Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

MONITORAGGIO DELLE ATTIVITÀ FISICHE DI PAZIENTI CARDIOPATICI MEDIANTE ANALISI DEI SENSORI EMBEDDED DELLO SMARTPHONE

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Federica Dell'Orletta

Correlatore:
Dott.
Ivan Corazza

Sessione II
Anno Accademico 2019/2020

Abstract

La fase fondamentale per il controllo dello stato di salute di un paziente con scompenso cardiaco grave o con dispositivo assistenziale LVAD, è la fase di follow up medico. Si tratta di una fase di controllo continuo e programmato del paziente.
Per stabile una pratica comune per tutte le tipologie di pazienti e instaurare una comunicazione continua tra paziente e medico, si porta avanti la realizzazione di un'applicazione Android.
Quello che si farà in questo lavoro è determinare un sistema di monitoraggio dell'attività motoria svolta dall'utente, concentrandoci sull'attività di deambulazione e sul dispendio calorico giornaliero.
Analizzeremo e discuteremo i diversi algoritmi applicabili per realizzare i nostri obiettivi, incentrando il lavoro sull'utilizzo dell'accelerometro e sulla computazione dinamica della lunghezza del passo.

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Introduzione

Il lavoro portato avanti all'interno di questa tesi, vuole ampliare un'applicazione precedentemente implementata per il controllo a distanza di pazienti in fase di follow up medico con scompenso cardiaco o portatori di LVAD.

Il campo d'interesse di questa tesi è lo smarthealth, ovvero l'introduzione delle nuove tecnologie, quali le intelligenze artificiali e l'internet of things, all'interno del sistema sanitario. Ciò rappresenta una concreta risposta alle statistiche OCSE, che riportano che entro il 2050 l'invecchiamento della popolazione mondiale sarà circa del 21%, e tale situazione porterà ad un aumento dei soggetti colpiti da malattie croniche.
Grazie all'introduzione delle nuove tecnologie dotate di sensori, diventerà più facile individuare situazioni di pericolo per un paziente e curarlo, evitando il verificarsi di situazioni critiche. Diversi studi, infatti, dimostrano come, l'utilizzo delle intelligenze artificiali o dell'internet of things, possa prevenire il verificarsi di un evento critico come un attacco di cuore [24].
Lo sviluppo dello smarthealth è reso ovviamente possibile grazie alla diffusione degli smartphone e alla creazione di applicazioni nel mondo del mobile health. Tali soluzioni consentono al paziente di visualizzare i suoi dati sanitari direttamente sul suo smartphone e di permettere la visualizzazione ed il monitoraggio remoto anche al medico curante.
In questa tesi, ci siamo concentrati su tecnologie abilitanti il monitoraggio remoto di pazienti affetti da scompensi cardiaci.

Le malattie cardiache corrispondono alla principale causa di morte al mondo, causando circa un decesso su quattro [8].
Le patologie cardiache, sono inoltre classificabili come il disturbo cronico con più larga diffusione. Tali patologie, qualsiasi area del cuore colpiscano, portano a gravi conseguenze e inibizioni, che nella maggior parte dei pazienti si manifestano a livello sistematico [15].

Una delle patologie cardiache più diffuse è lo scompenso cardiaco. Si tratta di una malattia che si compone di una serie di sintomi e manifestazioni fisiche dovute all'incapacità del cuore di assolvere alle normali funzioni contrattale di pompa e di soddisfare il corretto apporto di sangue a tutti gli organi. Nella maggior parte dei casi, i pazienti con questa patologia presentano un'alterazione denominata "blocco di branca sinistra" (BBS) [19].
Il policlinico bolognese Sant'Orsola-Malpighi si occupa della cura e dello studio di questa patologia. L'ospedale nel 2017 ha portato avanti una campagna di raccolta dei dati di pazienti con scompenso cardiaco e ha intrapreso uno studio sulla qualità della vita di tali pazienti con l'unità operativa di psicologia clinica e sperimentale. Questo ha portato ha una suddivisione dei pazienti in 2 blocchi: pazienti con scompenso cardiaco avanzato e pazienti LVAD (Left Ventricular Assist Device) - è un dispositivo di assistenza ventricolare sinistra che sopperisce ad alcune funzionalità che il muscolo non può svolgere in quanto compromesso [3].
Nonostante la terapia meno invasiva i pazienti con scompenso cardiaco avanzato presentano un profilo psicologico peggiore, rispetto ai pazienti LVAD, che invece hanno subito un intervento chirurgico [17].

I pazienti LVAD, date le medicazioni settimanali e il monitoraggio da remoto grazie al dispositivo che indossano, hanno una maggior coscienza sul loro stato di salute.
La complicata situazione psicologica dei pazienti con scompenso cardiaco grave, ha portato il policlinico Sant'Orsola-Malpighi, a voler creare un protocollo

standard per seguire le due tipologie di pazienti in modo simile. Per questo è stato proposto lo sviluppo di un'applicazione mobile, incentivata anche da una popolazione giovane di pazienti del reparto, per lo più di età inferiore ai 65 anni [14].

L'obiettivo di questa tesi sarà di ampliare le funzionalità dell'applicazione precedentemente proposta, inserendo un sistema di monitoraggio del movimento dell'utente e del calcolo del suo dispendio calorico giornaliero.
Per far ciò abbiamo discusso brevemente perchè la scelta di implementare il sistema in Android risulti maggiormente fruibile rispetto a realizzarlo in IOS.
L'ampia letteratura, in materia di monitoraggio del movimento, ci ha permesso di disaminare largamente i differenti algoritmi di computazione del movimento e analizzare quali siano i sensori di raccolta dei dati, accelerometro e pedometro, e selezionare quello che risponde meglio alle nostre necessità.
Per realizzare il secondo obbiettivo centrale della tesi, cioè calcolare il dispendio calorico dellâutente. Si è ritenuto fondamentale ampliare il discorso di come la stima della lunghezza del passo sia il punto focale per il calcolo del dispendio calorico, per questo ci si è focalizzati sul discutere il perchè il calcolo dinamico della lunghezza del passo dia risultati più stabili rispetto al calcolo statico della lunghezza del passo.
La fase di realizzazione del sistema ha messo in luce come il sensore di accelerazione sia la scelta migliore per rispondere alle nostre necessità, perchÃ¨ ci permette sia di computare il movimento, con il relativo conteggio del numero di passi eseguiti dallâutente, e sia di stimare dinamicamente la lunghezza del passo.
Per rafforzare la nostra scelta e controllare che esse siano state coerenti con lâobbiettivo implicito della tesi, cioè quello di creare un sistema di monitoraggio stabile. Per far ciò sono stati eseguiti diversi test per la raccolta dei dati prodotti dallâapplicazione allo scopo di eseguire unâanalisi statistica, che hanno confermato la stabilità dellâalgoritmo adottato e di come la stima

dinamica del passo risulti decisamente più efficiente, sia nel calcolo della kcal bruciate che nel calcolo della distanza percorsa, rispetto alla stima statica.
La tesi è dunque organizzata in 5 capitoli:

- Capitolo 1: (Stato dell'arte) in questo capitolo accenneremo brevemente la struttura delle applicazioni Android, concentrandoci sullo sviluppo del settore delle fitness app e delle mHealth application. Continueremo introducendo i sensori disponibili all'interno degli smartphone Android, concentrandoci sull'accelerometro ed il pedometro e guardando al loro utilizzo all'interno delle fitness app. Discuteremo, infine, ampiamente come avviene il calcolo delle kcal bruciate, facendo riferimento ai metodi di stima della lunghezza del passo e tratteremo largamente gli algoritmi più importanti per il conteggio del numero di passi eseguiti dall'utente.

- Capitolo 2: (Progettazione) in questo capitolo tratteremo gli obiettivi fissati per la realizzazione della tesi e verranno poi definite le specifiche relative alle funzionalità. Discuteremo l'architettura dell'applicazione, precedentemente implementata, e le funzionalità che sono state aggiunte grazie alla realizzazione di questo lavoro di tesi.

- Capitolo 3: (Implementazione) in questo capitolo accenneremo brevemente alle tecnologie utilizzate per la realizzazione dell'applicazione. Per poi discutere ampiamente la realizzazione dell'interfaccia utente e del database utilizzato. Per poi, infine, discutere l'implementazione dell'algoritmo scelto per la realizzazione del contapassi, e come avviene il calcolo del dispendio calorico utilizzando la stima dinamica della lunghezza del passo.

- Capitolo 4: (Validazione) in questo capitolo discuteremo i test effettuati per verificare la stabilità dell'applicazione implementata. Verrà messo a confronto l'algoritmo implementato per la realizzazione del contapassi con lo step counter di Android ed un altro algoritmo che utilizza l'accelerometro. Per stimarne l'attendibilità sono stati messi in pratica

due tipologie di test, quali: quello sulla velocità di camminata (bassa, media, alta) e quello sul conteggio del numero di passi (100, 200, 500 passi). Infine, si è realizzato il test per dimostrare l'affidabilità della stima dinamica della lunghezza del passo rispetto al calcolo con stima statica del passo.

- Capitolo 5: (Conclusioni e sviluppi futuri) in questo capitolo riepiloghe-remo brevemente quanto trattato all'interno della tesi e introdurremo dei possibili sviluppi futuri per l'applicazione.

Capitolo 1

Stato dell'arte

1.1 Android e le applicazioni mobili

Uno dei processi che ha segnato di più la vita degli abitanti di tutto il mondo, è stato il graduale passaggio, negli ultimi dieci anni, da telefono, visto come strumento per chiamare ed inviare messaggi, all'avvento, grazie alla diffusione di Internet, degli smartphone, strumenti che data la loro potenza di calcolo mettono a disposizione innumerevoli funzionalità per l'utente. Segnando un radicale rinnovamento nella mentalità della popolazione mondiale. Infatti, il settore della telefonia mobile, nonostante la crisi economica generale a livello globale, mostra un fatturato di circa 714,96 miliardi di dollari tra il 2019 e il 2020 e si prospetta un settore prosperoso, mostrando un trend positivo, nel previsioni di crescita del fatturato per i prossimi 5 anni, superiori al 90% [25].

Android e IOS risultano essere i sistemi operativi, nel campo degli smartphone, più diffusi a livello globale.
La loro supremazia dal 2009 ad oggi non risulta scalfita, nonostante l'ingresso all'interno del settore di nuovi competitor. Guardando ai dati di espansione di entrambi all'interno del mercato globale, Android mostra una diffusione superiore rispetto ad IOS, data la sua forte presenza, oltre che nei mercati dei

Paese occidentali, anche in quelli delle economie emergenti. Ad oggi, infatti, il "Google Play Store", conta circa 3.3 bilioni di utenti di qualsiasi parte del mondo e un catalogo di applicazioni di circa 2.714.499, numeri destinati ad aumentare per la fine del 2022 [22].

Un'applicazione non è altro che un software, scritto in un particolare linguaggio di programmazione, come Java o Kotlin per le app Android e in Swift per IOS. La sua natura è similare a quella di un software per PC ma esse vengono concepite con un design differente per migliorarne l'usabilità e la user experience, grazie soprattutto allo sviluppo di Material Design.

Il settore di sviluppo delle applicazioni mobili è vario, dai sistemi di messaggistica (come il più conosciuto WhatsApp), alle app in campo economico come quelle per l'home banking, passando per le applicazioni in campo fitness, fortemente importanti anche per lo sviluppo delle app dette mHealth ovvero quelle sviluppate in campo biomedico.

1.1.1 La struttura di Android

Android è una piattaforma che nasce nei primi anni 2000 per dispositivi mobili touchscreen, è basato su sistema Linux e include un sistema operativo e un middleware (tutto ciò che sta tra sistema operativo e app).

La struttura di Android è composta da più layer che sono:

- Kernel Linux: garantisce sicurezza e affidabilità, in più il fatto che sia basato su Linux garantisce un elevato grado di portabilità del sistema su un qualsiasi dispositivo hardware.

- HAL: la sigla sta per "Hardware Abstraction Layer", è il layer che permette al sistema operativo di interfacciarsi con le API. Il suo lavoro

è di tradurre le chiamate delle API in linguaggio macchina così che siano compressibili dal sistema operativo.

- Librerie Native: sono le librerie contenenti i principali servizi presenti nel sistema Android. Le NDK sono scritte in C/C++, un esempio sono le librerie base per la gestione della grafica.

- Android Runtime: è un layer che fa parte del middleware serve per interpretare e compilare le applicazioni scritte in Java, non è altro che un'implementazione della Java Virtual Machine. Si tratta di un layer inserito in seguito allo sviluppo di Android infatti è stato inserito a partire dalla versione 4.4 ed è diventato obbligatorio a partire dalla versione 5.

- Java API Framework: sono tutte le API messe a disposizione dell'implementatore, si tratta di una serie di funzionalità necessarie per lo sviluppo delle applicazioni. La maggior parte dei moduli che compongono questo livello sono detti manager perchè permettono la gestione del comportamento di una particolare area dell'applicazione, come ad esempio l'activity manager che permette di gestire il life cycle di un'activity.

- System Apps: è il layer più alto del sistema Android, è quello che va ad interfacciarsi con l'utente. Contiene tutte le applicazioni principali del sistema operativo, come ad esempio la rubrica telefonica.

1.1.2 Le fitness app

Tra i settori più in voga negli ultimi anni troviamo quello del fitness e della cura persona.
Per rispondere ai bisogni di molti utenti si è pensato di creare delle applicazioni per il monitoraggio e il miglioramento dell'attività fisica.

Figura 1.1: La struttura del sistema Android

Nel 2015 il numero di app relative a questo settore risultava essere di circa 165.000 [5]. Negli ultimi anni le applicazioni che promuovono uno stile di vita sano e consigli sullo svolgimento di esercizi per restare in forma è quasi raddoppiato. L'enorme sviluppo di questo ambito è dovuto, sicuramento, alla larga diffusione di dispositivi, come gli smartwatch, che permettono di monitorare costantemente il movimento e i parametri vitali degli utenti (ne è un esempio il battito cardiaco).
Il monitoraggio costante dell'attività motoria, utilizzando solo lo smartphone, avviene attraverso l'utilizzo dei sensori embedded allo smartphone e di particolari algoritmi di riconoscimento del movimento.

