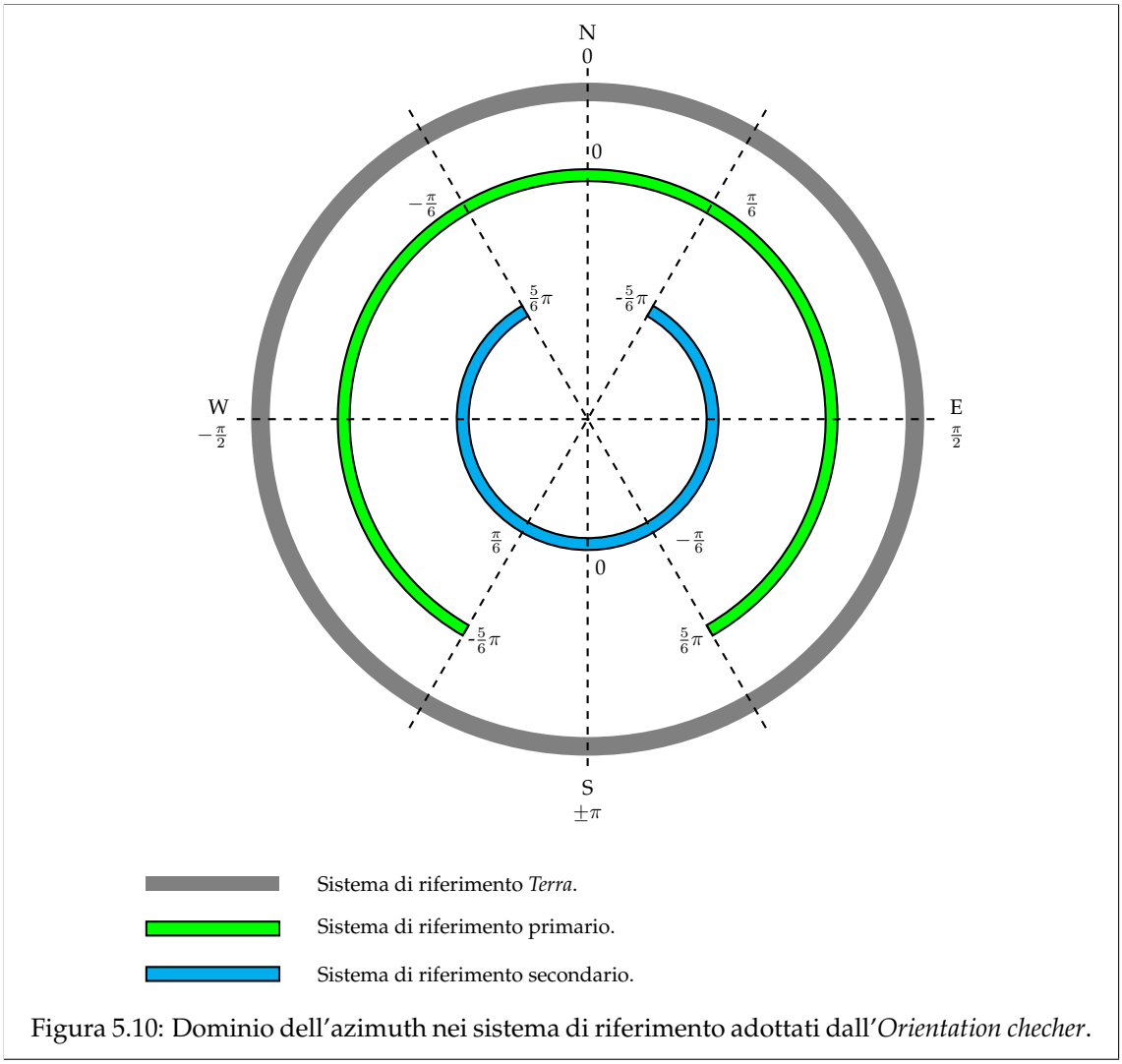


Figura 5.10: Dominio dell'azimuth nei sistema di riferimento adottati dall'*Orientation checher*.

5.2.4.5 Calcolo del pitch

Le prime sperimentazioni hanno evidenziato come l'operazione di estrazione ed inserimento dello smartphone induce lo *Strides counter* a rilevare falsi strides. Per limitare questo fenomeno, oltre all'inserimento della soglia $soglia_{min}$, si è scelto di disabilitare il rilevamento degli strides se lo

Altezza utente [cm]	Lunghezza stride [cm]
$h \leq 165$	140
$165 < h \leq 175$	145
$h > 175$	150

Tabella 5.1: Corrispondenza tra altezza dell'utente e la lunghezza dello stride assunte dall'applicazione.

smartphone non raggiunge un'orientazione simile alla posizione di lavoro. A tale scopo, si è scelto di valutare il beccheggio (*pitch*) dello smartphone, ovvero l'angolo di rotazione attorno all'asse x[a] del sistema di riferimento *Smartphone* originale. Per le formule di Eulero tale angolo vale:

$$pitch = arctg2(-e_x, -n_x)$$

Quando lo smartphone si trova in mano nella posizione naturale per essere usato, il $pitch$ vale circa $\pm\frac{\pi}{3}$, a seconda della mappa usata per il sistema di riferimento S. Quando il dispositivo è in tasca, il $pitch$ vale circa 0.

Da questa analisi si è scelto di inserire due soglie: $pitch_{min}$ e $pitch_{max}$. Quando $|pitch| < pitch_{min}$ lo smartphone entra nella zona di lavoro in cui lo *Stride counter* è attivo per il rilevamento degli strides. Viceversa, quando $|pitch| > pitch_{max}$ il dispositivo esce dalla zona di lavoro e il rilevamento degli strides è disattivato. Sperimentalmente si sono determinati due possibili valori: $|pitch_{min}| = \frac{\pi}{8}$ e $|pitch_{max}| = \frac{\pi}{4}$.

5.2.5 Walk converter

Data la coppia $(n_{strides}, azimuth)$, si determina l'incremento Δ_x e Δ_y del segmento di cammino attraverso le seguenti formule:

$$\begin{aligned} \Delta_x &= \sin(azimuth)\ \ l_{stride}\ \ n_{strides} \\ \Delta_y &= \cos(azimuth)\ \ l_{stride}\ \ n_{strides} \end{aligned} \tag{5.5}$$

Se (x_0, y_0) sono le coordinate iniziali del segmento, $(x_0 + \Delta_x, y_0 + \Delta_y)$ corrisponde al punto finale *(start, end)* del segmento di cammino.

La determinazione della lunghezza di uno stride è un problema aperto. Esistono numerosi modelli, di cui si può trovare una buona panoramica in [53]. Nel contesto dell'applicazione si semplifica notevolmente il problema assumendo un valore costante; nella tabella 5.1 è illustrata la lunghezza dello stride in funzione dell'altezza dell'utente (in accordo con [54]).

5.3 Progettazione software del modulo

5.3.1 Acquisizione dati dai sensori

A grandi linee, per acquisire i dati da un sensore dello smartphone è necessario:

1. Acquisire un'istanza sm di `SensorManager`.
2. Acquisire un'istanza s del sensore da cui si vuole ottenere le misurazioni.
3. Creare un'istanza sel di `SensorEventListener` che ha il compito di svolgere l'elaborazione all'arrivo di un nuovo campione dal sensore (evento "nuovo dato").

[a] In realtà poiché il sistema di riferimento dello smartphone è stato mappato, la rotazione che corrisponde al beccheggio naturale del dispositivo nel sistema di riferimento *Smartphone* corrisponde ad un angolo di *roll* nel nuovo sistema di riferimento mappato. Tuttavia, per non generare confusione si continua, erroneamente, a chiamare tale angolo con il termine *pitch*.

4. Registrare, tramite sm, la coppia (s, sel). Questo significa che ogni qualvolta il sensore, a cui s è associato, acquisisce un nuovo dato, l'evento "nuovo dato" è elaborato dall'istanza di sel.

Un primo approccio consiste nell'implementare una sottoclasse di `SensorEventListener` che si occupi di svolgere le funzione dello *Strides detector* e dell'*Orientation checker*. Tuttavia tale approccio presenta alcuni inconvenienti:

- Il sistema Android non fornisce nuovi campioni generati dal sensore se l'elaborazione del campione precedente, per opera dell'istanza sel, non è terminata. Questo implica che, per non perdere campioni, l'elaborazione deve essere efficiente e non superare il tempo che intercorre tra un campione e l'altro.
- Il campione del sensore fornito da Android è salvato su un oggetto temporaneo, il cui ciclo di vita (cioè la sovrascrittura del suo valore) è responsabilità del sistema Android stesso. Per tale motivo è necessario salvare il campione in modo tale che le finestre mobili adottate dallo *Strides detector* e dall'*Orientation checker* possano elaborare dati consistenti.

Si sceglie quindi di disaccoppiare l'acquisizione dei dati dall'accelerometro e dal magnetometro dalla loro elaborazione effettuata da parte dello *Strides detector* e dell'*Orientation checker*. Tale disaccoppiamento è realizzato in accordo alle seguente direttive:

- Per ogni sensore (accelerometro e magnetometro) è registrato una diversa istanza di `SensorEventListener`, ognuna con le seguenti caratteristiche:
  - L'esecuzione del codice in risposta all'evento "nuovo dato" è effettuata da un thread privato, diverso dal thread principale su cui risiede l'interfaccia grafica (UI thread). Si indica con T_A il thread che ha in carico la gestione dell'evento associato all'accelerometro e con T_M il thread associato agli eventi del magnetometro.
  - Il dato del campione è salvato nell'ultima posizione di una coda privata. Ciò implica la presenza di due code: la prima contenente i campioni dell'accelerometro (chiamata c_A), la seconda quelli del magnetometro (chiamata c_M).

  L'istanza di `SensorEventListener` associata all'accelerometro ha il compito di salvare, tramite T_A, il nuovo campione dell'accelerazione nella coda c_A. Invece, l'istanza di `SensorEventListener` associata al magnetometro ha il compito di salvare, tramite T_M, il nuovo campione del campo magnetico nella coda c_M.
- L'implementazione dello *Strides counter* avviene su un thread privato che acquisisce i campioni da c_A. Si indica con T_{SC} tale thread.
- L'implementazione dell'*Orientation checker* avviene su un thread privato che acquisisce i campioni sia da c_A che da c_M. Si indica con T_{OC} tale thread.

5.3.1.1 Struttura dati: blocking queue

Di seguito si discute come deve essere regolato l'accesso ai campioni di c_A e c_M, in quanto esse costituiscono due risorse condivise accessibili da più thread.
Hanno accesso a c_A i seguenti threads:

- T_A con l'operazione di *inserimento in coda*.
- T_{SC} con l'operazione di *estrazione in testa*.
- T_{OC} con l'operazione di *estrazione in testa*.

Hanno accesso a c_M i seguenti thread:

- T_M, con l'operazione di *inserimento in coda*.
- T_{OC}, con l'operazione di *estrazione in testa*.

È bene sottolineare che nascono esigenze di sincronizzazione. Infatti è necessario:

- Assicurare che non sia eseguita un'*estrazione in testa* se la relativa coda è vuota.
- Garantire che il T_{OC} sia in grado di reperire un campione dalla coda dell'accelerometro e una campione dalla coda del magnetometro tali che la differenza tra i relativi timestamp sia minima.

Per assicurare la prima condizione è possibile utilizzare una delle strutture dati messe a disposizione da Android (ad esempio, `ArrayBlockingDeQueue` o `LinkedBlockingDeque`). Queste classi consentono di realizzare un'operazione di *estrazione in testa* che implementa un meccanismo "intelligente" di attesa: infatti se non sono presenti elementi nella struttura dati, il thread chiamante entra in uno stato di pausa. Quando un nuovo dato è inserito, il sistema Android si fa carico di risvegliare il thead in pausa in modo tale che esso possa riprende l'esecuzione prelevando il nuovo campione inserito.

Viceversa, per garantire la seconda condizione si utilizza l'algoritmo 5.4. Segue una piccola disamina dell'algoritmo:

- La funzione deve essere invocata da T_{OC} subito dopo aver acquisito un campione dal magnetometro. In questo modo, il campione restituito ha un timestamp "ravvicinato" (temporalmente) al campione del magnetometro.
- A seconda della taglia della coda dell'accelerometro, ci sono diverse possibilità:
  - Se la coda è vuota, si deve attendere il prossimo campione (righe 10-13).
  - Se la coda contiene almeno un valore, si hanno due alternative:
    * Se in precedenza non è stata rilevata una taglia della coda pari a 0 (righe 16 e 17) allora è sufficiente prelevare il valore alla fine della coda. Infatti, è ragionevole estrarre il campione più recente.
    * Se in precedenza è stata rilevata una taglia della coda pari a 0 (righe 19 e 20) allora è sufficiente prelevare il valore in testa alla coda. Infatti, nel frattempo che T_{OC} è in attesa (riga 13), più campioni potrebbero essere inseriti nella coda per opera di T_A. Quindi è ragionevole estrarre il valore più vecchio, che si trova in testa alla coda.

Durante l'esecuzione della funzione si deve impedire che T_{SC} possa prelevare un campione al posto di T_{OC}. La variabile RL è il *lock*, cioè il meccanismo di sincronizzazione che limita l'accesso ad una risorsa condivisa, che assolve tale funzione. Viceversa, la variabile WL è il lock che regola la scrittura di c_A. Esso viene acquisito da T_{OC} per tutta la durata dell'esecuzione della funzione, ad eccezione del caso in cui T_{OC} entri nello stato d'attesa. In questo caso WL è rilasciato per consentire a T_A di inserire nuovi campioni.

Infine si noti che, adottando l'algoritmo 5.4, i campioni di c_A prelevati da T_{OC} sono poi rinseriti nella coda in modo tale da poter essere elaborati da T_{SC}. Questa esigenza nasce dal fatto che solo pochi campioni superano la soglia utilizzata dallo strides counter (come rappresentato in figura 5.6a).

5.3.2 Sincronizzazione tra Strides counter e Orientation checker

Lo *Strides counter* e l'*Orientation checker* necessitano di sincronizzazioni in diversi istanti:

- Se T_{SC} determina che $\sigma < \sigma_{static}$ allora T_{OC} deve iniziare il calcolo dell'azimuth. Tale sincronizzazione è ottenuta tramite un flag (chiamato *isStatic*), visibile globalmente ai due thread, che attiva (cioè seleziona l'opportuno ramo di un costrutto *if*) la porzione di codice di T_{OC} per il calcolo dell'azimuth.
- In analogia al caso precedente, se T_{SC} determina che $\sigma > \sigma_{static}$ allora impostando opportunamente il flag *isStatic* si arresta il calcolo dell'azimuth da parte del T_{OC}.

Algoritmo 5.4 Get campione acceleromentro.

Input: $coda\ accelerometro$
Output: $campione$, cioè il vettore accelerazione più recente.

```
1:  function GET CAMPIONE ACCELEROMENTRO
2:      RL è il read lock da acquisire per leggere dalla struttura dati (in testa o in coda)
3:      WL è il write lock da acquisire per scrivere nella struttura dati (in testa o in coda)
4:      in funzione ← TRUE
5:      prendi l'ultimo ← TRUE
6:      ACQUISISCI RL(coda accelerometro)              ▷ T_SC non può leggere nuovi campioni
7:      while in funzione do
8:          ACQUISISCI WL(coda accelerometro)          ▷ T_A non può inserire nuovi campioni
9:          n ← GET TAGLIA(coda accelerometro)
10:         if n = 0 then
11:             RILASCIA WL(coda accelerometro)        ▷ T_A può inserire nuovi campioni
12:             prendi l'ultimo ← FALSE
13:             WAIT(t)
14:         else
15:             if prendi l'ultimo = TRUE then
16:                 campione ←ESTRAI L'ULTIMO(coda accelerometro)
17:                 INSERISCI IN CODA(coda accelerometro, dato)
18:             else
19:                 campione ←ESTRAI IL PRIMO(coda accelerometro)
20:                 INSERISCI IN TESTA(coda accelerometro, dato)
21:             infunzione ← FALSE
22:     RILASCIA WL(coda accelerometro)
23:     RILASCIA RL(coda accelerometro)
24:     return campione
```

- Se T_{SC} rileva il passaggio dallo stato *static* allo stato *walk*, allora è necessario prelevare il valore della media dell'azimuth calcolato T_{OC}. Tuttavia la raccolta del dato deve avvenire dopo che T_{OC} abbia effettivamente terminato il calcolo della media: infatti i due thread, a causa del multi-threading del sistema operativo, potrebbero subire ritardi nel l'esecuzione del loro codice. A tale scopo si utilizza un lock (chiamato L_{mean}) che regola l'accesso ad una variabile condivisa su cui T_{OC} inserisce la media dell'azimuth:

  1. Quando il dispositivo entra nello stato di *static*, T_{OC} acquisisce il lock L_{mean} ed inizia il calcolo dell'azimuth.
  2. Quando il dispositivo passa nello stato di *walk*, T_{OC} inizializza la variabile condivisa con il valore della media dell'azimuth e rilascia lock L_{mean}.
  3. Prima di accedere alla variabile condivisa che mantiene la media dell'azimuth, T_{SC} acquisisce il lock L_{mean}. Questa operazione assicura che T_{SC} non rilevi un valore non aggiornato.

- Se T_{OC} determina che lo smartphone è uscito dalla zona di lavoro, allora è necessario disabilitare il rilevamento degli strides. Ciò avviene impostando un opportuno flag, visibile globalmente ai due thread, che esclude l'esecuzione della porzione di codice di T_{SC} per l'individuazione degli strides.

