Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

STUDIO, IMPLEMENTAZIONE E VALUTAZIONE COMPARATIVA DI ALGORITMI E APPLICAZIONI COMMERCIALI PER IL CONTEGGIO DEI PASSI

Relatore:
Prof.
FEDERICO MONTORI

Presentata da:
FORLANI FRANCESCO

Sessione II

Anno accademico 2024/2025

Sommario

L'elaborato descrive lo sviluppo e la valutazione di un sistema completo per l'analisi del passo umano basato su sensori di movimento. Il lavoro nasce dall'esigenza di disporre di strumenti affidabili per il conteggio dei passi, un'attività centrale in ambito fitness, sanitario e di monitoraggio dell'attività fisica. Tuttavia, la letteratura non offre metodologie standardizzate per confrontare in modo riproducibile le prestazioni degli algoritmi dedicati. Per affrontare questa criticità è stata realizzata un'applicazione Android in linguaggio Kotlin che integra differenti algoritmi di rilevazione dei passi e un'infrastruttura di testing automatizzato basata sulla riproduzione controllata di dati sensoristici. Il sistema permette di ricreare, tramite emulatore, le stesse condizioni registrate sul campo, consentendo una valutazione comparativa equa e replicabile. Le prove sperimentali, condotte su differenti tipologie di camminata e su varie posizioni del dispositivo, hanno consentito di confrontare l'accuratezza degli algoritmi implementati con quella di applicazioni commerciali, evidenziandone prestazioni e limiti. Il lavoro fornisce così un contributo utile alla comprensione e al miglioramento dei sistemi di pedometria basati su smartphone.

Introduzione

In un'epoca in cui il movimento naturale è sempre più sostituito da quello artificiale, la camminata conserva un valore centrale, non solo come attività fisica, ma come misura del benessere e della qualità della vita. Numerosi studi scientifici, a partire dalle linee guida dell'Organizzazione Mondiale della Sanità, evidenziano come il mantenimento di un livello regolare dell'attività fisica riduca il rischio di malattie e migliori le funzioni cognitive. In tale cornice, l'analisi e lo sviluppo di sistemi in grado di monitorare e interpretare il movimento umano assumono un'importanza crescente. Esistono numerosi articoli scientifici riguardanti lo studio di contapassi. In [13] l'autore li suddivide secondo due modalità, in tempo reale e non in tempo reale, e crea inoltre un'applicazione Android in linguaggio Java che implementa gli algoritmi di tipo Real-Time. In [19], invece, vengono descritti ed implementati alcuni algoritmi Non Real-Time nell'applicazione creata in [13]. L'elaborato in [27] descrive un sistema per il testing di applicazioni Android sull'emulatore di Android Studio. Lo scopo di questo elaborato è raffinare ed ampliare gli studi presentati in questi tre lavori. Il lavoro svolto è stato suddiviso in tre differenti fasi:

1. Raccolta dati, registrazione dei valori dell'accelerometro, magnetometro e giroscopio durante diversi tipi di camminata

2. Implementazione di una nuova applicazione Android in linguaggio Kotlin che implementa gli algoritmi descritti in [13] e [19]

3. Confronto dei dati raccolti utilizzando l'architettura proposta in [27]

Nei capitoli successivi verranno descritti nel dettaglio la metodologia adottata, le implementazioni software e i risultati ottenuti dal confronto sperimentale.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

I Pedometri

1.1 Storia e definizione dei pedometri

Un pedometro è un dispositivo che conta il numero di passi effettuati da una persona. Le prime versioni erano di tipo meccanico: un esempio celebre è quello ideato dall'orologiaio svizzero Abraham-Louis Perrelet, che utilizzava un meccanismo a leva e sospensione per registrare i passi. Con il progresso tecnologico, i pedometri sono divenuti dispositivi elettronici, basati su sensori di movimento in grado di rilevare le oscillazioni del corpo umano durante la camminata. Oggi sono integrati in sistemi più complessi, come smartphone e smartwatch, che aggiungono funzionalità di monitoraggio avanzate, come ad esempio la misura della frequenza cardiaca, delle calorie bruciate e della distanza percorsa. I sensori impiegati sono generalmente di tipo **MEMS** (Micro-Electro-Mechanical Systems), dispositivi miniaturizzati che combinano componenti meccaniche ed elettroniche su scala micrometrica. Dal punto di vista fisico, trasformano stimoli meccanici in segnali elettrici digitali. I sensori MEMS integrati negli smartphone presentano inevitabilmente fenomeni di *bias* statico, rumore casuale e deriva termica, che influenzano la stima della velocità o della posizione ottenuta tramite integrazione numerica dei dati. Nonostante ciò, la loro precisione è generalmente sufficiente per applicazioni come il rilevamento dei passi. I sensori maggiormente utilizzati nei pedometri moderni appartengono a tre categorie principali:

1. **Accelerometri**: misurano l'accelerazione lineare del corpo, ossia la variazione di velocità nel tempo, lungo uno o più assi (solitamente x, y, z). In un accelerometro MEMS, una massa di prova sospesa è collegata al substrato tramite micro-molle. Quando il dispositivo subisce un'accelerazione, la massa

si sposta leggermente, modificando la capacità elettrica tra elettrodi fissi e mobili. Il circuito integrato converte tale variazione di capacità in un segnale proporzionale all'accelerazione;

2. **Giroscopi**: misurano la velocità angolare del dispositivo, cioè quanto rapidamente e in quale direzione esso ruota intorno ai propri assi, di solito nelle tre dimensioni spaziali. Nei MEMS, il principio di funzionamento si basa sull'effetto Coriolis: una massa vibrante subisce una forza proporzionale alla velocità di rotazione, producendo una deviazione misurabile che viene convertita in segnale elettrico;

3. **Magnetometri**: misurano l'intensità e la direzione del campo magnetico terrestre, fornendo informazioni sull'orientamento del dispositivo rispetto al campo magnetico. Sono anch'essi implementati, in genere, lungo tre assi (x, y, z).

Va precisato che la qualità dei sensori può variare sensibilmente da un dispositivo all'altro, anche quando si tratta dello stesso tipo di sensore. Ciò dipende da fattori come la tecnologia costruttiva, la calibrazione di fabbrica e i filtri applicati dal sistema operativo. Tuttavia, la combinazione di più sensori consente di ottenere una precisione complessiva sufficiente per la maggior parte delle applicazioni. In ambito pedometrico, l'obiettivo principale è individuare un modello algoritmico che garantisca la massima accuratezza possibile, indipendentemente dal dispositivo utilizzato. Lo scopo di questo elaborato è confrontare diverse implementazioni — sia commerciali sia open-source — e analizzarne i risultati per trarre conclusioni sulla loro affidabilità e robustezza.

1.2 Stato dell'arte

Negli ultimi anni, la diffusione capillare di smartphone e dispositivi indossabili dotati di sensori inerziali ha favorito un notevole incremento della ricerca sui pedometri digitali. In letteratura sono stati proposti numerosi approcci per il rilevamento automatico dei passi, spesso con l'obiettivo di migliorarne l'accuratezza e l'affidabilità in condizioni d'uso reali. Nonostante i progressi tecnologici, il problema della corretta rilevazione del passo umano resta aperto. L'accuratezza dei pedometri dipende infatti non solo dalla qualità dei sensori, ma anche dall'algoritmo impiegato per interpretare i segnali. Le differenze tra modelli commerciali e implementazioni

accademiche derivano proprio da questa componente algoritmica, che rappresenta il cuore del sistema. Le sezioni seguenti presentano un excursus delle principali tipologie di algoritmi proposti in letteratura e un riepilogo dei lavori comparativi più significativi, fino a introdurre l'approccio metodologico adottato in [28], che costituisce la base sperimentale di questo elaborato.

1.2.1 Background e principi fisici

Secondo [10], un passo è definito come un evento di movimento che produce un segnale di accelerazione con caratteristiche specifiche di ampiezza e frequenza, ovvero, un picco positivo ed uno negativo attorno ad un punto di equilibrio. Il pattern che si forma corrisponde al movimento del piede che si stacca dal suolo, si muove in avanti e poi ritorna a toccare il terreno. Dal punto di vista fisico, il passo è rilevato quando la magnitudine dell'accelerazione supera una certa soglia predefinita, che può variare in base all'implementazione dell'algoritmo utilizzato per il rilevamento.

1.2.2 Algoritmi in letteratura

In letteratura sono stati proposti numerosi algoritmi per il conteggio dei passi. Negli studi, i dati rilevati dai sensori dei dispositivi vengono generalmente pre-elaborati attraverso filtri che hanno l'obiettivo di ridurre il rumore e migliorare la qualità del segnale. Questo viene poi analizzato per identificare i pattern caratteristici del passo umano. Gli algoritmi possono venire classificati in base a diversi criteri, come verrà illustrato in seguito. Secondo [19] e [14], una prima distinzione può essere fatta tra algoritmi di tipo Real-Time e Non Real-Time. Gli algoritmi Real-Time sono progettati per elaborare i dati in tempo reale, mentre quelli Non Real-Time richiedono dati precedenti a quelli attuali per effettuare il calcolo. Di seguito, verranno descritti i principali approcci utilizzati in letteratura per il conteggio dei passi, in modo da fornire un quadro generale delle tecniche più comuni.

Filtri

Lo studio [3] propone un algoritmo di filtraggio basato su un filtro passa-basso. Lo scopo, in questo caso, è lasciare passare le componenti lente del segnale, che sono quelle più rilevanti per il rilevamento dei passi, eliminando le componenti ad alta frequenza che rappresentano il rumore. La soglia che determina l'attenuazione del segnale è chiamata frequenza di taglio. Nello studio appena citato, la frequenza di

taglio è stata impostata a 20 Hz. Viene inoltre dichiarato che le informazioni sul movimento umano si verificano sotto la tale soglia. Molti altri studi utilizzano questo tipo di filtro, come ad esempio [7], [9], [20], [22]. Le differenze nelle implementazioni risiedono principalmente nelle frequenze di taglio scelte. Relativamente al filtro passa-basso, una variante comune è il tipo Butterworth che permette di ottenere una risposta in banda passante più piatta possibile come descritto in [11]. In particolare, questo studio propone un filtro con frequenza di taglio adattiva, che varia in base alla velocità della camminata.

Oltre al filtro passa-basso, è utilizzato in alternativa il filtro passa-alto, come in [1]. A differenza del primo, questo filtro lascia passare le componenti ad alta frequenza del segnale, eliminando quelle a bassa frequenza. L'obiettivo è rimuovere le componenti statiche del segnale, come la gravità, che possono influenzare negativamente il rilevamento dei passi.

Un alternativa ai precedenti, presente frequentemente in letteratura, è il filtro passa-banda. Questo lascia passare solo le componenti del segnale comprese tra due frequenze di taglio, eliminando sia le componenti a bassa che ad alta frequenza. Un esempio di questo approccio è lo studio [21], dove viene usato un filtro passa-banda di tipo butterworth per rimuovere sia le componenti lente che quelle veloci del segnale.

Un ulteriore approccio, spesso descritto negli articoli analizzati, è l'uso della matrice di rotazione, come lo studio [29], [8], [26] o [18]. Riguardo al funzionamento del filtro, questi articoli utilizzano accelerometro, magnetometro e giroscopio in modo da calcolare una matrice che proietta i dati nel sistema di riferimento del mondo reale ed ottenere un segnale più pulito. Un altro studio analizzato [15] propone un approccio differente, necessita soltanto dei dati dell'accelerometro ed introduce una trasformazione delle coordinate per rendere il segnale indipendente dall'orientamento del dispositivo.

L'ultimo tipo di filtro preso in considerazione come tra i più comuni in letteratura è il filtro di Kalman, viene utilizzato ad esempio in [25] per ricostruire i passi mancanti che vengono nascosti a causa dell'oscillazione del braccio. In generale, il filtro di Kalman è un algoritmo ricorsivo che stima lo stato interno di un sistema dinamico a partire da una serie di misurazioni rumorose.

Rilevamento dei passi

Si possono classificare diversi approcci per il rilevamento dei passi in base al dominio di analisi del segnale. Un primo gruppo di metodi analizza il segnale

nel dominio del tempo. Indubbiamente, l'approccio più comune in questo caso, ed in generale per il rilevamento dei passi, è l'individuazione dei picchi nel segnale dell'accelerometro. Questo metodo si basa sull'osservazione che durante la camminata, l'accelerazione verticale del corpo umano mostra un pattern caratteristico di picchi positivi e negativi. La maggioranza degli studi analizzati utilizza questo metodo, come ad esempio [3], [15] o [1]. Esistono tuttavia diverse varianti di questo approccio. Ad esempion in [9] viene proposto un algoritmo che utilizza l'intersezione dei picchi con l'asse delle ascisse, ovvero utilizza il giroscopio per individuare i punti in cui il segnale angolare cambia segno. Ogni volta che il giroscopio cambia segno si considera il passo iniziato o terminato.

Un diverso approccio è l'analisi del dominio della frequenza. In questo caso, il segnale viene di solito trasformato in frequenza utilizzando la Trasformata di Fourier (FFT). Un esempio di questo approccio lo abbiamo in [21] dove la FFT viene utilizzata per individuare le frequenze dominanti del passo. Questo articolo descrive inoltre un algoritmo denominato autocorrelazione, che calcola la correlazione del segnale con sè stesso, ovvero dopo esser filtrato sul segnale viene calcolata una funzione che misura la somiglianza del segnale con sè stesso nel tempo. I picchi periodici dell'autocorrelazione corrispondono ai passi.

Gli algoritmi più moderni, invece, utilizzano tecniche di machine learning per il rilevamento dei passi. In questo approccio, non vengono usate soglie o regole fisse, ma il sistema viene addestrato su un insieme di dati etichettati per imparare a riconoscere i pattern caratteristici del passo umano. Esistono già applicazioni commerciali che utilizzano questo approccio, come Google Fit che rileva automaticamente l'attività svolta ed elabora i dati dei sensori tramite modelli di machine learning.