Ogni dispositivo Android, al momento dell'acquisto, trova installato su di esso un'applicazione fitness, ovvero quella della casa madre, come ad esempio Samsung Health, presente su tutti i dispositivi Samsung di qualsiasi fascia di prezzo.
Mentre, tra le applicazioni scaricabili dallo store, la più diffusa è troviamo "MyFitness Pal" [26]. App famosissima per le sue numerose funzionalità, sponsorizzata dall'azienda produttrice di abbigliamento sportivo "Under Armound", che oltre a mettere a disposizione il monitoraggio del movimento dell'utente, gli permette di contare le calorie che ha consumato durante i pasti, di salvare l'attività motoria svolta come ad esempio registrare che l'utente ha fatto 30 minuti di sala pesi in palestra, producendo un calcolo approssimato delle calorie che ha bruciato.

1.1.3 Le mHealth app

Lo sviluppo delle app in campo fitness ha portato alcune aziende sviluppatrici a migliorare la propria tecnologia per ampliare le funzionalità del proprio software, dando così il via al settore del mHealth.

Con le applicazioni mHealth, abbreviazione di "Mobile Health", si fa riferimento alla "digitalizzazione" dei dati relativi allo stato di salute generale

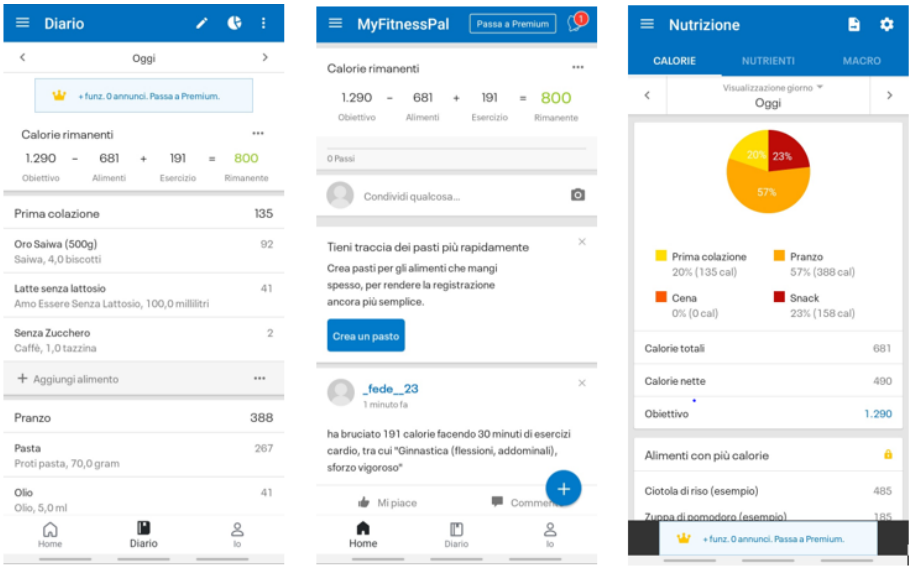

Figura 1.2: Esempio di come si presenta l'app MyFitnessPal

1. Rappresenta il diario giornaliero dell'utente dove può inserire l'attività svolta e il cibo che ha consumato.

2. È la home dell'app dove viene mostrato le kcal assunte e consumate, in più vengono mostrate anche delle note da parte dell'utente. 3. Possiamo vedere graficamente l'andamento dal punto di vista calorico della giornata.

dell'utente. Attraverso l'utilizzo di queste applicazioni gli utenti sono in grado di tener traccia dei propri dati sanitari (come il battito cardiaco) e, in caso di anomalie, rilevare le problematiche in modo semplice per esporle al proprio medico.

Nel 2017, le applicazioni mHealth presenti negli store di Android e IOS si aggiravano attorno alle 318.000 e negli anni sono aumentate a dismisura. Differenti categorie di monitoraggio sanitario si sono sviluppate all'interno del settore, dal monitorare lo stato di avanzamento della gravidanza, a tener sotto controllo la situazione di pazienti con malattie croniche come il diabete, per passare anche ad un aiuto concreto contro malattie più insidiose come possono essere i disturbi alimentari e le malattie mentali [16].

Lo sviluppo di queste applicazioni, negli anni avvenire, punta a ridurre i costi della sanità, cercando di diminuire il numero di visite mediche e di trattamenti sanitari non necessari, aumentando i contatti a distanza tra medico e paziente. Rafforzare lo sviluppo della telemedicina, è fondamentale per il ridisegno del sistema strutturare della sanità.

1.2 I sensori Android

Gli smartphone che ci ritroviamo ad utilizzare tutti i giorni sono delle vere e proprie macchine sofisticate, che hanno attraversato un processo di evoluzione notevole nell'ultimo decennio.
Oggi gli smartphone sono in grado di controllare il nostro movimento, il nostro battito cardiaco e tanto altro. Tutto ciò è permesso, grazie alla combinazione dei dati rilevati dai sensori hardware presenti all'interno dello smartphone, combinati con particolari algoritmi di computazione di tali dati.

I sensori fisici presenti all'interno dello smartphone sono numerosi. Un dispositivo mobile, di qualsiasi dimensione, conta circa una cinquantina di sensori

al suo interno. Non tutti gli smartphone sono uguali, a seconda della casa produttrice essi potrebbero presentare delle differenze interne al sensore inteso come componente hardware o differenze nelle capabilities, ma in linea generale la loro gestione è similare.

Ogni sensore viene utilizzato per compiere un'analisi dello stato e del comportamento dell'utente. Tra i sensori più conosciuto troviamo il giroscopio che fornisce importanti informazioni sulla direzione e sull'inclinazione dello smartphone, oppure, il sensore di prossimità che rileva se il telefono si trova in prossimità di oggetti esterni (come il nostro volto quando effettuiamo una chiamata) e va a disabilitare l'input del touch screen per evitare input indesiderati.

In aggiunta ai sensori finisci, troviamo anche una serie di sensori detti virtuali, come il sensore di gravità e il sensore i rotazione. Tali sensori vanno a computare i dati dai sensori fisici per fornire all'utilizzatore dati aggregati più semplici da analizzare.

1.2.1 Utilizzo dei dati raccolti dai sensori

L'utilizzo dei dati prodotti dai sensori è reso possibile grazie all'interazione del sistema operativo.
L'interazione avviene mediante una sorta di "comunicazione" instaurata con il sensore tramite lo sfruttamento della classe "Sensor Manager" che permette di attivare e mettere in pausa un sensore a seconda dell'uso per evitare inutili dispendi di batteria.
Tale comunicazione viene poi raffinata attraverso l'utilizzo della classe Sensor che rappresenta il sensore a livello software. Infine, ultima classe fondamentale da prendere in considerazione è la 'Sensor Event' che rappresenta un evento e contiene le informazioni relative al sensore, come ad esempio la sua accuratezza.

Tutte le classi dell'applicazione che al loro interno utilizzano un sensore devono implementano l'interfaccia "Sensor Event Listener" necessario per "raccogliere" le notifiche inviate dal SensorManager quando vi sono nuovi dati dal sensore.

1.2.2 L'utilizzo dei sensori nelle fitness app

Nel paragrafo precedente si è fatto riferimento alla diffusione delle app per il fitness e di quelle in campo biomedicale.
Queste applicazioni per migliorare i servizi che forniscono utilizzano sistemi di monitoraggio periodici del movimento dell'utente, ciò è reso possibile grazie allo sfruttamento, dei sopracitati, sensori messi a disposizione dagli smartphone.

Il monitoraggio del movimento dell'utente può avvenire in diversi modi, tralasciamo momentaneamente gli algoritmi utilizzabili per l'analisi dei dati prodotti da un sensore di cui parleremo in seguito. Possiamo dire che ogni implementazione dell'analisi del movimento può scegliere, tra due sensori, e prediligere quello che ritiene migliore per la raccolta dei dati e la loro computazione.
La scelta del sensore avviene tra l'accelerometro e il pedometro, entrambi presentano vantaggi e svantaggi.

Accellerometro

L'accelerometro, da come suggerisce il nome, è il sensore che misura e rileva l'accelerazione dello smartphone.

L'utilizzo comune di questo sensore è il riconoscimento dell'orientamento del telefono per passare in maniera rapida ed automatica da modalità portrait a modalità landascape.

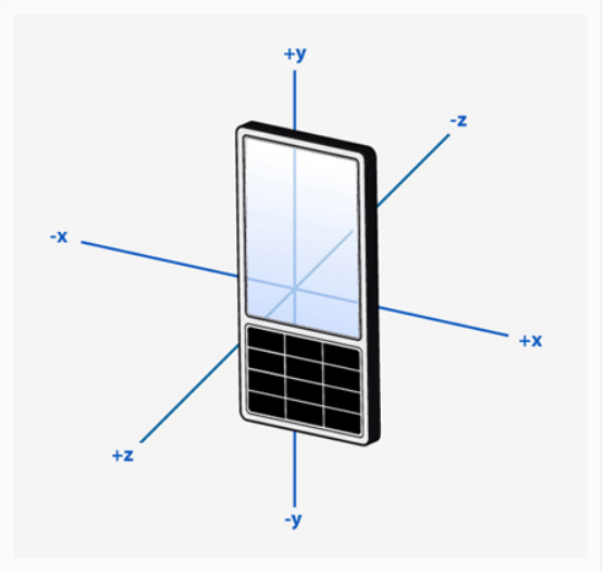

Figura 1.3: Rappresentazione del piano tridimensionale dell'accelerometro

L'accelerometro permette, quindi di estrarre la posizione nello spazio dello smartphone.
La posizione del device viene presa su un piano tridimensionale di asse x, y, z, ove i tre assi indicano rispettivamente il movimento in orizzontale , il movimento verticale e il movimento dello smartphone fuori dal piano x,y [6], come è possibile vedere anche in figura 1.3.

Questo sensore, dal punto di vista implementativo restituisce attraverso l'utilizzo del SensorEvent i tre valori degli assi (x,y,z), che poi verranno computati a seconda delle necessità.
Un sensore di questo tipo in sostanza permette di misurare l'accellerazione applicata ad un dispositivo (Ad). Concettualmente lo fa misurando la forza applicata (Fs) al sensore stesso, tramite la formula: [21]

$$Ad = \sum Fs/massa$$

Usare questo sensore nella rilevazione del movimento, risulta essere una pratica molto utilizzata. Tramite la computazione dei dati ottenuti è infatti

possibile arrivare a rilevare il numero di passi compiuti dall'utente.

Pedometro

Il pedometro è un sensore di utilizzo più recente rispetto all'accelerometro.
Infatti, se il sensore di accelerazione era disponibile già nei primissimi smartphone, il pedometro è stato introdotto in seguito, dagli smartphone che di fabbrica montavano un sistema operativo Android 4.4 o superiori.

La differenza sostanziale tra accelerometro e pedometro è delineata dalla natuara virtuale di quest'ultimo. Infatti, quello che va a fare il pedometro è leggere i dati dell'accelerometro e computarli, tramite l'utilizzo di un particolare algoritmo interno.
In sostanza, in questo caso dal punto di vista dell'utilizzo implementativo non viene preso in considerazione nessun piano tridimensione, ma a partire da Android 4.4 e l'API 19 sono state introdotte due nuove classi di sensori per il monitoraggio dei passi, che sono:

- Lo step detector che attiva un evento ogni volta che l' utente esegue un passo, restituendo come valore 1.0 e il timestamp di quando si è verificato il passo. La constatazione dell'evento corrisponde a quando il piede ha toccato il suolo, infatti questo movimento causa una forte variazione dell'accelerazione dello smartphone. La funzionalità di questo sensore è quindi quella di generare un evento ogni volta che viene eseguito un passo.

- Lo step counter attiva anche un evento all'esecuzione di ogni passo, ma ci fornisce il numero di passi totale che l'utente ha eseguito dal momento della registrazione di questo sensore sull'applicazione. Si tratta di un sensore implementato nel hardware a bassa potenza. [20]

Dobbiamo però tener presente che questi sensori non presentano sempre gli stessi risultati, infatti lo step counter ha una latenza maggiore allo step detector, questo perchè l'algoritmo del counter va ad eseguire più computazioni per eliminare i falsi positivi.
Possiamo quindi evidenziare che, anche se il counter potrebbe essere più lento, esso risulta essere più affidabile e in aggiunta un altro punto a suo favore è il fornire come risposta il numero di passi aggregati.

La raccolta di dati da questi sensori è resa possibile solo se vengono dichiarati gli user permission di "ACTIVITY RECOGNITION" nel manifest dell'applicazione.

1.3 Il calcolo del dispendio calorico

Punto focale del monitoraggio del movimento dell'utente e di estrema importanza per questo lavoro di tesi, oltre che la scelta del sensore da utilizzare, sta nel capire quale algoritmo di conteggio del passo sia il migliore da applicare.
In realtà la scelta fondamentale sta nel cercare l'algoritmo che risponda nel modo migliore alle nostre esigenze di calcolo della lunghezza del passo.

Sia il calcolo della lunghezza del passo, che il conteggio del numero di passi percorsi dall'utilizzatore risultano essere molto importanti, in quanto ci aiutano a capire, oltre che la distanza che l'utente ha percorso nel corso della giornata, anche il suo dispendio calorico.
Di fatti, nelle fitness app il calcolo delle calorie bruciate da un utente è un dato molto importante da misurare e da prendere in considerazione.
La formula a cui faremo riferimento per determinare le calorie bruciate è stata elaborata nel 1938 dal Prof. Rodolfo Margaria ed è così definita [4]:

$$kcal = 0.5kcal * kgpesocorporeo * kmpercorsi.$$

Si tratta di un'espressione che ovviamente non è precisa al 100% in quanto il dispendio calorico dipende oltre che da i fattori qui presi in considerazione, anche da altri fattori esogeni come il metabolismo dell'utente.