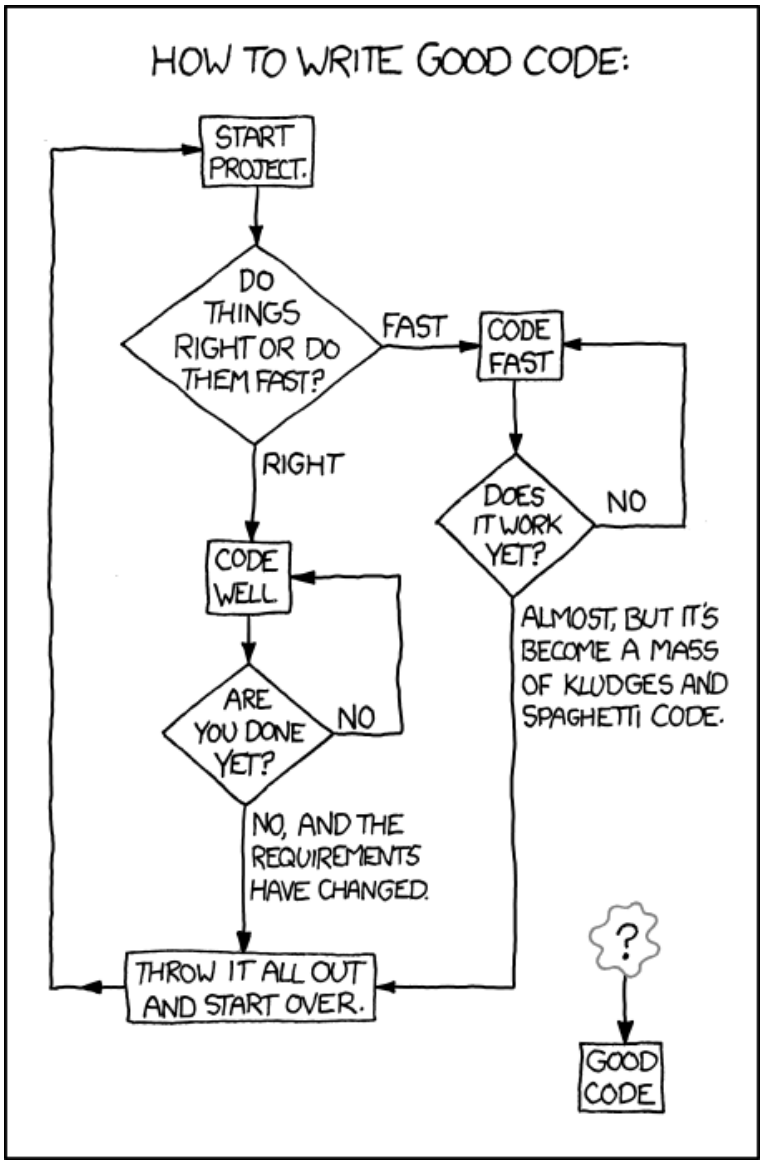
5.4 Test

Purtroppo non è possibile, con gli strumenti a disposizione, simulare i valori generati dal sensori per stimolare e valutare il comportamento del modulo *Contapassi*. Per tale ragione, la validazione

avviene insieme al modulo *Core*, il quale contente di eseguire l'applicazione direttamente nello smartphone.

Capitolo 6

Modulo Core



Fonte: *xkcd* website. Ultima visita: 3 dicembre 2014.

6.1 Definizione delle specifiche

Il modulo *Core* rappresenta il componente principale dell'applicazione, il quale gestisce (cioè alloca e libera) il ciclo di vita degli altri moduli. In prima analisi esso deve garantire l'esecuzione di due task distinti:

- *Creazione di una mappa*
- *Navigazione di una mappa*

Nello sviluppo di questa parte di progetto, si è scelto di implementare solamente il codice necessario a garantire il corretto funzionamento dell'applicazione quando essa è utilizzata per creare una nuova mappa. Non è stato progettato e implementato il task *Navigazione di una mappa*: infatti, affinché il risultato di tale task sia attendibile è necessario realizzare un efficace modulo *Contapassi*.

Il task *Creazione di una mappa* ha il compito di:

1. Ricevere i dati acquisiti dal modulo *UI*.
2. Creare le istanze necessarie affinché i moduli *Dati*, *Creazione mappa* e *Contapassi* siano operativi.
3. Creare un'istanza di una mappa.
4. Avviare il modulo *Contapassi* in modo tale che il rilevamento degli strides e il calcolo dell'orientazione del cammino siano attivi.
5. Garantire il funzionamento del programma anche se l'applicazione perde il focus o viene premuto il tasto "blocca schermo" dello smartphone.
6. Assicurare che l'informazione fornita dal modulo *Contapassi* sia elaborata dal modulo *Creazione mappa*.
7. Garantire che la mappa creata sia salvata sul database tramite il modulo *Dati*.
8. Arrestare, su richiesta, i threads del modulo *Contapassi* e chiudere correttamente le risorse allocate dal *Modulo dati*.

6.2 Progettazione software del modulo

6.2.1 Estensione della classe `Service`

Si è scelto di concentrare lo sviluppo del task *Creazione di una mappa* attorno ad una sottoclasse di `Service` di Android.

In primo luogo si è scelto di implementare un `Service` "autonomo". Con riferimento alla documentazione Android[1], un `Service` "autonomo" è un generico servizio che è avviato ed arrestato da un componente Android (`Activity`, `Service`, `BroadcastReceiver`). Una volta avviato, il `Service` rimane attivo fino al momento in cui termina autonomamente la sua funzione o viene esplicitamente arrestato. L'esistenza del `Service` è quindi disaccoppiata dall'esistenza del componente che ha richiesto il servizio stesso. Chiaramente, questa forma del servizio si adatta allo scopo dell'applicazione: l'utente avvia il programma, mette in tasca lo smartphone ed arresta l'applicazione solo quando è arrivato a destinazione.

In aggiunta, si è scelto di aggiungere al `Service` implementato anche la forma "bound". Sempre in riferimento alla documentazione ufficiale, un `Service` è nella forma "bound" quando esiste un legame tra esso e il componente che ne ha fatto richiesta. Generalmente, ed anche nel contesto di questa applicazione, il legame si traduce nella possibilità di ottenere una referenza dell'istanza

[1] Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.

del `Service` d'interesse e, tramite essa, invocare i metodi del `Service`. Ad esempio, nel caso i cui si debba realizzare un riproduttore musicale, un'istanza di `Service` ha il compito di riprodurre la traccia audio (anche quando l'applicazione è in background) mentre un'`Activity` che si lega al `Service` ha il compito di cambiare la traccia.

Nell'applicazione sviluppata, poiché di norma viene premuto il tasto "blocca schermo" dello smartphone prima del suo inserimento in tasca, il componente che ha avviato il `Service` è distrutto da Android. Alla riattivazione del terminale (disattivazione del "blocca schermo"), un nuovo componente viene creato. Legandosi al `Service` è possibile quindi recuperare i risultati d'interesse (o eventuali dati di debug).

6.2.2 Design del `Service`

L'obiettivo di questa sezione consiste nel descrivere come sono progettate le funzioni indicate nelle specifiche 6.1. In primo luogo si descrivere quando tali funzioni devono essere realizzate nel contesto del ciclo di vita del `Service`:

1. *Start lifetime*: costituisce la fase iniziale in cui il `Service` è inizializzato. Qui sono eseguite le specifiche da 1 a 5.
2. *Running lifetime*: costituisce la fase in cui il `Service` è attivo. Qui deve essere realizzata la specifica 6.
3. *Stop lifetime*: costituisce la fase di arresto del `Service`. Deve assicurare la realizzazione delle specifiche 7 e 8.

La fase d'inizializzazione del `Service` comprende una serie di istruzioni per adempire alle sue specifiche. Più interessante è la realizzazione delle fase di *running* e *arresto* del `Service`. Il problema principale di queste fasi consiste nel progettare come deve avvenire la comunicazione (e la sincronizzazione) tra i moduli del programma. Si è scelto di realizzare un meccanismo simile alla soluzione del problema del *produttore-consumatore*. Nel dettaglio:

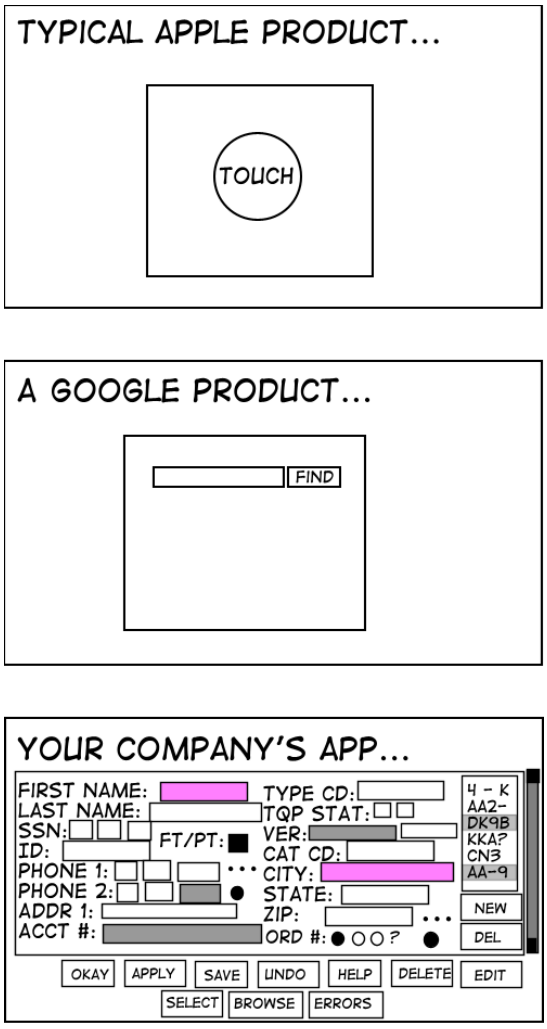
- L'istanza di `Service` alloca:
  - Una coda che contiene messaggi, ognuno dei quali mantiene un'istruzione da eseguire. Esistono almeno due tipi di istruzioni:
    - *Aggiorna mappa($n_{strides}$, angolo)*.
    - *Salva mappa*.
  - Un thread *consumatore* che ha il compito di svolgere le istruzioni contenute in ogni messaggio della coda.
- Lo *Strides counter* svolge il ruolo di *produttore*, inserendo nella coda il messaggio *Aggiorna mappa($n_{strides}$, angolo)* quando un nuovo segmento del cammino è terminato.
- Prima che il `Service` termini il suo ciclo di vita, esso inserisce nella coda il messaggio *Salva mappa* in modo tale che la mappa creata diventi persistente in memoria.

Riassumendo:

- La comunicazione tra i moduli *Contapassi* e *Core* avviene tramite l'inserimento di un messaggio nell'apposita coda. Affinché ciò sia possibile è stato scelto di legare ("bound") il thread T_{SC} al `Service` e, tramite un opportuno metodo, ottenere una referenza della coda dei messaggi.
- La comunicazione tra i moduli *Core* e *Creazione mappa* avviene grazie alle istanze di `Selettore di celle` e `Aggiornamento celle` che il `Service` ha allocato.
- La comunicazione tra i moduli *Core* e *Dati* avviene grazie alle istanze di `DAO factory` e `DAO` che il `Service` ha allocato.

Capitolo 7

Modulo UI



Fonte: *Stuffthathappens* website (sito principale offline). Ultima visita: 3 dicembre 2014.

7.1 Definizione delle specifiche

7.1.1 Acoustic User Interface

La release finale dell'applicazione utilizza un'interfaccia utente che, tramite opportuni messaggi acustici, fornisce indicazioni sulla navigazione che l'utente deve eseguire. Una possibile progettazione di questa Acoustic User Interface, prevede di segnalare all'utente l'indicazione dello stride appena esso viene individuato (quindi in tempo reale). Questa strategia ha il vantaggio di rassicurare la navigazione dell'utente in quanto esso può direttamente confrontare se il rilevamento dello stride avviene in modo corretto e quindi avere conferma che l'applicazione sta funzionando correttamente. Ad esempio, si immagini l'utilizzo in un centro commerciale in cui la presenza di ostacoli (panchine, vasi, scale) è frequente: se l'applicazione non fornisce un certo "feedback" immediato all'utente, esso assumerà un atteggiamento più prudente e cauto, in quanto aumenta l'incertezza percepita sul cammino che l'applicazione sta indicando.

In aggiunta, per agevolare l'utilizzo dell'applicazione anche in ambienti affollati da altre persone, si potrebbe prevedere un'integrazione di una Vibration User Interface che fornisca all'utente le indicazioni necessarie tramite sequenze di vibrazione dello smartphone di durata diversa. In questo modo, si riduce l'invasività del prodotto in quanto il suo utilizzo risulta praticamente indistinguibile dalle normali funzioni usate comunemente con uno smartphone.

7.1.2 Degub User Interface

Nella porzione di progetto realizzato, si è scelto di sviluppare un'interfaccia grafica che sia di supporto al test dell'applicazione. Essa pertanto non è adatta e distribuita nella versione finale del prodotto.

A differenza degli altri capitoli, la seguente documentazione descrive direttamente i risultati del modulo, cioè l'interfaccia del prodotto. In questo modo è possibile costruire una piccola guida su come utilizzare l'applicazione tramite questa interfaccia.

7.1.2.1 Create map task

All'avvio dell'applicazione (figura 7.1a), selezionando *Creation map task* compare la schermata principale da cui poter modificare tutte i parametri trattati negli algoritmi esposti nei precedenti capitoli (figura 7.1b):

- Alla voce "*Sensor delay*" è possibile modificare indipendentemente la frequenza di campionamento "suggerita" per l'accelerometro e il magnetometro (figura 7.1c).
- Alla voce "*Strides counter: Lifemap*" si impostano tutti i parametri relativi allo strides counter in accordo con la tabella 7.1. In aggiunta:

Voce menù	**Parametro corrispondente**
Size of window	n
Min size of window	n_{min}
Max size of window	n_{max}
Operating k of std. dev.	k
Transitory k of std. dev.	$k_{transitorio}$
Threshold on std. dev. to switch k	σ_k
Min threshold for acceleration	$soglia_{min}$
Static user threshold of std. dev.	σ_{static}
Min timeout strides in ms.	$soglia_{\Delta T}$

Tabella 7.1: Corrispondenza tra le voci del menù del task *Creazione mappa* (figure 7.1d e 7.1e) e i parametri dell'algoritmo dello strides counter.

  - La voce "*Is window size fixed?*" abilita o disabilta l'estensione della finestra. Se la dimensione è fissa, "*Size of window*" indica la durata della finestra; viceversa se la dimensione è variabile "*Min size of window*" e "*Max size of window*" corrispondono ai parametri n_{min} e n_{max}.
  - La voce "*User heigh in m.*" corrisponde all'altezza dell'utente da cui ricavare la lunghezza dello strides (tabella 5.1).

- Alla voce "*Compass: Lifemap*" si impostano tutti i parametri relativi all'*Orientation checker* in accordo con la tabella 7.2.

- Alla voce ''*Map options*" è possibile scegliere di (figura 7.1g):

  - Creare una nuova mappa. In questo caso, attraverso la voce "*Map parameters*" (figure 7.1h e 7.1i) è possibile:
    * Impostare il numero di righe, di colonne e il passo di discretizzazione della mappa.
    * Attivare la voce "*Automatic generation of EPM*": questa funziona imposta automaticamente come *EPM* la prima cella che contiene lo *start* del primo segmento del cammino. Poiché è obbligatorio la presenza di almeno un *EPM* affinché la mappa possa essere salvata nel database, è indispensabile attivare questa opzione.
    * Impostare manualmente le coordinate GPS dell'*EPM* (voce "*Set GPS coordinates*"). Attualmente, tali coordinate non sono utilizzate ad eccezione dei programmi di test.
    * Attivare la voce "*Automatic generation of IP*": in analogia a quanto avviene per *EPM*, questa funzione imposta automaticamente l'ultima cella che contiene l'*end* dell'ultimo segmento del cammino.
    * Attivare la voce "*Enable spread of weight*", la quale distribuisce il peso di una porzione del cammino non solo nelle celle della griglia attraversate dal segmento ma anche nelle celle adiacenti (algoritmo 4.3).
  - Caricare una mappa dal database. In questo caso è possibile (figura 7.1j):
    * Caricare una nuova mappa, tramite la voce "*Load map from db*". All'apertura della nuova schermata (7.1k), che contiene tutte le mappe presenti nel database, è sufficiente premere sulla mappa da caricare.
    * Attivando l'opzione "*Overwrite the loaded map*", la mappa caricata è eliminata dopo che la mappa aggiornata è stata salvata nel database. Per evitare perdite accidentali di dati, ogni qualvolta l'`Activity`, è distrutta (più precisamente è eseguito il metodo `onDestroy`) tale opzione è disabilitata.
  - Salvare nel database solamente le coppie *($n_{strides}$, azimuth)* nell'apposita tabella *Strides data* (sezione 3.5.1).

- Alla voce "*Debug options*" è possibile (figura 7.1m):

  - Abilitare un segnale acustico riprodotto al rilevamento di ogni stride.
  - Impostare un ritardo tra la pressione del pulsante "*Start the create map task*" (figura 7.1b) e l'effettivo inizio dell'esecuzione dei thread T_{SC} e T_{OC}.

Voce menù	**Parametro corrispondente**
Size of window for pitch	n_{pitch}
Size of window for azimuth	$n_{azimuth}$
Min pitch threshold of work zone in degrees	$pitch_{min}$
Max pitch threshold of work zone in degrees	$pitch_{max}$

Tabella 7.2: Corrispondenza tra le voci del menù del task *Creazione mappa* (figura 7.1d e 7.1e) e i parametri dell'algoritmo dello strides counter.

- Alla pressione del pulsante "*See results*" è possibile visualizzare i risultati dell'elaborazione dello *Strides counter* e dell'*Orientation checker*. Nella metà superiore della schermata (figura 7.1n) sono indicate le informazioni relative agli strides individuati. Di seguito, si descrive il significato dei campi seguendo l'ordine di apparizione:
  - Indice dello stride individuato.
  - Timestamp del campione che ha individuato lo stride.
  - $\|A\|$ del campione che ha individuato lo stride.
  - Soglia con cui $\|A\|$ si è confrontato.

  Nella parte inferiore della schermata è possibile visualizzare i segmenti del cammino individuati dal modulo contapassi, cioè l'insieme delle coppie *($n_{strides}$, azimuth)*.