1.2.3 Lavori comparativi

I sensori integrati negli smartphone sono tendenzialmente abbastanza precisi da permettere il rilevamento dei passi attraverso algoritmi come quelli precedentemente descritti. Il nodo cruciale risiede però nella scelta dell'algoritmo più adatto per interpretare i dati. A tal proposito, sono stati condotti numerosi studi comparativi per valutare l'accuratezza di diversi algoritmi di conteggio dei passi, con l'obiettivo di identificare le soluzioni più efficaci e robuste e guidare verso lo sviluppo di metodi sempre più performanti. I tipi di confronto presenti negli articoli analizzati sono di diversi tipi. In [4] viene valutata l'influenza della posizione del dispositivo e della velocità di camminata sul conteggio dei passi, utilizzando quattro diversi tipi di

dispositivi commerciali. Velocità e passo sono regolati con metronomo e cordicella alle caviglie, l'ambiente è controllato su percorso a terra. I partecipanti vengono ripresi in modo da poter contare i passi manualmente e confrontarli con quelli rilevati dai dispositivi. In [24] viene analizzata l'accuratezza del rilevamento dei passi e del riconoscimento delle attività di sette monitor di attività fisica, in vari contesti di utilizzo, come camminata indoor, outdoor, scale, viene anche cambiata la velocità di camminata. Lo studio [12] valuta l'accuratezza di tre app di pedometria su smartphone Android. I test vengono effettuati su tapis roulant alternando le posizioni del dispositivo (tasca, fascia alla vita, braccio) e confrontati con i risultati in "free-living" dei partecipanti. Per il conteggio in laboratorio, è un ricercatore che si occupa del conteggio manuale dei passi. Le tre app vengono eseguite in contemporanea sullo stesso dispositivo in modo che il riferimento della camminata sia identico per tutte e tre. Generalmente, gli studi comprendono un numero limitato alle poche decine di partecipanti, il più solido tra i precedenti rimane quest'ultimo citato con 48 soggetti coinvolti. Come viene evidenziato infatti in [23], su 25 studi analizzati, la media è di circa 20 partecipanti per studio. Per quanto riguarda le modalità di confronto fra i diversi pedometri, l'approccio è simile. Viene effettuato un conteggio manuale dei passi, che funge da riferimento per valutare l'accuratezza dei dispositivi; le camminate vengono svolte in ambienti controllati ed i risultati vengono confrontati. Un approccio alternativo è quello proposto in [28], che costituisce la base sperimentale di questo elaborato. In questo studio, viene sviluppata un'architettura di testing automatizzato per il confronto di diverse applicazioni Android di pedometria tramite emulatore, sia commerciali sia open-source. In [14] viene inoltre effettuato un confronto tra diversi algoritmi implementati in un'applicazione Android open-source, con la peculiarità di utilizzare dati registrati in precedenza per il confronto, piuttosto che camminate effettuate in tempo reale.

1.2.4 Motivazioni e obiettivi

La ricerca e la validazione di algoritmi per il conteggio dei passi rappresentano un nodo cruciale nello sviluppo di strumenti di monitoraggio dell'attività fisica affidabili. Le procedure sperimentali tradizionali si basano su sessioni ripetute di cammino in condizioni controllate. Nonostante l'adozione di protocolli rigorosi, ogni prova resta soggetta a variazioni intrinseche: differenze nella cadenza, nella lunghezza del passo, nella posizione del dispositivo o nella stabilità del segnale, che introducono rumore sperimentale e limitano la capacità di isolare il contributo dell'algoritmo rispetto a

quello della variabilità umana. Inoltre, la raccolta di dati reali è onerosa dal punto di vista temporale e logistico: per testare efficacemente molte configurazioni, parametri e versioni di algoritmo servirebbero molteplici ripetizioni per ciascuna combinazione, con costi sperimentali rapidamente insostenibili. Per affrontare queste limitazioni, il lavoro qui presentato propone e adotta un meccanismo di sensor injection: una pipeline che registra tracce grezze dei sensori (accelerometro, giroscopio, magnetometro e timestamp associati) durante camminate reali e consente di riprodurle su emulatori Android. Questo approccio trasforma sequenze di movimento reali in dataset riutilizzabili, somministrabili a molteplici implementazioni di algoritmi di individuazione dei passi. Riprodurre lo stesso segnale consente di confrontare algoritmi sulla base di uno stesso input di riferimento, eliminando la componente di variazione dovuta alla ripetizione delle prove dal vivo e permettendo di attribuire con maggiore certezza le differenze di output alle scelte progettuali degli algoritmi stessi. Gli obiettivi principali dell'approccio sono tre:

1. aumentare il rigore metodologico dei confronti: usando dati identici per ogni test, si migliora la validità interna degli esperimenti e si riduce il rischio che le conclusioni vengano contaminate da differenze nelle esecuzioni;

2. ampliare la scalabilità sperimentale: l'automazione della riproduzione dei segnali permette di eseguire rapidamente grandi batterie di test senza la necessità di riunire soggetti o ripetere camminate reali ogni volta;

3. favorire la riproducibilità: i dataset registrati possono essere documentati, versionati e condivisi, offrendo alla comunità la possibilità di replicare valutazioni e confronti con i medesimi input originali.

Implementando questo metodo l'obiettivo è quindi accelerare e rendere più rigorose le valutazioni comparative degli algoritmi contapassi, ridurre il carico sperimentale e aumentare la solidità delle conclusioni ottenute, contribuendo così allo sviluppo di soluzioni più accurate e affidabili per il monitoraggio dell'attività fisica.

Capitolo 2

Architettura del progetto

In questo capitolo viene presentata l'architettura generale del sistema sviluppato, descrivendone i principali componenti e le loro interazioni ad un livello alto di astrazione. L'obiettivo è fornire una visione d'insieme dell'ecosistema sperimentale realizzato per l'analisi e la validazione di algoritmi di rilevamento dei passi pedonali, rimandando ai capitoli successivi la trattazione dettagliata delle singole implementazioni. L'infrastruttura complessiva è articolata in tre moduli principali, che cooperano tra loro per consentire la raccolta, la riproduzione e l'analisi dei dati sensoristici:

1. **Applicazione Android StepLab**, che integra diversi algoritmi di conteggio dei passi;

2. **Applicazione Android MotionTracker**, utilizzata per registrare i segnali grezzi provenienti dai sensori durante camminate reali e salvarli online per la successiva elaborazione;

3. **Iniezione dei dati**, che consente di riprodurre negli emulatori Android i tracciati registrati, simulando in modo controllato le camminate e permettendo di confrontare le prestazioni di diverse applicazioni di pedometria.

Ciascun modulo svolge un ruolo distinto ma complementare, contribuendo alla creazione di un ambiente sperimentale riproducibile per lo studio comparativo degli algoritmi di step detection. Nei paragrafi successivi verrà descritto come tali componenti si integrano e quali tecnologie ne supportano il funzionamento.

2.1 Panoramica dell'architettura

Nel sistema concepito, ogni modulo interagisce con gli altri in maniera complementare per realizzare un flusso di lavoro coerente. Il punto di partenza è l'applicazione **MotionTracker**, che viene eseguita su dispositivi reali per registrare i dati grezzi dei sensori durante camminate effettuate da volontari. Questi hanno installato l'app sul proprio smartphone personale oppure, nel caso non fossero in possesso di un dispositivo Android, viene fornito loro uno smartphone dedicato. I dati raccolti vengono salvati su uno spazio di archiviazione online, accessibile successivamente per l'elaborazione. Le sessioni di registrazione sono state condotte su camminate reali in ambienti esterni (parchi, strade cittadine), seguendo un protocollo controllato per garantire l'uniformità dei dati raccolti. Ogni registrazione corrisponde a una camminata di 50 passi, conteggiati manualmente per fungere da riferimento. I dati raccolti vengono poi utilizzati nel modulo di **Iniezione dei dati**, che consente di riprodurre le tracce registrate. Viene utilizzato un emulatore Android, parte integrante di Android Studio, che permette di simulare il funzionamento di un dispositivo reale. Lo script di iniezione carica i dati dallo spazio di archiviazione online, *Firebase Storage*, e li invia all'emulatore, che li interpreta come se fossero stati acquisiti in tempo reale dai sensori. In questo modo, è possibile testare diverse applicazioni di pedometria in condizioni identiche, utilizzando gli stessi segnali di input. Tra le applicazioni testate, vi è **StepLab**. Questa integra diversi algoritmi di conteggio dei passi, implementati in linguaggio Kotlin, e consente di visualizzare i risultati in tempo reale. L'applicazione è stata sviluppata appositamente per questo progetto, con l'obiettivo di confrontare le prestazioni degli algoritmi open-source con quelle di applicazioni commerciali, utilizzando i dati iniettati dall'emulatore. Nelle sezioni successive verranno descritti ad alto livello i singoli moduli che compongono l'architettura.

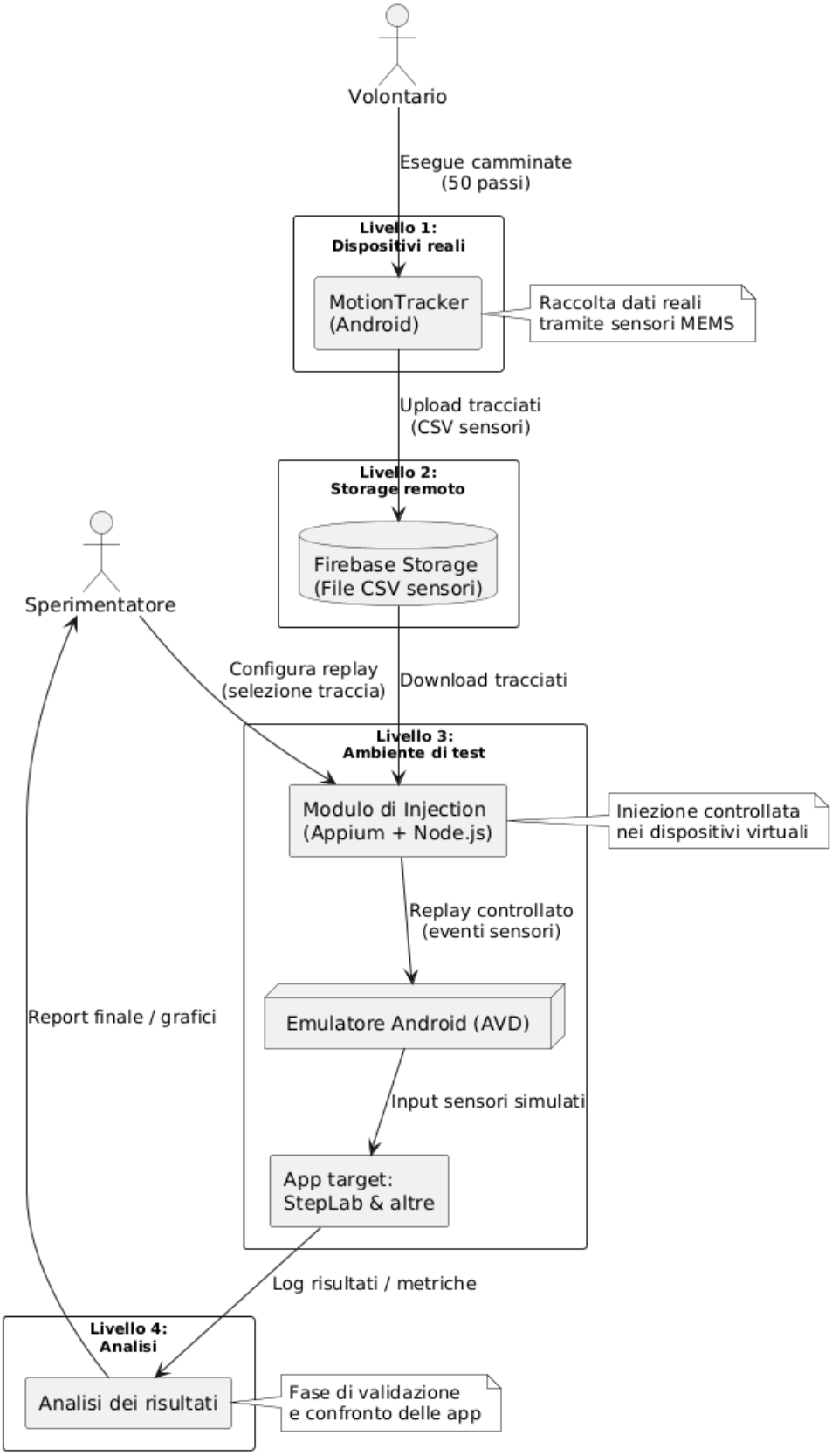

Figura 2.1: Architettura del sistema: diagramma dei componenti e flusso dei dati.

2.2 Contributi

L'applicazione Android **StepLab** nasce prendendo come riferimento le implementazioni in Java presentate in [13, 19], utilizzate esclusivamente come base di requisiti e catalogo degli algoritmi da considerare. La versione qui presentata è stata riscritta da zero in Kotlin e mantenendo le caratteristiche comuni e non funzionali dei lavori precedenti, ma con un'implementazione nuova e indipendente, introducendo correzioni, migliorie e funzionalità aggiuntive specifiche per questo progetto. Per quanto riguarda gli algoritmi, StepLab implementa tutte le famiglie considerate in [13, 19], introducendo modifiche e ottimizzazioni ove necessario. In particolare, invece dell'algoritmo denominato "autocorrelazione" in [19] (implementato in Matlab e non integrato nell'app), StepLab integra l'algoritmo di autocorrelazione descritto in [21]. Per quanto riguarda l'applicazione **MotionTracker**, al momento dell'inizio del progetto era già disponibile in [5]. Tuttavia, sono state apportate alcune modifiche per adattarla alle esigenze specifiche di questo lavoro, come l'integrazione con *Firebase Storage* utilizzato per la memorizzazione dei dati e l'ottimizzazione delle funzionalità di registrazione per garantire la coerenza e l'affidabilità dei dati raccolti, oltre a correzioni di bug che non permettevano un corretto funzionamento dell'applicazione.