Concentrandoci brevemente sulla formula per il conteggio delle kcal bruciate, possiamo accorgerci che si tratta di una semplice moltiplicazione tra termini ben definiti, che sono:

- 0.5 kcal: è un moltiplicatore fisso che non varia nel corso del tempo. Moltiplicare per 0.5 è utile a dimezzare la grandezza del prodotto così da rendere il dato più attendibile e vicino alla realtà.
- kg peso corporeo: rappresenta semplicemente il peso espresso in kg dell'utente.
- km percorsi: indica la distanza percorsa dall'utente durante il periodo di utilizzo dell'applicazione. Esistono due metodologie di stima del passo.
  - Si può calcolare facendo una semplice moltiplicazione:

$$kmpercorsi = npassi * lpassom$$

    I due termini della moltiplicazione sono quindi interessanti al fine del nostro studio:

    * npassi: sta per numero di passi effettuati dall'utente. Si farà riferimento, nel caso specifico di un'applicazione, al numero di passi contati dall'algoritmo che si è scelto di utilizzare.
    * lpasso m: sta per lunghezza del passo in m. Questo è un dato fondamentale per il calcolo delle calorie perchè dipende da diversi fattori endogeni e esogeni al contesto dell'applicazione. In questo caso si prende in considerazione la lunghezza statica: si tratta di una lunghezza definita all'avvio dell'app e non

varia. Questo fattore dipende dall'altezza del soggetto, se si prende in considerazione un utente alto 1 m e 74 cm la lunghezza del suo passo sarà di circa 74cm e grazie ad una semplice equivalenza possiamo utilizzarlo come dato per il calcolo 0.00074m.

– Oppure, stimando la lunghezza dinamica: il calcolo della lunghezza dinamica del passo non è banale, per arrivare ad avere una stima il più vicino possibile alla lunghezza reale del passo in quel momento bisogna prendere in considerazione diversi fattoti.
Per prima cosa bisogna prendere inconsiderazione altezza, peso e sesso dell'utente utilizzatore e moltiplicarli tra loro per ottenere un fattore k che rappresenti l'utente. Il fattore importante all'interno di questa moltiplicazione è la presa in considerazione del sesso, in quando a seconda di esso vi sarà un fattore moltiplicativo k differente, se si tratta di un uomo il fattore di moltiplicazione sarà 0.415, mentre se si prende in considerazione un utente donna esso sarà 0.413. Il moltiplicatore in base al sesso è fondamentale perchè la differenza sostanziale tra passa eseguito da una donna o da un uomo sta nella forza che egli impiega nel compierlo.
Ottenuto il fattore moltiplicativo k, bisogna utilizzarlo per calcolare la lunghezza del passo (step size), tale calcolo avviene utilizzando i dati che sono stati raccolti dal sensore di accelerazione, quindi:

$$StepSize = k * \sqrt[4]{potenzaPasso^2 - potenzaPassoMedia^2}$$

vedremo in seguito come viene stimata la potenza del passo [29].
Ottenuta la stima della step size, va ricavato $\propto$, che stima il "peso" per quel determinato passo, andando a confrontare la step size stimata con la potenza del passo media (magAvg), discutermo

successivamente il calcolo della potenza del passo media, quindi:

$$\propto= \begin{cases} 0.5 & if \;\; stepSize < magAvg \\ 1.0 & if \;\; stepSize = magAvg \\ 1.5 & if \;\; stepSize > magAvg \end{cases}$$

La stima della distanza non sarà altro che la:

$$Distanza = \sum i = 1^{n} Peak(i) * \propto (i)$$

Questo calcolo ci permette di arrivare ad avere una distanza percorsa il più vicino possibile alla distanza realmente percorsa dall'utente [18].
Dimosteremo più avanti come il calcolo dinamico della lunghezza del passo, si dimostra più affidabile rispetto all'utilizzo della lunghezza statica combinata con il numero di passi.

La distanza percorsa dall'utente è quindi il dato più importante da calcolare per la stima del dispendio calorico.

L'analisi di questa formula ci mostra come i primi due termini siano poco rilevanti ai fini del nostro studio, in quanto sono fissi e non cambiano nel tempo, o meglio il peso di una persona nel tempo varia ma la sua frequenza di modifica non è alta ed in più e un dato che varia in maniera statica dato che dev'essere l'utente a modificarlo.
La nostra analisi, come già è stato dette in precedenza, consiste in uno studio sperimentale sulla migliore metodologia per arrivare a calcolare il numero di passi e la loro lunghezza.

1.3.1 Algoritmi per la stima del numero dei passi eseguiti

Come precedentemente è stato introdotto, all'interno dei dispositivi Android sono due i particolari sensori utilizzabili per il conteggio dei passi eseguiti dall'utente, essi sono lo step counter e l'accelerometro.
Da questo momento ci concentriamo sugli algoritmi implementabili per il calcolo del numero di passi effettuati grazie all'utilizzo dell'accelerometro.

Il sensore di accelerazione ci fornisce un insieme di dati grezzi (x,y,z) e tali dati andranno per questo raffinati. Ci concentreremo sul diverse metodologie di raffinamento dei dati prodotti dall'accelerometro.
Dobbiamo sottolineare come la letteratura relativa al settore del riconoscimento delle attività umane a partire da sensori embedded sia particolarmente ampia, per ovvi motivi tratteremo solo alcuni degli algoritmi più importanti.

Prima di concentrarci sul conteggio dei passi, introduciamo brevemente il concetto di PDR. La sigla PDR sta per "Pedestrian dead reckoning ", si tratta di una delle tecniche più famose per l'indoor position utilizzando i sensori di uno smartphone. È quindi una tecnica che ci permette di rilevare la posizione attuale dell'utente, coordinando i dati prodotti dal sensore e la posizione rilevata in precedenza.

Il primo algoritmo su cui ci concentreremo, prende in considerazioni i dati di accelezione del piano tridimensionale x,y,z rilevati appunto dall'accelerometro.
Per prima cosa viene calcolata la vibrazione sui tre assi, che chiameremo magnitude, il calcolo è basilare:

$$mag = \sqrt{x^2 + y^2 + z^2}$$

Per escludere dal calcolo della magnitudine i valori della gravità si esegue una computazione attraverso il calcolo delle magnitudini medie ottenute dal

calcolo di mag.

$$netmag = mag - avgmag$$

Quindi dalla sottrazione tra magnitudine e magnitudine media, possiamo quindi ricavare la net magnitude che rappresenta un dato quasi preciso sulla potenza di esecuzione del passo da parte dell'utente[18].
Il problema principale di questo algoritmo è dato dalla tendenza all'in accuratezza dei sensori di uno smartphone. Infatti, il problema principale dell'accelerometro sta nella sua acuta sensibilità al movimento.
Per ovviare a questo problema viene fatto un livellamento dei dati, andando ad escludere eventuali valori evidenziabili come outliners, questo renderà il rilevamento dei picchi di dati tendenzialmente più stabile. In questa fase vengono utilizzati due particolari tecniche di filtraggio dei dati:

- il primo detto filtro Kalman per eliminare eventuali falsi picchi generati dalla forza di gravità. Il filtro Kalman è un algoritmo di filtraggio dei dati, genera la stima ottimale delle quantità desiderate attraverso un processamento ricorsivo. Tale filtro non guarda alla precisione dei dati, bensì va a processare tutte le informazioni possibili. [7]

- il secondo detto high pass filter, è una tipologia di filtraggio che permette solo a quei segnali superiori ad una determinata frequenza di taglio di essere presi in considerazione. [30]

L'applicazione di questi filtri permette di "smussare" i dati ottenuti, per arrivare ad un conteggio del passo più stabile. Tale algoritmo risulterà particolarmente rilevante anche per essere utilizzato nel calcolo della distanza percorsa da un utente con applicazione del calcolo della lunghezza dinamica del passo.

Il secondo algoritmo che prenderemo in considerazione, cerca di sfruttare ampiamente la continuità della deambulazione andando a filtrare e differenziare i dati ricevuti dall'accelerometro.
In questo caso si cerca di comprendere il "sistema di coordinate del corpo",

infatti l'andamento fisico dell'utente può essere calcolato prendendo in considerazione le coordinate geografiche (n), a tali coordinate fanno riferimento tre angoli sequenziali di rotazione che sono: heding(φ), l'inclinazione(θ) e la rotazione (ϕ). I valori dei tre angoli vengono associati nella matrice di trasformazione C_n^b, che è così rappresentata:

$$C_n^b = \begin{bmatrix} cos\varphi\cos\theta & \sin\varphi\cos\theta & \text{-}\sin\theta \\ -sin\varphi\cos\phi{+}\cos\varphi\sin\theta\sin\phi & \cos\varphi\cos\phi{+}\sin\varphi\sin\theta\sin\phi & \cos\theta\sin\phi \\ sin\varphi\sin\phi{+}\cos\varphi\sin\theta\cos\phi & \text{-}\cos\varphi\sin\phi{+}\sin\varphi\sin\theta\cos\phi & \cos\theta\sin\phi \end{bmatrix}$$

Data la matrice di trasformazione, possiamo definire all'interno del frame n il vettore x^n e all'interno del frame b il vettore x^b, così da definire la matrice di trasformazione che dal frame n va in b, quindi:

$$x^b = C_n^b x^n.$$

L'utilizzo di questo sistema aiuta il filtraggio dei dati. Si assume che in questo caso i valori raccolti abbiano un andamento di oscillazione a forma d'onda e che l'insieme di onde, a causa dei movimenti del corpo, abbiano un andamento oscillatorio.
Definita la matrice di trasformazione possiamo passare al vero e proprio calcolo del passo, prendiamo in considerazione la matrice C_n^b, il valore degli assi del piano tridimensionale ricavato dall'accelerometro e la gravità rilevante solo per il valore z che ricopre il ruolo di dato migliore caratteristica di camminata.

$$a^{'} = C_n^b \begin{bmatrix} a_x \\ a_y \\ az \end{bmatrix} - g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

La matrice $a^{'}$ sarà composta da $a_x^{'}$, $a_y^{'}$ e $a_z^{'}$. Nello studio dell'accelerazione di fondamentale importanza sarà $a_x^{'}$, per le ragioni sopracitate. Infatti l'accelerazione è calcolabile facendo la differenza tra $a_x^{'}$ corrente e la media dei $a_x^{'}$:

$$acc = a_z^{'} - avg(a_z^{'}).$$

Ricavato il valore dell'accelerazione, possiamo calcolare la ΔAccelerazione che potrà essere utilizzata per rilevare e contare il passo. Il suo calcolo è dato da una semplice sottrazione tra accelerazione corrente e l'accelerazione precedente.
L'utilizzo delle variabili ΔAccelerazione e acc che abbiamo calcolato è propedeutico al validazione e conteggio del passo effettuato, difatti vengono considerate 5 condizioni che devono risultare tutte vere così che il passo sia considerato reale [29].
I due algoritmi che abbiamo sopra discusso risultano entrambi validi, il secondo è più complesso da un punto di vista matematico, mentre il primo risulta più intuitivo.

Capitolo 2

Progettazione

2.1 Obiettivi

Il lavoro svolto nella tesi si va principalmente a concentrare sull'ampliare le funzionalità di un'app MobileHealth, precedentemente iniziata in un altro lavoro di tesi, per il monitoraggio di pazienti con scompenso cardiaco. Il campo di incremento funzionale dell'applicazione, come introdotto nel capitolo precedente, è quello del controllo a distanza dell'attività motoria svolta dall'utente. Nonostante le applicazioni in ambito mHeath e non, che svolgono quest'attività siano numerose, l'obbiettivo del lavoro sta nel rendere un'app già utilizzata ampliata anche delle funzionalità di monitoraggio del movimento fisico dell'utente.

Quello che si andrà quindi a presentare, all'interno di questa tesi, è l'aggiunta delle funzionalità di un'applicazione mobile in campo biomedicale, utilizzata all'interno del Policlinico Sant'Orsola-Malpighi, per l'assistenza di pazienti con due differenti patologie: i portatori di LVDA e quelli con scompenso cardiaco grave.

Gli scopi quindi della tesi saranno: quelli di aggiungere le funzionalità di monitoraggio del movimento e del dispendio calorico dell'utente per una

MHealth application.

In questo capitolo, andremo a discutere le scelte progettuali ed architetturali fatte.

2.1.1 L'applicazione base

Descriviamo brevemente le funzionalità contenute nell'applicazione base. L'app si compone di cinque macro funzionalità, quali: Record, Remind, Communicate e Inform. Da queste è stata delineata la pool delle funzionalità implementate.
L'estensione delle macro funzionalità ha portato a definire le seguenti componenti:

- Registrazione - in cui il paziente definisce i loro dati identificativi e il suo account, in più definisce anche i dati da monitorare descrivendo la sua patologia.
- Autenticazione - il paziente inserisce le credenziali (email e password) per eseguire il login.
- Monitor: il paziente visualizza i suoi dati personali e i parametri inseriti relativamente a quella giornata.
- Inserisci Parametri: il paziente inserisce il set di parametri quotidiano, questi parametri si differenziano in base alla tipologia di paziente, o con scompenso cardiaco grave o LVDA.
- Contapassi: il paziente visualizza i passi compiuti nella giornata corrente, ma per la realizzazione di questo sistema è stato utilizzato il contapassi integrato del sistema Android (il sensore virtuale Android).
- Allarme: registra alcuni parametri pericolosi (nel caso di pazienti LVDA) e chiama un numero di emergenza

- Calendario: il paziente può visualizzare i parametri inseriti raggruppati giorno per giorno.
- Visualizza giorno: il paziente visualizza nel dettaglio i parametri inseriti per quella giornata.
- Documentazione: il paziente visualizza una serie di PDF di carattere informativo
- Invio report: se il paziente registra i dati da almeno 2 settimane può estrarre il report bisettimanale e inviarlo al medico.
- Grafici: il paziente visualizza dei grafici a linee della variazione di determinati parametri nel tempo.
- Notifica: invia un remainder al paziente se entro le 16:00 non ha inserito i parametri giornalieri.

Queste sono state definite come le funzionalità basi dell'applicazione, che possono essere ampliate ed estese, non solo con il progetto portato avanti da queta tesi ma anche con altri lavori [14].

Nel corso di questo capitolo descriveremo le scelte architetturali definite nell'app base e che sono state seguite anche per la realizzazione di questo progetto.

2.2 Specifiche

L'intento al momento della decisione dell'argomento di tesi era quella di fissare e realizzare una serie di incontri con dei responsabili del reparto di cardiologia del Policlinico S. Orsola per definire i punti fondamentali e la struttura del progetto. Ciò non è stato possibile a causa della pandemia. Per questa ragione le specifiche sono state definite tramite un incontro con il professore e con lo sviluppatore che precedentemente ha iniziato tale lavoro.