Per eseguire l'applicazione è necessario premere sul pulsante *Start the creation map task*.

L'applicazione salva su file l'ultimo stato di tutte le impostazioni quindi non è necessario impostare nuovamente i parametri per ogni diversa esecuzione.

7.1.2.2 Navigation map task

Con riferimento alla figura 7.1a, alla pressione della voce "*Navigation map task*" compare un messaggio che indica che la funzione non è disponibile.

7.1.2.3 Extra

La voce "*Extra*" della figura 7.1a consente di:

- Esportare la tabella *MP Raw* nel formato `csv` elaborabile da altri strumenti di lavoro (voce "*Export MatrixWeightRaw table*" della figura 7.2a). Infatti, essendo la matrice di una *MMP* salvata nel database come tipo di dato binario `BLOB` non è possibile esportare i record di tale tabella con strumenti esterni poiché la corrispondenza tra array di bit e celle della matrice non può essere ricostruita (essendo non nota la serializzazione adottata). Pertanto, il processo di esportazione deve essere realizzato internamente all'applicazione.

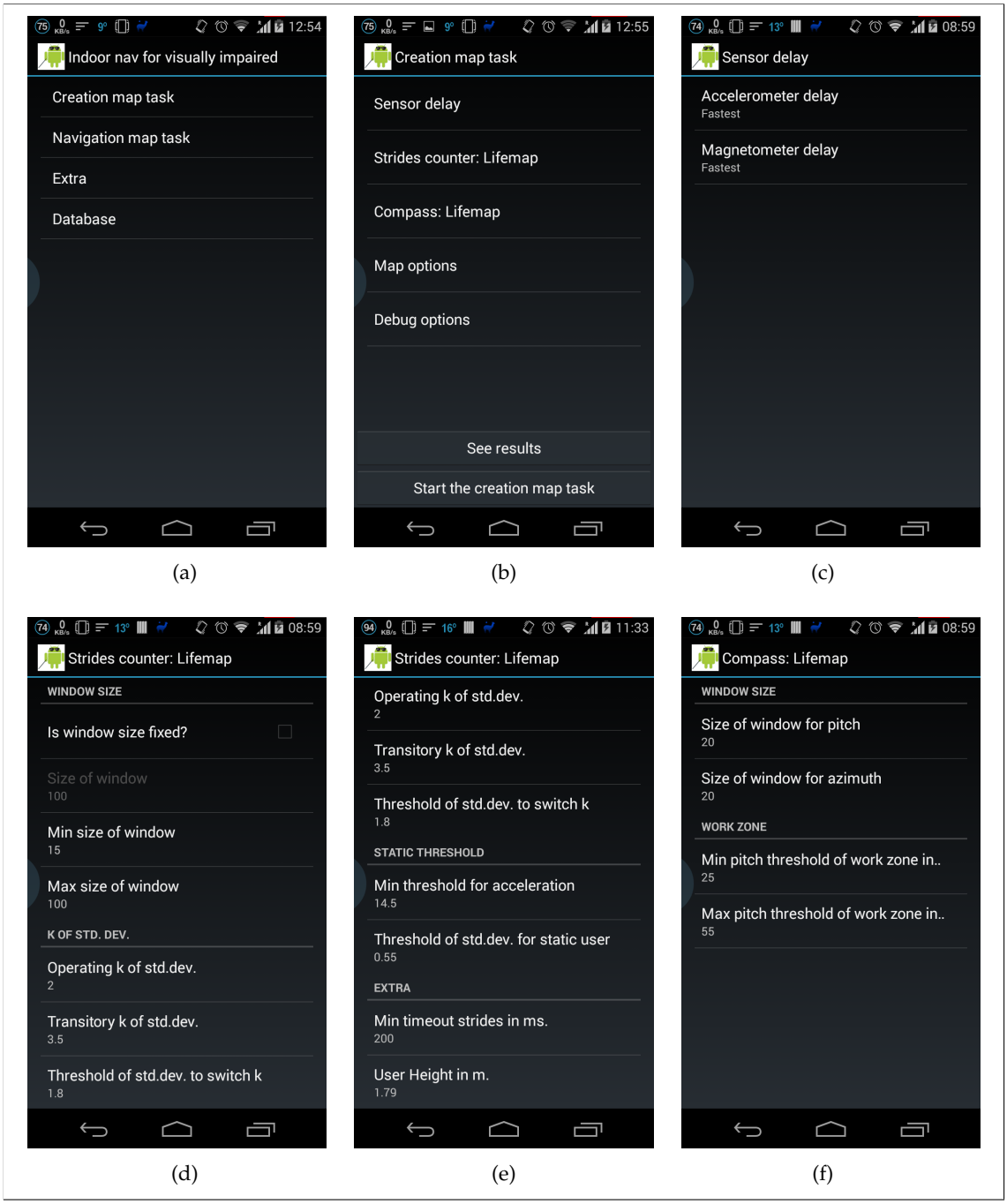
  Sono disponibili due formati d'esposizione (figura 7.2b):

  - Premendo sulla voce "*Export to CSV*", ogni record della tabella è convertito in un formato testuale composto dalla successione dei valori dei suoi attributi, separati da un carattere speciale. Come indicato in figura 7.2c è possibile:
    - Impostare il carattere separatore.
    - Avvertire l'applicazione di ripetere l'intestazione (cioè lo schema della relazione) per ogni nuova mappa rilevata.
  - Premendo sulla voce "*Export as pivot table*", tutti i record relativi ad una tabella sono convertiti nel formato `csv` in accordo con la formattazione di "*Export to CSV*'. Inoltre:
    - È aggiunto un nuovo record finale contente i valori degli indici di colonna della griglia.
    - È aggiunto all'inizio di ogni record della tabella, il valore dell'indice di riga corrente del record.

    Un esempio di tabella pivot (opportunamente acquisito con *Microsoft Excel*) è rappresentato in figura 9.1.

  Per entrambe le funzioni, l'avvio della conversione avviene premendo il pulsante "*Start*".

- Estrarre i record della tabella *Strides data* e generare una o più mappe con tali informazioni (voce "*Aggregate data with same tag*" della figura 7.2a). La porzione di codice che si occupa di questa funzione deve essere scritta ad hoc. Per il momento, l'unica opzione disponibile è la voce "*Insert tag for a pairs*" (figura 7.2e): la stringa s inserita in questo campo è utilizzata per estrarre dalla tabella *Strides data* tutti i record con il valore del campo *Tag* pari ad s e generare una o più mappe. Un esempio di come usare tale campo è indicato nella sezione 9.2.

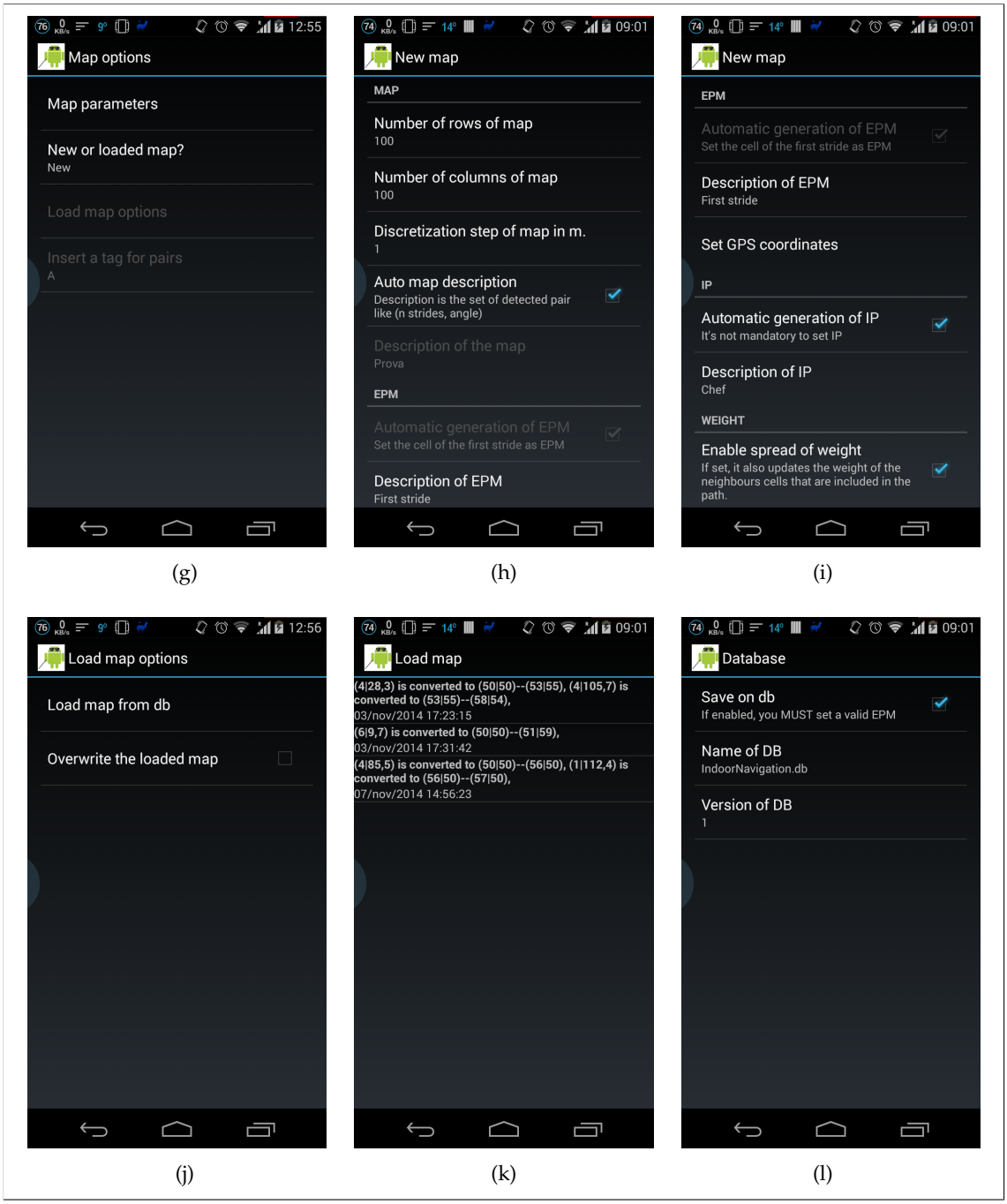


7.1.2.4 Database

Alla voce "*Database*" è possibile settare le impostazioni relative al database (figura 7.1l). Non è possibile scegliere il percorso di destinazione del database; di default è posizionato nella cartella `/IndoorNavigation` della memoria esterna del dispositivo.

7.2 Progettazione software del modulo

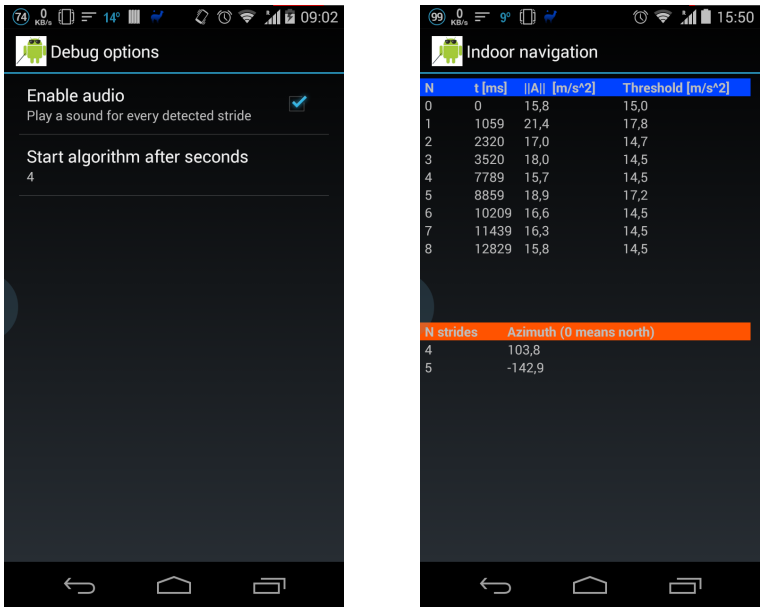
Si è scelto di realizzare la *Debug User Interface* facendo uso di tre `Activity`:



- Un `Activity` principale che estende la classe `PreferenceActivity`. La funzione di questa interfaccia è quello di consentire l'inserimento di tutti i parametri dell'applicazione.
- Un `Activity` per la visualizzazione dei risultati (figura 7.1n).
- Un `Activity` per la scelta della mappa da caricare (figura 7.1k).

L'utilizzo di un'estensione della classe `PreferenceActivity` consente di:

- Definire un'interfaccia utente coerente con l'esperienza d'uso che l'utente ha quando naviga all'interno del menù delle impostazioni del sistema Android.



(m) (n)

Figura 7.1: Screenshot del task *Creazione mappa*.

- Agevolare lo sviluppo dell'interfaccia poiché è sufficiente definire quali elementi devono essere visualizzati nell'interfaccia e non occuparsi della loro rappresentazione (layout). All'atto pratico, l'insieme delle voci che costituiscono l'interfaccia sono realizzati (scritti) in un file `xml`, opportunamente strutturate in forma gerarchica (cioè il main come radice e i sottomenù come foglie).

Gran parte della navigazione utente all'interno dell'applicazione avviene nell'istanza di `PreferenceActivity`. Il passaggio alle `Activity` per la visualizzazione dei risultati dello *Strides counter* o per la visualizzazione delle mappe presenti nel database avviene mediante il meccanismo degli `Intent`. Ad esempio, alla pressione del pulsante "*See results*" si ha che:

1. Un'istanza di `Intent` per l' `Activity` "*Visualizza risultati*" è allocata.
2. L'`Intent` creato viene inizializzato con i risultati ottenuti dal task *Creazione mappa*.
3. Viene "lanciata" una nuova `Activity` fornendo come parametro l'`Intent` appena creato. Il sistema operativo si fa carico di creare un'istanza dell'`Activity` indicata fornendo come parametro d'ingresso l'`Intent` che ha fatto richiesta.
4. Una volta avviata, la nuova `Activity` estrae i risultati dal task *Creazione mappa* dall'`Intent` ricevuto come parametro e li visualizza a schermo.

Considerazione analoghe sono valide anche per l'`Activity` che visualizza le mappe salvate nel database. In questo caso però risulta conveniente estendere la classe `ListActivity`, la quale consente agevolmente di visualizzare una lista di record la cui backend è costituita da un cursore ottenuto come risultato di un interrogazione su di un database. In altra parole, al suo avvio la classe provvede a:

1. Creare un'istanza di una `Factory` relativa a *SQLite* (sezione 3.3.3.1).
2. Ottenere un'istanza del `DAO` per la tabella *MP*.
3. Estrarre tutti i record della tabella *MP* tramite il `DAO`.
4. Visualizzare tutti i record in una lista.

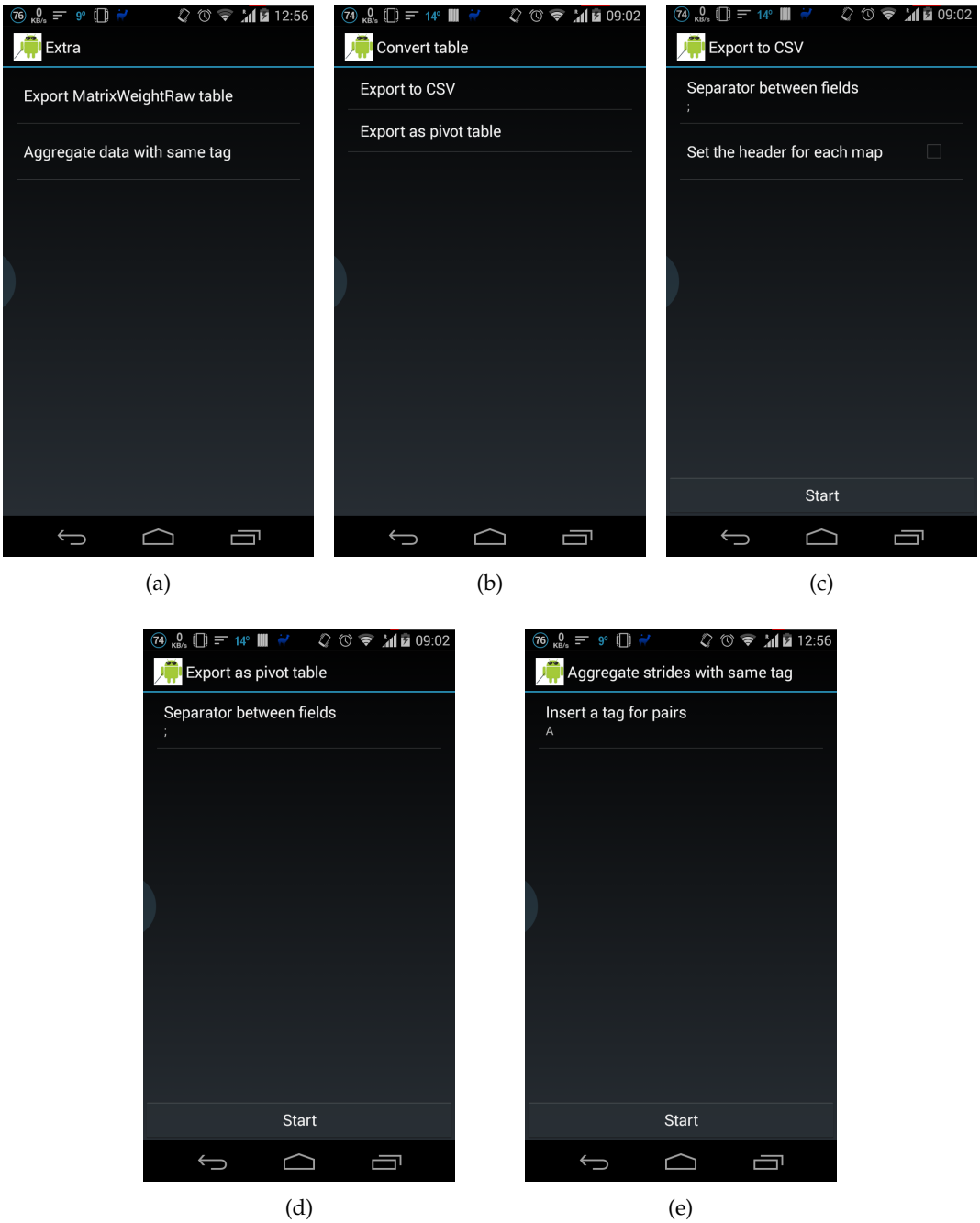


Figura 7.2: Screenshot delle funzioni *Extra* dell'applicazione.