2.3 StepLab

StepLab è un'applicazione Android completa per sperimentare e valutare algoritmi di rilevamento dei passi pedonali. Consente di registrare dati dei sensori, eseguire il rilevamento in tempo reale con algoritmi e opzioni di filtraggio configurabili e confrontare le prestazioni di diverse configurazioni su test registrati in precedenza. L'applicazione è pensata principalmente per la ricerca e la didattica, permettendo il confronto tra diversi tipi di configurazioni e visualizzando l'output di queste.

2.3.1 Ruolo nel sistema

Nel contesto dell'architettura complessiva di questo progetto, StepLab svolge una duplice funzione:

1. **Implementazione e confronto di algoritmi open-source**: acquisizione di tracce sensoristiche ed applicazione di algoritmi di step detection che offrono un'alternativa alle soluzioni commerciali, permettendo il confronto diretto tra implementazioni open-source e proprietarie;

2. **Strumento di validazione del sistema di iniezione**: come sarà illustrato nei capitoli successivi, StepLab offre funzionalità per l'analisi delle registrazioni di camminate, consentendo di valutare l'accuratezza del sistema di iniezione dei dati, ovvero verificare le corrispondenze tra i risultati ottenuti con l'architettura descritta in questo elaborato e quelli che si otterrebbero senza di essa.

2.3.2 Materiali e tecnologie utilizzate

L'applicazione è sviluppata interamente in **Kotlin** e la build è gestita tramite **Gradle Kotlin DSL**, sfruttando un catalogo versioni centralizzato per la gestione delle dipendenze.

L'interfaccia utente è realizzata con **layout XML tradizionali**, supportati da **AppCompat** e **Material Components**. Questa scelta, preferita a Jetpack Compose, assicura maggiore compatibilità con tutte le versioni di Android supportate e con le librerie preesistenti, mantenendo una netta separazione tra logica e presentazione e garantendo performance prevedibili nelle sezioni a tempo reale. La rappresentazione grafica dei dati in tempo reale avviene tramite **MPAndroidChart**, una potente libreria open source per grafici interattivi su Android. La persistenza è affidata al framework **Room ORM** e supporta migrazioni di schema additive, garantendo l'evoluzione del database senza perdita di dati utente. L'elaborazione del segnale utilizza librerie specializzate: **JTransforms** per la FFT (Fast Fourier Transformation) necessaria all'algoritmo di autocorrelazione, e **iirj** per i filtri digitali IIR (Infinite Impulse Response) di tipo Butterworth.

2.3.3 Architettura interna

L'applicazione è organizzata in quattro package funzionali, ciascuno con una responsabilità ben definita che contribuisce alla modularità e manutenibilità del codice:

Package UI Gestisce l'intero livello di presentazione e interazione con l'utente. Include le schermate per la configurazione degli algoritmi, la registrazione dei test, l'esecuzione del pedometro in tempo reale e la visualizzazione dei confronti tra configurazioni. L'interfaccia è realizzata con layout XML tradizionali, garantendo compatibilità e separazione tra logica e presentazione.

Package Algorithms Costituisce il nucleo computazionale dell'applicazione. Implementa gli algoritmi di step detection, i filtri digitali per il condizionamento del segnale e le operazioni matematiche di supporto. Gestisce sia l'elaborazione in tempo reale che quella batch su tracce registrate.

Package Data Si occupa della persistenza dei dati attraverso il framework Room. Memorizza i test acquisiti con i relativi metadati (frequenza di campionamento, durata, passi reali) e gli snapshot delle configurazioni utilizzate nei confronti.

Package Utils Fornisce funzionalità di supporto per la conversione tra formati (JSON e CSV), permettendo l'import di tracce da fonti esterne e l'export dei dati registrati per analisi successive. Garantisce la compatibilità con i file generati da MotionTracker [5].

2.3.4 Principali funzionalità

L'applicazione offre un ambiente per la sperimentazione e la valutazione degli algoritmi di rilevamento dei passi. Le principali funzionalità possono essere riassunte come segue:

- **Raccolta e gestione dei test**: acquisizione e archiviazione di tracce sensoristiche.
- **Configurazione ed esecuzione degli algoritmi**: possibilità di selezionare filtri e strategie di rilevamento, eseguire analisi in tempo reale e visualizzare graficamente l'andamento dei segnali.
- **Confronto e analisi dei risultati**: valutazione comparativa delle diverse configurazioni di algoritmo mediante grafici sovrapposti.
- **Interoperabilità**: import ed export dei dati in formati standard (`JSON`, `CSV`), garantendo la compatibilità con i tracciati raccolti tramite **MotionTracker**.

2.4 MotionTracker

MotionTracker [5] è un'applicazione open-source per la registrazione dei dati di movimento, progettata per funzionare con smartphone Android ma anche con dispositivi bluetooth.

2.4.1 Ruolo nel sistema

Nel contesto dell'architettura complessiva di questo progetto, MotionTracker svolge il ruolo di strumento di acquisizione dati. Viene utilizzata per registrare i segnali grezzi provenienti dai sensori (in particolare accelerometro, giroscopio e magnetometro) durante camminate reali. I dati raccolti vengono salvati su *Firebase Storage*, accessibile successivamente per la riproduzione tramite il modulo di iniezione dei dati. In questo modo, MotionTracker fornisce le tracce di movimento necessarie per testare e confrontare le prestazioni delle diverse applicazioni di pedometria.

2.4.2 Materiali e tecnologie utilizzate

MotionTracker è sviluppata in **Kotlin** con build system basato su **Gradle Kotlin DSL**. L'interfaccia utente adotta **Jetpack Compose** per la creazione dichiarativa delle schermate e i **Material 3 Components** per garantire coerenza visiva con le linee guida Android moderne. La navigazione tra le schermate è gestita tramite **Navigation Compose**, mentre la gestione dello stato dell'interfaccia si avvale di **ViewModel** e **LiveData**. L'iniezione delle dipendenze è affidata a **Dagger Hilt**, che permette una configurazione modulare e testabile dei componenti applicativi. Per la persistenza remota e la sincronizzazione delle tracce registrate, l'applicazione si integra con i servizi **Firebase Storage** (archiviazione file) e **Firebase Realtime Database**. L'applicazione supporta inoltre la connessione a dispositivi esterni tramite **Bluetooth Low Energy**, utilizzando la libreria **RxAndroidBle** per la gestione reattiva delle comunicazioni BLE e la libreria proprietaria **mdslib** per l'integrazione con sensori Movesense. Questi ultimi sono dispositivi indossabili, sviluppati da Suunto, progettati appositamente per la rilevazione accurata dei dati inerziali. Combina nove assi, tre per ogni sensore: accelerometro, giroscopio e magnetometro. La piattaforma Movesense è programmabile e include un *SDK* che consente di personalizzare il comportamento dei sensori e di accedere ai dati registrati tramite *API* dedicate.

2.4.3 Architettura interna

L'applicazione è organizzata in package funzionali che separano le diverse responsabilità:

Package screen Contiene le schermate dell'interfaccia utente realizzate con Jetpack Compose. Include i composable per la registrazione dei dati, la gestione del ciclo di vita della registrazione e i dialoghi di conferma. Integra la logica di navigazione e la comunicazione con il servizio di monitoraggio in background.

Package sensor Incapsula lo strato di acquisizione dei segnali. Normalizza le letture hardware e le consegna al resto del sistema.

Package data Realizza il modello dati dell'applicazione, funge da strato di astrazione tra l'acquisizione e la persistenza.

Package navigation Definisce le destinazioni di navigazione dell'applicazione e coordina il flusso tra le diverse schermate attraverso Navigation Compose.

Package bluetooth Implementa lo strato di comunicazione con dispositivi esterni tramite Bluetooth dell'applicazione, responsabile della scoperta e gestione dei dispositivi esterni.

Package di Definisce i moduli di iniezione delle dipendenze utilizzando Dagger Hilt, fornisce i contesti e i componenti condivisi tra i diversi layer, mantenendo le dipendenze disaccopiate e riusabili.

2.4.4 Principali funzionalità

Lato utente, l'applicazione offre principalmente funzionalità relative alla registrazione dei dati di movimento:

- **Configurazione camminata**: mostra un form per inserire i metadati della camminata (età, altezza, peso, sesso, posizione del dispositivo, tipo di camminata) prima di iniziare la registrazione.
- **Registrazione dati sensoristici**: acquisizione in tempo reale dei dati provenienti dai sensori integrati dello smartphone o da dispositivi esterni collegati via Bluetooth. Mostra un dialog che invita l'utente a effettuare 50 passi e permette di terminare la registrazione.
- **Associazione Bluetooth**: connessione a sensori esterni che permettono di ampliare le possibilità di raccolta informazioni oltre i sensori integrati.

2.5 Iniezione dei dati

A differenza delle altre componenti del sistema, il modulo di iniezione dei dati non è un'applicazione Android, ma uno script javascript che viene eseguito su un computer host per interagire con l'emulatore Android.

2.5.1 Ruolo nel sistema

Il modulo di iniezione dei dati svolge il ruolo cruciale di ponte tra le tracce registrate con MotionTracker e le applicazioni di pedometria eseguite nell'emulatore Android (inclusa StepLab). Pertanto, il suo ruolo nell'architettura complessiva è fondamentale per consentire la riproduzione controllata delle camminate registrate.

2.5.2 Materiali e tecnologie utilizzate

Il modulo di iniezione è sviluppato in **Node.js** e utilizza il framework **Appium** con driver **UiAutomator2** per comunicare con gli emulatori Android (**AVD**) tramite la console sensori. L'automazione è realizzata con la libreria **WebdriverIO**, che consente di inviare comandi come `sensor set acceleration/gyroscope/magnetic-field` per riprodurre fedelmente le tracce raccolte. L'accesso ai file CSV avviene sia in locale sia da **Firebase Storage**, grazie al **Firebase Admin SDK**, mentre **dotenv** gestisce la configurazione tramite variabili d'ambiente e **csv-parse** si occupa del parsing dei dati. L'ambiente richiede inoltre un'installazione di **Android SDK/Emulator**, gli **APK** delle app target e l'avvio di Appium con i permessi necessari.

2.5.3 Flusso di esecuzione

Lo script di iniezione esegue in modo automatizzato la riproduzione delle tracce sensoristiche registrate da MotionTracker sugli emulatori Android. Dopo l'avvio, il sistema consente di selezionare la sorgente dei dati (file locale o **Firebase Storage**) e scarica automaticamente i file CSV necessari. Successivamente, se necessario, Appium e WebdriverIO gestiscono le interazioni con l'interfaccia dell'app target per prepararla alla ricezione dei dati. Una volta pronta, lo script legge i dati sensoristici dal file CSV e li invia all'emulatore. La riproduzione avviene in tempo reale, mantenendo la stessa temporizzazione dei dati originali. Al termine della riproduzione di un file, lo script interagisce con l'utente. Viene data la possibilità di salvare i risultati ottenuti dall'applicazione (ad esempio i conteggi dei passi), procedere con la riproduzione

di un nuovo file oppure terminare il processo. In questo modo è stato possibile automatizzare (seppur parzialmente) la pipeline di esecuzione dei test.

Modulo	**Piattaforma**	**Ruolo principale**
StepLab	Android (Kotlin)	Analisi e confronto di algoritmi di step detection
MotionTracker	Android (Kotlin)	Registrazione dei dati sensoristici e salvataggio su Firebase
Iniezione dati	Node.js / Appium	Riproduzione controllata delle tracce su emulatore

Tabella 2.1: Sintesi dei moduli software del sistema

Capitolo 3

Implementazione di StepLab

In questo capitolo viene presentata l'implementazione dettagliata dell'applicazione Android **StepLab**, descritta ad alto livello nel Capitolo 2, e resa disponibile open source in [6]. Vengono illustrati i componenti software, le logiche operative, i flussi applicativi e le principali scelte progettuali adottate durante lo sviluppo. L'applicazione è organizzata in quattro package principali, come già introdotto nel Capitolo 2. In questa sezione se ne approfondisce la struttura interna.

3.1 UI

Questo package gestisce l'interfaccia utente e l'interazione con l'utente. Contiene le *activity* e i *fragment* per le diverse schermate. Questi sono organizzati in sottopackage, separati in base alle loro responsabilità in modo da rendere il codice il più coeso possibile.

3.1.1 Main

Il package `main` contiene due classi fondamentali per l'avvio e la gestione dell'applicazione: `StepLabApplication` e `MainActivity`.

La prima rappresenta il punto di ingresso dell'intera applicazione. La sua responsabilità principale è l'inizializzazione e la gestione del database Room.

Nel metodo *onCreate()*, eseguito una sola volta all'avvio dell'applicazione prima di qualsiasi *activity*, viene creata l'istanza del database `tests.db` utilizzando il *pattern Singleton*. Questa scelta architetturale garantisce che:

- il database venga inizializzato una sola volta durante l'intero ciclo di vita dell'applicazione;

- si evitino istanze multiple e potenziali *race condition*;
- l'accesso al database sia disponibile da qualsiasi punto dell'applicazione tramite `StepLabApplication.database`.

Durante l'inizializzazione vengono registrate le migrazioni del database, permettendo l'evoluzione dello schema senza perdita di dati utente. Questa architettura separa la logica di inizializzazione globale dalla logica UI delle *activity*, migliorando la manutenibilità e la testabilità del codice.

La `MainActivity` funge da *hub* centrale dell'applicazione. All'avvio, verifica in una *coroutine* su *thread* IO se esistono dati salvati, abilitando o disabilitando di conseguenza le funzioni che ne dipendono (confronto, *export* e visualizzazione salvataggi).

La navigazione è implementata tramite *intent* espliciti verso le seguenti schermate:

- contapassi *live*;
- registrazione di un nuovo *test*;
- confronto configurazioni;
- *export* dei dati;
- visualizzazione dei salvataggi di confronti.