Abbiamo così definito come argomento centrale del mio progetto quello di estendere la funzionalità di tracking di parametri attraverso l'uso dei sensori Android. Sottolineando in particolar modo l'interesse, per il calcolo del consumo calorico dell'utente.

L'intento iniziale è stato quello di migliorare il sistema di conteggio e riconoscimento dei passi eseguiti dall'utente, cercando di applicare il miglior algoritmo possibile.
Terminata la fase di conteggio del passo, abbiamo definito come punto fondamentale quello del riconoscimento della lunghezza del passo in modo dinamico, da ciò è stato possibile ricavare la distanza percorsa durante la giornata dall'utente.
Il miglioramento del sistema di conteggio del passo e il riconoscimento della lunghezza del passo, sono stati necessari al raggiungimento dell'obbiettivo finale nell'ampliamento delle funzionalità, ovvero il calcolo del dispendio calorico nel corso della giornata dell'utente.

2.3 Le scelte architetturali e funzionali

Definiti gli obiettivi della tesi, quello che si è fatto e prendere le decisioni progettuali per portare avanti l'applicazione. Dato che si tratta di un ampliamento ad un lavoro precedentemente iniziato, le scelte a livello architetturale non sono state numerose. Mentre per quanto riguarda le funzionalità le scelte sono state coerenti con le specifiche fornite, cercando la migliore metodologia di sviluppo.

2.3.1 L'architettura

L'applicazione è stata precedentemente implementata in modo robusto, così da essere facilmente sviluppabile, sostenibile e mantenibile.

Per la realizzazione dell'applicazione sono state fatte delle scelte di software engineering, di fatti come punto di riferimento è stata presa un'architettura che rispetta i principi di programmazione ad oggetti detta "Clear Architecture". Ciò fa riferimento ai cinque principi SOLID [14] che sono:

- S: "Single responsibility principle", un modulo o una classe del sistema deve far riferimento ad una singola funzionalità del software.
- O: "Open/Close principle", le entità del software sono aperte alle estensioni ma chiuse alle modifiche.
- L: "Liskov substitution principle", una classe che deriva da un'altra classe detta classe derivata, può essere sempre sostituita dalla classe da cui deriva.
- I: "Interface segregation principle", nessun client dovrebbe dipendere da interfacce generiche, bensì dovrebbe dipendere sempre da interfacce specifiche che implementano solo i metodi al client necessari.
- D: "Dependece inversion principle", si tratta del principio che disaccoppia i moduli del software, affermando che: i moduli di basso livello non dovrebbero dipendere da moduli di alto livello, e viceversa; in più le astrazioni non devono dipendere dai dettagli, ma contrariamente le dipendenze dovrebbero dipendere dalle astrazioni [12].

L'utilizzo di queste best practies, permette di ampliare l'applicazione con nuove funzionalità, infatti estendere l'applicazione è possibile grazie all'applicazione di Open Close principle.

Riferendoci brevemente al concetto di clear architecture, possiamo definirla come una struttura "pulita" che permette di aggregare in modo semplice ed esplicativo concetti di alto livello. Gli obbiettivi principali di tale struttura sono:

- Fornire equilibrio tra usabilità, semplicità, astrazione e concetti di alto livello.
- Un'architettura concreta con un layer di base e che include un insieme di regole e convenzioni.
- è indipendente dall'ambiente di programmazione.

Una clear architecture, si presenta su 3 layer costruiti uno sopra l'altro, l'esistenza del sottostante layer è indispensabile all'esistenza dei livelli superiori. I tre livelli sono:

1. Domain Layer: Vengono definite le business rule e gli oggetti del dominio.
2. Application Layer: Vengono definiti gli use cases necessari all'orchestrazione delle complementi del dominio. In più in questo livello, viene definita il criterio per l'interazione del client con la logica di dominio.
3. Client Layer: vengono definiti: tutti i dettagli dell'infrastruttura come il database e i framework utilizzati; le porte pubbliche per l'utilizzo di servizi esterni (per esempio le api); i dettagli per l'esecuzione del testing quali: unit, functional e integration. [10]

Per la realizzazione dell'applicazione, combinato con la clear architecuture, è stato utilizzato il design pattern "Model-View-ViewModel". La sua utilità è quella di aiutare a separare la logica di business dall'interfaccia utente di un'applicazione; questo aiuta a migliorare il testing, l'evoluzione e il mantenimento di un'applicazione [13].
All'interno di questo modello possiamo individuare tre elementi fondamentali:

- View: responsabile della struttura, del layout e dell'interfaccia grafica con cui l'utente lavora.

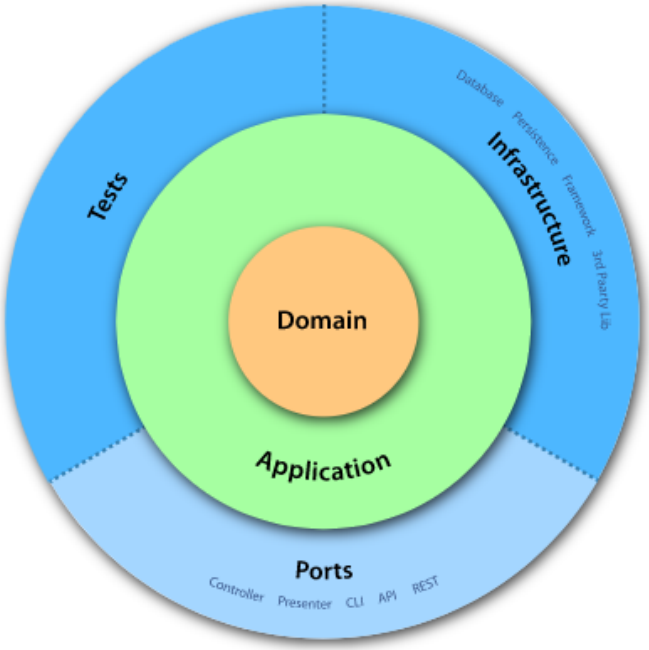

Figura 2.1: Rappresentazione concentrica di una clear architecture. [10]

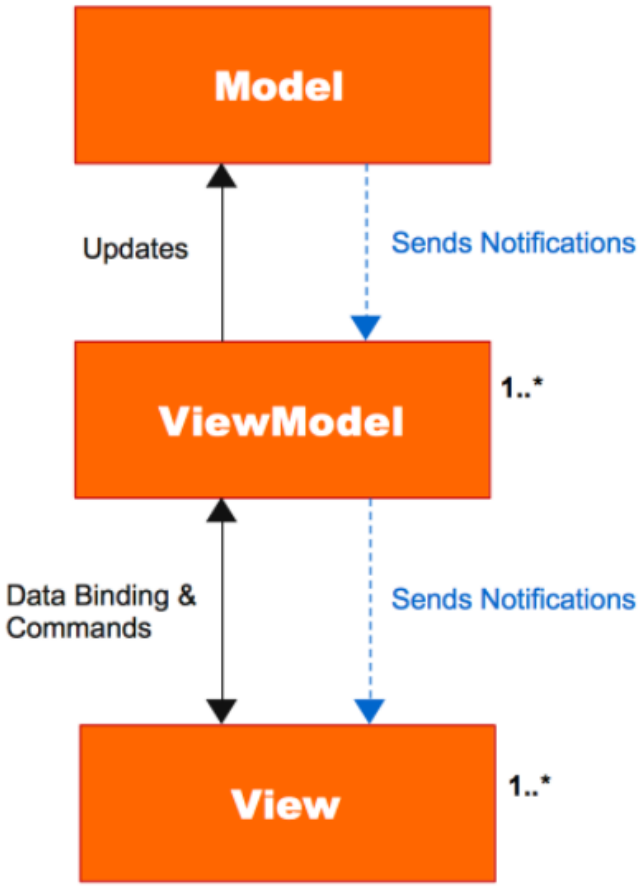

Figura 2.2: Rappresentazione grafica del pattern model-view-ViewModel.

- Model: sono classi non visibili all'interno dell'applicazione che incapsulano i dati raccolti dall'app.

- ViewModel: implementa le proprietà e i comandi a cui la View è collegata, in più fa riferimento anche allo stato e all'eventuale sistema di notifiche di cambio di stato dei dati. [27].

Raggiungendo così il fine di avere un'applicazione mantenibile e solida.

2.3.2 Le funzionalità

Prendendo in considerazione l'obbiettivo finale da raggiungere cioè l'ampliamento dell'app, con il calcolo del dispendio calorico nel corso della giornata da parte dell'utente.
Da ciò abbiamo ricavato le due funzionalità principali necessaria al raggiungimento di tale obbiettivo. Tali funzionalità sono:

- Il riconosciemento del movimento dell'utente con annesso calcolo del numero dei passi eseguiti.
- Il calcolo della lunghezza del passo in modo dinamico così che il calcolo del dispendio calorico sia il più affidabile possibile.

Per la loro realizzazione di entrambe si è fatto un largo utilizzo della letteratura largamente discussa nel primo capitolo, cercando la metodologia più affidabile e aderente alle nostre necessità.
In più, si è lavorato cercando di rispettare il più possibile la struttura dell'applicazione precedentemente predisposta.

In seguito, tratteremo ampiamente le scelte implementative fatte per realizzare tali funzionalità.

Capitolo 3

Implementazione

Nel corso di questo capitolo discuteremo le scelte implementative che hanno consentito di mettere in pratica le decisioni discusse in fase progettuale.

L'applicazione a cui facciamo riferimento all'interno di questa tesi è stata implementata in modo disgiunto dall'app principale, anche se sono state rispettate tutte le linee guida precedentemente stabilite.

3.1 Tecnologie

Il sistema operativo che è stato scelto per l'implementazione dell'applicazione a cui facciamo riferimento è Android.
Le ragioni che hanno portato a preferire Android piuttosto che IOS, sono innumerevoli, tra cui la sua larga diffusione. A rafforzare ciò la documentazione che mette a disposizione Google è maggiormente completa e approfondita, anche dal punto di vista della gestione dei sensori, tema centrale di questa tesi. Di fatti in IOS l'utilizzo di sensori, come l'accelerometro, risulta molto più macchinoso rispetto ad Android.
Facendo però riferimento alla parte di progettazione definita nel capitolo precedente, avremmo potuto implementare l'applicazione in IOS senza apportare modifiche o avere particolari problematiche.

L'ambiente di sviluppo integrato che è stato utilizzato è "Android Studio" per l'implementazione dell'applicazione, questo perchè è lo strumento universale per la realizzazione di applicazioni per questo sistema operativo. Infatti permette di creare app per smartphone e tablet, per Wear OS dedicate ai dispositivi indossabili, per Android TV e tutti gli altri dispositivi che utilizzino Android come sistema operativo.
Il codice è stato scritto definendo come minimum SDK (Software development kit) version Android 5.0 (Lollipop) che fa riferimento all'API 21. Tale versione del sistema operativo è disponibile a partire dal 2014, è stato l'aggiornamento di sistema che ha incluso tutte le più grandi novità che oggi siamo abituati a vedere. Infatti, Lollipop include Material Design, che permette un utilizzo più intuitivo delle applicazioni, e la nuova gestione delle notifiche dalla schermata di blocco [11].
Il codice è stato compilato su dispostivi Android che utilizzano l'API 29 (Target SDK), cioè la versione del sistema operativo 10.
Ad oggi, la nostra applicazione, potrebbe essere installata su circa il 94.1% degli smartphone in utilizzo (dato riportata da Android Studio).

I linguaggi utilizzabili per implementare un'app Android sono Java e Kotlin. I due linguaggi risultano essere facilmente compatibili tra loro e in aggiunta Android Studio mette a disposizione una funzionalità che permette di convertire il codice Java in codice Kotlin, e viceversa. Per realizzare il nostro progetto è stato utilizzato Kotlin. Le ragioni che hanno portato alla scelta di Kotlin a discapito di Java, sono:

- Facile comprensione: Kotlin risulta più intuitivo sia dal punto di vista del linguaggio che dal punto di vista semantico rispetto a Java.

- Programmazione più breve: Kotlin permette di avere lo stesso risultato che avremmo in Java con meno righe di codice, quindi meno lavoro per il programmatore.

Figura 3.1: Tutte le versioni di Android su cui l'app è funzionante

- La gestione del puntatore nullo: uno dei grandi problemi di Java è la gestione delle null reference, che Kotlin gestisce senza difficoltà [9].

3.2 Le componenti dell'architettura

Un'applicazione Android è composta da codice e risorse.
Le risorse sono tutto ciò che non è codice, ovvero tutti i dati che l'app utilizza in maniera statica. Vengono definite all'interno di file di tipo xml, che a runtime l'app valuterà. Le risorse vengono definite all'interno della cartella del sottoalbero di un'applicazione detta "res", che è composta da tre sottocartelle standard, quali: layout, value e drawable.

La componente fondamentale all'interno di un'applicazione Android è l'Activity.
Un'activity può essere definita come il livello "visivo" dell'applicazione, si tratta infatti, nella maggior parte dei casi, di quello che l'utente vede. Può essere definita come una singola visualizzazione di ciò che l'utente può fa-

re. Il passaggio da un'activity ad un'altra avviene attraverso l'utilizzo degli intent, un sistema con un comportamento simile a quello di un sistema di messaggistica ma tra le componenti dell'applicazione.
Un'activity fondamentale a cui dobbiamo far riferimento è la "Main Activity", rappresenta, nella stragrande maggioranza dei casi, il punto di accesso all'applicazione. Si tratta dell'activity che permette di gestione il flusso di dati dell'applicazione. In più, essa permetterà di richiedere l'utilizzo di funzionalità esterne all'applicazione, come l'utilizzo delle funzionalità del sistema di messaggistica, o l'utilizzo delle componenti hardware come la fotocamera.
Parlando delle Activity, dobbiamo introdurre brevemente il suo ciclo di vita (life cycle).
Un utente può navigare all'interno dell'applicazione, questo è permesso dalle classi di tipo Activity, che attraverso una serie di callback, permettono di capire cosa sta cambiando all'interno del sistema. I metodi di callback che compongono l'Activity lifecycle sono:

1. onCreate(): è lo stato in cui l'Activity viene creata.
2. onStart(): lo stato in cui l'Activity diventa visibile all'utente.
3. onResume(): l'utente può interagire con l'Activity, quindi è nel suo stato di utilizzo.
4. onPause: in questo caso l'Activity diventa "semitrasparente", in quanto viene coperta da un'altra Actvity che è in stato onResume.
5. onStop: l'activity è completamente nascosta, quindi non visibile all'utente.
6. onRestart: dallo stato onStop(), l'Activity viene ricaricata per essere resa di nuovo disponibile all'utente.
7. onDestroy: l'Activity viene rimossa dalla memoria [28].