Capitolo 8

Modulo Navigazione

Fonte: *toonclips.com* website. Ultima visita: 3 dicembre 2014.

8.1 Possibile progettazione

Un prerequisito necessario per poter realizzare un'affidabile modulo *Navigazione* consiste nello sviluppo di un modulo *Contapassi* che fornisca risultati attendibili. Infatti, in assenza di tale condizione, non sarebbe possibile stabilire con un'adeguata precisione la posizione dell'utente all'interno della mappa. Per tale motivo, lo sviluppo del modulo *Navigazione* non viene trattata all'interno di questa tesi. Tuttavia è opportuno descrivere sommariamente una prima analisi di una possibile progettazione, in modo tale da rendere più esplicito il significato del campo *peso* di ciascun elemento di una matrice di una mappa *MMP*.

Gli elementi di una *MP* tengono traccia della frequenza con cui l'utente ha attraversato una determinata cella. Un possibile algoritmo di navigazione potrebbe prendere spunto dalla seguente idea: se l'utente percorre più volte lo stesso percorso da uno punto *start* ad un punto *end* è ragionevole ipotizzare che egli attraversi (all'incirca) le stesse celle della mappa. Supponendo che tale ipotesi sia vera, gli elementi della *MP* che corrispondo a tali celle della griglia assumerebbero un *peso* più elevato rispetto agli altri elementi della matrice.

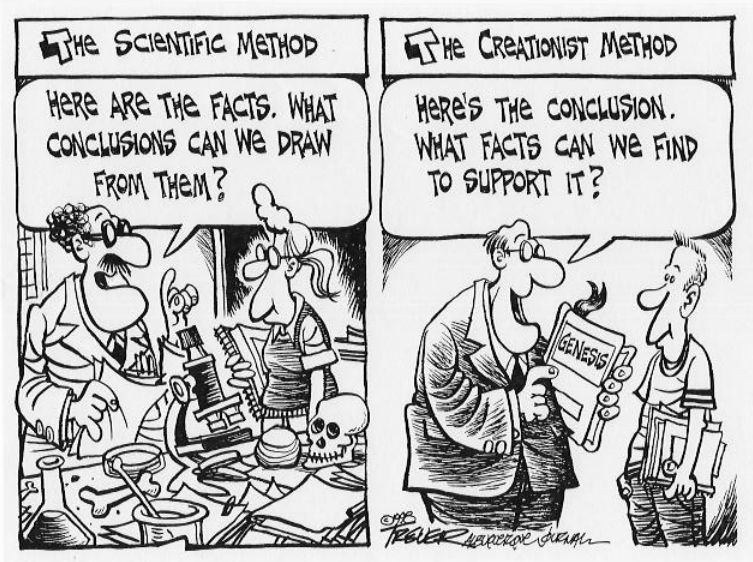
Sfruttando tale idea è possibile realizzare un semplice algoritmo di navigazione. Data una *MMP*, acquisita con il modulo *Creazione mappa*, si supponga che esista un *EPM* e un *IP*. Poiché la mappa costituisce una "registrazione" di un percorso precedente fatto, è noto a priori l'esistenza di almeno una soluzione per l'algoritmo di navigazione, cioè di un percorso che unisce l'*EPM* all'*IP*. Pertanto, il problema può essere classificato come un problema di ricerca $P = (G, S, s_0, A, s_g)$ in cui:

- $G = (N, E)$ è un grafo tale che:
  - N è l'insieme dei nodi. Ogni cella della griglia visitata dell'utente costituisce un nodo.
  - E è l'insieme degli archi (non orientati). Per ogni cella c esistono otto archi che uniscono c con ciascuna cella vicina secondo il vicinato di Moore (figura 4.3).
- S è l'insieme degli stati, cioè di tutte le possibili celle "visitabili" dall'algoritmo di navigazione. Poiché la mappa costituisce una registrazione di un precedente percorso, S contiene tutte le celle della griglia visitate dall'utente.
- s_0 è lo stato iniziale, cioè l'*EPM*.
- A è l'insieme delle azioni possibili. Nel contesto dell'applicazione, A è costituito dalla transazione da uno cella c ad una cella appartenente al vicinato di Moore di c.
- s_g è il goal, cioè *IP*.

Dato il problema P, l'obiettivo è trovare un percorso da s_0 a s_g. A tale scopo è possibile attingere tra diversi algoritmi proposti in lettura, di cui si trova un'ampia panoramica in [55]. Tali algoritmi navigano all'interno dello spazio degli stati (che nel contesto dell'applicazione costituiscono le celle attraversate dall'utente durante la creazione della mappa), in accordo a specifiche regole, al fine di determinare una soluzione. Una scelta ragionevole, consiste nell'adottare un algoritmo di ricerca "*informata*" (*best first search*): l'idea consiste nell'utilizzare l'informazione legata alla frequenza con cui l'utente ha attraversato le varie celle per guidare la ricerca all'interno dello spazio degli stati S, al fine di trovare una soluzione in modo più efficiente. Ad esempio, un'idea plausibile consiste nel guidare la ricerca valutando una funzione g, la quale favorisce l'espansione verso i nodi (dello spazio degli stati S) a cui corrispondo celle con peso più elevato rispetto alle altre.

Capitolo 9

Risultati sperimentali



Fonte: *abqjournal.com* website. Ultima visita: 3 dicembre 2014.

9.1 Procedura di taratura

A causa dei numerosi fattori che influiscono l'andamento delle tracce dell'accelerometro e del magnetometro acquisite dai sensori dello smartphone (ad es. la frequenza del cammino, la tipologia di calzature, di pavimentazione, di abbigliamento…), l'applicazione necessita di essere adeguatamente tarata in modo da poter funzionare correttamente. Di default, il software prevede già un'inizializzazione ragionevole di tutti i suoi parametri. Tuttavia, essa non è sufficiente a garantire il corretto funzionamento del software.

Nella versione finale del prodotto, l'operazione di taratura dovrebbe essere il più possibile automatica poiché, molto probabilmente, l'utilizzatore non possiede il bagaglio di conoscenze tecniche per poterla eseguire correttamente. Nel contesto della *Debug User Interface* non vi è necessità di una procedura automatica di taratura poiché l'utilizzatore di tale interfaccia corrisponde a chi effettivamente sta sviluppando l'applicazione.

Di seguito si propone un'insieme di Frequently Asked Questions (FAQ) che possono aiutare nella procedura di taratura:

1. **Cosa devo fare se è rilevato uno stride durante il movimento necessario ad inserire lo smartphone in tasca?**

   È necessario aumentare il ritardo alla voce "*Start algorithm after seconds*" (figura 7.1m), cioè il tempo che intercorre tra l'esecuzione dei threads T_{SC} e T_{OC} e la pressione del tasto "*Start the creation map task*".

2. **Cosa devo fare se è rilevato uno stride durante il movimento necessario ad estrarre lo smartphone dalla tasca?**

   È necessario aumentare il valore alla voce "*Min threshold for acceleration*" (figura 7.1e). Non si dovrebbe incrementare troppo tale valore poiché c'è il rischio di non rilevare correttamente alcuni strides. Inoltre, appena lo smartphone è completamente estratto dalla tasca, si consiglia di posizionarlo parallelo al terreno, cioè con un angolo di *pitch* che vale circa 90 gradi (come indicato nella sezione 5.2.4.5): in questo modo l'*Orientation checher* disabilita la funzione di rilevamento degli strides. È possibile modificare tale soglia alla voce "*Max pitch threshold of work zone in degrees*" (figura 7.1f).

3. **Cosa devo fare se sono rilevati anche gli steps?**

   Probabilmente è necessario aumentare la durata della finestra mobile, cioè le voci "*Size of window*" o "*Max size of window*" (figura 7.1d). Inoltre potrebbe essere necessario aumentare il valore $k_{operativo}$, cioè la voce "*Operating k of std. dev.*" (figura 7.1d).

4. **Perché, dopo la rilevazione corretta di alcuni strides, l'applicazione non rileva più i successivi?**

   Probabilmente è necessario diminuire la taglia della finestra mobile, cioè le voci "*Size of window*" o "*Min size of window*" (figura 7.1d).

5. **Cosa devo fare se solo alcuni strides sono rilevati?**

   Le cause possono essere molteplici. Si provi a:

   (a) Diminuire il valore $k_{operativo}$, cioè la voce "*Operating k of std. dev.*" (figura 7.1d).

   (b) Diminuire la soglia $k_{transitorio}$, cioè la voce "*Threshold of std.dev. to switch k*" (figura 7.1d).

   (c) Aumentare la durata minima della finestra mobile, cioè le voci "*Size of window*" o "*Max size of window*" (figura 7.1d).

   (d) Diminuire il valore $soglia_{min}$ alla voce "*Min threshold for acceleration*" (figura 7.1e).

6. **Cosa devo fare se anche il primo step è rilevato (anche se non è uno stride)?**

   È necessario utilizzare una finestra mobile estensibile di durata ridotta in modo che il sistema sia più reattivo. È sufficiente modificare la "*Min size of window*" (figura 7.1d). Inoltre potrebbe essere necessario aumentare il valore $k_{transitorio}$, cioè la voce "*Transitory k of std. dev.*" (figura 7.1d).

7. **Cosa devo fare se non è rilevato nessuno strides?**

   Le cause possono essere molteplici. Si provi a:

   (a) Diminuire il valore $k_{operativo}$, cioè la voce "*Operating k of std. dev.*" (figura 7.1d).

   (b) Aumentare la dimensione minima della finestra mobile, cioè le voci "*Size of window*" o "*Max size of window*" (figura 7.1d).

   (c) Diminuire il valore $soglia_{min}$ alla voce "*Min threshold for acceleration*" (figura 7.1e).

8. **Cosa devo fare se, quando mi fermo e ruoto, l'applicazione segnala l'avvenuta identificazione di strides?**

   È necessario alzare la soglia della deviazione standard che stabilisce quando l'utente è fermo. È sufficiente modificare la "*Threshold of std. dev. for static user*" (figura 7.1e).

9. **Perché sono rilevati due strides quando in realtà ne ho compiuto uno solo?**

   Le cause possono essere molteplici. Si provi quindi a:

   (a) Aumentare il valore $k_{operativo}$, cioè la voce "*Operating k of std. dev.*" (figura 7.1d).

   (b) Aumentare il timeout tra due strides, cioè la voce "*Min timeout strides in ms.*" (figura 7.1e).

   (c) Aumentare la durata minima della finestra mobile, cioè le voci "*Size of window*" o "*Max size of window*" (figura 7.1d).

10. **Cosa devo fare se l'applicazione sotto/sovra stima la distanza realmente percorsa?**

    Purtroppo l'applicazione utilizza un modello approssimativo della distanza percorsa in funzione della lunghezza dello stride. Si può selezionare un'altezza diversa dell'utente (parametro "*User heigh in m.*", figura 7.1e) per diminuire tale errore.

11. **Ho effettuato più volte lo stesso percorso, arrestando e avviando di volta in volta l'applicazione. Cosa devo fare se il valore dell'azimuth non è uguale tra i vari cammini?**

    Le cause possono essere molteplici. Si provi quindi a:

    (a) Inserire lo smartphone in tasca sempre nella stessa posizione. Per via della superficie curva della coscia, molto spesso lo smartphone è tangente alla curva in posizione diversa per ogni prova. Questo implica che l'angolo d'uscita della porzione del cammino è diverso per ogni sessione.

    (b) Aumentare la durata della finestra per il calcolo dell'azimuth, cioè la voce "*Size of window for azimuth.*" (figura 7.1f). Tuttavia, tale incremento dovrebbe essere seguito da un'attesa maggiore da parte dell'utente tra una rotazione e l'inizio della successiva porzione di cammino.

12. **Ho effettuato 3 passi. Cosa devo fare se l'applicazione spezza il cammino in due porzioni? Entrambe hanno lo stesso angolo d'uscita ed una è costituita da due passi e l'altra da uno.**

    Questo comportamento è dovuto al fatto che l'angolo di *pitch* dello smartphone assume un valore tale da superare la soglia d'uscita dalla zona di lavoro (voce "*Max pitch threshold of work zone in degrees*", figura 7.1f) mentre l'utente sta camminando. In questa situazione, il dispositivo passa dallo stato di *walk* ad uno stato di *not working*. Successivamente, quando l'angolo di *pitch* rientra nella zona di lavoro (voce "*Min pitch threshold of work zone in degrees*", figura 7.1f) il dispositivo non rientra nello stato di *static* perché effettivamente l'utente si sta muovendo. In questa situazione non c'è modo di calcolare un angolo iniziale per questa porzione di cammino.

    Si è scelto quindi di assegnare a questa porzione di cammino l'ultimo angolo noto, cioè quello calcolato per la porzione precedente. Questa soluzione è adeguata nel caso in cui il dispositivo esca dalla zona di lavoro mentre l'utente cammina poiché genera soluzione comunque corrette.

    Tuttavia, se il dispositivo esce dallo zona di lavoro durante una rotazione e rientra in prossimità dell'istante in cui l'utente inizia a camminare, allora l'angolo d'uscita potrebbe non essere effettivamente stato calcolato con campioni coerenti all'orientazione dell'utente e quindi fornire un risultato errato. Per il momento, non è stato previsto nessun comportamento in risposta a questa situazione. Una prima strategia potrebbe consistere nell'avvisare l'utente (tramite un messaggio acustico) che l'applicazione non ha avuto modo di calcolare un angolo di azimuth stabile seguito da una transazione di stato da *static* a *walk*. In questo modo, l'utente è al corrente che l'applicazione non sta funzionando correttamente.

    Per evitare tale fenomeno è possibile:

    (a) Aumentare la durata della finestra per il calcolo del pitch (voce "*Size of window pitch*", figura 7.1f).

(b) Aumentare la soglia massima per l'uscita dalla zona di lavoro (voce "*Max pitch threshold of work zone in degrees*", figura 7.1f).

13. **Cosa devo fare se ricevo una `ArrayIndexOutOfBoundsException`?**

    L'applicazione imposta l'EPM della mappa alla cella $(\frac{n\ rows}{2}, \frac{n\ cols}{2})$. Se durante la creazione della mappa, il percorso dell'utente supera le dimensioni della griglia associata alla mappa allora viene lanciata tale eccezione. Incrementando la dimensione della mappa è possibile risolvere tale problema (voci "*Number of rows of map*" e "*Number of cols of map*", figura 7.1h)

14. **Cosa devo fare se ricevo una `SQLiteException`?**

    Molto probabilmente il salvataggio della mappa corrente viola i vincoli d'integrità referenziale (ad es. si sta cercando di salvare una mappa senza un EPM).

9.2 Modalità d'esecuzione dei test

Per valutare il comportamento dell'applicazione, si è scelto di eseguire due test, che si differenziano per la durata del cammino, cioè del numero di strides e dai cambi di orientazione effettuati.

9.2.1 Test breve: cammino rettilineo

Le caratteristiche del test sono le seguenti:

- Il cammino effettuato è composto da una sequenza rettilinea di 10 strides.
- L'intero percorso è stato ripetuto per 12 volte, camminando sempre dallo stesso punto iniziale allo stesso punto finale.
- L'azimuth iniziale reale di ogni cammino è all'incirca il medesimo.
- La gamba che effettua il primo passo è quella in cui non è presente lo smartphone nella rispettiva tasca.
- I parametri d'esecuzione del test sono indicati nelle tabelle 9.1, 9.2 e 9.3.
- Si è scelto di disaccoppiare l'operazione di acquisizione delle coppie *($n_{strides}$, azimuth)* dall'effettiva creazione della mappa. Nel dettaglio di procede con due fasi distinte:

  1. Fase 1: acquisizione.
     (a) Impostare l'applicazione in modo tale da acquisire e salvare le coppie *($n_{strides}$, azimuth)* di ciascun cammino. Per far ciò è necessario accedere al menù "*Map options*" (figura 7.1g della sezione 7.1.2.1), premere sulla voce "*New or loaded map?*" e selezionare "*Save only pairs like (n strides, azimuth)*".
     (b) Alla voce "*Insert a tag for pairs*", impostare uno stesso tag *t* per tutte le 12 acquisizioni (figura 7.1g). Questo passaggio è fondamentale poiché indica che i 12 insiemi di coppie *($n_{strides}$, azimuth)* acquisiti devono essere utilizzati per comporre un'unica mappa.
  2. Fase 2: elaborazione.
     (a) Nel menù "*Extra*", selezionare la voce "*Aggregate data with same tag*" (figura 7.2a).
     (b) Alla voce "*Insert a tag for pairs*", inserire lo stesso tag t usato durante le acquisizioni.
     (c) Avviare l'esecuzione tramite il pulsante "*Start*". Un metodo, scritto ad hoc, della classe `AggregateDataWithSameTagActivity` del package `ui` provvede a costruire n mappe in cui:
        – Ogni mappa deriva dalla sovrapposizione dei 12 cammini eseguiti.

– Per ogni mappa, la posizione iniziale di ogni cammino (cioè l'*EPM*) è lo stesso per ogni acquisizione, corrispondente alla cella in posizione centrale della griglia.
– Ogni mappa è caratterizzata da valori di parametri diversi. Si sceglie di costruire una mappa per ogni possibile combinazione ricavabile dai seguenti parametri ($w_1, w_{2m}, w_{2s}, w_{3m}, w_{3s}$ e ϵ sono i coefficienti della distribuzione del peso dell'algoritmo 4.3): passo di discretizzazione $\in \{0.5, 1\}$, $w_1 = 1$, $(w_{2m}, w_{2s}) \in \{(0.6, 0.4), (0.8, 0.2)\}$, $(w_{3m}, w_{3s}) \in \{(0.5, 0.25), (1, 0)\}$ e $\epsilon = 10\%$ del passo di discretizzazione. Quindi, in totale sono costruite 8 mappe.