La funzionalità più articolata presente in questa *activity* è l'*import* di file esterni. Non dispone di una schermata dedicata, ma viene avviata tramite un pulsante nella MainActivity. Il flusso operativo è il seguente:

1. Viene lanciato un `ActivityResultLauncher` che apre il *file picker* Android, utilizzando il contratto `GetMultipleContents()` per permettere la selezione di più file contemporaneamente.
2. Un *dialog* non cancellabile tiene informato l'utente sul progresso dell'importazione.
3. Per ogni file selezionato, l'intero contenuto viene letto come stringa e processato in base al formato:
    - se il file ha estensione `.csv`, il contenuto viene convertito in formato JSON tramite la classe di utilità `CsvToJsonConverter`;

- altrimenti, viene tentato direttamente il *parsing* del contenuto come JSON.

4. Una copia del JSON viene salvata nello *storage* interno con un *timestamp* come nome del file.

5. I metadati del *test* vengono inseriti nel database Room.

6. Al termine, il *dialog* mostra il riepilogo dell'importazione con il numero di file importati con successo ed eventuali errori.

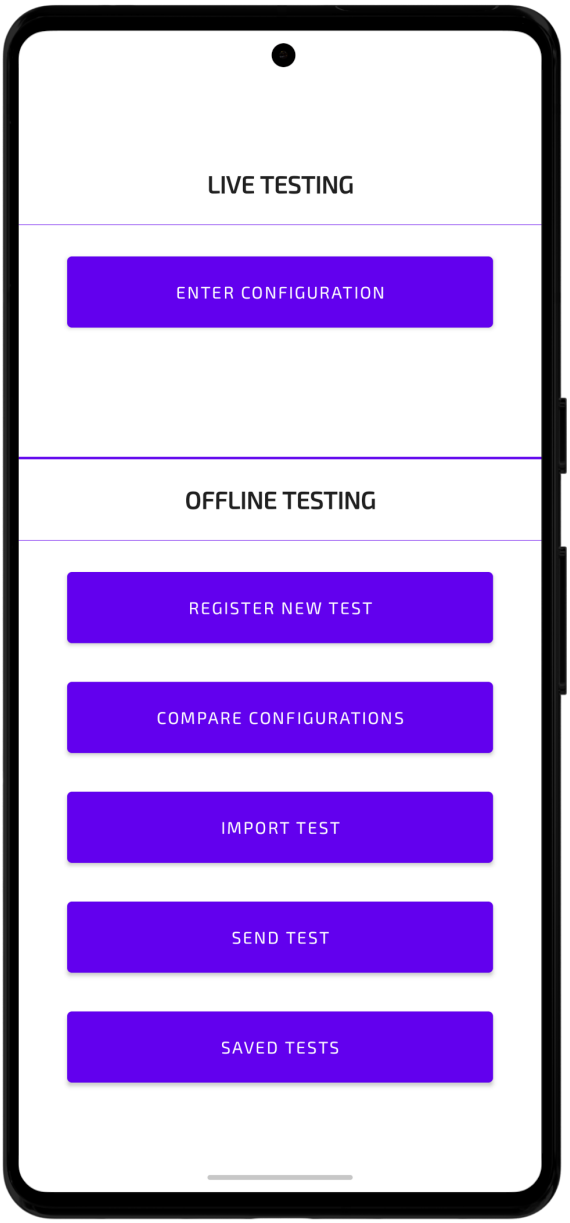

Figura 3.1: Schermata principale dell'applicazione StepLab.

3.1.2 Configuration

La cartella `configuration` raccoglie i componenti UI dedicati alla selezione degli elementi da utilizzare per il conteggio dei passi. Ciò comprende la costruzione della configurazione, ovvero la combinazione di filtri digitali e algoritmi di riconoscimento dei passi, e la scelta dei *test* pre-registrati da analizzare. A causa della dipendenza con tutti i file presenti in questo *package*, è stata qui localizzata anche l'*activity* dedicata al confronto delle configurazioni.

Andando nello specifico dell'implementazione, un elemento chiave dell'applicazione è `EnterSettingsFragment`, che è il *form* centralizzato per tradurre le scelte dell'utente in una configurazione concreta. Viene riutilizzato sia nella schermata di *live testing* sia in quella di confronto configurazioni, offrendo un'interfaccia coerente per la personalizzazione dell'analisi. Viene passato come argomento, dall'*activity* che contiene il *fragment*, un oggetto `algorithms.Configuration` che rappresenta la configurazione corrente. Il riferimento è condiviso tra *activity* e *fragment*, permettendo a quest'ultimo di modificare direttamente l'oggetto in memoria. Al salvataggio, l'*activity* può recuperare la configurazione aggiornata e procedere con l'esecuzione del pedometro o del confronto. Il *fragment* supporta due scenari distinti:

- ***Live testing***: disabilita in UI l'autocorrelazione poiché non compatibile e mostra la frequenza di campionamento alla quale si vogliono registrare i sensori.
- **Comparazione *offline***: tutti gli algoritmi sono disponibili ma la frequenza di campionamento non è modificabile, in quanto dipende dal *test* selezionato.

Il tipo di configurazione creata viene identificata in maniera automatica:

- **Non *real-time*** se è selezionata almeno un'opzione fra `Autocorrelation`, `False step detection`, `Butterworth filter` o `Time filtering`.
- ***Real-time*** in tutti gli altri casi.

L'indicatore di modalità viene aggiornato di conseguenza. L'autocorrelazione è trattata come modalità speciale ed esclusiva: quando selezionata, azzera tutte le altre scelte; alla deselezione, ripristina immediatamente la configurazione predefinita standard. Questa modalità segue una *pipeline* completamente separata, incompatibile con gli altri algoritmi. Viene dunque utilizzato il *pattern* **Model View Controller**, scelto per la semplicità del caso. Questa struttura è stata creata per permettere il riutilizzo del *fragment* in diverse *activity* e garantire la solidità e la correttezza degli algoritmi applicati alle varie modalità.

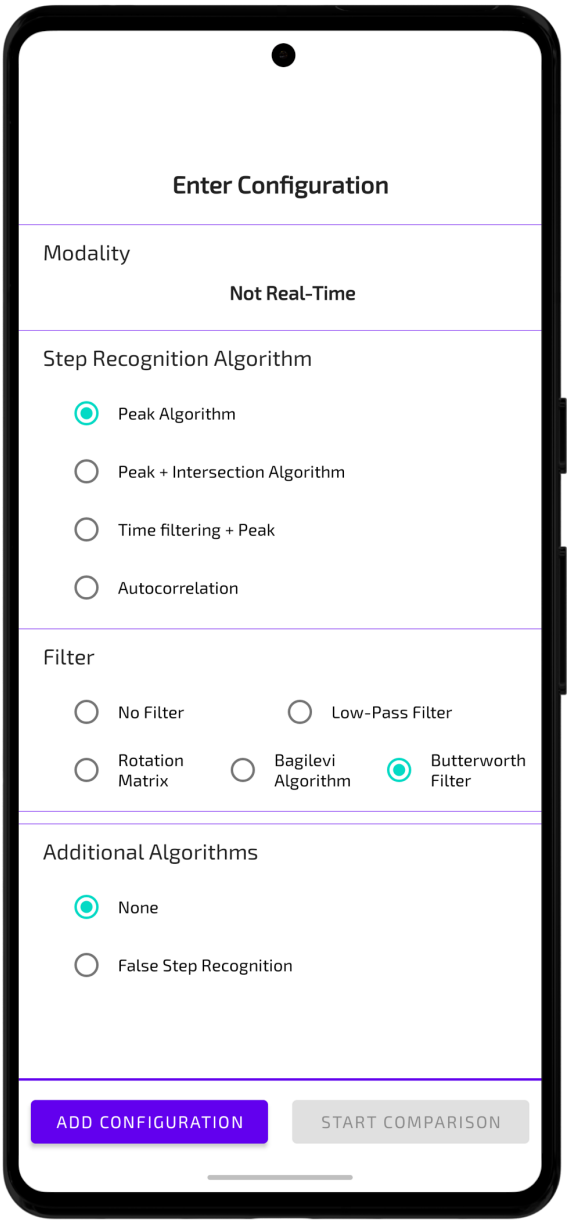

Figura 3.2: Schermata di configurazione degli algoritmi in StepLab.

Il *package* contiene anche due *activity* utilizzate soltanto per la costruzione della comparazione nel caso in cui si voglia confrontare in modalità non *live*:

- `SelectConfigurationsToCompare` funge da contenitore per il *fragment* di configurazione e gestisce la memorizzazione delle configurazioni create dall'utente. Dopo aver creato un oggetto `Configuration` che viene modificato dal *fragment*, permette tramite un pulsante di aggiungerlo ad una lista interna. Viene istanziato poi un nuovo *fragment* con una configurazione vuota per permettere la creazione di una nuova configurazione. Possono esserne aggiunte fino a sei in

totale. Al termine, viene avviata l'*activity* di selezione del test da utilizzare per il confronto passando come extra nell'intent la lista delle configurazioni create.

- `SelectTest` è una *Activity* intermedia che permette all'utente di selezionare il test registrato su cui effettuare il confronto delle configurazioni appena create. Carica i test salvati utilizzando una coroutine su thread IO e li mostra in una *RecyclerView*. A tal proposito, viene utilizzato un `AdapterForTestCard` per popolare la lista. Al click su un elemento, viene avviata l'*activity* di confronto passando come extra nell'intent il test selezionato e la lista delle configurazioni.

`ConfigurationsComparison` è l'Activity finale del workflow di confronto configurazioni. Nel suo flusso di esecuzione, le prime operazioni che svolge sono:

1. caricare il test dal database cercandolo tramite l'ID passato
2. inizializzare le view e il grafico per la visualizzazione dei dati della libreria [16]

In seguito viene disegnata sul grafico, in rosso e sull'asse delle ordinate, la magnitudine dell'accelerazione indicata nel test registrato, effettuando un ciclo su tutte le entry ordinate nel file JSON del test. In questo modo è dunque possibile visualizzare i dati grezzi registrati dai sensori sul grafico. Il passo successivo che l'*Activity* svolge è preparare la lista delle *Card* che saranno inserite per mostrare i risultati delle configurazioni processate. Viene istanziato un `AdapterForConfigurationsCard` e viene associato alla *RecyclerView* dedicata. Arriva poi la core logic di `ConfigurationsComparison`. Esegue un thread in background che si occupa di processare tutte le configurazioni. Mentre l'operazione è in svolgimento nasconde la *RecyclerView* e mostra una barra di progresso. Ad ogni configurazione viene assegnato un colore e ognuna di queste viene incapsulata in un'istanza della classe interna *ConfigurationContext*. Quest'ultima utilizza il pattern GOF *Facade* per semplificare l'utilizzo degli algoritmi, delegando le operazioni da svolgere alla classe `StepDetectionProcessor`, di cui discuteremo più avanti, ed esponendo al resto dell'activity i metodi semplificati. Il flusso subisce poi una biforcazione:

- utilizza *context.processAutocorrelationAlgorithm()* se l'algoritmo di autocorrelazione è attivo;
- negli altri casi utilizza *context.myOnSensorChanged()*.

In entrambe le opzioni, viene effettuata una delega al processore che restituisce i passi individuati. La scelta di dividere il flusso in un blocco condizionale è dovuta

dal fatto che, come accennato in precedenza, l'algoritmo di autocorrelazione segue una pipeline separata ed è stato dunque necessario separarlo da tutte le altre opzioni. Nel primo caso viene passato il file JSON completo del test al processore, mentre nel secondo caso vengono simulati i sensori passando i valori letti dal file uno ad uno. Una volta che le configurazioni sono state processate, viene aggiornata l'interfaccia utente. Per ogni configurazione processata, sul grafico vengono disegnati e collegati con una linea i punti in cui i passi sono stati rilevati. Infine la *RecyclerView* viene resa visibile e vengono aggiunte le *Cards* che indicano la configurazione processata ed il relativo colore sul grafico. Il disegno dei punti e linee sul grafico avviene aggiungendo a questo degli oggetti di tipo *Entry*, specifici della libreria [16]. Per la magnitudine grezza, vengono creati dinamicamente per ogni campione del file JSON, mentre per i passi rilevanti vengono creati e memorizzati dal processore algoritmico centrale e recupearti dal contesto di configurazione. Un'ulteriore funzionalità resa disponibile in `ConfigurationsComparison` è il salvataggio della comparazione che si sta visualizzando. Viene richiesto un nome univoco tramite un *dialog*, dopodichè su una coroutine serializza le configurazioni e le memorizza nella *entity* dedicata nel *database*.

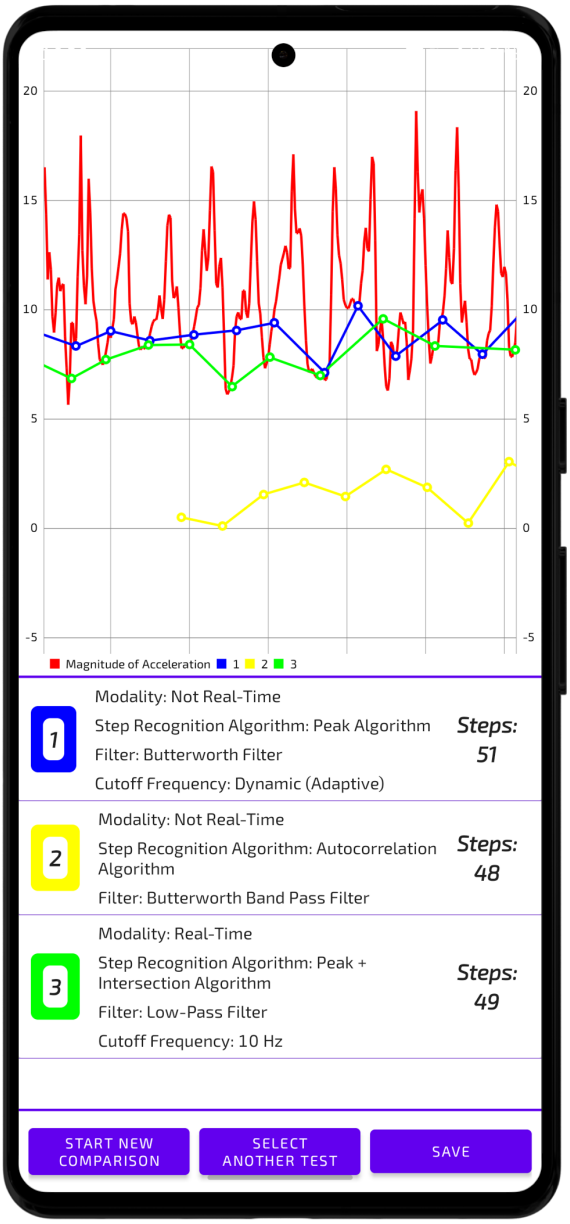

Figura 3.3: Schermata di confronto delle configurazioni in StepLab.