I dati all'interno di un'Activity che si trova in stato onStop o onPause, vengono salvati in una particolare struttura dati detta "bundle". Cioè un dato

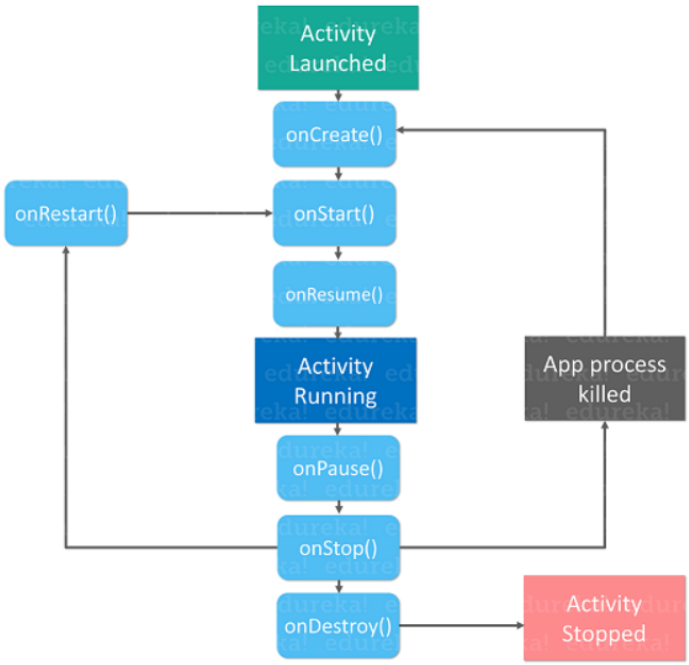

Figura 3.2: L'Activity life cycle [28]

di tipo blob, accessibile tramite un'associazione chiave-valore.
Tutte le Activity presenti in un'app, devono essere dichiarate all'interno dell'"Android Manifest", cioè un file xml che al suo interno descrive le informazioni essenziali sull'applicazione.

Quindi la differenza principale tra un'applicazione Android e un software per pc è che in un'app non esiste un unico punto di accesso, quindi il "main", ma esistono più punti d'accesso che possono svolgere diverse funzionalità a seconda dell'esigenza.

3.2.1 La user interface

Per l'implementazione della nostra applicazione è stata realizzata una sola Activity (la "Main Activity"). Tale scelta è derivata dal fatto che l'applicazione relativa a questa tesi è stata realizzata in modo indipendente rispetto

al resto dell'applicazione.
Un'app graficamente può essere realizzata utilizzando diverse tipologie di layout.
Un Layout è utilizzato per definire la user interface per un'activity dell'applicazione. L'interfaccia utente è composta da raccolte di oggetti che sono:

- View: rappresenta la classe base per tutti gli elementi della UI ed è utilizzata per creare: gli elementi interattivi (es. i radio button), è anche responsabile per la realizzazione delle animation e dei drawer.

- Vew Group: è una sottoclasse delle View, si tratta di un container per definire le proprietà di layout.

In generale quindi un'app Android avrà una o più Activity, ognuna delle quali avrà una sua interfaccia utente.
Il layout in Android può essere definito in due modi, o mediante dichiarazione degli elementi nella UI in XML, oppure, istanzianto gli elementi in fase di esecuzione (quindi nel nostro caso nella classe Kotlin). Grazie però ai framework, possiamo realizzare i layout della nostra applicazione in modo ibrido [2].
Ogni Layout ha una serie di attributi contenuti nel layout parameters che riguarderà anche i suoi "figli". Gli attributi fondamentali sono l'altezza e la larghezza. Come è già stato introdotto precedentemente, esistono diverse tipologie di layout, esse sono:

- Linear Layout: si tratta di un view group utilizzato per fare il rendering di tutte le view figlie una per una, o in orizzontale o in verticale.

- Relative Layout: è una tipologia di view group utilizzato per posizionare una view in un punto collegato ad un'altra View, oppure in una posizione definita rispetto al genitore.

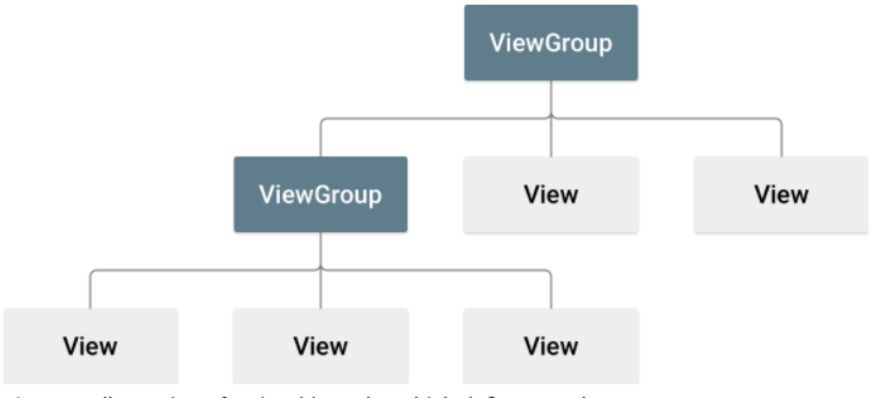

Figura 3.3: Illustrazione della struttura gerarchica delle view, per definire l'UI Layout.

- Frame Layout: si tratta di un tipo di view group contenitore per altre view. Infatti, si utilizza per sovrapporre una view ad un'altra.
- Table Layout: Permette di visualizzare le view che compongono il view group in righe e colonne.
- Constraint Layout: è un view group che al momento dell'inserimento delle view le allinea tutte in alto a sinistra, le view dovranno poi essere dislocate all'interno del view group attraverso il posizionamento tramite le proprietà del tag xml.

All'interno dell'applicazione realizzata per la tesi è stata realizzata un'interfaccia UI con layout di tipo Linear, i cui elementi al suo interno vengono posizionati in modo orizzontale.
Per la realizzazione di questo layout è stato utilizzato un ibrido di programmazione XML e Kotlin. Gli elementi sono stati definiti all'interno della struttura XML (come possiamo vedere dal listing 3.1), ma la gestione dei singoli elementi in modo dinamico è avvenuto all'interno della classe Kotlin. Riferendoci alla gestione dinamica degli elementi del layout ci riferiamo a

come variano i valori delle variabile all'interno dell'interface, questo è visibile dalla figura 3.4.
I principali elementi utilizzati all'interno della UI sono:

- TextView: questo elemento viene utilizzato per la visualizzazione del testo per l'utente. I valori stampati nella nostra interfaccia nella view dell'applicazione sono per lo più settati all'interno della classe Kotlin in quanto i valori variano al variare del numero di passi, della distanza percorsa e delle calorie bruciate.

- ImageView: permette l'inserimento e la visualizzazione delle immagini all'interno della view. La maggior parte delle immagini viene settata all'interno del file xml, l'immagine della bandierina (visualizzata nell'immagine B della figura 3.4), inizialmente non viene mostrata, ne viene settata la visibilità in modo programmatico tramite il metodo "setVisibility(true)".

- ProgressBar: si tratta di un elemento che fornisce un feedback visivo dell'attività che si svolgendo. Infatti, inizialmente la round progress bar è viene visualizzata tutta grigia e progressivamente in base al numero di passi che vengono svolti essa viene colorata di verde, ciò avviene tramite il metodo "setProgess(valore numerico)" inserito nella classe Kotlin.

Si è voluto quindi realizzare una UI intuitiva per l'utente.

Nei prossimi paragrafi parleremo di come avviene il calcolo delle kcal, dei km percorsi e del conteggio del numero di passi eseguiti.

```
1
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/
     res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
```

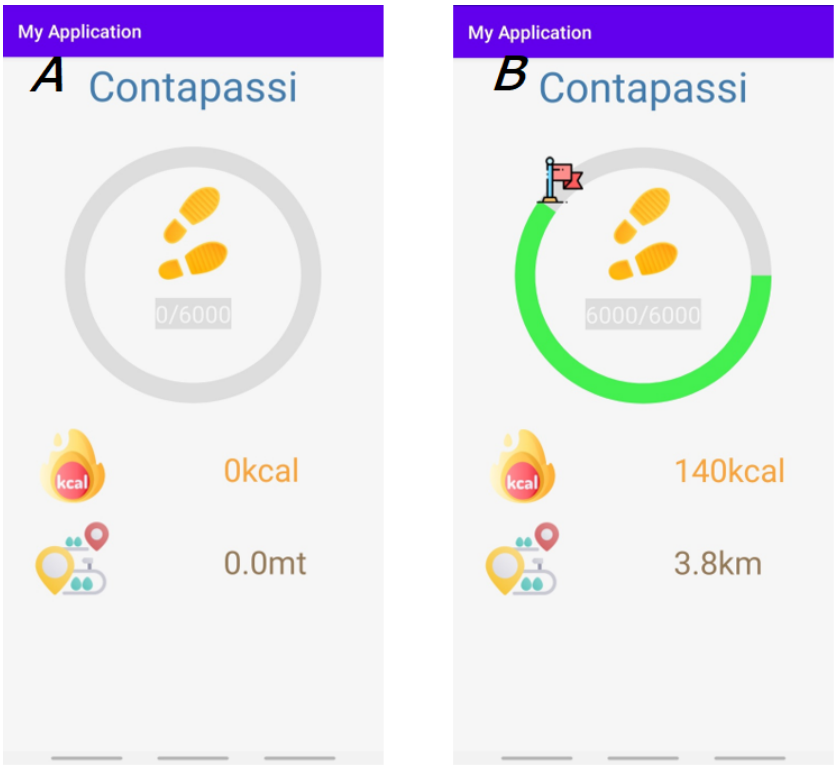

Figura 3.4: Esempio dell'UI dell'applicazione in due diverse situazioni:
A. Applicazione ancora in utilizzata con 0 passi fatti.
B. Applicazione con obiettivo giornaliero di 6000 passi raggiunto.

```
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity"
8     android:background="@color/bgOther"
9     android:orientation="vertical">
10
11
12    <TextView
13        android:id="@+id/textView2"
14        android:layout_width="wrap_content"
15        android:layout_height="wrap_content"
16        android:text="Contapassi"
17        android:textColor="@color/title"
18        android:textSize="45dp"
19        android:layout_gravity="center"
20        app:layout_constraintBottom_toBottomOf="parent"
21        app:layout_constraintHorizontal_bias="0.465"
22        app:layout_constraintLeft_toLeftOf="parent"
23        app:layout_constraintRight_toRightOf="parent"
24        app:layout_constraintTop_toTopOf="parent"
25        app:layout_constraintVertical_bias="0.021" />
26
27    <ProgressBar
28        android:id="@+id/progressBar2"
29        android:layout_width="350dp"
30        android:layout_height="350dp"
31        android:indeterminateOnly="false"
32        android:progress="0"
33        android:layout_gravity="center"
34        android:progressDrawable="@drawable/circle"
35        app:layout_constraintBottom_toBottomOf="parent"
36        app:layout_constraintHorizontal_bias="0.454"
37        app:layout_constraintLeft_toLeftOf="parent"
38        app:layout_constraintRight_toRightOf="parent"
39        app:layout_constraintTop_toTopOf="parent"
40        app:layout_constraintVertical_bias="0.12" />
41
```

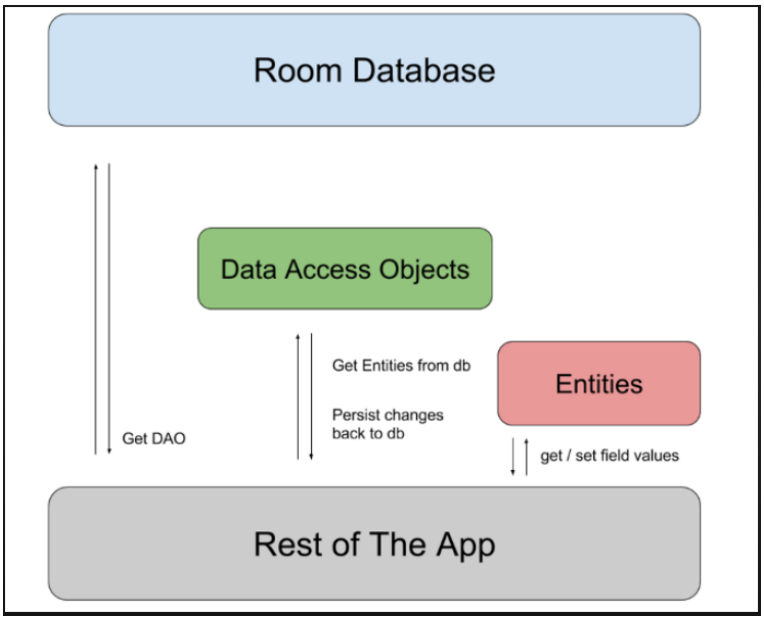

Figura 3.5: La struttura di Room db

```
42      .....
43
44
45  </LinearLayout>
```

Listing 3.1: Uno stralcio implementato all'interno dell'applicazione

3.2.2 Il database

La creazione di un database relazionale in un'applicazione Android piè essere portat avanti attraverso l'uso delle API di SQLite. Tale scelta però non è consigliata, in quanto rappresenta un'implementazione macchinosa e di basso livello.

La metodologia più comune e diffusa è quella di utilizzare la "Room Persisten Library". Si tratta di una libreria che fornisce un livello di astrazione di SQLite, per consentire un accesso più affidabile al database sfruttando al contempo tutta la potenza di SQLite.

La struttura di Room Database è divisa in tre componenti (come possiamo vedere anche dalla figura 3.5):

- Database: si tratta del punto di accesso al database vero e proprio, infatti permette la creazione delle relazioni tra entità e le interfacce DAO. Può essere quindi definito come holder del sistema.

```
1
2 import androidx.room.Database
3 import androidx.room.RoomDatabase
4
5 @Database(version = 1, entities = [PedometerDatas::class
     ], exportSchema = false)
6 abstract class MyDB : RoomDatabase() {
7     abstract val subscriber: PedometerDatasDAO?
8 }
```

Listing 3.2: Codice della creazione del database usato all'interno dell'applicazione.