Parametro	Valore
Min size of window	15
Max size of window	100
Operating k of std. dev. for strides counter	2
Transitory k of std. dev.	3.5
Threshold on std. dev. to switch k	1.8 m/s^2
Min threshold for acceleration	14.5 m/s^2
Static user threshold of std. dev.	0.55 m/s^2
Min timeout strides in ms.	200 ms

Tabella 9.1: Parametri dello strides counter.

Parametro	Valore
Size of window for pitch	30°
Size of window for azimuth	30°
Min pitch threshold of work zone in degrees	25°
Max pitch threshold of work zone in degrees	55°

Tabella 9.2: Parametri dell'orientation checker.

Parametro	Valore
Delay for accelerometer	Fastest
Delay for magnetometer	Fastest
Number of rows	100
Number of columns	100

Tabella 9.3: Altri parametri di test.

9.2.2 Test prolungato: cammino lungo il perimetro di un rettangolo

Le caratteristiche del test sono le seguenti:

- Il cammino effettuato corrisponde ad un percorso rettangolare composto da:
  1. Una sequenza rettilinea di 10 strides con orientazione all'incirca verso nord-ovest.
  2. Una rotazione di $90°$ in senso antiorario: la nuova direzione del cammino punta all'incirca verso sud-ovest.
  3. Una sequenza rettilinea di 15 strides.
  4. Una rotazione di $90°$ in senso antiorario: la nuova direzione del cammino punta all'incirca verso sud-est.

5. Una sequenza rettilinea di 10 strides.
6. Una rotazione di $90°$ in senso antiorario: la nuova direzione del cammino punta all'incirca verso nord-est.
7. Una sequenza rettilinea di 15 strides. L'utente arriva al punto iniziale da cui ha avuto origine il cammino.

- L'intero percorso è stato ripetuto per 10 volte, camminando sempre dallo stesso punto iniziale allo stesso punto finale.
- L'azimuth iniziale reale dell'inizio di ogni sessione rettilinea di strides è all'incirca il medesimo.
- La gamba che effettua il primo passo è quella in cui non è presente lo smartphone nella rispettiva tasca.
- I parametri d'esecuzione del test sono indicati nelle tabelle 9.1, 9.2, 9.3 e 9.4.
- All'esecuzione del primo percorso è stata creata una nuova mappa (opzione "*New*" alla voce "*New or loaded map?*", figura 7.1g). Prima di eseguire il percorso n-esimo si è impostata l'applicazione in modo da caricare e aggiornare la mappa costruita per il percorso $n-1$ (opzione "*Load*" alla voce "*New or loaded map?*", figura 7.1g): in questo modo, la mappa creata con il decimo percorso mantiene il contributo di tutti i percorsi precedentemente effettuati.
- Tra un percorso e l'altro si è registrato manualmente (su carta) i valori delle coppie *($n_{strides}$, azimuth)* attraverso l'interfaccia per la visualizzazione dei risultati (figura 7.1n). Si noti che tali coppie non sono salvate nel database.

Parametro	Valore
Passo di discretizzazione	0.5 m
w_1	1
w_{2m}	0.6
w_{2s}	0.4
w_{3m}	0.5
w_{3s}	0.25
ϵ	0.2

Tabella 9.4: Parametri di aggiornamento della mappa relativi al test prolungato. $w_1, w_{2m}, w_{2s}, w_{3m}, w_{3s}$ e ϵ sono i coefficienti della distribuzione del peso dell'algoritmo 4.3.

9.3 Analisi dei risultati

9.3.1 Test breve

La tabella 9.5 espone i risultati acquisiti dal modulo *Contapassi*. Come si può osservare, l'applicazione ha rilevato uno strides in più rispetto al numero realmente effettuato (cammino numero 2): grazie alla AUI, si è potuto constatare che il falso-positivo rilevato è dato dall'ultimo step effettuato, cioè dal penultimo passo eseguito. Probabilmente, l'eventuale cambio di frequenza del cammino per far in modo che l'utente si arresti al decimo strides, ha favorito il verificarsi di tale fenomeno. Tuttavia, l'accuratezza dello strides counter (cioè la corrispondenza tra valore teorico e valore sperimentale) supera il 99%.

Invece, i risultati dell'angolo d'uscita dei vari cammini evidenzia una certa variabilità del valore ottenuto. Nel caso di cammini sufficientemente lunghi (come nel caso del test, in cui 10 strides corrispondono ad una distanza di 15 m), una variazione di pochi gradi nell'angolo di partenza

Indice cammino	$n_{strides}$	Azimuth[°]
1	10	-65.7
2	11	-87.0
3	10	-66.6
4	10	-72.4
5	10	-79.0
6	10	-64.0
7	10	-65.7
8	10	-61.4
9	10	-80.7
10	10	-72.8
11	10	-78.3
12	10	-74.4

Tabella 9.5: Risultati sperimentali di 12 cammini composti da 10 strides e con all'incirca lo stesso azimuth (test breve). In rosso è segnalato l'errore di rilevazione.

corrispondente ad una differenza in metri tra i punti finali del cammino. Ad esempio, si supponga che l'angolo di partenza corretto sia $-73°$ (si è scelto il valore della media degli angoli). Denotando con $(0, 0)$ il punto iniziale espresso in metri allora la destinazione e_1 di questo cammino corrisponde al punto $(-14.3, 4.4)$, in accordo con le formule 5.5. Prendendo in considerazione il cammino numero 8, il punto finale e_2 è $(-13.2, 7.2)$. La distanza euclidea tra le destinazioni dei due cammini risulta essere di circa 3 metri.

Per tale motivo, analizzando le mappe generate dalla sovrapposizione dei 12 cammini (figura 9.1), si può osservare una certa dispersione dei pesi delle celle della griglia. Ovviamente, aumentare o diminuire la suddivisione del peso tra più celle costituisce un trade off per la costruzione della mappa. Infatti, nel caso di limitata spartizione verso le celle adiacenti (ad es. figura 9.1h), la distribuzione del peso è concentrata solo in poche celle. Di conseguenza, alcune di esse costituiscono dei massimi locali per la distribuzione. A questo punto, osservando la mappa, nasce il problema di determinare quale sia la destinazione "più probabile" del cammino poiché più celle, anche non adiacenti tra loro, hanno pesi paragonabili e di intensità elevata rispetto alle altre (riprendendo l'esempio, sia in un intorno della cella $(36, 54)$ che la cella $(37, 56)$ hanno un peso pari a 3).

Invece, nel caso di una spartizione più accentuata, quest'ultimo fenomeno risultato attenuato (in figura 9.1e, la cella $(37, 56)$ non costituisce più un picco intenso come in figura 9.1h); tuttavia, si presenta il problema di non avere una traccia nella griglia ben marcata.

	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
65								0.3																									
64							0.3	0.5	0.3																								
63							0.8	0.5	0.8	0.6	0.3																						
62						0.6	1	1	1	0.7	0.8	0.4	0.3																				
61						0.4	0.8	1.3	1	1	1	0.9	0.8	0.3	0.3																		
60					0.3		0.3	0.5	0.8	1.3	1.4	1.2	1.3	0.8	0.8	0.3	0.3																
59					1.4	0.9	0.5	0.5	0.5	0.5	0.7	1.4	1.3	1.6	1.3	1	0.8	0.3	0.3													
58					1.2	1.5	1.3	1.3	1.2	0.5	0.5	0.5	0.5	1.2	1.3	1.5	1.3	1	0.8	0.3												
57					0.5	0.9	1	1	1.4	1.4	1.3	1.2	0.5	0.8	0.8	1	1.3	1.7	1.3	1.3	0.9	0.3										
56				0.3	2	1	1	1	1	1.4	1	1.4	1.6	1.3	1.4	1.2	1	1.6	1.3	1.8	1.6	1.3	0.7	0.3								
55				0.5	1.6	1.7	1.8	1.8	1.8	1.8	1.6	1	1.7	1.8	1.4	1.6	1.4	1.3	1.4	1.5	1.9	1.8	2	1.3	0.8	0.3						
54				0.3	0.3	1.1	1.3	1.3	1.3	1.8	1.9	1.9	1.8	2	2.2	1.9	2.1	1.8	2.4	1.8	1.5	1.7	1.9	2.3	1.9	1.4	0.8	0.3				
53	0.3	0.3					0.3	0.3	0.3	0.3	0.8	1.4	1.3	1.3	1.6	1.9	2	2.2	2.3	2.7	2.2	2.6	2.8	3.1	2.9	2.6	2.9	1.4	1	0.3			
52	0.5	0.5	0.6	0.6	0.4	0.4	0.3	0.3	0.3	0.3	0.3	0.3	0.5	0.5	0.5	1	1.3	1.6	1.5	1.8	2.6	2.5	2.8	2.8	3.5	3.2	3.7	4.3	3.8	2.4	1.5		
51	0.3	0.3	0.4	0.4	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.8	0.8	1.3	1.6	1.6	1.8	2.2	3.1	3	4.1	4.8	5.7	5	3.8	3
50							0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.4	0.4	0.6	0.9	1.3	1.5	1.8	2.3	3.4	4.5	6.2	6
49																										0.3	0.3	0.3	0.3	0.3	1	2	3

(a) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.6$, $w_{2s} = 0.4$, $w_{3m} = 0.5$, $w_{3s} = 0.25$, $\epsilon = 0.05$.

64								1																							
63							1		1	0.6																					
62						0.6	1	1	1	0.4	1	0.4																			
61						0.4	1	2	1	1	1	0.6	1																		
60									1	2	1.6	1.4	1	1	1																
59					1.6	0.6					0.4	1.6	2	1.6	1	1	1														
58					1.4	2	2	2	1.4					1.4	2	2	1	1	1												
57						0.4	1	1	1.6	2.4	2	1.4				1	2	2.4	1	1	0.6										
56					2.2	1				0.6	1	1.6	2.6	2	1.6	0.4		0.6	2	3	1.8	1	0.4								
55				1	1.8	2.4	3	3	3	2	0.6		0.4	1	1.4	2.6	2.4	2	0.4		1.6	3	2.2	1							
54						0.6	1	1	1	2	3.4	3.4	3	3	2.2	0.6	0.6	1	2.6	3	2	0.4	1.4	3	3.4	1.4					
53												0.6	1	1	1.8	3.4	4	3.4	3	2.2	1.4	2.6	3	2.6	1.6	2.6	3.6	1.4			
52	1	1	0.6	0.6	0.4	0.4												0.6	1	1.8	3.6	4	4	3.8	4.2	3.4	3.4	3.8	4	1.4	
51			0.4	0.4	0.6	0.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.6	0.6	1	2.2	3.6	4	5.8	7	8.2	7	1.8
50																						0.4	0.4	0.6	0.6	1	1	1	1	2.4	5	10	12
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50

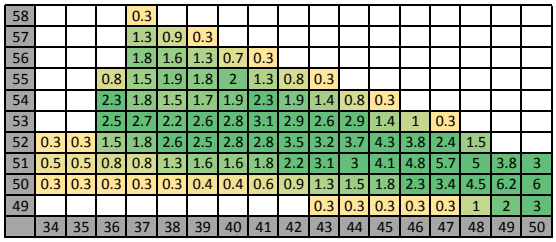
(b) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.6$, $w_{2s} = 0.4$, $w_{3m} = 1$, $w_{3s} = 0$, $\epsilon = 0.05$.

65								0.3																							
64							0.3	0.5	0.3																						
63							0.8	0.5	0.8	0.8	0.3																				
62						0.8	1	1	1	0.5	0.8	0.2	0.3																		
61						0.2	0.8	1.3	1	1	1	1.1	0.8	0.3	0.3																
60					0.3		0.3	0.5	0.8	1.3	1.6	1	1.3	0.8	0.8	0.3	0.3														
59					1.6	1.1	0.5	0.5	0.5	0.5	0.5	1.6	1.3	1.8	1.3	1	0.8	0.3	0.3												
58					1	1.5	1.3	1.3	1	0.5	0.5	0.5	0.5	1	1.3	1.5	1.3	1	0.8	0.3											
57					0.5	0.7	1	1	1.6	1.2	1.3	1	0.5	0.8	0.8	1	1.3	1.5	1.3	1.3	1.1	0.3									
56				0.3	2.4	1	1	1	1	1.6	1	1.6	1.8	1.3	1.6	1	1	1.8	1.3	1.8	1.2	1.3	0.5	0.3							
55				0.5	1.2	1.5	1.8	1.8	1.8	1.8	1.8	1	1.5	1.8	1.2	1.8	1.2	1.3	1.2	1.5	2.1	1.8	2.4	1.3	0.8	0.3					
54				0.3	0.3	1.3	1.3	1.3	1.3	1.8	1.7	1.7	1.8	2	2.6	2.1	2.3	1.8	2.6	1.8	1.5	1.5	1.7	2.3	1.7	1.2	0.8	0.3				
53	0.3	0.3					0.3	0.3	0.3	0.3	0.8	1.6	1.3	1.3	1.2	1.7	2	2	2.3	3.1	2	2.8	2.8	3.3	3.1	2.8	3.1	1.2	1	0.3		
52	0.5	0.5	0.8	0.8	0.2	0.2	0.3	0.3	0.3	0.3	0.3	0.3	0.5	0.5	0.5	1	1.3	1.8	1.5	1.4	2.8	2.5	2.8	2.4	3.9	3	3.5	4.9	3.8	2.2	1.5	
51	0.3	0.3	0.2	0.2	0.8	0.8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.8	0.8	1.3	1.8	1.8	1.8	1.6	3.3	3	3.7	4.8	6.1	5	3.4	3
50							0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.2	0.2	0.8	1.1	1.3	1.5	1.8	2.3	3.2	4.5	6.6	6
49																										0.3	0.3	0.3	0.3	0.3	1	2	3
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50

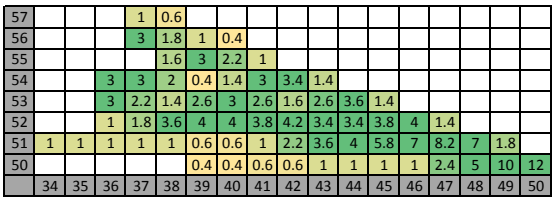
(c) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.8$, $w_{2s} = 0.2$, $w_{3m} = 0.5$, $w_{3s} = 0.25$, $\epsilon = 0.05$.

64								1																							
63							1		1	0.8																					
62						0.8	1	1	1	0.2	1	0.2																			
61						0.2	1	2	1	1	1	0.8	1																		
60									1	2	1.8	1.2	1	1	1																
59					1.8	0.8					0.2	1.8	2	1.8	1	1	1														
58					1.2	2	2	2	1.2					1.2	2	2	1	1	1												
57						0.2	1	1	1.8	2.2	2	1.2				1	2	2.2	1	1	0.8										
56					2.6	1				0.8	1	1.8	2.8	2	1.8	0.2		0.8	2	3	1.4	1	0.2								
55				1	1.4	2.2	3	3	3	2	0.8		0.2	1	1.2	2.8	2.2	2	0.2		1.8	3	2.6	1							
54						0.8	1	1	1	2	3.2	3.2	3	3	2.6	0.8	0.8	1	2.8	3	2	0.2	1.2	3	3.2	1.2					
53												0.8	1	1	1.4	3.2	4	3.2	3	2.6	1.2	2.8	3	2.8	1.8	2.8	3.8	1.2			
52	1	1	0.8	0.8	0.2	0.2												0.8	1	1.4	3.8	4	4	3.4	4.6	3.2	3.2	4.4	4	1.2	
51			0.2	0.2	0.8	0.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0.8	0.8	1	1.6	3.8	4	5.4	7	8.6	7	1.4
50																						0.2	0.2	0.8	0.8	1	1	1	1	2.2	5	11	12
	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50

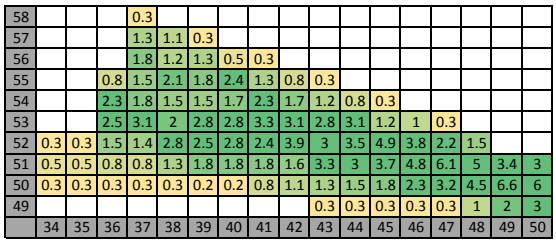
(d) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.8$, $w_{2s} = 0.2$, $w_{3m} = 1$, $w_{3s} = 0$, $\epsilon = 0.05$.



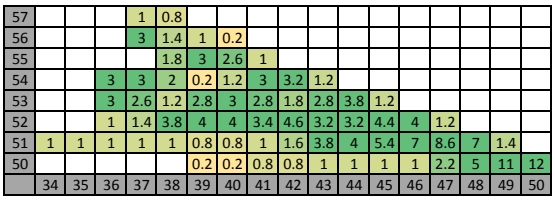
(e) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.6$, $w_{2s} = 0.4$, $w_{3m} = 0.5$, $w_{3s} = 0.25$, $\epsilon = 0.1$.



(f) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.6$, $w_{2s} = 0.4$, $w_{3m} = 1$, $w_{3s} = 0$, $\epsilon = 0.1$.



(g) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.8$, $w_{2s} = 0.2$, $w_{3m} = 0.5$, $w_{3s} = 0.25$, $\epsilon = 0.1$.



(h) Passo di discretizzazione= 0.5 m, $w_1 = 1$, $w_{2m} = 0.8$, $w_{2s} = 0.2$, $w_{3m} = 1$, $w_{3s} = 0$, $\epsilon = 0.1$.

Figura 9.1: Mappe ottenute dalla sovrapposizione dei 12 cammini del test breve. Ogni cammino inizia dalla cella $(50, 50)$. I parametri indicati nelle didascalie fanno riferimento all'algoritmo 4.3.