3.1.3 Test

Questo sottopackage contiene i componenti UI dedicati alla registrazione dei test ed alla gestione di questi. Qui viene definita anche la schermata per la visualizzazione live del pedometro, l'*activity* `LiveTesting`. Questa utilizza il *fragment* di configurazione descritto in precedenza per permettere la selezione degli algoritmi. La creazione della configurazione avviene nello stesso modo descritto per la modalità non live. Tuttavia, viene esposto all'utente soltanto un pulsante per avviare il pedometro che, quando premuto, sostituisce il *fragment* corrente con un

`PedometerRunningFragment` che mostra il numero di passi individuati ed un grafico per la visualizzazione della magnitudo dell'accelerazione. Anche il pulsante viene sostituito con un pulsante di stop che interrompe il pedometro e ripristina la UI precedente. Dunque viene implementato in questa activty una sorta di *State Pattern* nella quale lo stato è rappresentato dalla modalità di visualizzazione (configurazione o pedometro in esecuzione). Il *fragment* di esecuzione del pedometro si occupa di registrare i sensori alla frequenza di campionamento specificata dall'utente e deregistrarli. Quando rilevano un nuovo campione, delegano al processore algoritmico centrale la gestione del dato, in maniera analoga alla pipeline di confronto delle configurazioni. Vengono ritornate varie informazioni, tra cui se è stato rilevato un passo e se il dato passato è un evento accelerometrico. Se il primo caso è affermativo, il contatore dei passi viene incrementato e la UI aggiornata. Se il secondo caso è affermativo, viene aggiunto un nuovo punto al grafico per la visualizzazione della magnitudo dell'accelerazione filtrata in base alla configurazione utilizzata.

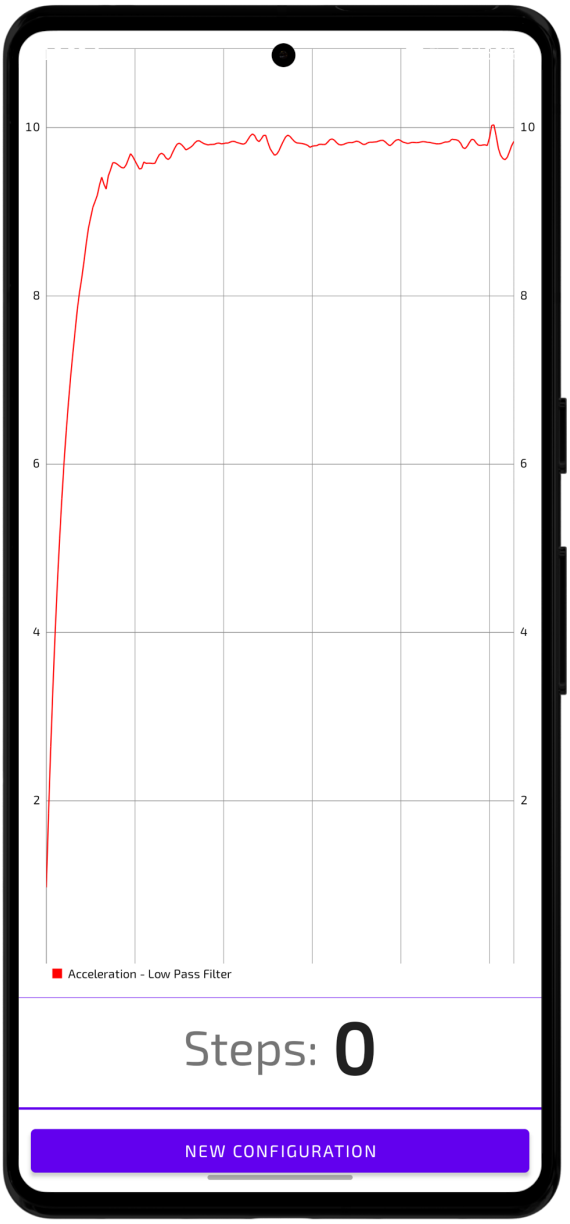

Figura 3.4: Schermata di live testing in StepLab.

Un ulteriore funzionalità inclusa in questa cartella è la registrazione di un nuovo test. Viene utilizzata l'*activity* `NewTest` che registra i sensori e aggiorna, analogamente a ciò che succede in *live testing*, un grafico quando vengono rilevati eventi accelerometrici. I sensori sono registrati con *SENSOR DELAY GAME*, poichè si tratta di un buon compromesso tra consumo energetico, frequenza di campionamento e fedeltà del segnale. Per ogni evento rilevato dai sensori, costruisce un *json object* e lo inserisce in una mappa indicizzata da time stamp. I sensori utilizzati sono l'accelerometro, il magnetometro, il sensore di rotazione e di gravità. Se l'evento è accelerometrico, viene anche calcolata ed inserita nella mappa anche

la magnitudo dell'accelerazione, utilizzata anche per il disegno del grafico. Quando si ferma la registrazione, si apre un *dialog* e vengono richiesti alcuni metadati opzionali (numero di passi e note aggiuntive) all'utente. Questo mostra un pulsante di salvataggio, che quando premuto effettua le operazioni di salvataggio necessarie: salva il file nello *storage* interno con un nome basato sul *timestamp* e crea una `EntityTest` con i metadati e l'ID del file salvato e la inserisce nel database Room.

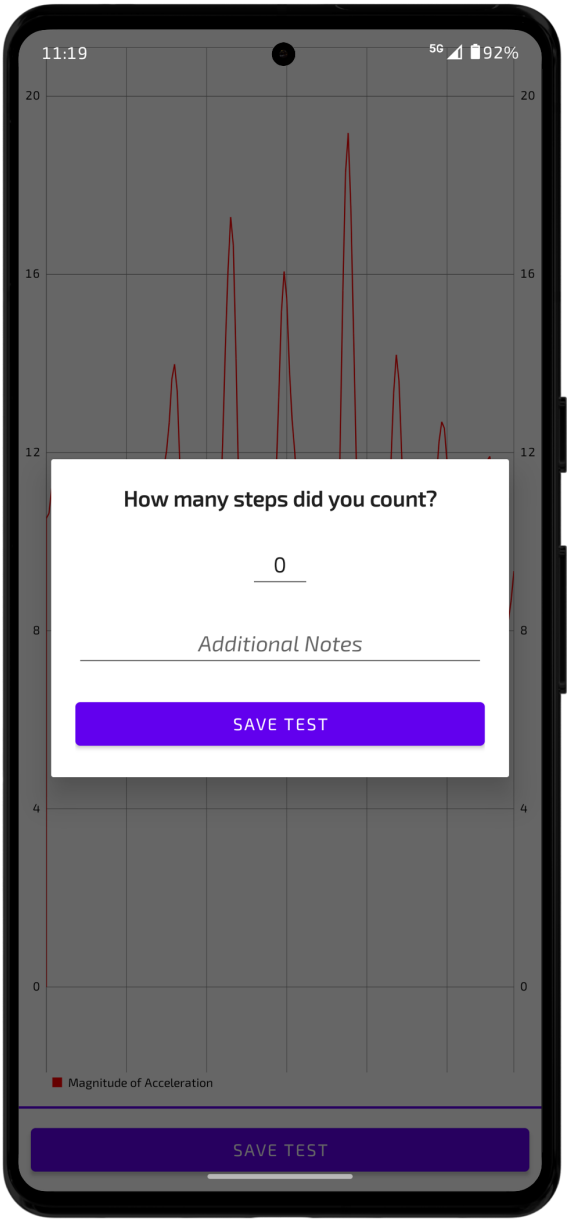

Figura 3.5: Schermata di registrazione di un nuovo test in StepLab.

Nella cartella corrente sono presenti anche i componenti dedicati all'esportazione dei test registrati. L'activity `SendTest` al momento dell'avvio legge il contenuto del database locale e popola una lista di elementi che rappresentano ciascun test,

mostrando per ognuno le informazioni principali (numero di passi, note e nome del file). Ciò avviene utilizzando una *RecyclerView* e un `AdapterForSendTestCard` per la gestione degli elementi. L'utente può selezionare uno o più test tramite la RecyclerView e, con un semplice pulsante, aprire una finestra di dialogo in cui scegliere il formato di esportazione preferito, JSON o CSV. Una volta confermata la scelta, l'applicazione recupera i file da esportare. Se il formato scelto è CSV, applica prima

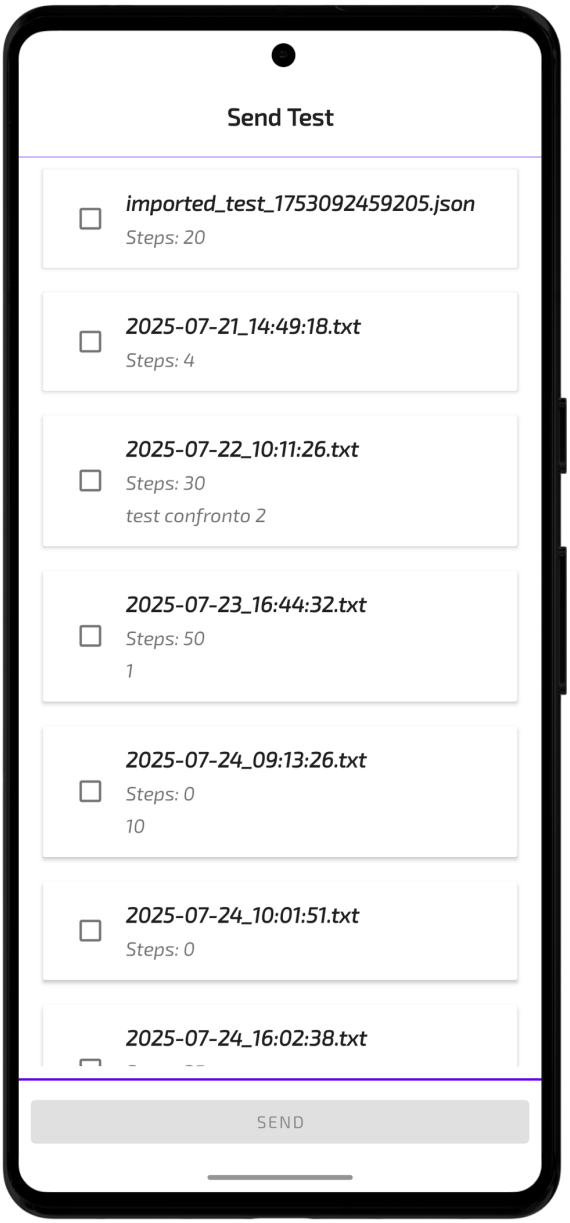

Figura 3.6: Schermata di esportazione dei test in StepLab.

una conversione tramite la classe `JsonToCsvConverter`, in quanto i test utilizzabili dall'applicazione vengono sempre salvati in JSON e perciò quando recuperati possiedono questo formato. Infine, utilizza *FileProvider* per generare gli URI e raccgolierli

in un lista, poi utilizza un *Intent* implicito di tipo `ACTION_SEND_MULTIPLE` per inviare i file.

L'ultima funzionalità presente in questa sezione è la visualizzazione dei salvataggi delle comparazioni effettuate. L'*activity* `SavedTests` è la schermata principale per la visualizzazione dei test salvati. Carica l'elenco delle comparazioni salvate, ognuna contiene il nome assegnato, la data di creazione e le configurazioni utilizzate durante l'esperimento.

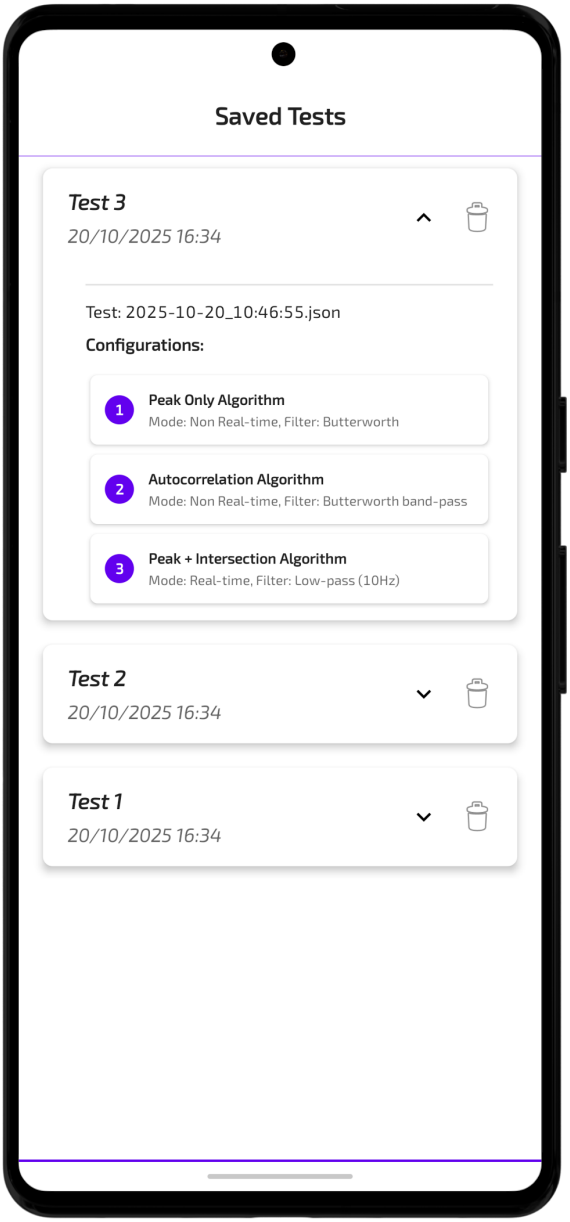

Figura 3.7: Schermata di visualizzazione dei salvataggi in StepLab.