- DAO: la sigla sta per "Data Access Objects", si tratta dell'interfaccia che permette di accedere ai dati all'interno del database. Tali interfacce sono riconoscibili dall'annotation @DAO. È l'elemento che permette di eseguire le operazioni CRUD(Create, Read, Update, Delete). Queste operazioni vengono indicate, anch'esse, tramite l'utilizzo delle annotation quali: "@Insert", "Delete", "Update" e "Query" (utilizzato per eseguire interrogazioni complesse all'interno del database).
  Per rendere il codice più leggibile si utilizza un'interfaccia DAO per ogni entità del nostro database.

```
1
2 interface PedometerDatasDAO {
3     @Query("SELECT * FROM PedometerDatas")
4     fun getDatas(): PedometerDatas?
5
```

```
6       @Insert(onConflict = OnConflictStrategy.REPLACE)
7       fun insertDatas(Subscribers: PedometerDatas?)
8
9   }
```

Listing 3.3: Codice di un'interface DAO con esempi di operazione di inserimento e interrogazione.

- Entity: sono la rappresentazione astratta dalle tabelle di un database. Vengono definite attraverso la creazione di una Plain Old Java Object, cioè viene definita tramite un oggetto che può essere definito "ordinario", nel nostro caso specifico un'entità viene definita come una data class Kotlin. Per ogni elemento dell'oggetto Room DB definirà un campo della tabella del database.
  Per indicare e definire il nome di un'entità si utilizza l'annotation "@Entity".
  Essenziale per ogni entità è la presenza di una primary key, cioè un campo della tabella che indentifichi univocamente ogni sua riga. Tale elemento viene indicato attraverso un'annotation "@PrimaryKey"

```
1
2   @Entity(tableName = "PedometerDatas")
3   class PedometerDatas {
4       @ColumnInfo(name = "dateOfDetection")
5       @PrimaryKey
6       var dateOfDetection: String? = null
7
8       @ColumnInfo(name = "steps")
9       var steps = 0
10  }
```

Listing 3.4: Codice di definizione di un'entità

Quello che fa il database utilizzato dalla nostra applicazione è salvare giorno per giorno il numero di passi eseguiti dall'utente. Questo avviene tramite

un controllo sull'orario e sulla data. La data viene quindi utilizzata come primary key.
La ragione per cui non vengono salvate all'interno del database le calorie bruciate giorno per giorno è che si tratterebbe di una ridondanza nello schema logico del database, in quanto il dato è facilmente ricavabile applicando la formula del calcolo delle calorie.

3.3 L'utilizzo dei sensori

L'utilizzo dei dati forniti dai sensori all'interno di un'applicazione Android è utile a migliorare l'esperienza d'uso per gli utenti. Infatti, ad oggi la maggior parte delle applicazioni ne fanno uso.
I sensori forniscono i dati grezzi con elevata precisione e accuratezza e sono utili se si desidera monitorare il movimento o il posizionamento tridimensionale del dispositivo o se si desidera monitorare i cambiamenti nell'ambiente circostante vicino a un dispositivo.
I sensori a disposizione di uno smartphone sono divisibili in tre categorie principali:

- Sensori di movimento: si tratta dei sensori utili per calcolare la forza di accelerazione e rotazione in un piano tridimensionale. A far parte di questa categoria di sensori sono l'accelerometro, il giroscopio e i vettori di rotazione.

- Sensori di posizione: sono i sensori che permettono di prendere inconsiderazione i parametri ambientali, come la temperatura. Questa categoria di sensori include il barometro, fotometri e termometro.

- Sensori ambientali: si usano per misurare la posizione fisica del sensore. Esso comprende sensori come il magnetometro.

Il campo di studio della nostra tesi è la categoria di "motion sensors".

Android ha messo a disposizione un framework chiamato "sensor framework" per accedere a tutti i sensori, fisici o virtuali, disponibili sul dispositivo e per ottenere tutti i dati grezzi da essi prodotti.
Da tale framework sono state definite delle particolari classi ed interfacce utili ad ottenere i dati prodotti dai sensori. Tali strumenti sono:

- SensorManager: si tratta della classe che ci permette di creare un'istanza del sensore, quindi si tratta della classe che contiene i metodi per elencare tutti i sensori, accedervi, registrare ed annullare la registrazione dell'EventListener.

- Sensor: si tratta della classe che permette di accedere ad un sensore specifico, metodo a disposizione tutti i metodi necessari a determinare la capacità di un sensore.

- SensorEvent: è la classe che fornisce i dati grezzi rilevati dal sensore, spesso questo dato è accompagnato dal valore dell'accuratezza del sensore e dal timestamp di rilevazione del dato.

- SensorEventListner: si tratta di un'interfaccia che permette di ricevere notifiche a seconda del comportamento del sensore [1].

L'utilizzo combinato di queste classi e interfacce permette di ottenere i dati grezzi del sensore che poi possiamo analizzare secondo le nostre necessità.

Il sensore utilizzato all'interno della nostra tesi è stato l'accelerometro, di cui abbiamo ampiamente discusso nel capitolo 1. Ricordiamo brevemente che si tratta di una componente hardware elettromeccanica che misura la forza di accelerazione causata dal movimento o dalla vibrazione del dispositivo [23]. I dati prodotti da questo sensore vengono rilevati su un piano tridimensionale virtuale di coordinate (x,y,z), per questo si tratta di uno dei sistemi di rilevamento del movimento dell'utente più diffuso.

Vedremo adesso come in fase implementativa i dati grezzi rilevati dall'accelerometro sono stati utili per portare a termine le funzionalità di conteggio del passo e del calcolo del consumo calorico da parte di un utente.

3.3.1 Contapassi e calcolo delle calorie bruciate

Discuteremo congiuntamente l'implementazione delle funzionalità di conteggio del numero di passi eseguiti e di calcolo delle calorie bruciate, essendo entrambe basate sull'utilizzo di un unico sensore (l'accelerometro).

Per l'implementazione delle funzionalità richieste, siamo quindi partiti dall'acquisizione dei dati grezzi rilevati dall'accelerometro come possiamo vedere dalle prime righe del codice nel listing 3.5.

Per la realizzazione del contapassi, una volta selezionato l'algoritmo che rispondesse meglio alle nostre necessità, tra quelli discussi nel primo capitolo, siamo passati alla sua implementazione.
Ricordiamo brevemente che l'algoritmo scelto per il raffinamento dei dati grezzi raccolti dall'accelerometro è quello che rilevati i dati sul piano tridimensionale (x,y,z), andando a calcolare la vibrazione totale prodotta dai tre assi (formula a cui nel primo capitolo ci siamo riferiti con mag) e il calcolo della potenza del passo (mediante l'uso della formula a cui nel capitolo 1 ci siamo riferiti chiamando netmag). Calcolati questi valori, è stato fatto un filtraggio sui dati tramite l'algoritmo di Kalman e mediante l'utilizzo del high pass filter, già discussi nel capitolo 1 (pp.25-26).
L'implementazione di tale algoritmo non presenta un alto livello di complessità, di fatti il significato del codice è facilmente intuibile (listing 3.5). I dati dall'accelerometro vengono facilmente acquisiti tramite l'interrogazione del sensore. Successivamente eseguiamo i calcoli della magnitudine (a cui nel codice ci siamo riferiti con "potenzaPasso") e della magnitudine media (a cui nel codice ci riferiamo con "forzaPasso").

Terminata l'esecuzione dei calcoli, abbiamo implementato le metodologie di filtraggio dei valori precedentemente citati. Tale fase è utile a capire se prendere in considerazione un passo o meno.
Terminata la fase di conteggio del passo, siamo passati a calcolare il dispendio calorico.
Per prima cosa abbiamo calcolato il valore k, che prende in considerazione tre parametri: sesso, altezza e peso. Tale valore sarà necessario per calcolare la lunghezza del passo, che come possiamo vedere dal listing 3.5 avviene prendendo in considerazione diversi valori. Come abbiamo già spiegato k che fungerà da moltiplicatore per il valore della radice quarta della sottrazione tra la forza del passo corrente e la media della forza dei passi precedenti.
Tale valore, lpasso, che abbiamo calcolato sarà necessario per stimare la distanza percorsa, infatti esso viene confrontata con la potenza iniziale del passo e a seconda della stima maggiore, minore o uguale, la potenza iniziale del passo verrà moltiplicata per un diverso fattore moltiplicativo che sommato ai risultati precedenti darà la distanza percorsa.
Infine, possiamo calcolare il dispendio calorico che sarà dato dalla semplice moltiplicazione del fattore costante 0.5 per il peso per la distanza percorsa.

```
1
2  val sensorManager =
3              getSystemService(Context.SENSOR_SERVICE) as
       SensorManager
4  val sensor =
5              sensorManager.getDefaultSensor(Sensor.
       TYPE_ACCELEROMETER)
6
7  val stepDetector: SensorEventListener = object :
       SensorEventListener {
8          override fun onSensorChanged(sensorEvent: SensorEvent
       ) {
9              if (sensorEvent != null) {
10                 val x = sensorEvent.values[0]
11                 val y = sensorEvent.values[1]
```

```
12                  val z = sensorEvent.values[2]
13                  val potenzaPasso = Math.sqrt(
14                      Math.pow(
15                          x.toDouble(),
16                          2.0
17                      ) + Math.pow(y.toDouble(), 2.0) +
    Math.pow(
18                          z.toDouble(),
19                          2.0
20                      )
21                  )
22                  val forzaPasso: Double = potenzaPasso -
    potenzaPassoIniziale
23
24                  lPasso = k * Math.sqrt(
25                      Math.sqrt(
26                          Math.pow(
27                              potenzaPasso,
28                              2.0
29                          ) - Math.pow(potenzaPassoIniziale
    , 2.0)
30                      )
31                  )
32
33                  if (lPasso < potenzaPassoIniziale) {
34                      distance = distance + 0.5 *
    potenzaPassoIniziale
35                  } else if (lPasso == potenzaPassoIniziale
    ) {
36                      distance = distance + 1.0 *
    potenzaPassoIniziale
37                  } else {
38                      distance = distance + 1.5 *
    potenzaPassoIniziale
39                  }
40
41                  kcal = 0.5 * kgPeso * distance
42
```

```
43                      potenzaPassoIniziale = potenzaPasso
44                  }
45              }
46               override fun onAccuracyChanged(sensor: Sensor, i:
    Int) {}
47              }
48
49              sensorManager.registerListener(stepDetector, sensor,
    SensorManager.SENSOR_DELAY_NORMAL)
50  }
```

Listing 3.5: Alcune parti del codice per l'implementazione del contapassi e del calcolo del dispendio calorico.

Capitolo 4

Validazione

Completata la fase di implementazione dell'applicazione, adottando le scelte sopracitate, si è passati alla fase di validazione del software e collezionamento dei dati.
In una prima fase di validazione del lavoro quello che è stato fatto è implementare algoritmi alternativi per la stima del passo così da verificare le prestazioni dell'algoritmo scelto.
Terminato tale step di verifica dell'accuratezza dell'algoritmo, si è passati a testare l'affidabilità della tecnica di calcolo dinamico della lunghezza del passo rispetto a quello statico.
Per ogni test sono state eseguite sei prove.

Eseguiti i test per la raccolta dei dati, è stato utilizzato come metodo di analisi statistico dei dati raccolti, per capirne il comportamento.
Gli elementi di calcolo statistico che sono stati utilizzati sono i seguenti:

- Media: ovvero il valore derivante dalla somma aritmetica di un insieme di valori, divisa per il numero dei dati raccolti.

$$mean = \frac{\sum_{i=1}^{n} x_i}{n}$$

- Errore relativo: è quell'errore che deriva dalla divisione dell'errore assoluto per il valore medio. Ricordiamo che l'errore assoluto è dato dalla

sottrazione tra valore massimo e valore minimo.

$$Err_{rel} = \frac{V_{max} - V_{min}}{mean}$$

Per eseguire queste valutazioni mi sono avvalsa dell'utilizzato dell'ambiente di sviluppo R Studio e delle sue funzioni. Sottolineo principalmente l'utilizzo della sua funzione "mean" per il calcolo della media e delle sue librerie "ggplot2" per la stampa dei grafici.

4.1 Validazione della stima del passo

Gli algoritmi che sono stati utilizzati per l'esecuzione del test sono:

- Un sistema simile all'algoritmo implementato, in cui le soglie di raffinamento dei valori calcolati dai dati restituitici dall'accelerometro sono meno precise e affidabili rispetto all'algoritmo scelto. D'ora in poi chiameremo questo sistema di conteggio del passo con "algoritmo 1" (ALG 1).

- Il secondo sistema utilizzato è stato il contapassi integrato di Android a cui si fa riferimento tramite l'interrogazione al sensore:

```
sensorManager?.getDefaultSensor(Sensor.TYPE_STEP_COUNTER)
```

  D'ora in poi chiameremo questo sistema di conteggio del passo con "algoritmo 2" (ALG 2).

- L'algoritmo implementato all'interno dell'applicazione e ampiamente discusso nei capitoli precedenti. D'ora in poi chiameremo questo sistema di conteggio del passo con "algoritmo 3" (ALG 3).