9.3.2 Test prolungato

Indice percorso	1				2				3			
$\mathbf{n_{strides}}$	10	15	10	15	10	15	10	16	10	15	10	15
Azimuth[°]	-57	-135	82	5	-55	-161	87	-2	-55	-148	78	6

Indice percorso	4				5				6			
$\mathbf{n_{strides}}$	10	15	10	15	10	15	10	14	10	15	10	15
Azimuth[°]	-58	-146	77	9	-53	-147	70	6	-53	-137	90	14

Indice percorso	7				8				9			
$\mathbf{n_{strides}}$	10	15	10	15	10	15	10	15	10	15	10	15
Azimuth[°]	-56	-140	66	-2	-55	-137	69	4	-61	-151	69	1

Indice percorso	10			
$\mathbf{n_{strides}}$	10	15	10	15
Azimuth[°]	-22	-149	74	3

Tabella 9.6: Risultati sperimentali di 10 percorsi rettangolari (test prolungato). In rosso è segnalato l'errore di rilevazione mentre in blu è indicato un'errore nell'esecuzione del test: l'utente, ha eseguito un stride in meno rispetto ai 15 previsti. Tuttavia l'applicazione ha rilevato in modo corretto il numero di strides eseguito.

Anche per il test prolungato, il modulo *Contapassi* ha esibito una buona accuratezza, commettendo un solo errore di rilevazione (tabella 9.6) rispetto ai 449 strides effettuati. Come per il test breve, anche in questo caso l'errore commesso si tratta di un falso rilevamento in corrispondenza dell'ultimo step effettuato.

In figura 9.2 si può notare l'insufficiente accuratezza della stima offerta da questa implementazione dell'*Orientation checker* per percorsi lunghi: solamente la prima porzione del cammino (i primi 10 strides) riescono a generare una traccia ben marcata nella mappa mentre le restanti parti del cammino non seguono un andamento comune per via dell'errore accumulato. Pertanto è evidente che questo problema debba essere affrontato mediante nuovi algoritmi e/o facendo uso di nuovi dispositivi e tecniche che integrino la stima effettuata con il solo smartphone.

9.4 Bug noti

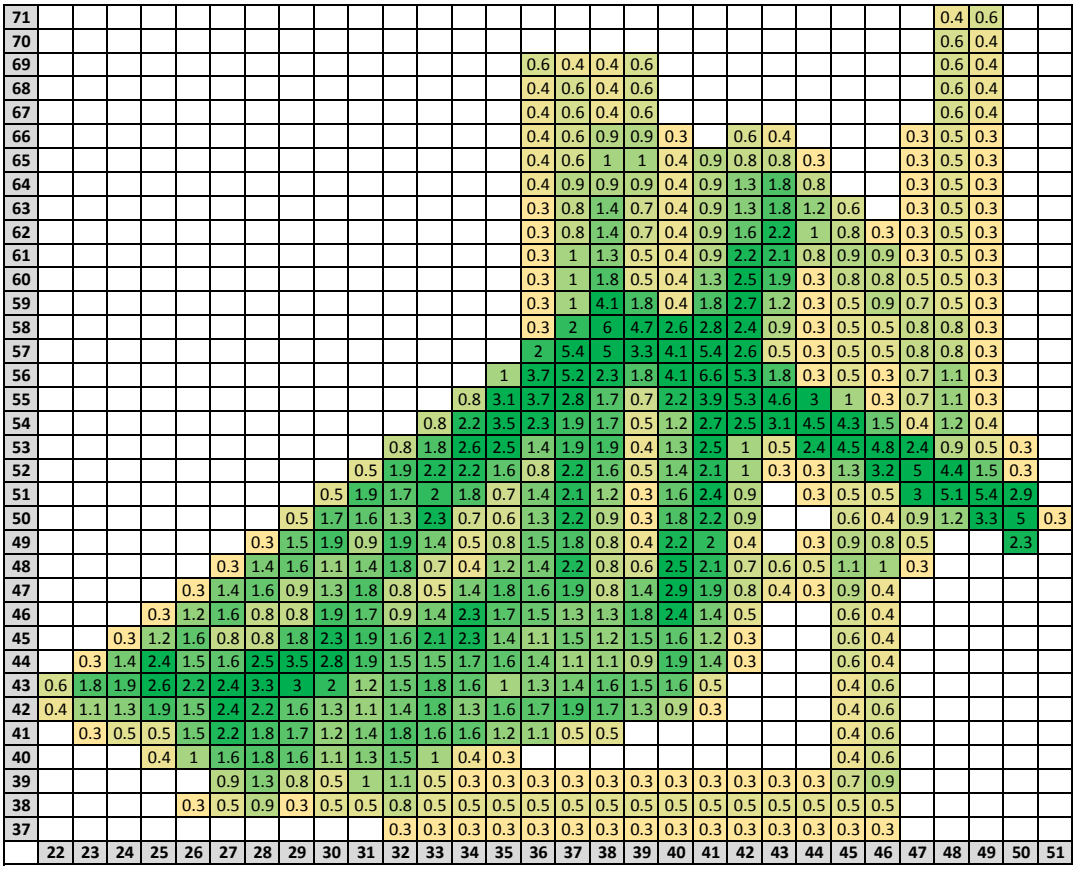
In questa sezione si descrivono i bug noti dell'applicazione a cui non è stato ancora trovato un rimedio:

Bound e Unbound di `PreferenceFragment`.

La classe interna `CreationMapPreferenceFragment` di `IndoorNavigationMainActivity` effettua il *bound* e *unbound* al servizio `CreateMapService` ogni qualvolta esso è attivo. La documentazione ufficiale di Android[1], consiglia di eseguire il *bound* e l'*unbound* nei metodi `onStart` e `onStop` di una `Activity`. Tuttavia, in alcune circostanze, si è verificato sperimentalmente che, in seguito alla pressione del tasto blocca schermo, Android distrugge il componente `Activity` principale invocando solamente il metodo `onPause` (in accordo con la documentazione[2]) e senza richiamare `onStop`. Di conseguenza, si crea un memory leak poiché l'oggetto che mantiene il legame tra `Activity` e `Service` non è più valido.

Purtroppo tale situazione, oltre a generare un'eccezione visibile nel file di log, inficia il corretto arresto del servizio, poiché l'*unbound* non è avvenuto correttamente. Infatti, anche in caso di ar-

[1] Bound services. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.
[2] Activity lifecycle. Fonte: documentazione Android. Link. Ultima visita: 3 dicembre 2014.



	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
71																											0.4	0.6		
70																											0.6	0.4		
69															0.6	0.4	0.4	0.6									0.6	0.4		
68															0.4	0.6	0.4	0.6									0.6	0.4		
67															0.4	0.6	0.4	0.6									0.6	0.4		
66															0.4	0.6	0.9	0.9	0.3		0.6	0.4				0.3	0.5	0.3		
65															0.4	0.6	1	1	0.4	0.9	0.8	0.8	0.3			0.3	0.5	0.3		
64															0.4	0.9	0.9	0.9	0.4	0.9	1.3	1.8	0.8			0.3	0.5	0.3		
63															0.3	0.8	1.4	0.7	0.4	0.9	1.3	1.8	1.2	0.6		0.3	0.5	0.3		
62															0.3	0.8	1.4	0.7	0.4	0.9	1.6	2.2	1	0.8	0.3	0.3	0.5	0.3		
61															0.3	1	1.3	0.5	0.4	0.9	2.2	2.1	0.8	0.9	0.9	0.3	0.5	0.3		
60															0.3	1	1.8	0.5	0.4	1.3	2.5	1.9	0.3	0.8	0.8	0.5	0.5	0.3		
59															0.3	1	4.1	1.8	0.4	1.8	2.7	1.2	0.3	0.5	0.9	0.7	0.5	0.3		
58															0.3	2	6	4.7	2.6	2.8	2.4	0.9	0.3	0.5	0.5	0.8	0.8	0.3		
57															2	5.4	5	3.3	4.1	5.4	2.6	0.5	0.3	0.5	0.5	0.8	0.8	0.3		
56														1	3.7	5.2	2.3	1.8	4.1	6.6	5.3	1.8	0.3	0.5	0.3	0.7	1.1	0.3		
55													0.8	3.1	3.7	2.8	1.7	0.7	2.2	3.9	5.3	4.6	3	1	0.3	0.7	1.1	0.3		
54												0.8	2.2	3.5	2.3	1.9	1.7	0.5	1.2	2.7	2.5	3.1	4.5	4.3	1.5	0.4	1.2	0.4		
53											0.8	1.8	2.6	2.5	1.4	1.9	1.9	0.4	1.3	2.5	1	0.5	2.4	4.5	4.8	2.4	0.9	0.5	0.3	
52										0.5	1.9	2.2	2.2	1.6	0.8	2.2	1.6	0.5	1.4	2.1	1	0.3	0.3	1.3	3.2	5	4.4	1.5	0.3	
51									0.5	1.9	1.7	2	1.8	0.7	1.4	2.1	1.2	0.3	1.6	2.4	0.9		0.3	0.5	0.5	3	5.1	5.4	2.9	
50								0.5	1.7	1.6	1.3	2.3	0.7	0.6	1.3	2.2	0.9	0.3	1.8	2.2	0.9			0.6	0.4	0.9	1.2	3.3	5	0.3
49							0.3	1.5	1.9	0.9	1.9	1.4	0.5	0.8	1.5	1.8	0.8	0.4	2.2	2	0.4		0.3	0.9	0.8	0.5			2.3	
48						0.3	1.4	1.6	1.1	1.4	1.8	0.7	0.4	1.2	1.4	2.2	0.8	0.6	2.5	2.1	0.7	0.6	0.5	1.1	1	0.3				
47					0.3	1.4	1.6	0.9	1.3	1.8	0.8	0.5	1.4	1.8	1.6	1.9	0.8	1.4	2.9	1.9	0.8	0.4	0.3	0.9	0.4					
46				0.3	1.2	1.6	0.8	0.8	1.9	1.7	0.9	1.4	2.3	1.7	1.5	1.3	1.3	1.8	2.4	1.4	0.5			0.6	0.4					
45			0.3	1.2	1.6	0.8	0.8	1.8	2.3	1.9	1.6	2.1	2.3	1.4	1.1	1.5	1.2	1.5	1.6	1.2	0.3			0.6	0.4					
44		0.3	1.4	2.4	1.5	1.6	2.5	3.5	2.8	1.9	1.5	1.5	1.7	1.6	1.4	1.1	1.1	0.9	1.9	1.4	0.3			0.6	0.4					
43	0.6	1.8	1.9	2.6	2.2	2.4	3.3	3	2	1.2	1.5	1.8	1.6	1	1.3	1.4	1.6	1.5	1.6	0.5				0.4	0.6					
42	0.4	1.1	1.3	1.9	1.5	2.4	2.2	1.6	1.3	1.1	1.4	1.8	1.3	1.6	1.7	1.9	1.7	1.3	0.9	0.3				0.4	0.6					
41		0.3	0.5	0.5	1.5	2.2	1.8	1.7	1.2	1.4	1.8	1.6	1.6	1.2	1.1	0.5	0.5							0.4	0.6					
40				0.4	1	1.6	1.8	1.6	1.1	1.3	1.5	1	0.4	0.3										0.4	0.6					
39						0.9	1.3	0.8	0.5	1	1.1	0.5	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.7	0.9					
38					0.3	0.5	0.9	0.3	0.5	0.5	0.8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5					
37											0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3					

Figura 9.2: Mappa ottenuta dalla sovrapposizione dei 10 cammini del test prolungato (tabella 9.6). Ogni cammino inizia dalla cella $(50, 50)$. I parametri di generazione della mappa sono indicati nella tabella 9.4.

resto da parte dell'utente (tramite la pressione del pulsante "*Stop the creation map task*") il servizio non viene arrestato, come se esistesse ancora un componente ad esso legato.

Per tale ragione, si è scelto di effettuare il *bound* e l'*unbound* in altri metodi del ciclo di vita del `Fragment` (nel caso particolare si sono scelti i metodi `onAttach` e `onResume`). Sebbene tale soluzione risolvi il problema del leak di memoria, emerge un ulteriore problema: come indicato in figura 9.3, quando l'utente ruota continuamente lo schermo (passando quindi dalla modalità portrait a landscape e viceversa), si verifica una doppia esecuzione del metodo `onAttach` la quale provvede ad effettuare il *bound* al servizio per ben due volte. Non è noto il motivo che porta Android a svolgere tale comportamento. Come conseguenza il service non è in grado di eseguire l'arresto correttamente (figura 9.4), come nel caso precedente.

Nonostante siano state sperimentate diverse possibilità in cui effettuare il *bound/unbound*, questo problema non è ancora stato risolto.

```
.unipd.com.indoornavigationforvisuallyimpairedpeople I/CreationMapFragment:  Bound to CreateMapService: main onAttach
.unipd.com.indoornavigationforvisuallyimpairedpeople I/art:  GcCauseBackground sticky partial concurrent mark sweep GC free
.unipd.com.indoornavigationforvisuallyimpairedpeople I/CreationMapFragment:  Unbound from CreateMapService: onPause
.unipd.com.indoornavigationforvisuallyimpairedpeople I/CreationMapFragment:  Bound to CreateMapService: main onAttach
.unipd.com.indoornavigationforvisuallyimpairedpeople I/CreationMapFragment:  Bound to CreateMapService: main onAttach
.unipd.com.indoornavigationforvisuallyimpairedpeople I/art:  GcCauseBackground partial concurrent mark sweep GC freed 14431
.unipd.com.indoornavigationforvisuallyimpairedpeople I/CreationMapFragment:  Unbound from CreateMapService: onPause
.unipd.com.indoornavigationforvisuallyimpairedpeople I/CreationMapFragment:  Bound to CreateMapService: main onAttach
```

Figura 9.3: Errore durante il *bound/unboud* con il servizio. Due operazioni di *bound* non sono intervallate da un'operazione di *unbound*. Di conseguenza il servizio non è successivamente arrestato correttamente (figura 9.4).

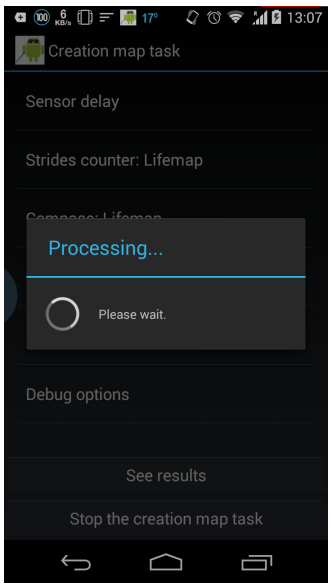


Figura 9.4: Arresto del servizio non riuscito.

Capitolo 10

Sviluppi futuri

Fonte: *jantoo* website. Ultima visita: 3 dicembre 2014.

10.1 Contributo scientifico

10.1.1 Modello per la stima della lunghezza dello stride

Nel progetto si fa uso di un modello che stima con una misura costante la lunghezza dello stride dell'utente. In realtà, in accordo con [56], ad una stessa velocità del cammino, l'estensione dello stride può variare del $\pm 40\%$ da persona a persona e dipende, in largo modo, dalla lunghezza della gamba. In aggiunta, anche la frequenza del cammino influenza fino al $\pm 50\%$ l'estensione dello stride. Pertanto, anche conoscendo la lunghezza media di uno stride, non è possibile determinare con precisione la distanza percorsa poiché la misura stessa cambia nel tempo.

Poiché un metodo ZUPT non è implementabile nella posizione di lavoro dello smartphone richiesta dall'applicazione, alcune tecniche di stima dell'estensione dello stride non possono essere utilizzate (ad esempio [57]). Un'interessante alternativa, consiste nell'algoritmo presente in [56]: l'idea principiale risiede nel fatto che la lunghezza dello stride è proporzionale allo spostamento verticale della parte superiore del corpo (dall'anca in su). Rilevando l'accelerazione verticale del pedone è possibile quindi stimare la lunghezza dello stride in accordo con la seguente approssimazione (empirica):

$$Stride_{length} = K \sqrt[4]{a_{max} - a_{min}}$$

in cui $K \in [0.5, 0.57]$ è una costante sperimentale, calibrata individualmente in funzione dell'utilizzatore [53].

Tale metodo risulta interessante poiché la posizione di lavoro dello smartphone per questo progetto è simile alla posizione richiesta dall'algoritmo.

10.1.2 Strides counter

Sebbene lo strides counter realizzato funzioni in maniera più che soddisfacente, è possibile individuare un insieme di fattori che contribuiscono al miglioramento della rilevazione:

- Inserimento di un'altra soglia che consenta di rilevare anche gli steps.
- Pre-processing del segnale mediante un filtro passa basso per eliminare le fluttuazioni ad alta frequenza. Tuttavia bisogna valutare l'impatto sul carico di lavoro che il filtro impone alla cpu dello smartphone.
- Introduzione di una procedura semi-automatica di taratura dei parametri. Al primo utilizzo dell'applicazione, una procedura di taratura dovrà chiedere all'utente di effettuare un cammino e fornire il numero di strides effettuati. Avendo a disposizione il numero esatto e il numero rilevato di strides è possibile implementare una procedura di taratura che minimizzi la probabilità di errore nell'identificazione.

Altre realizzazioni dello strides counter potrebbero analizzare il segnale dell'accelerometro nel dominio della frequenza e quindi individuare la componente periodica che si manifesta quando un pedone cammina.

10.1.3 Calcolo dell'azimuth

Anche in letteratura, risulta difficile acquisire misure accurate dell'azimuth di un cammino effettuato dall'utente. Molto spesso, si integrano i risultati forniti dal magnetometro con le indicazioni di un sistema esterno (ad es. tramite sistemi che fanno uso di triangolazioni di segnali radio) o di altri strumenti di misura, (ad es. giroscopio, come utilizzato in [58]).

In accordo con [30, 59], un possibile sviluppo futuro, che utilizza la stessa piattaforma hardware, consiste nel monitorare costantemente l'azimuth del cammino. Tuttavia, come osservato in figura 5.9, è presente una componente periodica nelle tracce acquisite dal magnetometro.