I dati vengono quindi mostrati in una lista interattiva, tramite *RecyclerView*. Utilizza un `SavedTestsAdapter` per popolare la lista che nasconde però le confi-

gurazioni utilizzate. Queste diventano visibili quando l'utente espande la scheda del test, permettendo di esplorare i dettagli. A tal proposito, ogni *card* della lista quando espansa, annida dentro di sè una seconda *RecyclerView*, popolata da un `SavedConfigurationAdapter` che mostra le singole configurazioni utilizzate nel test salvato. L'utente può interagire con ogni scheda in due modi principali: toccando la scheda stessa per aprire il test in modalità di sola visualizzazione, avviando l'*activity* di confronto configurazioni con i dati del test selezionato, oppure premendo l'icona di eliminazione per rimuovere il test. Questa architettura è stata scelta in modo da permettere all'applicazione di offrire una memoria storica strutturata delle prove effettuate e mantenere la tracciabilità delle esperienze di test.

3.2 Algoritmi e building blocks del rilevamento passi

Prima di presentare l'implementazione degli algoritmi nel codice, è utile fornire una tassonomia dei principali blocchi che compongono la pipeline di rilevamento passi. Ogni configurazione creabile nell'applicazione è composta da una serie di moduli che operano in sequenza per elaborare i dati grezzi provenienti dai sensori, questi vengono di seguito descritti.

3.2.1 Filtri

I filtri digitali sono componenti essenziali per la pulizia del segnale prima dell'analisi. Rimuovono il rumore e permettono di isolare le caratteristiche rilevanti per il riconoscimento dei passi, rappresentando di fatto il primo passo per il conteggio. Di seguito vengono spiegati i filtri implementati nell'applicazione.

Filtro passa-basso

Il filtro passa-basso implementato è un filtro digitale applicato separatamente ai tre assi dell'accelerometro. Il suo obiettivo è attenuare le componenti ad alta frequenza del segnale, tipicamente dovute al rumore dei sensori e alle oscillazioni rapide del dispositivo che non sono correlate al passo. Il filtro è definito dalla seguente equazione ricorsiva:

$$y_t = y_{t-1} + \alpha(x_t - y_{t-1}),$$

dove x_t è il valore grezzo del sensore, y_t è il valore filtrato e $\alpha \in (0, 1)$ è un coefficiente che controlla la reattività del filtro. Valori piccoli di α rendono il filtraggio più aggressivo, mentre valori più grandi avvicinano il filtro al segnale originale. Il valore y_{t-1} rappresenta lo stato interno del filtro, ovvero l'ultimo valore filtrato, che viene aggiornato ad ogni nuovo campione. Il parametro α è calcolato in funzione della frequenza di campionamento f_s e della *frequenza di taglio* f_c desiderata, secondo la formula:

$$\alpha = \frac{\Delta t}{RC + \Delta t},$$

dove $\Delta t = 1/f_s$ è il periodo di campionamento e $RC = 1/(2\pi f_c)$ rappresenta la costante di tempo del filtro, che indica quanto rapidamente il filtro risponde ai cambiamenti del segnale. Una frequenza di taglio alta corrisponde a un valore RC piccolo, rendendo il filtro più reattivo; viceversa, una frequenza di taglio bassa produce un valore RC più grande, rallentando la risposta del filtro. Il valore di α viene inoltre limitato nell'intervallo $[0, 1]$ per garantire la stabilità numerica del filtro. La frequenza di taglio f_c rappresenta la frequenza oltre la quale il filtro inizia ad attenuare significativamente le componenti del segnale. La scelta di questo parametro ha un impatto determinante sul comportamento del filtro: più è alta, minore sarà l'attenuamento e più il segnale filtrato rimarrà vicino a quello originale. Viceversa, frequenze di taglio basse comportano un filtraggio più aggressivo, che rimuove efficacemente il rumore ma può anche attenuare i picchi associati ai passi.

Matrice di rotazione

Questo filtro non agisce sul segnale tramite attenuazione selettiva delle frequenze, ma tramite una trasformazione di coordinate. Il suo obiettivo è rendere il segnale dell'accelerometro invariabile rispetto all'orientamento del dispositivo, in modo da stabilizzare il riconoscimento dei passi. Il filtro utilizza la matrice di rotazione fornita dall'API `SensorManager`. Questa stima l'orientamento del dispositivo rispetto al sistema di riferimento del mondo attraverso la gravità e il nord magnetico. Una volta calcolata la matrice, il vettore di accelerazione viene trasformato nel sistema mondo tramite:

$$a^{(w)} = R\, a^{(d)},$$

dove $a^{(d)}$ è il vettore dell'accelerazione nel sistema del dispositivo e $a^{(w)}$ è quello nel sistema di riferimento globale, mentre R è la matrice di rotazione calcolata. Questo

filtro si limita a una trasformazione lineare per campione e risulta particolarmente utile quando l'orientamento del dispositivo è variabile.

Filtro passa-basso Butterworth

Il filtro Butterworth implementato in *StepLab* è un filtro digitale di tipo IIR (*Infinite Impulse Response*), ovvero un filtro che utilizza non soltanto il campione di ingresso corrente, ma anche i valori filtrati delle iterazioni precedenti. Questa caratteristica consente una risposta più selettiva rispetto al passa-basso semplice, a costo di dover mantenere uno stato interno che evolve nel tempo. Il filtro è applicato separatamente ai tre assi dell'accelerometro ed è configurato come passa-basso del secondo ordine. La forma ricorsiva di un tale filtro è:

$$y_t = b_0 x_t + b_1 x_{t-1} + b_2 x_{t-2} - a_1 y_{t-1} - a_2 y_{t-2},$$

dove i coefficienti b_i controllano il contributo dei nuovi campioni mentre i coefficienti a_i rappresentano l'influenza delle uscite precedenti. Tali coefficienti sono calcolati a partire dalla frequenza di taglio f_c e dalla frequenza di campionamento f_s, ma in *StepLab* non vengono calcolati manualmente, si appoggia invece alla libreria open source [17]. Un aspetto distintivo di questa implementazione è la *frequenza di taglio dinamica*. L'applicazione parte da un valore iniziale di 3 Hz, ma può modificarlo automaticamente durante l'elaborazione in risposta al comportamento del segnale. La regolazione tiene conto di:

- variazioni della magnitudine tra passi consecutivi,
- differenze temporali tra le durate dei passi,
- frequenza di campionamento stimata dinamicamente.

Se tali indicatori suggeriscono un andamento anomalo, la frequenza di taglio viene adattata entro un intervallo sicuro [2 Hz, $f_s/3$]. Questo comportamento rende il filtro più robusto ma richiede una situazione di campionamento relativamente stabile, poiché l'adattamento continuo dipende dalla memoria interna del filtro e dalla coerenza del segnale in ingresso.

Filtro passa-banda Butterworth

Questo filtro non viene reso disponibile direttamente nell'applicazione, ma viene implementato unicamente per supportare l'algoritmo di autocorrelazione. Quando

quest'ultimo è utilizzato, viene applicato il filtro descritto in questa sezione. A differenza del filtro passa-basso, che attenua solo le frequenze alte, il filtro passa-banda Butterworth isola un intervallo specifico di frequenze, attenuando sia le componenti troppo basse che quelle troppo alte. Il filtro è configurato specificando due parametri di frequenza:

- *frequenza inferiore* f_{low}: limite inferiore della banda passante, al di sotto del quale il segnale viene attenuato;
- *frequenza superiore* f_{high}: limite superiore della banda passante, al di sopra del quale il segnale viene attenuato.

La banda passante $\Delta f = f_{\text{high}} - f_{\text{low}}$ determina l'ampiezza dell'intervallo di frequenze conservato, mentre la frequenza centrale $f_c = (f_{\text{low}} + f_{\text{high}})/2$ indica il punto medio della banda. Matematicamente, il filtro è descritto da un'equazione ricorsiva simile a quella del passa-basso, ma con coefficienti che dipendono da entrambe le frequenze di taglio:

$$y_t = b_0 x_t + b_1 x_{t-1} + \cdots + b_n x_{t-n} - a_1 y_{t-1} - \cdots - a_n y_{t-n},$$

dove l'ordine n del filtro (in questo caso 6) determina la selettività della risposta in frequenza. Il filtro viene applicato al segnale di magnitudine dell'accelerazione producendo un segnale che enfatizza le oscillazioni periodiche tipiche del cammino, facilitando così l'analisi di autocorrelazione per il riconoscimento dei passi. Anche questo filtro si appoggia alla libreria [17] per l'implementazione dei coefficienti.

Filtro Bagilevi

Il filtro Bagilevi non è un filtro digitale nel senso tradizionale, ma una trasformazione euristica applicata ai tre assi del magnetometro. Il suo scopo è produrre un segnale scalare più stabile e più sensibile alle variazioni utili per il riconoscimento dei picchi. Il filtro sfrutta il valore massimo atteso del campo magnetico terrestre $B_{\max}$ (fornito dalla costante di sistema `MAGNETIC_FIELD_EARTH_MAX`) e combina i tre assi del magnetometro secondo la seguente formula:

$$y = \frac{1}{3} \sum_{i=x,y,z} 4 \cdot (B_{\max} - m_i).$$

Questa operazione amplifica le deviazioni del segnale rispetto al valore teorico del campo geomagnetico e ne calcola la media sui tre assi, riducendo l'effetto di

anomalie localizzate su un singolo asse. Il filtro Bagilevi fa parte di una pipeline dedicata: quando selezionato, attiva un algoritmo di riconoscimento specifico che sostituisce quello standard per l'individuazione dei picchi.

3.2.2 Strategie di rilevamento passi

Il secondo blocco fondamentale della pipeline di conteggio dei passi è rappresentato dalle strategie di rilevamento. Questi algoritmi analizzano il segnale pre-elaborato dai filtri per identificare i momenti in cui l'utente compie un passo. Di seguito vengono descritte le strategie implementate.

Rilevamento dei picchi

Questa strategia identifica i passi analizzando l'andamento del segnale filtrato. L'algoritmo rileva un picco quando la magnitudine dell'accelerazione smette di crescere ($m_t \leq m_{t-1}$ dopo una fase crescente), mentre una valle viene rilevata quando il segnale smette di decrescere dopo un picco. Durante l'esecuzione vengono mantenuti il massimo locale $M_{\max}$, il minimo locale $M_{\min}$ e i relativi timestamp. Una volta identificata una coppia picco-valle, si calcola la differenza

$$\Delta_{\text{ext}} = |M_{\max} - M_{\min}|.$$

Per ridurre i falsi positivi, l'algoritmo rileva un nuovo passo solo se le condizioni seguenti sono soddisfatte:

- il massimo locale supera un valore minimo fisso ($10.5\,\text{m/s}^2$);
- la differenza tra gli estremi supera una soglia dinamica ridotta:

$$\Delta_{\text{ext}} > \frac{3}{5}\,\theta.$$

La soglia θ viene aggiornata secondo una media mobile pesata:

$$\theta_{n+1} = \frac{n\,\theta_n + \Delta_{\text{ext}}}{n+1},$$

che rende l'algoritmo adattativo all'intensità del movimento dell'utente. Questo metodo è leggero dal punto di vista computazionale e funziona bene quando il segnale presenta picchi regolari e ben separati.

Rilevamento dei picchi con filtraggio temporale

Questa strategia estende il rilevamento dei picchi introducendo vincoli temporali adattativi, in modo da migliorare la robustezza in presenza di oscillazioni irregolari o rumore. L'algoritmo mantiene due serie distinte di timestamp: una per i picchi (massimi locali) e una per le valli (minimi locali). Per ciascun tipo di estremo vengono conservati i due timestamp più recenti, T_{n-1} e T_{n-2}, utilizzati per definire soglie temporali che verificano la regolarità della cadenza. Per ogni coppia di estremi dello stesso tipo vengono calcolate due soglie:

- soglia di inizio
$$\theta_S = 0.35\,|T_{n-1} - T_{n-2}|,$$
che verifica la coerenza dell'intervallo corrente con quelli precedenti;

- soglia di fine
$$\theta_E = 0.20\,(t_{\text{curr}} - T_{n-1}),$$
che richiede un'adeguata separazione temporale tra candidati successivi.

Richiede dunque almeno due cicli precedenti per attivare la logica. Quando un nuovo estremo candidato viene rilevato, l'algoritmo verifica che l'intervallo corrente $\delta = t_{\text{curr}} - T_{n-1}$ rispetti $\delta > \theta_S$. Se un secondo candidato appare prima del completamento del ciclo, viene accettato solo se soddisfa $\delta_{\text{new}} \geq \theta_E$. Un passo viene confermato solo al termine di un'intera sequenza picco-valle che rispetta i vincoli temporali. Questo permette di filtrare oscillazioni che non seguono la cadenza naturale del cammino.

Rilevamento tramite intersezione con l'asse

Questa strategia non sostituisce il rilevamento dei picchi, ma ne costituisce una verifica aggiuntiva. Il normale algoritmo picco-valle individua un massimo locale $T_{\max}$ e un minimo locale $T_{\min}$, questi valori definiscono un ciclo candidato al riconoscimento del passo. Parallelamente, questo algoritmo registra il timestamp dell'ultimo attraversamento del segnale rispetto all'asse di riferimento, ovvero il momento in cui la magnitudine cambia segno. Tale istante viene memorizzato come

$$T_{\text{cross}} = \texttt{lastXAxisIntersectionTime}.$$

Il passo viene confermato solo se l'intersezione avviene all'interno del ciclo picco-valle. In altre parole, l'algoritmo verifica se l'asse è stato attraversato tra i due estremi:

$$T_{\max} < T_{\text{cross}} < T_{\min}.$$

Questa condizione funge da guardia aggiuntiva, un passo viene riconosciuto solo se il ciclo picco-valle attraversa effettivamente l'asse, riducendo i falsi positivi in tutti quei casi in cui il segnale presenta oscillazioni che non superano lo zero o non rappresentano un vero movimento ciclico del passo.