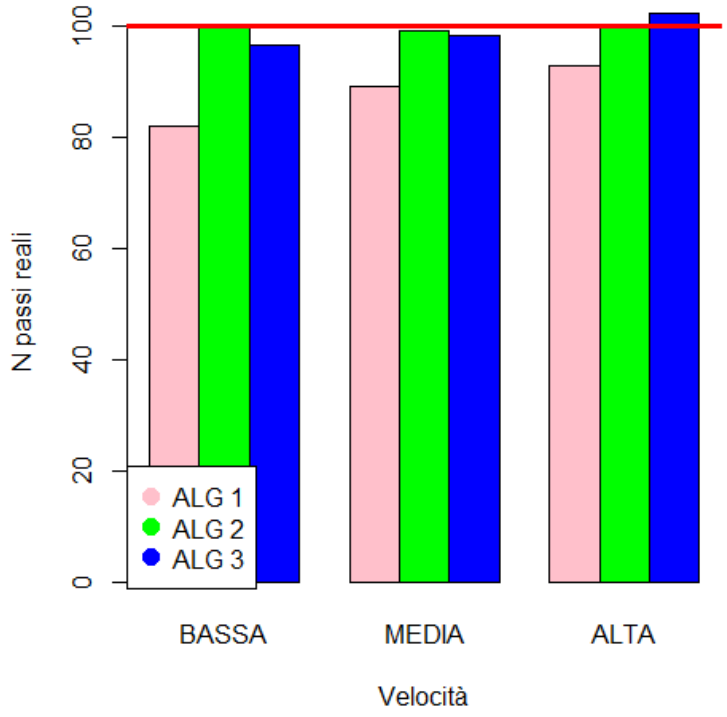

Figura 4.1: Grafico testing sulla velocità: andamento degli algoritmi

4.1.1 Test sulla velocità della camminata

Come primo test eseguito è stata presa in considerazione la velocità della camminata, prendendo tre tipologie di andatura del passo:

- Bassa: cioè una camminata lenta.
- Media: una camminata a passa svelto.
- Alta: presa in considerazione la velocità della corsa.

Per ogni velocità sono stati eseguiti 6 test, per ogni prova sono stati eseguiti 100 passi, come possiamo vedere dalla tabella 4.1. Esamineremo il comportamento, ad ogni velocità, degli algoritmi per studiare la loro accuratezza nelle diverse andature del passo.

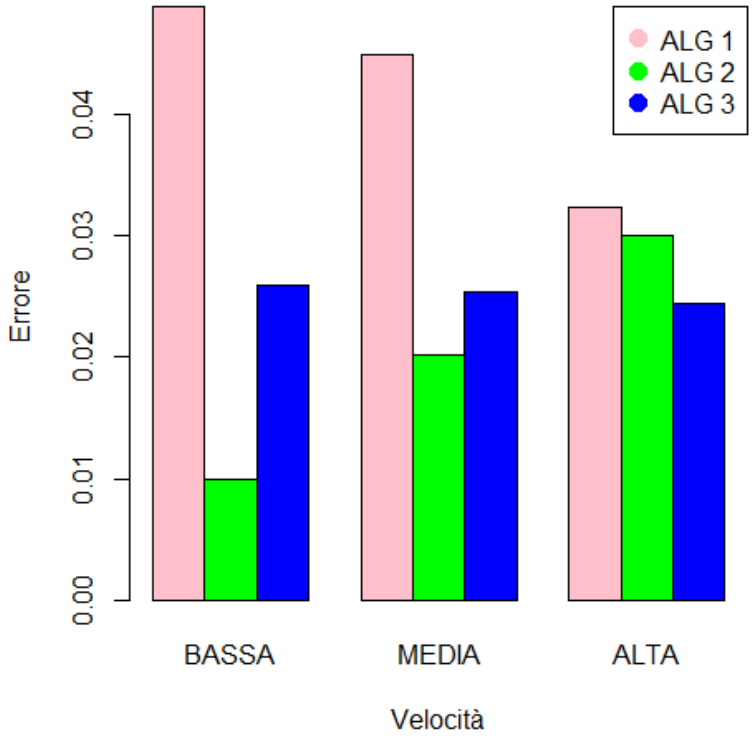

Figura 4.2: Grafico dell'errore relativo degli algoritmi nel testing sulla velocità

	Bassa			Media			Alta		
	ALG1	ALG2	ALG3	ALG1	ALG2	ALG3	ALG1	ALG2	ALG3
1	79	100	95	86	101	98	90	103	100
2	83	99	97	90	97	100	94	99	102
3	81	100	100	87	100	95	90	100	105
4	86	99	97	88	100	99	96	101	102
5	78	101	95	90	98	98	94	97	102
6	85	100	95	94	99	100	92	100	102

Tabella 4.1: Tabella dei dati raccolti per la velocità del passo

Velocità bassa

Prendendo in esame un'andatura del passo lenta, possiamo analizzare un comportamento dei tre algoritmi differenti come si vede dalle prime 3 barre del grafico in figura 4.1.
Guardando anche alla 4.1, ci accorgiamo che ad una velocità bassa, l'algoritmo che risulta essere meno affidabile è l'algoritmo 1. Di fatti, la media dei passi ottenuti durante il testing è di 82 su 100. La sua instabilità è confermata anche dal grafico degli errori relativi (figura 4.2), è appunto l'algoritmo che a tale velocità presenta l'errore più forte.
Per quanto riguarda invece gli algoritmi 2 e 3, cioè lo step counter di Android e l'algoritmo utilizzato all'interno dell'applicazione, possiamo notare che l'algoritmo 2 è presenta un affidabilità superiore la media dei passi calcolata infatti è quasi pari 100 e il suo errore da come possiamo facilmente vedere dal grafico 4.2 si aggira interno a 0.01. L'algoritmo 3 sottostima leggermente il numero dei passi, il valore medio dei passi compiuti è all'incirca 97 e il suo errore relativo è inferiore a 0.03, questo lo rende comunque un algoritmo accettabile.

Velocità media

Facendo riferimento ad una velocità media, possiamo analizzare i comportamenti dei 3 algoritmi facendo un analisi combinata delle parti centrali del grafici in figura 4.1 e 4.2. Come possiamo vedere l'algoritmo 1 si presenta sempre come il più instabile: ciò appare sia del grafico del testing che dal grafico degli errori, anche se mostra un miglioramento nel comportamento rispetto alla velocità bassa, aumentando leggermente la sua affidabilità nel conteggio del numero di passi eseguiti.
Anche in questo caso, l'algoritmo 2 si mostra come il più affidabile per il conteggio dei passi, ma guardando alla figura 4.2 possiamo ben notare che il suo errore relativo mostra un incremento di quasi il doppio del valore. Vice versa, l'algoritmo 3 oltre a migliorare la sua stima nel conteggio del passo avvicinandosi di molto alla soglia del 100, mostra anche un errore assoluto del tutto costante con quanto già calcolato per velocità bassa.

Velocità alta

Prendendo in esame l'ultima categoria di dati raccolti della corsa, il comportamento dei tre algoritmi è simile al comportamento che hanno avuto a velocità bassa e media. vi sono però dei punti da notare: dobbiamo rilevare un deciso miglioramento della stima del passo da parte dell'algoritmo 1 sottolineato anche da una netta diminuzione dell'errore assoluto.
In più, in questo caso c'è una sovrastima del numero dei passi eseguiti da parte dell'algoritmo 3, infatti la media dei passi eseguiti è intorno a 102, ma nonostante ciò il suo errore assoluto rimane in linea con le due predenti situazioni.
Le prestazioni dell'algoritmo 2 sono comparabili a quelle degli altri algoritmi.

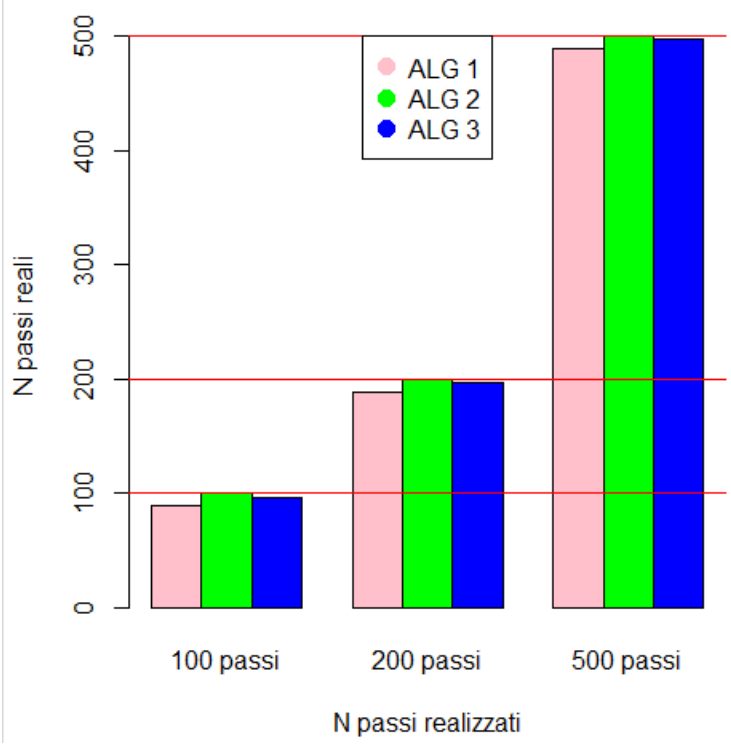

Figura 4.3: Grafico dei test sul numero di passi

4.1.2 Test sul conteggio del numero dei passi

Un altro test che è stato eseguito per misurare l'affidabilità dei 3 algoritmi è il test sul numero di passi.
Per eseguire tali test è stata mantenuta una velocità media e sono stati eseguiti 6 test per ogni numero di passi e per ogni algoritmo (riferito alla tabella 4.2).
Esamineremo singolarmente l'andamento della situazione per numero di passi. Si ometterà solo il commento della situazione relativa ai 100 passi, per evitare inutili ripetizioni in quanto si tratta della stessa situazione che è stata precedentemente discussa nella sotto sezione "Velocità media".

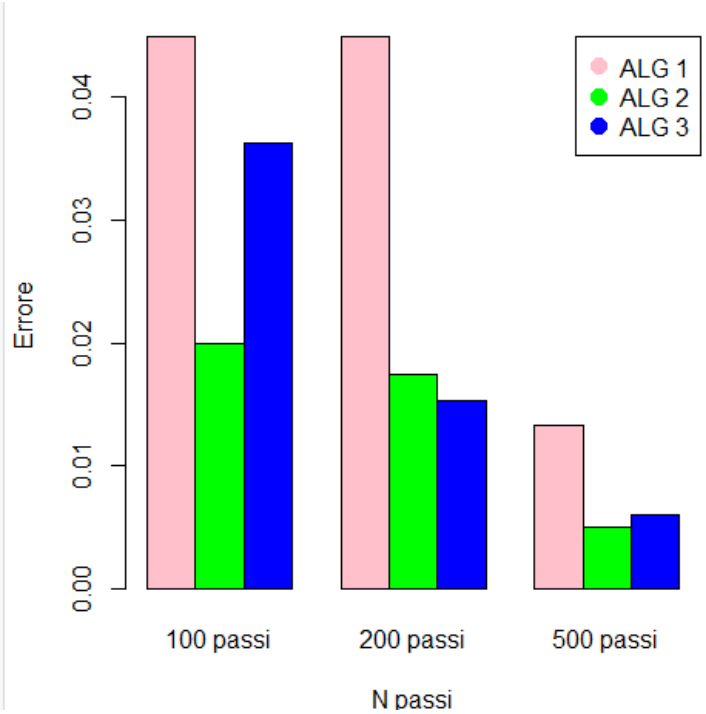

Figura 4.4: Grafico dell'errore relativo degli algoritmi nel testing sul numero di passi

	100 passi			200 passi			500 passi		
	ALG1	ALG2	ALG3	ALG1	ALG2	ALG3	ALG1	ALG2	ALG3
1	86	101	95	179	199	195	495	501	495
2	90	100	98	192	200	196	484	500	499
3	87	102	93	187	201	200	493	498	500
4	88	100	95	196	197	194	486	503	498
5	90	98	98	192	204	200	482	500	495
6	94	99	100	190	200	195	495	499	501

Tabella 4.2: Tabella dei dati raccolti per il conteggio del numero di passi

200 passi

Se si prende come soglia di conteggio dei passi 200, quello che possiamo ricavare osservando il grafico in figura 4.3 è che il comportamento di tutti e tre gli algoritmi non si discosta troppo dalla realtà. L'algoritmo che dimostra più errori, come già visto in altri casi è l'algoritmo 1, ciò e sottolineato anche dal comportamento del suo errore che come possiamo evidentemente capire dal grafico 4.4, rimane sì in linea con la situazione precedente, ma comunque molto alto in valore assoluto.
L'algoritmo 2 si dimostra il più affidabile dal punto di vista del conteggio del passo, ma è l'algoritmo 3 a presentare un errore assoluto inferiore tra i tre. In più possiamo anche ben notare che la media di conteggio dei passi dell'algoritmo 3 è strettamente vicino a 200.

500 passi

Nell'ultimo test eseguito per il controllo dell'affidabilità dei tre algoritmi, si continua a confermare la situazione precedentemente sottolineata.
L'algoritmo 1 continua a dimostrare la sua inesattezza nel conteggio dei passi, anche se in questo caso il suo errore risulta essere decisamente ridimensionato, aggirandosi poco sopra lo 0.01.

L'algoritmo 2 continua a mostrare, come in tutti i test eseguiti, la sua affidabilità.
Infine, l'algoritmo 3 mostra anch'esso una forte affidabilità, sottolineando un comportamento quasi stabile in tutte le occasioni in cui è stato utilizzato.

Possiamo quindi concludere che a diverse velocità e a diversi conteggi di passi sono l'algoritmo 2 e l'algoritmo 3 a garantire prestazioni migliori, mentre l'algoritmo 1 da come si evince sia dai grafici dei valori dei test (figure 4.1 e 4.3) che dai grafici degli errori (figure 4.2 e 4.3) la sua stabilità è troppo bassa. Possiamo però sottolineare che l'algoritmo 2 potrebbe essere non affidabile quanto l'algoritmo 3, perchè il suo errore assoluto cresce in modo intenso al crescere della velocità.
In conclusione, questi test hanno confermato che l'algoritmo implementato all'interno dell'applicazione è stabile e affidabile, in quanto ha un ottimo comportamento sia a diverse velocità che a diverse distanze percorse. In più continuare ad utilizzare questo algoritmo permette di eseguire una sola interrogazione sul sensore accelerometro, che sarà utilizzato anche per il calcolo dinamico della lunghezza del passo.

4.2 Lunghezza della distanza percorsa

Come ultimo test è stato eseguito il test per la verifica dell'accuratezza del calcolo di misurazione del passo statico e del calcolo dinamico. Nei precedenti capitoli si è ampiamente discusso le differenze tra queste due metodologie di calcolo. In più ricordiamo che come scelta implementativa si è scelto di utilizzare il calcolo della lunghezza dinamica del passo.
Durante la nostra analisi abbiamo preso inconsiderazione il precorrimento di tre distanze: 50, 100 e 500 metri; come nei test precedenti abbiamo eseguito 6 test, i cui valori sono visualizzabili nella tabella 4.3.