L'idea di tali algoritmi si basa sul fatto che osservando i valori dell'azimuth in corrispondenza del ripetersi di uno stesso evento, allora tali quantità assumono un valore simile. Nel dettaglio la procedura consiste in due passi:

1. Si analizza la traccia relativa all'accelerometro: ogni qualvolta il valore $\|A\|$ assume il valore g (cioè pari all'accelerazione di gravità) allora si registra il timestamp t_A del valore di $\|A\|$.
2. Per ogni timestamp t_A si estrae il valore dell'azimuth ricavato dai campioni del magnetometro.

Poiché le componenti periodiche delle tracce relative all'accelerometro e al magnetometro esibiscono una frequenza simile è ragionevole ipotizzare che l'insieme di valori dell'azimuth estratti siano simili. Tuttavia le tracce in figura 5.9 esibiscono, oltre alla componente periodica, anche un fenomeno di *drift*: per tale motivo, un'eventuale applicazione delle tecniche citate in precedenza richiede anche un'adeguata elaborazione del segnale da analizzare.

Figura 10.1: Esempio di dispositivo *wearable*. Fonte: *Motorola* website. Ultima visita: 3 dicembre 2014.

10.2 Piattaforma hardware-software

10.2.1 Android Wear

Nel giugno 2014, Google ha introdotto Android *Wear*, cioè una versione modifica di Android progettata per una nuova famiglia di prodotti "indossabili", chiamati anche *wearable* (figura 10.1). Questi dispositivi possiedono un System on Chip (SoC) con potenza di elaborazione paragonabile a molti smartphone di fascia media-bassa del mercato (cioè dal costo inferiore a 200 €) ed integrano un ampio numero di sensori, tra cui l'accelerometro, il magnetometro, il sensore di battiti cardiaci, l'altimetro ecc.... Tali apparecchi sono quindi adeguati ad ospitare l'applicazione realizzata durante questo progetto. In aggiunta, l'elevata similitudine tra le API Android e le API Android *Wear* (in molti casi si tratta delle stesse classi e metodi) facilitano la portabilità del programma tra le due piattaforme.

Uno dei vantaggi nell'utilizzo dei *weareable* rispetto agli smartphone consiste nel fatto che questi apparecchi sono indossati dall'utente e quindi assumono una posizione vincolata ad ogni utilizzo. Di conseguenza la misurazione effettuata dal dispositivo è soggetta ad un disturbo di natura diversa rispetto ad uno smartphone inserito in una tasca: sperimentalmente si è notato che, seppur lo smartphone sia posto in uno spazio ristretto, esso scivola lungo la stoffa interna della tasca, non risultando più perpendicolare al terreno. Invece, con un *wearable* tale problema non si presenta poiché la pressione esercitata dal cinturino assicura di mantenere in posizione il dispositivo.

Tuttavia è bene sottolineare che la nuova posizione dell'apparecchio porterà sicuramente alla rilevazione di valori di accelerazione di intensità maggiore rispetto ai valori acquisiti in prossimità delle anche poiché, durante un cammino, il braccio è più sollecitato rispetto al bacino. Pertanto, sarà necessaria un'adeguata rivalutazione dei parametri di esercizio dell'algoritmo.

10.2.2 Aumento della frequenza di campionamento

Ispezionando le caratteristiche dei sensori dello smartphone di test (Motorola Moto G, 1° generazione) attraverso i metodi della classe `Sensor` si è individuato che l'accelerometro installato è il modello LIS3DH, prodotto dalla *STMicroelectronics*, mentre il magnetometro è il modello AK8963 (non è specificata quale variante, cioè C o N), prodotto dalla *Asahi Kasei MicroDevice Corporation (AKM)*. Dalla lettura del datasheet delle specifiche tecniche si osserva che:

- Accelerometro: *"it is capable of measuring accelerations with output data rates from* 1 *Hz to* 5 *kHz"* [60].

- Magnetometro: *Continuous measurement mode 2. Sensor is measured periodically in* 100 *Hz* [61].

In linea teorica, l'accelerometro è in grado di campionare il segnale ad una frequenza maggiore rispetto ai circa 90 Hz rilevati. Un possibile sviluppo futuro riguarda:

- Uno studio preliminare che valuti la fattibilità dello sviluppo di un modulo *Contapassi* con Android NDK in modo tale da prelevare campioni ad una frequenza maggiore.
- Uno studio che valuti la portabilità del modulo *Contapassi* realizzato al punto precedente anche su dispositivi che utilizzano sensori hardware diversi. In questo caso è opportuno determinare quale sia il driver più adatto con cui realizzare il modulo.

Appendice

Appendice A

Strumenti di sviluppo

Fonte: *glasbergen* website. Ultima visita: 3 dicembre 2014.

A.1 Sensor data carrier

A.1.1 Definizione delle specifiche

L'applicazione Android *Sensor data carrier* costituisce uno strumento di sviluppo separato dall'applicazione principale del progetto. La sua funzione consiste nell'esportare i valori dei sensori dello smartphone in un formato elaborabile da altri strumenti software (come *NI LabVIEW* o *Mathworks Matlab*).

Con il committente si sono accordate le seguenti specifiche per l'applicazione:

- Acquisizione dei dati dall'accelerometro e dal magnetometro. In aggiunta si è prevista la possibilità di acquisire i campioni da un sensore software, cioè da un insieme di valori ottenuti combinando i dati di più sensori fisici. Ad esempio, i valori di orientazione ricavati dall'*Orientation checker* (sezione 5.2.4) costituisco i possibili valori restituiti da un sensore software.

- Invio dei dati acquisiti tramite il protocollo *UDP*.

A.1.2 Progettazione software

L'applicazione consiste di quattro moduli separati:

- Il modulo *Interfaccia Utente* (*UI*) ha il compito di acquisire i parametri di configurazione dei sensori forniti dall'utente.
- Il modulo *Acquisisci dati* ha la funzione di acquisire i dati dai sensori.
- Il modulo *Spedisci dati* ha il compito di inviare i dati dei sensori ad un ricevitore *UDP* esterno tramite rete wi-fi.
- Il modulo *Core* alloca le classi che realizzano gli altri moduli e si occupa di mantenere attiva l'applicazione.

A.1.2.1 Modulo UI

La *UI* dell'applicazione ha due compiti:

1. Acquisire i parametri d'ingresso dell'applicazione, cioè:
   - L'*IP* e la porta di destinazione del receiver *UDP*.
   - La tipologia di sensori da cui acquisire i campioni.
   - La frequenza "suggerita" di campionamento.
2. Visualizzare alcune informazioni di debug, cioè:
   - Valore corrente dei campioni acquisiti nella seguente forma ordinata:
     (a) Indice campione.
     (b) Timestamp in ns.
     (c) Valore della prima componente.
     (d) Valore della seconda componente.
     (e) Valore della terza componente.
   - Numero di pacchetti inviati.

In aggiunta sono disponibili altre due voci:

- "*On background, stop after*" indica il numero di minuti entro cui l'applicazione si arresterà in modo autonomo se la schermata principale della stessa non è in primo piano. Tale precauzione assicura di non consumare troppo la batteria del dispositivo nel caso in cui l'applicazione resti accidentalmente attiva.
- La checkbox "*Debug*" attiva la modalità debug in cui l'applicazione si arresta dopo aver inviato 100 pacchetti e visualizza, nel riquadro sottostante, l'elenco di tutti i payloads inviati.

L'applicazione è avviata premendo il tasto *Start*. I simboli "?" forniscono indicazioni sul significato dei vari parametri.

A.1.2.2 Modulo Acquisisci dati

In analogia a quanto avviene per l'applicazione principale del progetto, le attività di acquisizione dei campioni dai sensori e la successiva elaborazione, costituita dall'invio tramite pacchetti *UDP*, sono disaccoppiate. Tale struttura è realizzata di pari passo come descritto nella sezione 5.3.1. Si ha quindi:

- Un thread T_A che acquisisce i campioni dall'accelerometro e li inserisce in una coda c_A.
- Un thread T_M che acquisisce i campioni dal magnetometro e li inserisce in una coda c_M.
- Un thread T_S che acquisisce i campioni dal sensore software e li inserisce in una coda c_s.

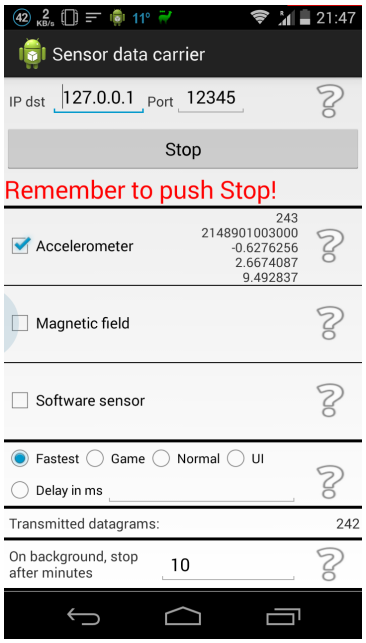


Figura A.1: Screenshot di *Sensor data carrier* durante il suo funzionamento.

A.1.2.3 Modulo Spedisci dati

Il modulo *Spedisci dati* esegue le seguenti operazioni:

1. Estrae un campione da c_A, uno da c_M e uno da c_S.
2. Da ciascun campione estrae i valori di *timestamp*, *prima componente*, *seconda componente* e *terza componente*.
3. Converte i valori estratti in un unica stringa di testo *s*: ogni valore è separato da un carattere *separatore* (di default vale |).
4. Antepone ad *s* l'indice del campione inviato e un'etichetta identificativa del tipo di campione che vale:
   - *A* se il campione è acquisito dall'accelerometro.
   - *M* se il campione è acquisito dal magnetometro.
   - *S* se il campione è acquisito dal sensore software.

   La seguente stringa costituisce un esempio di payload inviato:
   1|A|0.2|-0.3|-9.8
5. Crea un pacchetto *UDP* e lo affida al sistema operativo per l'invio all'host di destinazione.

A.1.2.4 Modulo Core

Il modulo core ha il compito di:

- Estrarre i parametri forniti dal modulo *UI*.
- Allocare gli oggetti delle classi dei moduli *Acquisisci dati* e *Spedisci dati*.
- Avviare l'acquisizione dei campioni e l'invio dei pacchetti.

- Assicurare l'esecuzione dell'applicazione anche quando l'utente preme il tasto "blocca schermo".
- Arrestare correttamente l'applicazione.

L'intero modulo è realizzato tramite una sottoclasse di `Service` in analogia a quanto avviene per il modulo *Core* dell'applicazione principale (capitolo 6).

A.1.3 Implementazione

In questa sezione si sottolineano alcuni aspetti dell'implementazione realizzata:

- La funzione del modulo *Spedisci dati* è realizzata da un thread T_{UDP} in modo tale da favorire la parallelizzazione nel caso di processori multicore.
- Affinché l'applicazione funzioni correttamente è necessario che l'elaborazione di T_{UDP}, cioè il prelievo dei campioni dalle rispettive code e il successivo invio, sia più rapido del tempo medio necessario ai thread T_A, T_M e T_S per generare i loro campioni. Se così non fosse, man mano che l'applicazione resta in esecuzione, la taglia delle code c_A, c_M e c_S è destinata ad aumentare fino a generare un'eccezione che arresta l'applicazione. Una valida soluzione a questo problema consiste nel diminuire la frequenza di campionamento "suggerita" ai sensori.
- Sebbene sia possibile registrare l'acquisizione dei campioni dei tre sensori con tre frequenza di campionamento diverse, nell'applicazione si è implementato solamente la registrazione che fa uso dell'unica frequenza acquisita dal modulo *UI*. Ovviamente, modificando l'interfaccia utente, è possibile correggere tale limite.
- Al fine di evitare che l'applicazione dimenticata in background consumi tutta la batteria del dispositivo, quando l'`Activity` principale perde il focus dell'utente (più precisamente quando è invocato il metodo `onPause`) viene creato un task che arresta l'acquisizione e l'invio dei campioni. Tuttavia la sua esecuzione è posticipata di un numero di minuti indicato nell'apposito campo dell'interfaccia utente.
- All'atto pratico, il sensore software è costituito da codice che:
  - Acquisisce dati da più sensori.
  - Effettua una qualche forma di elaborazione sui campioni.
  - Ritorna un nuovo campione.

  Nello sviluppo dell'applicazione si è sfruttato il sensore software per valutare la qualità e affidabilità dei dati generati dai metodi `getRotationMatrix` e `getOrientation` della classe `SensorManager` di Android attraverso *LabVIEW*; in questo caso il payload di un pacchetto UDP è costituito dal valore degli angoli *azimuth*, *pitch* e *roll* del sistema di riferimento *Smartphone*.

A.1.4 Note finali

Nella piattaforma hardware di test, l'invio contemporaneo dei campioni acquisiti dal sensore software e da uno dei due sensori hardware porta ad un comportamento anomalo dell'applicazione (ad es. rallentamento di tutta l'interfaccia utente). Probabilmente, il motivo di tale situazione dipende dall'eccessivo carico indotto da T_{UDP}: il numero di pacchetti da generare (circa 3 pacchetti ogni 10 ms) sovraccarica la gestione dello stack del protocollo *UDP* compiuta da Android. In questo caso è opportuno:

- Ridurre la frequenza di campionamento.

- Realizzare una nuova implementazione che faccia uso del protocollo *TCP*. Costruendo un unico pacchetto per più campioni dei sensori è possibile ridurre il numero dei pacchetti generati e semplificare la gestione dello stack del protocollo compiuta da Android.

L'intero progetto è disponibile al path `Codice/Android/Sensor data carrier`.

A.2 Acquire sensor data

L'applicazione *LabVIEW Acquire sensor data* provvede a:

- Acquisire i dati dal socket *UDP* specificato (di default vale 12345).
- Visualizzare in tempo reale i dati acquisiti.
- Salvare su disco i dati acquisiti in accordo con la formattazione `csv`. Per ogni sensore, da cui avviene l'acquisizione, è generato un file che contiene tutti i suoi campioni. È possibile identificare a quale sensore appartengono i campioni di un file osservando la lettera finale del suo nome, in accordo con le seguenti regole:
  - $A \rightarrow$ campioni accelerometro.
  - $M \rightarrow$ campioni magnetometro.
  - $S \rightarrow$ campioni sensore software.

Il programma è disponibile al path `Codice/LabVIEW/Acquire sensor data.vi`

A.3 Generica Labview VI

In figura A.2 è illustrato il *block diagram* di *LabVIEW* che può essere utilizzato per leggere i dati salvati su file da *Acquire sensor data*.

A.4 Esportazione delle tabelle del database

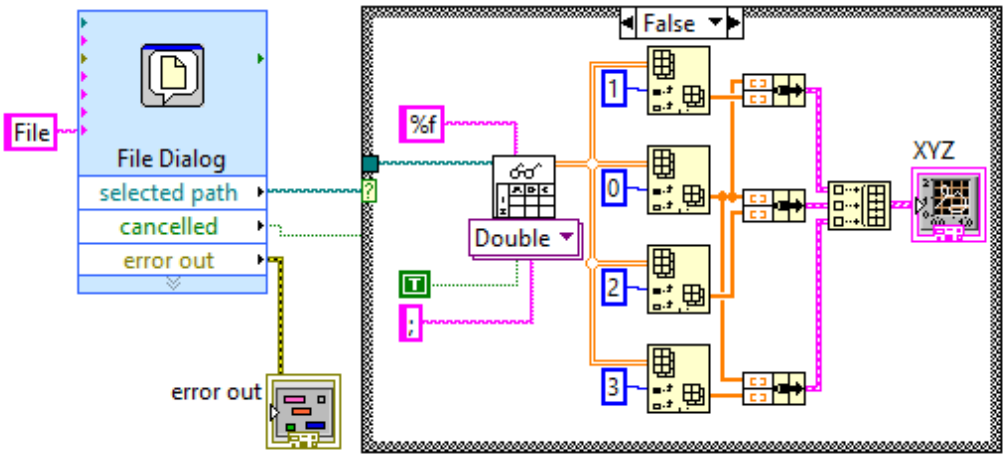
Per esportare le tabelle del database nel formato `csv` si procede in due modi:

- Per le tabelle *MP*, *Elemento EPM* e *Elemento Goal* è necessario:
  1. Copiare il database su un pc.
  2. Aprire il database tramite il programma *SQLite broswer* (o con un'applicazione equivalente) disponibile qui o al seguente path: `Tool/sqlitebrowser-3.4.0-win32.exe`
  3. Cliccare su `File/Export/Table as CSV file...`
  4. Selezionare la tabella e le opzioni desiderate, quindi premere `OK`.
- Per la tabella *MP Raw* è necessario procedere come indicato nella sezione 7.1.2.3.

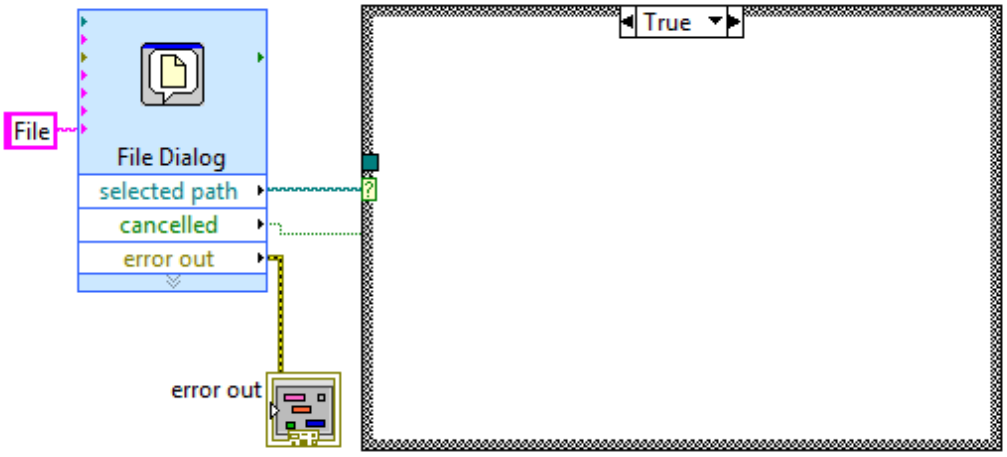
A.5 Tool utili

Applicazioni Android gratuite reperibili dal *Play store*:

- *SQLiteManager*: legge e modifica un database *SQLite*.
- *CSV Viewer*: legge un file formattato con lo stile `csv`.