Algoritmo Bagilevi

Questa strategia è progettata per funzionare esclusivamente in combinazione con il filtro Bagilevi e sostituisce il rilevamento dei picchi standard. A differenza degli altri metodi, che identificano direttamente picchi e valli, questo algoritmo analizza la variazione direzionale del segnale filtrato per individuare le inversioni di andamento, che interpretate come estremi locali. L'algoritmo osserva il segno della derivata discreta:

$$\mathrm{dir}_t = \mathrm{sign}(m_t - m_{t-1}),$$

dove m_t è il valore corrente del segnale filtrato. Una variazione del segno indica un'inversione di pendenza:

- da negativa a positiva ⇒ minimo locale (valle);
- da positiva a negativa ⇒ massimo locale (picco).

Quando viene rilevato un estremo, la magnitudine corrispondente viene salvata come

$$M_{\min} = \texttt{lastLocalMinAccel}, \qquad M_{\max} = \texttt{lastLocalMaxAccel},$$

, mentre i tempi di rilevamento vengono memorizzati in

$$T_{\min} = \texttt{lastStepFirstPhaseTime}, \qquad T_{\max} = \texttt{lastStepSecondPhaseTime}.$$

La decisione finale sul riconoscimento di un passo utilizza una logica di coerenza basata sulla differenza di ampiezza tra gli estremi locali.

$$\Delta_t = \left| M_{\max,t} - M_{\min,t} \right|$$

La quantità Δ_t rappresenta l'ampiezza del movimento verticale associato a un potenziale passo; valori troppo piccoli suggeriscono rumorosità o falsi estremi. Nel codice, la decisione utilizza tre criteri:

1. Ampiezza minima del movimento:

$$\Delta_t > 10.$$

2. Coerenza con il ciclo precedente:

$$\Delta_t > \tfrac{2}{3}\,\Delta_{t-1} \qquad \text{e} \qquad \Delta_{t-1} > \tfrac{1}{3}\,\Delta_t.$$

3. Alternanza corretta del tipo di estremo, ovvero, l'algoritmo richiede che il tipo di estremo corrente (picco/minimo) alterni rispetto al precedente. Ciò evita di contare due picchi o due valli consecutivi come passi distinti.

Solo quando tutte e tre le condizioni sono soddisfatte, il passo viene confermato. Questo rende l'algoritmo particolarmente adatto a contesti in cui il segnale sia rumoroso o presenti variazioni irregolari di ampiezza.

Algoritmo di autocorrelazione

L'algoritmo di autocorrelazione rappresenta un approccio diverso rispetto ai metodi basati sui picchi. Invece di analizzare il segnale campione per campione, esso considera porzioni più ampie del segnale e ne studia la periodicità, che nel cammino umano tende ad essere costante. Dal vettore tridimensionale dell'accelerazione viene calcolata la magnitudine:

$$m_t = \sqrt{x_t^2 + y_t^2 + z_t^2}.$$

Per stabilizzare l'analisi viene rimossa la componente continua (media del segnale):

$$m'_t = m_t - \frac{1}{N}\sum_{i=1}^{N} m_i.$$

Se la frequenza di campionamento è molto elevata, il segnale viene decimato verso 50–60 Hz per ridurre la complessità computazionale senza perdere informazioni utili alla camminata. Per determinare la frequenza caratteristica del passo (f_0), il segnale viene trasformato nel dominio della frequenza tramite FFT (Fast Fourier Transform). La ricerca del picco viene limitata al range plausibile per l'andatura umana (1–3.5 Hz). Utilizzando la stima di f_0, il segnale viene filtrato tramite un Butterworth passa-banda (vedi 3.2.1), così da isolare le oscillazioni tipiche del passo ed eliminare sia movimenti troppo lenti sia vibrazioni ad alta frequenza. Per rilevare gli intervalli in cui è effettivamente presente la camminata si calcola la deviazione standard mobile (MSD):

$$\mathrm{MSD}_t = \sqrt{\frac{1}{w}\sum_{i=t-w+1}^{t} (y_i - \bar{y})^2}.$$

Dove indichiamo con y_t il segnale filtrato, con $\bar{y}$ la media del segnale nella finestra mobile, mentre w indica l'ampiezza della finestra stessa (in numero di campioni). I

campioni per cui MSD_t supera una soglia dinamica vengono raggruppati in segmenti continui, successivamente fusi e filtrati in modo da mantenere solo intervalli compatibili con almeno un ciclo completo di camminata. Su ogni segmento valido viene calcolata l'autocorrelazione normalizzata:

$$\rho(k) = \frac{\sum_{t=0}^{N-k-1} y_t \, y_{t+k}}{\sum_{t=0}^{N-1} y_t^2}, \qquad \rho(0) = 1.$$

Il primo picco significativo di $\rho(k)$ fornisce il lag k^*, ossia il numero di campioni corrispondenti a due passi successivi. Il numero di passi stimato nel segmento è quindi:

$$\hat{S} = \frac{L}{k^*},$$

dove L è la lunghezza del segmento. Poiché la cadenza del cammino non può variare arbitrariamente, la stima viene vincolata entro un intervallo coerente con la frequenza fondamentale globale:

$$S_{\min} \leq \hat{S} \leq S_{\max}, \qquad S_{\min} \approx 0.7 \, \frac{L}{f_0}, \quad S_{\max} \approx 1.3 \, \frac{L}{f_0}.$$

In sintesi, l'algoritmo di autocorrelazione non cerca picchi locali, ma individua la periodicità del movimento. Risulta essere molto robusto anche in presenza di rumore, richiede tuttavia segmenti di camminata sufficientemente lunghi per produrre stime affidabili.

3.2.3 Algoritmi aggiuntivi

Oltre ai filtri e alle strategie di rilevamento, nella pipeline sono includibili anche eventuali algoritmi aggiuntivi per manipolare ulteriormente il segnale. Nel caso di questa implementazione, tra questi rientra soltanto una guardia utilizzata per riconoscere e scartare i passi falsi.

Algoritmo dei passi falsi

L'idea alla base del metodo è che ogni passo reale produce un andamento del campo magnetico relativamente stabile da un ciclo all'altro, mentre oscillazioni casuali generano deviazioni anomale. Ad ogni passo individuato dagli algoritmi principali viene raccolta la magnitudine del magnetometro per l'intera durata del passo:

$$m_1^{(k)}, m_2^{(k)}, \ldots, m_{n_k}^{(k)},$$

, e si calcola la media del passo k:

$$\bar{m}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} m_i^{(k)}.$$

Per i primi quattro passi rilevati, il sistema costruisce una finestra contenente:

$$\{\bar{m}_1, \bar{m}_2, \bar{m}_3, \bar{m}_4\}.$$

Prima che la finestra sia piena, il controllo di falsi passi è molto semplice: se la differenza tra la media corrente e quella immediatamente precedente è eccessiva,

$$|\bar{m}_k - \bar{m}_{k-1}| > 3.6,$$

il passo viene marcato come falso. Una volta popolata la finestra con quattro valori, il riconoscimento dei falsi passi utilizza un criterio più robusto. Siano:

$$m_4 = \frac{1}{4} \sum_{i=1}^{4} \bar{m}_{k-i}, \qquad m_5 = \frac{1}{5} \left(\sum_{i=1}^{4} \bar{m}_{k-i} + \bar{m}_k \right).$$

Il passo corrente viene considerato un falso positivo se:

$$|m_4 - m_5| > \tau, \qquad \tau = 0.8.$$

Questa misura confronta la stabilità dei passi precedenti con l'effetto dell'inserimento del nuovo valore: se l'aggiunta della media del passo corrente modifica troppo la media mobile, il passo è considerato anomalo. Dopo ogni verifica, la finestra dei quattro passi più recenti viene aggiornata in modo scorrevole, mantenendo sempre memoria delle ultime quattro medie valide.

3.3 Algorithms

Questo package contiene la logica centrale dell'applicazione, ovvero l'implementazione degli algoritmi di step detection e dei filtri digitali. Vengono quindi descritte le classi necessarie e il loro utilizzo.

3.3.1 Configurazione e dati dei sensori

La classe `Configuration` è una semplice data class, inizializzata nella parte UI e modificata e letta dagli algoritmi durante la pipeline di rilevamento passi, fungendo da contenitore unico di stato e parametri. Questa è la spina dorsale del contesto: non solo memorizza i tipi di algoritmi da applicare indicati durante la creazione

da parte dell'utente, ma memorizza anche alcuni valori numerici utilizzati dagli algoritmi e modificati da questi. Un esempio sono i timestamp dei picchi rilevati che vengono utilizzati per calibrare il filtraggio temporale. Ciò permette a tutta la pipeline di operare in modo adattivo, coerente e robusto. Un ruolo simile lo svolge la classe `SensorData` la quale funge da contenitore dei dati provenienti dai vari sensori, mantenendo distinti i valori grezzi e quelli filtrati o trasformati durante la pipeline. Per ogni sensore memorizza il vettore tridimensionale, le versioni filtrate , le eventuali componenti nel sistema di riferimento del mondo e le relative magnitudini. Ogni istanza dispone inoltre di un flag di validità per evitare elaborazioni su dati non inizializzati. Nel processore principale, che sarà descritto in questa sezione, più oggetti `SensorData` lavorano in parallelo per separare i diversi flussi di segnale, garantendo ordine, isolamento dello stato e coerenza tra le varie fasi di elaborazione.

3.3.2 Calcoli ed algoritmi

Gli algoritmi sono incapsulati in diverse classi, ognuna di queste con una propria responsabilità. Sono "attrezzi" utilizzati dal processore principale per elaborare i dati. Di seguito si fornisce una panoramica di queste specificando la rispettiva responsabilità di ognuna:

- `Calculations`: fornisce un insieme di funzioni matematiche e supporto numerico utilizzate da tutti i moduli della pipeline, ad esempio, contiene il metodo per calcolare la magnitudine, ampliamente utilizzato dalle altre classi come messo in evidenza precedentemente. Non si occupa direttamente della classificazione dei passi ma rappresenta la base numerica sulla quale i moduli di filtraggio e rilevamento si fondano.

- `Filters`: incapsula le operazioni di filtraggio del segnale necessarie per eliminare rumore prima dell'individuazione del passo. Mantiene diversi tipi di filtro e gli stati di questi, ove necessario. Inoltre, nella UI viene rese disponibile come filtro anche la matrice di rotazione, che effettua una trasformazione di coordinate per rendere il segnale invariabile rispetto all'orientamento del dispositivo. Tuttavia, non è presente in questa classe in quanto viene calcolata direttamente nel processore principale.

- `KeyValueRecognition`: si occupa dell'analisi dei valori chiave dei segnali dei sensori, ovvero del riconoscimento dei picchi e delle valli che caratterizzano

il passo umano. Offre anche altre funzioni: espone un algoritmo di riconoscimento dei picchi utilizzato solo nel caso in cui si sia prima applicato un filtro bagilevi e permette di applicare ulteriori correzioni al segnale. Una di filtraggio temporale sui picchi rilevati e un'altra di filtraggio in base all'attraversamento dell'accelerazione dello zero. L'individuamento dei valori chiave dipende dai valori precedentemente rilevati, che vengono memorizzati come stato interno in un'istanza della classe `Configuration` ed alla quale questa classe fa riferimento. In ogni caso, i metodi esposti restituiscono un valore booleano che indica se il valore passato è un massimo locale. Su ciò viene basata poi la logica di rilevamento del passo.

- `StepDetection`: rappresenta il nucleo logico della rilevazione dei passi. Implementa diverse strategie di riconoscimento: differenza dei picchi, intersezione con l'asse delle ascisse, autocorrelazione. Il funzionamento generale è simile a quello della classe precedentemente illustrata: ogni metodo espone un algoritmo di rilevamento dei passi che restituisce un valore booleano che indica se il campione passato corrisponde ad un passo. Anche in questo caso, il riconoscimento dipende da valori precedentemente rilevati e memorizzati come stato interno nell'istanza di `Configuration`.

3.3.3 Processore principale

La classe `StepDetectionProcessor` funge da *facade* per orchestrare l'intera *pipeline* di elaborazione dei dati sensoristici e l'applicazione degli algoritmi di *step detection*, nascondendo la complessità dell'implementazione.

Riceve la dipendenza da `Configuration` al costruttore, assicurando che tutti i collaboratori ricevano gli stessi dati e garantendo coerenza nella *pipeline* di elaborazione. A questa classe viene delegato il lavoro che sarebbe altrimenti svolto direttamente nelle modalità di *live testing* e confronto configurazioni.

Poiché vi è differenza nel processare dati in *live* rispetto all'elaborazione in *batch*, `StepDetectionProcessor` presenta una biforcazione principale, definendo due metodi distinti per le due modalità operative.

ProcessRealTimeSensorData() Questo metodo viene invocato dal `Pedometer RunningFragment` ogni volta che arrivano nuovi campioni dai sensori. Riceve in ingresso il tipo di sensore, il vettore dei valori e il *timestamp* del campione.

In base al tipo di sensore, distingue tra accelerometro, magnetometro, gravità e rotazione. Per contenere le allocazioni ed evitare pressione sul *Garbage Collector*, utilizza un piccolo *pool* di array pre-allocati (uno per ciascun sensore gestito) su cui copia i valori prima di convertirli in `BigDecimal`: questo approccio riduce sensibilmente le nuove istanze per secondo rispetto a una conversione *naive*.