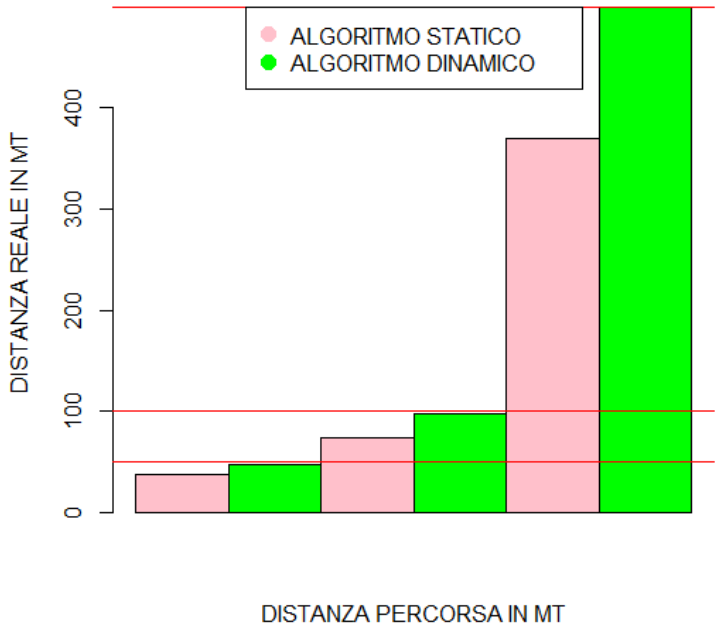

Figura 4.5: Grafico misura della lunghezza percorsa

Nel grafico riportato in figura 4.5 possiamo vedere il comportamento dei due sistemi di calcolo sulle 3 distanze. In questo caso si ritiene che sia intuitivo apprezzare la differenza tra i due calcoli. Infatti, l'algoritmo di calcolo dinamico del passo, indicato nel grafico in verde, si dimostra senza ombra di dubbio più affidabile dell'algoritmo di calcolo del passo statico, ciò è riconducibile alla natura dei valori che compongono il calcolo.
Quindi ciò va a confermare la correttezza nella scelta dell'algoritmo di calcolo del passo dinamico come componente per il calcolo delle calorie bruciate.

	50m		100m		500m	
	Statico	Dinamico	Statico	Dinamico	Statico	Dinamico
1	37	47	74	98	370	499
2	39	47	75	98	371	499
3	37	48	74	97	370	500
4	39	47	74	98	369	499
5	37	47	75	97	370	500
6	37	48	74	98	371	499

Tabella 4.3: Tabella dei dati raccolti per il calcolo della distanza percorsa

Capitolo 5

Conclusioni e sviluppi futuri

Il lavoro di tesi propone lo sviluppo di un sistema di monitoraggio remoto di pazienti affetti da patologie cardiache e portatori di LVDA. Nello specifico, la tesi si è concentrata sulle fasi di rilevazione e monitoraggio delle attività motorie quotidiane e e del dispendio calorico giornaliero.

Possiamo quindi concludere, ricapitolando ciò che abbiamo discusso all'interno di questo lavoro di tesi.
Si è brevemente trattato il contesto di sviluppo dell'applicazione, concentrandoci sull'architettura del sistema Android. Sulla diffusione delle fitness app e dello sviluppo del settore del mobile health. Abbiamo ampiamente approfondito la discussione sui sensori disponibili all'interno degli smartphone Android, incentrando principalmente l'analisi sugli algoritmi di stima del passo. Andando anche ad analizzare il calcolo del dispendio calorico, concentrando la nostra argomentazione sulla stima della lunghezza del passo, esplicando come l'uso della computazione dinamica della lunghezza del passo possa risultare conveniente per la stima della distanza percorsa dall'utente.
Terminata la discussione sullo stato della letteratura relativa ai sensori di Android. Si sono definiti testualmente gli obiettivi del lavoro da svolgere, individuando ed esaminando le specifiche progettuali da portare avanti. Abbiamo discusso le funzionalità già presenti nell'app da ampliare e siamo poi

andati a introdurre la sua architettura, facendo riferimento principalmente ai design pattern utilizzati. Giundendo poi all'individuazione funzionalità da realizzare, che possiamo racchiudere nel conteggio del numero di passi e nella distanza percorsa dall'utente, entrambi i dati utili al raggiungimento dell'obiettivo finale, ovvero il calcolo del dispendio calorico dell'utente.
Nella seconda parte del lavoro di tesi ci siamo concentrati sul discutere le tecnologie utilizzate per l'implementazione dell'applicazione, argomentando la scelta di Android e il perchè si sia preferito l'uso di Kotlin come linguaggio implementativo rispetto al più affermato Java. Si è poi guardato alla realizzazione delle componenti fondamentali quali la UI, per cui oltre a discuterne la struttura abbiamo anche mostrato e discusso le scelte grafiche adottate nell'app, e la struttura del database. Siamo entrati nei dettagli implementativi delle principali funzionalità realizzate, concentrandoci sull'utilizzo dei sensori e argomentando largamente lo sviluppo del contapassi e del calcolo delle calorie bruciate.
In fine, abbiamo esposto i test realizzati per verificare l'attendibilità delle scelte progettuali fatte per la realizzazione dell'applicazione. Portando avanti 2 tipologie di test, i test per la validazione del sistema di stima del passo e il test per la validazione della stima della lunghezza del passo. Mostrando come le decisioni implementative si siano mostrate stabile nell'utilizzo pratico dell'applicazione.

5.1 Sviluppi futuri

Il lavoro di tesi finora discusso, come abbiamo già detto, è un ampliamento di un'applicazione già esistente.
Elencheremo alcune componenti software che potrebbero essere aggiunte.

- Sempre nell'ottica di utilizzo dei sensori, permettere la misurazione del battito cardiaco mediante l'utilizzo dell'apposito sensore Android.

- Lo sviluppo medical side dell'applicazione, per permettere ai medici di inviare tramite app messaggi ai pazienti, quindi potrebbe essere implementato anche un sistema di messaggistica istantanea.
- Ampliare le piattaforme di utilizzo dell'applicazione, quindi creare l'equivalente di quest'applicazione Android anche in IOS.

Ringraziamenti

Ringrazio il professor Di Felice, per avermi dato la possibilità di realizzare una tesi oltre che molto interessante dal punto di vista informatico, anche molto appagante dal punto di vista umano.

Ringrazio la mia famiglia, in particolare i miei genitori per tutte le possibilità che mi state dando e per essermi a fianco nonostante lo sconforto e la sensazione di non farcela di certi momenti. Grazie più di ogni cosa, per supportarmi (e sopportami) in tutte le mie scelte. È grazie anche ai vostri incoraggiamenti se oggi ce l'ho fatta.
Ringrazio Marco, per essermi sempre stato accanto, anche se in un modo tutto tuo.
Ringrazio i miei nonni, per tutto l'amore che mi hanno sempre dimostrato e fatto sentire nonstante la lontananza.
Ringrazio Silvia, ci sarebbe tanto da ringraziare a Silvia. Premetto però che ogni persona ha la sua croce e tu sei la mia. Grazie per esserci stata in ogni caso, per essere stata la compagna di tante avventure (per lo più disavventure) belle e divertenti, ma soprattutto grazie per essere stata la mia ancora nei momenti in cui non sapevo come affrontare le situazioni e pensavo di non farcela. Più di ogni cosa, grazie per aver creduto tanto in me, ci hai creduto più tu che io.

Ringrazio Claudia, abbiamo iniziato quest'esperienza bolognese da quasi sconosciute e adesso non riuscirei ad immaginare la mia vita senza di te. Grazie

per avermi fatto conoscere la vita vista da un'altra prospettiva. Non serve aggiungere altro tu sai.

Ringrazio i miei coinquilini attuali e meno attuali Claudia, Silvia, Stefano, Fabiotti, Giovanni e Alex, per avermi fatto sentire sempre a casa, siete stati una sorta di seconda famiglia. In particolare voglio dire grazie a Stefano per essere stato semplicemente se stesso e per avermi sopportato.

Ringrazio i miei compagni di corso, o meglio i miei amici, con cui in questi 3 intesi anni ho condiviso gioie, ma soprattutto dolori per gli esami. Grazie per le risate, soprattutto per quelle durante le lezioni più pesanti.

Ringrazio Bea, mai avrei pensato che l'università mi avrebbe permesso di conoscere "Una come te". Grazie per avermi appoggiato e incoraggiato nelle mie scelte poco serie, nelle situazioni in cui tutti mi avrebbero scoraggiato.

Ringrazio tutti i miei amici di Roseto, in particolare le mie amiche, anzi le mie amike befane, che sono state in grado, anche se inconsapevolmente, di strapparmi una risata o di riuscire a farmi evadere un po' in un momento che non è stato dei migliori.

Ringrazio la professoressa Prosperi per avermi fatto innamorare dell'informatica e per avermi spinto a continuare su questa strada. È grazie ai suoi stimoli se oggi sono qua.

Ringrazio Dalilina, nonostante tutto, sei sempre nel mio cuoricino e ti voglio bene.

Ringrazio, infine, tutti quelli che in questi tre lunghi anni hanno fatte o fanno ancora parte della mia vita. Vi voglio bene!

Bibliografia

[1] *Android Sensors with Examples*. URL: https://www.tutlane.com/tutorial/android/android-sensors-with-examples.

[2] *Android UI Layouts (Linear, Relative, Frame, Table, ListView, GridView, WebView)*. URL: https://www.tutlane.com/tutorial/android/android-ui-layouts-linear-relative-frame-table-listview-gridview-webview.

[3] Patrizia Chimera. *LVAD: cosâÃ¨ e quando serve un sistema di assistenza ventricolare sinistra*. URL: https://www.benessereblog.it/post/34596/lvad-cose-e-quando-serve-un-sistema-di-assistenza-ventricolare-sinistra.

[4] *Come calcolare le calorie bruciate camminando*. URL: https://dilei.it/benessere/quante-calorie-si-bruciano-camminando-come-calcolarle/625839/.

[5] *Fitness app*. URL: https://www.definitions.net/definition/fitness%20app.

[6] Sashen Govender. *Using the Accelerometer on Android*. URL: https://code.tutsplus.com/tutorials/using-the-accelerometer-on-android--mobile-22125.

[7] Rodolfo Guzzi. *Introduzione ai metodi inversi*. Collana di Fisica e Astronomia, vol 32. Springer, 2012. ISBN: 978-88-470-2494-6. URL: https://link.springer.com/chapter/10.1007%2F978-88-470-2495-3_8.

[8] *Heart Disease*. URL: https://www.cdc.gov/heartdisease/.

[9] *Kotlin vs Java â Why Kotlin is better than Java?* URL: https://www.tinyquip.com/kotlin-vs-java-why-kotlin-is-better-than-java/.

[10] Joschi Kuphal. *A Clear Architecture*. URL: https://jkphl.is/articles/clear-architecture-php/.

[11] *La dolce novitÃ di Android - Android 5.0, Lollipop*. URL: https://www.android.com/intl/it_it/versions/lollipop-5-0/.

[12] Simon LH. *SOLID Principles: Explanation and examples*. URL: https://itnext.io/solid-principles-explanation-and-examples-715b975dcad4.

[13] Jeremy Likness. *Model-View-ViewModel (MVVM) Explained*. URL: https://www.wintellect.com/model-view-viewmodel-mvvm-explained/.

[14] Andrea Lombardo. *PROGETTAZIONE E IMPLEMENTAZIONE DI UNâAPPLICAZIONE MOBILE PER IL MONITORAGGIO DI PAZIENTI CON SCOMPENSO CARDIACO*.

[15] *Malattie Cardiache*. URL: https://labtestsonline.it/conditions/malattie-cardiache.

[16] Alicia Phaneuf. *How mHealth apps are providing solutions to the healthcare market's problems*. URL: https://www.businessinsider.com/mhealth-apps-definition-examples?IR=T.

[17] Dott. Luciano Potena. "et al. Â≪Archivio elettronico per la terapia medica ed interventistica avanzata nello scompenso cardiaco severoÂ≫." In: ().

[18] Ahmad Abadleh - Eshraq Al-Hawari - Esra's Alkafaween - Hamad Al-Sawalqah. "Step Detection Algorithm For Accurate Distance Estimation Using Dynamic Step Length". In: *International conference on Mobile Data Management* ().

[19] *Scompenso cardiaco*. URL: https://www.humanitas.it/malattie/scompenso-cardiaco.

[20] *Sensor*. URL: https://developer.android.com/reference/android/hardware/Sensor.

[21] *Sensor Event*. URL: https://developer.android.com/reference/android/hardware/SensorEvent#values.

[22] Avinash Sharma. *Top Google Play Store Statistics 2019-2020 You Must Know*. URL: https://appinventiv.com/blog/google-play-store-statistics/.

[23] Sagar Sharma. *What Is Accelerometer? How to Use Accelerometer in Mobile Devices?* URL: https://www.credencys.com/blog/accelerometer/.

[24] *SMART HEALTH: COSâÃ E COME CAMBIERÃ IL LAVORO DI MEDICI E PROFESSIONISTI DELLA SALUTE*. URL: https://www.gipo.it/blog/sanita-digitale/smart-health-come-cambiera-sanita/.

[25] *SMARTPHONES MARKET - GROWTH, TRENDS, AND FORECASTS (2020 - 2025)*. URL: https://www.mordorintelligence.com/industry-reports/smartphones-market.

[26] Abby Stassen. *The 10 Best Fitness Apps to Download in 2020*. URL: https://www.verywellfit.com/best-fitness-apps-4173707.

[27] *The Model-View-ViewModel Pattern*. URL: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm.

[28] Neha Vaidya. *What is Android Activity Life Cycle?* URL: https://www.edureka.co/blog/activity-lifecycle/.

[29] Anyi Wang - Xue Ou - Bin Wang. "Improved Step Detection And Step Length Estimation Based On Pedestrian Dead Reckoning". In: ().

[30] *What is a High Pass Filter? Circuit Diagram, Characteristics, and Applications*. URL: https://www.elprocus.com/what-is-a-high-pass-filter-circuit-diagram-characteristics-and-applications/.