(a) $FALSE$ case.



(b) $TRUE$ case.

Figura A.2: Block diagram introduttivo per leggere i dati salvati su file da *Acquire sensor data*.
File: `Codice/LabVIEW/Intro block diagram.vi`

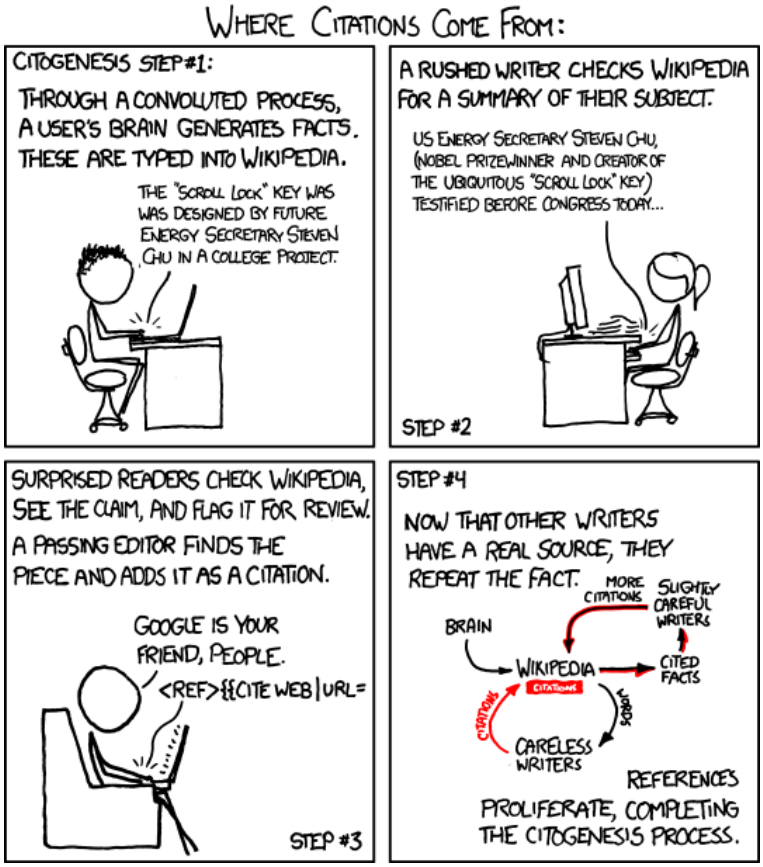
Acronimi

Fonte: dilbert website. Ultima visita: 3 dicembre 2014.

GNSS Global Navigation Satellite System . . . 2
RSS Received Signal Strengths . . . 3
TOA Time Of Arrival . . . 3
TDOA Time Difference Of Arrival . . . 3
RTOF Round Trip Of Flight . . . 3
RFID Radio Frequency Identifier Description . . . 3
MEMS Sensors MicroElectroMechanical Systems . . . 3
IMU Inertial Measurement Unit . . . 3
PDR Pedestrian Dead Reckoning . . . 2
ZUPT Zero velocity UPdaTe . . . 4
BVI Blind and Visual Impaired . . . 2
GUI Graphic User Interface . . . 13
AUI Acoustic User Interface . . . 13
VUI Vibration User Interface . . . 13
SDK Software Development Kit . . . 16
NDK Native Development Kit . . . 17
SIMD Single Instruction, Multiple Data . . . 17
JNI Java Native Interface . . . 17
IDE Integrated Development Environment . . . 16
ADT Android Developer Tools . . . 17
JDK Java Development Kit . . . 17
VCS Version Control System . . . 17

API Application Programming Interface 16
EPM Entry Point Map 22
IP Interesting Point 22
MMP Mappa Matrice di Pesi 24
MP Matrice di Pesi 24
DBMS DataBase Management System 31
E-R Entità-Relazione 27
O-O Object-Oriented 36
XML eXtensible Markup Language 32
DAO Data Access Object 36
FAQ Frequently Asked Questions 107
SoC System on Chip 121

Bibliografia



Fonte: xkcd website. Ultima visita: 3 dicembre 2014.

[1] World Health Organization, *Visual impaired and blindness*, Fact sheet N°282, ago 2014. Indirizzo: http://www.who.int/mediacentre/factsheets/fs282/en/, ultima visita il 21-11-2014 (cit. a p. 1).

[2] International Classification of Diseases, *H53-H54: Visual disturbances and blindness, 10th revision (update 2015)*. Indirizzo: http://apps.who.int/classifications/icd10/browse/2015/en#/H54.9, ultima visita il 21-11-2014 (cit. a p. 1).

[3] M. Hersh e J. M.A., "Electronic Travel Aids: An Assessment Learning Objectives", in *Assistive Technology for Visually Impaired and Blind People*, Springer, 2008, cap. 9, pp. 289–321 (cit. a p. 2).

[4] N. Fallah, I. Apostolopoulos, K. Bekris e F. E., "Indoor Human Navigation Systems: A Survey", *Interacting with Computers*, vol. 25, n. 1, pp. 21–33, feb. 2013 (cit. a p. 2).

[5] H. Liu, H. Darabi, P. Banerjee e J. Liu, "Survey of Wireless Indoor Positioning Techniques and Systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 37, n. 6, pp. 1067–1080, nov. 2007 (cit. a p. 2).

[6] A. J.N. e P. L.C., "Sensor Network Localization via Received Signal Strength Measurements with Directional Antennas", in *In Proceedings of the Allerton Conference on Communication, Control, and Computing*, 2004, pp. 1861–1870 (cit. a p. 3).

[7] N. Patwari, A. Hero, M. Perkins, N. Correal e R. O'Dea, "Relative location estimation in wireless sensor networks", *IEEE Transactions on Signal Processing*, vol. 51, n. 8, pp. 2137–2148, ago. 2003 (cit. a p. 3).

[8] R. Fontana, E. Richley e J. Barney, "Commercialization of an ultra wideband precision asset location system", in *IEEE Conference on Ultra Wideband Systems and Technologies*, nov. 2003, pp. 369–373 (cit. a p. 3).

[9] G. C. e J. T., "Indoor Positioning using Time Difference of Arrival between Multipath Components", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, ott. 2013 (cit. a p. 3).

[10] A. Günther e C. Hoene, "Measuring Round Trip Times to Determine the Distance Between WLAN Nodes", in *In Proceedings of Networking*, Springer-Verlag, 2005, pp. 768–779 (cit. a p. 3).

[11] D. Niculescu e B. Nath, "Ad hoc positioning system (APS) using AOA", in *International Conference of the IEEE Computer and Communications (INFOCOM)*, vol. 3, mar. 2003, pp. 1734–1743 (cit. a p. 3).

[12] E. D'Atri, C. Medaglia, A. Serbanati e U. Ceipidor, "A system to aid blind people in the mobility: A usability test and its results", in *International Conference on Systems (ICONS)*, apr. 2007 (cit. a p. 3).

[13] B. Ding, H. Yuan, X. Zang e L. Jiang, "The Research on Blind Navigation System Based on RFID", in *International Conference on Wireless Communications, Networking and Mobile Computing (WiCom)*, set. 2007, pp. 2058–2061 (cit. a p. 3).

[14] H. Huang, G. Gartner, M. Schmidt e Y. Li, "Smart Environment for Ubiquitous Indoor Navigation", in *International Conference on New Trends in Information and Service Science (NISS)*, giu. 2009, pp. 176–180 (cit. a p. 3).

[15] T. Aguilera, J. Paredes, F. Alvarez, J. Suarez e A. Hernandez, "Acoustic local positioning system using an iOS device", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, ott. 2013, pp. 1–8 (cit. a p. 3).

[16] J. Baus, A. Krüger e W. Wahlster, "A Resource-adaptive Mobile Navigation System", in *International Conference on Intelligent User Interfaces*, ser. IUI '02, ACM, 2002, pp. 15–22 (cit. a p. 3).

[17] G. Retscher e M. Thienelt, "NAVIO – A Navigation and Guidance Service for Pedestrians", *Journal of Global Positioning Systems*, vol. 3, n. 1-2, pp. 208–217, 2004 (cit. a p. 3).

[18] W. Elloumi, K. Guissous, A. Chetouani, R. Canals, R. Leconge, B. Emile e S. Treuillet, "Indoor navigation assistance with a Smartphone camera based on vanishing points", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, ott. 2013 (cit. a p. 3).

[19] Y. Chon e H. Cha, "LifeMap: A Smartphone-Based Context Provider for Location-Based Services", *IEEE Pervasive Computing*, vol. 10, n. 2, pp. 58–67, apr. 2011 (cit. alle pp. 3, 76).

[20] L. Ojeda e J. Borenstein, "Personal Dead-reckoning System for GPS-denied Environments", in *IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR)*, set. 2007, pp. 1–6 (cit. alle pp. 3, 4).

[21] H. Benzerrouk, A. Nebylov, S. H. e C. P., "MEMS IMU/ZUPT Based Cubature Kalman Filter Applied to Pedestrian Navigation System", in *International Electronic Conference on Sensors and Applications*, giu. 2014 (cit. alle pp. 3, 4).

[22] E. Foxlin, "Pedestrian tracking with shoe-mounted inertial sensors", *IEEE Computer Graphics and Applications*, vol. 25, n. 6, pp. 38–46, nov. 2005 (cit. a p. 3).

[23] S.-E. Kim, Y. Kim, J. Yoon e E. S. Kim, "Indoor positioning system using geomagnetic anomalies for smartphones", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, nov. 2012, pp. 1–5 (cit. alle pp. 3, 5).

[24] I. Skog, J.-O. Nilsson, D. Zachariah e P. Handel, "Fusing the information from two navigation systems using an upper bound on their maximum spatial separation", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, nov. 2012, pp. 1–5 (cit. alle pp. 3, 4).

[25] J. Link, P. Smith, N. Viol e K. Wehrle, "FootPath: Accurate map-based indoor navigation using smartphones", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, set. 2011, pp. 1–8 (cit. alle pp. 3, 5, 76).

[26] C. Ascher, S. Werling, G. Trommer, L. Zwirello, C. Hansmann e T. Zwick, "Radio-asissted inertial navigation system by tightly coupled sensor data fusion: Experimental results", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, nov. 2012, pp. 1–7 (cit. a p. 4).

[27] T. Gallagher, E. Wise, B. Li, A. Dempster, C. Rizos e E. Ramsey-Stewart, "Indoor positioning system based on sensor fusion for the Blind and Visually Impaired", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, nov. 2012, pp. 1–9 (cit. alle pp. 4, 5).

[28] M. Romanovas, V. Goridko, A. Al-Jawad, M. Schwaab, M. Traechtler, L. Klingbeil e Y. Manoli, "A study on indoor pedestrian localization algorithms with foot-mounted sensors", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, nov. 2012, pp. 1–10 (cit. a p. 4).

[29] J. Mendez, J. Lorenzo e M. Castrillon, "Comparative Performance of GPU, SIMD and OpenMP Systems for Raw Template Matching in Computer Vision", in *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, 2011, pp. 9–16 (cit. a p. 5).

[30] J. Qian, J. Ma, R. Ying, P. Liu e L. Pei, "An improved indoor localization method using smartphone inertial sensors", in *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, ott. 2013, pp. 1–7 (cit. alle pp. 5, 76, 120).

[31] IDC, *Smartphone OS Market Share, Q2 2014*, 2014. Indirizzo: http://www.idc.com/prodserv/smartphone-os-market-share.jsp, ultima visita il 24-11-2014 (cit. a p. 5).

[32] C. Ghezzi, M. Jazayeri e D. Mandrioli, *Ingegneria del software: fondamenti e principi*, 2ª ed. Pearson Education Italia, 2004 (cit. a p. 6).

[33] "IEEE Standard Glossary of Software Engineering Terminology", *IEEE Std 610*, dic. 1990 (cit. alle pp. 7, 12).

[34] "Systems and software engineering – Vocabulary", *ISO/IEC/IEEE 24765*, dic. 2010 (cit. a p. 7).

[35] I. Sommerville, *Software Engineering*, 9ª ed. Pearson/Addison-Wesley, 2011 (cit. a p. 7).

[36] R. Pressman, *Software Engineering: A Practitioner's Approach*, 7ª ed. McGraw-Hill, 2010 (cit. a p. 7).

[37] W. W. Royce, "Managing the development of large software systems: concepts and techniques", in *Technical Papers of Western Electronic Show and Convention (WesCon)*, ago. 1970, pp. 1–9, reprinted in *Proceedings of the 9th ICSE*, pp. 328–338, mar. 1987 (cit. a p. 8).

[38] H. D. Mills, D. O'Neill, R. C. Linger, M. Dyer e Q. R.E., "The Management of Software Engineering: Principles of Software Engineering", *IBM System Journal*, vol. 19, n. 4, pp. 414–477, dic. 1980 (cit. a p. 9).

[39] M. M. Lehman e L. A. Belady, *Program Evolution: Processes of Software Change*. Academic Press Professional, 1985 (cit. a p. 9).

[40] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland e D. Thomas, *Manifesto for Agile Software Development*, 2001. Indirizzo: http://www.agilemanifesto.org/ (cit. a p. 11).

[41] A. S. Tanenbaum e M. v. Steen, *Distributed Systems: Principles and Paradigms*, 2ª ed. Prentice-Hall, 2006, pp. 36–43 (cit. a p. 15).

[42] P. Pin-Shan Chen, "The Entity-Relationship Model: Toward a Unified View of Data", *ACM Transactions on Database Systems*, vol. 1, pp. 9–36, 1976 (cit. a p. 27).

[43] P. Atzeni, S. Ceri, S. Paraboschi e R. Torlone, *Basi di dati. Modelli e linguaggi di interrogazione*, 3ª ed. McGraw-Hill, 2009 (cit. alle pp. 27, 32).

[44] R. Elmasri, S. Navathe e S. Castano, *Sistemi di basi di dati. Fondamenti*, 6ª ed. Pearson, 2011 (cit. a p. 31).

[45] E. F. Codd, "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks", *IBM Research Report*, vol. RJ599, 1969 (cit. a p. 32).

[46] H. Kim, N. Agrawal e C. Ungureanu, "Revisiting Storage for Smartphones", *ACM Transactions on Storage*, vol. 8, n. 4, pp. 1–14, dic. 2012 (cit. a p. 42).

[47] S. Microsystems, *Core J2EE Patterns - Data Access Object*, 2002. Indirizzo: http://www.oracle.com/technetwork/java/dataaccessobject-138824.html, ultima visita il 01-09-2014 (cit. a p. 45).

[48] E. Gamma, R. Helm, R. Johnson e J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman, 1995, pp. 99–109 (cit. a p. 45).

[49] S. Miyazaki, "Long-term unrestrained measurement of stride length and walking velocity utilizing a piezoelectric gyroscope", *IEEE Transactions on Biomedical Engineering*, vol. 44, n. 8, pp. 753–759, ago. 1997 (cit. a p. 71).

[50] B. Picasso, *Fondamenti di Meccanica e Biomeccanica: Meccanica dei corpi rigidi articolati*. Springer, 2013, pp. 23–66 (cit. a p. 85).

[51] P. J. Schneider e D. Eberly, *Geometric Tools for Computer Graphics*. Elsevier Science, 2002, pp. 847–852 (cit. a p. 85).

[52] D. Eberly, *Euler Angle Formulas*, 1999. Indirizzo: http://www.geometrictools.com/Documentation/EulerAngles.pdf, ultima visita: 13/11/14 (cit. a p. 85).

[53] E. Martin, "Novel method for stride length estimation with body area network accelerometers", in *International Conference on Biomedical Wireless Technologies, Networks, and Sensing Systems (BioWireleSS)*, IEEE, gen. 2011, pp. 79–82 (cit. alle pp. 89, 120).

[54] D. N. Rômulo, M. Aline S. e D. Victor Z., "Usual gait speed assessment in middle-aged and elderly Brazilian subjects", *Revista Brasileira de Fisioterapia*, vol. 15, n. 2, pp. 117–122, mar. 2011 (cit. a p. 89).

[55] S. J. Russell e P. Norvig, *Artificial Intelligence: A Modern Approach*, 3ª ed. Pearson Education, 2010, pp. 64–112 (cit. a p. 106).

[56] H. Weinberg, "Using the ADXL202 in Pedometer and Personal Navigation Applications", *Analog Devices AN-602 application Note*, 2002 (cit. alle pp. 119, 120).

[57] D. Alvarez, R. Gonzalez, A. Lopez e J. Alvarez, "Comparison of Step Length Estimators from Weareable Accelerometer Devices", in *International Conference on Engineering in Medicine and Biology (EMBS)*, ago. 2006, pp. 5964–5967 (cit. a p. 120).

[58] L. Klingbeil e T. Wark, "A Wireless Sensor Network for Real-Time Indoor Localisation and Motion Monitoring", in *International Conference on Information Processing in Sensor Networks (IPSN)*, apr. 2008, pp. 39–50 (cit. a p. 120).

[59] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu e F. Zhao, "A Reliable and Accurate Indoor Localization Method Using Phone Inertial Sensors", in *International Conference on Ubiquitous Computing (UbiComp)*, ACM, 2012, pp. 421–430 (cit. a p. 120).

[60] *LIS3DH: MEMS digital output motion sensor ultra low-power high performance 3-axes "nano" accelerometer*, STMicroelectronics, mag. 2010, pp. 1, 30 (cit. a p. 121).

[61] *AK8963 3-axis Electronic Compass*, Asahi Kasei MicroDevice Corporation (AKM), feb. 2013, pp. 13, 15 (cit. a p. 122).