Quando l'evento è accelerometrico, attiva la *pipeline*:

1. applica il filtro scelto in configurazione;
2. eventualmente esegue la correzione di intersezione se l'algoritmo di riconoscimento selezionato lo richiede;
3. convalida i falsi positivi;
4. aggiorna il contapassi.

L'esito viene restituito come un oggetto `ProcessingResult`, che riporta se è stato rilevato un passo, il valore filtrato utile alla visualizzazione e l'etichetta da mostrare sul grafico.

ProcessBatchSensorData() Questo metodo è utilizzato durante il confronto di configurazioni per riprodurre *test* registrati in precedenza, processando una alla volta le *entry* del file JSON.

In ingresso riceve l'istante (in millisecondi) e l'oggetto JSON dell'evento. Il *parsing* è flessibile: può leggere file che contengono solo accelerometro oppure più sensori nello stesso record (diversamente dal *live*, dove gli eventi arrivono separati).

In questo contesto non *real-time* privilegia la semplicità: crea i `BigDecimal` direttamente dalle stringhe del JSON senza *object pooling*, scelta accettabile perché l'elaborazione non deve rispettare vincoli di latenza. La frequenza di campionamento viene stimata dinamicamente dai *timestamp* tramite `updateFsFromMillis()`.

Quando è attiva la raccolta per i grafici, aggiunge automaticamente le *entry* utili alla visualizzazione finale. Dopo il *parsing*, la *pipeline* è la stessa della modalità *live*: filtri, riconoscimento e gestione dei falsi passi. Il metodo restituisce un booleano che indica se in quel *frame* è stato rilevato un passo.

ProcessAutocorrelationAlgorithm() Vi è inoltre un'ulteriore *pipeline* dedicata all'autocorrelazione, che richiede un file completo e non funziona *frame-by-frame* come quelle precedenti. Le operazioni qui effettuate sono indipendenti e diverse da quelle disponibili negli altri due metodi.

Il metodo riceve in ingresso l'intero oggetto JSON contenente tutti gli eventi registrati. Inizialmente, le chiavi vengono ordinate cronologicamente e per ciascun evento viene estratta la magnitudine dell'accelerazione, costruendo una lista di campioni con i relativi *timestamp*.

La frequenza di campionamento viene stimata analizzando l'intervallo temporale totale e il numero di campioni disponibili. Successivamente, viene invocato il metodo `countStepsAutocorrelation()`, che applica:

- rimozione della componente continua dal segnale;
- filtraggio *band-pass* Butterworth sulla banda di frequenza identificata come ottimale per il riconoscimento dei passi;
- segmentazione del segnale e calcolo dell'autocorrelazione su ciascun segmento;
- individuazione dei picchi periodici, che corrispondono ai passi rilevati.

Questa modalità è particolarmente efficace per l'analisi *offline* di camminate regolari, dove la periodicità del passo è ben definita.

3.4 Data

Questo *package* ha la responsabilità di gestire la persistenza locale attraverso entità Room, DAO e la configurazione del database.

La persistenza utilizza Room con due entità principali: `EntityTest` memorizza i *test* acquisiti, mentre `EntitySavedConfigurationComparison` conserva gli *snapshot* delle configurazioni scelte dall'utente per confronti ripetibili. Questa separazione riflette due casi d'uso distinti: i *test* sono dati di misura, le configurazioni sono insiemi di parametri. Tutto è esposto tramite un DAO (`DatabaseDao`) con metodi `suspend`: le query sono naturalmente integrabili con le *coroutine*, evitando blocchi dell'UI e rendendo evidente che ogni accesso al disco avviene *off-main-thread*.

Per i *test*, la scelta architetturale adottata prevede il salvataggio dei dati sensoristici come file JSON separati nello *storage* interno, referenziati tramite il campo `fileName` nell'entità. Questa soluzione evita di memorizzare grandi *payload* direttamente nel database, mantenendo Room snello e performante. I campioni sono eterogenei e la loro struttura può evolvere nel tempo: preservare fedelmente il tracciato completo in file JSON consente di evitare migrazioni frequenti dello schema solo per aggiungere colonne. Il caricamento dei dati avviene tramite i metodi

`loadTestData()` e `loadTestValues()`, che leggono il file dal filesystem solo quando necessario all'elaborazione algoritmica.

Il contro di questa scelta è rinunciare a query SQL sui singoli campi dei campioni, ma qui non servono: l'analisi avviene nella *pipeline* algoritmica in memoria, non nel database. Per i metadati effettivamente utili alla UI (numero passi, note, nome file, data di registrazione) sono invece presenti colonne dedicate, che rendono snella la schermata di elenco senza dover aprire i file.

Per i confronti di configurazione (`EntitySavedConfigurationComparison`) è stato adottato un modello *testa e corpo*: alcuni campi atomici (nome, riferimento al *test*, *timestamp*) e un campo `configurationsJson` che contiene l'elenco delle configurazioni serializzate. La motivazione è la stabilità: la classe `Configuration` evolve con l'algoritmica, e serializzarla in JSON evita di rendere rigido Room con decine di colonne sensibili a modifiche. Gli *snapshot* possono essere riaperti su versioni successive dell'applicazione, e dove un campo nuovo manca, la deserializzazione può assegnare un valore predefinito senza compromettere i dati storici.

La serializzazione/deserializzazione è incapsulata in `ConfigurationSerializer`, un oggetto *stateless* che costruisce e ricostruisce `Configuration` da e verso JSON.

Il DAO è minimale e leggibile: metodi per ottenere ed eliminare *test*, cercare per ID o nome file, e le equivalenti operazioni per i confronti salvati. Per l'evoluzione dello schema è presente una migrazione esplicita che introduce la tabella dei confronti senza distruggere dati, testimoniando l'orientamento a migrazioni additive e non distruttive.

Il database aggrega le due entità necessarie al dominio. È presente una *foreign key* da `EntitySavedConfigurationComparison` verso `EntityTest`, la quale garantisce che l'eliminazione di un *test* rimuova automaticamente anche i confronti associati, mantenendo la coerenza referenziale.

In sintesi, l'architettura locale privilegia robustezza all'evoluzione e semplicità operativa: Room per i metadati stabili, file JSON per i *payload* complessi e le configurazioni versionabili, DAO sospensivi per *threading* corretto, migrazioni additive per non perdere dati. Il risultato è un sottosistema che non intralcia la ricerca algoritmica, ma la supporta.

3.5 Utils

Il *package* `utils` raccoglie le componenti di supporto dedicate alla conversione dei formati dati utilizzati dal sistema. Le classi, `CsvToJsonConverter` e

`JsonToCsvConverter`, consentono rispettivamente di importare *test* esterni in formato CSV e di esportarli nel medesimo formato. Entrambe gestiscono in modo flessibile diversi schemi di file, riconoscendoli automaticamente tramite analisi dell'intestazione. L'applicazione è resa compatibile anche con i file CSV registrati tramite l'applicazione MotionTracker [5], in modo da poter riutilizzarli in StepLab. Vengono convertiti in un formato conforme allo standard di quest'ultima, consentendo l'importazione senza perdita di dati e l'utilizzo in modalità di comparazione.

3.6 Considerazioni sull'implementazione

L'implementazione di StepLab presentata in questo capitolo riflette un approccio ingegneristico orientato alla modularità, alla manutenibilità e all'estensibilità. L'architettura a *package* separati consente di isolare le responsabilità: l'interfaccia utente gestisce la presentazione e l'interazione, gli algoritmi incapsulano la logica di elaborazione, la persistenza garantisce la conservazione dei dati e le utilità forniscono l'interoperabilità con sistemi esterni. Il risultato è un'applicazione funzionale allo scopo di questo progetto, in linea con quanto descritto riguardo al suo ruolo nell'architettura generale (Sezione 2.3).

Capitolo 4

Implementazione del sistema di iniezione dei dati

In questo capitolo si illustra l'implementazione del sistema di iniezione dei dati sensoristici nella sua completezza. Si descrive come avviene la registrazione dei dati, la loro memorizzazione e lo script di iniezione utilizzato per riprodurli.

4.1 Registrazione e memorizzazione dei dati

Come anticipato nel Capitolo 2, la registrazione dei dati sensoristici avviene tramite l'applicazione **MotionTracker**. Per gli esperimenti di questo elaborato non sono state utilizzate le funzionalità Bluetooth né sensori esterni, ma esclusivamente i sensori interni del dispositivo Android (accelerometro, giroscopio e magnetometro). Dal punto di vista implementativo, la responsabilità principale della raccolta dei dati è delegata ad un *foreground service*, `MonitoringService`, avviato dalla schermata principale (*HomeScreen*). Quando l'utente avvia una nuova registrazione, la *HomeScreen* genera (o recupera) un identificativo univoco e un timestamp di sessione, che vengono passati al servizio insieme ai metadati inseriti dall'utente.

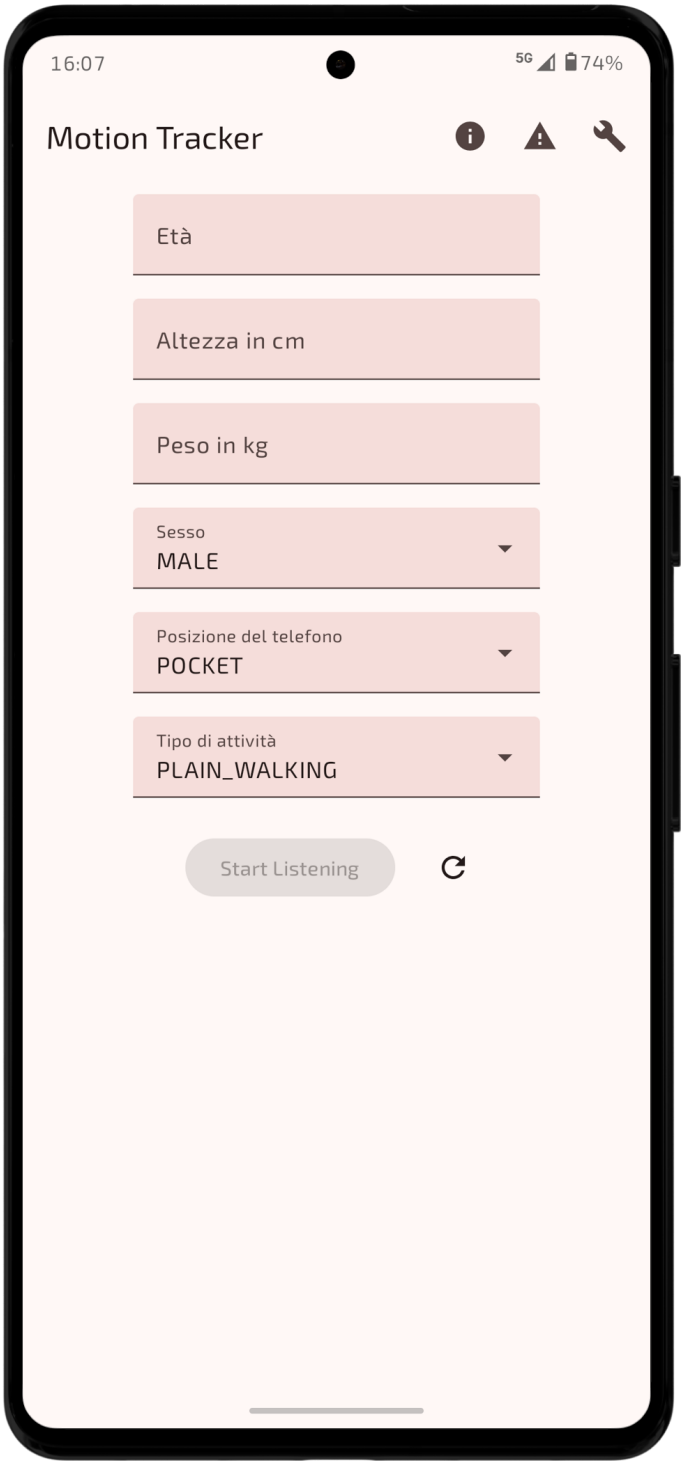

Figura 4.1: Schermata principale di MotionTracker.

I sensori sono gestiti tramite la classe astratta `AndroidSensor`, che incapsula la logica di interazione con il `SensorManager`. Nel codice originale, i sensori venivano campionati con `SENSOR_DELAY_NORMAL` ($\sim$ 5 Hz), producendo tracce molto povere e non utilizzabili ai fini della tesi: una camminata di 50 passi generava soltanto 150–200

campioni complessivi. Tale densità era insufficiente per qualsiasi analisi temporale e, soprattutto, i tre sensori aggiornando i valori in momenti diversi causavano un marcato disallineamento temporale tra accelerometro, giroscopio e magnetometro, rendendo il file incompatibile con il sistema di iniezione. Per risolvere questi problemi, la frequenza di campionamento è stata aumentata utilizzando `SENSOR_DELAY_GAME`, che porta la frequenza a circa 50 Hz, e la logica di raccolta è stata completamente ristrutturata. In `MonitoringService`, ogni listener aggiorna solo l'ultimo valore noto del rispettivo sensore, memorizzato in variabili condivise. La costruzione del campione da salvare è demandata a una funzione centrale (`checkAndStoreSampleImproved()`), che produce un nuovo record completo (accelerometro + giroscopio + magnetometro) ogni $\sim$ 20 ms, utilizzando i valori più recenti disponibili. Questo meccanismo elimina i problemi di disallineamento e produce tracce con timestamp regolari e densità elevata (1500–2000 campioni per 50 passi), massimizzando l'utilizzabilità del tracciato. Alla conclusione della registrazione, `MonitoringService` scrive i dati raccolti su un file CSV, che contiene sia i valori sensoristici sia i metadati della sessione. La tabella seguente mostra l'intestazione del file prodotto:

Colonna	**Descrizione**
Timestamp	Millisecondi dal sistema
AccelerometerX/Y/Z	Componenti dell'accelerazione (m/s^2)
GyroscopeX/Y/Z	Velocità angolare (rad/s)
MagnetometerX/Y/Z	Componenti del campo magnetico (µT)
Sex	Sesso dell'utente
Age	Età
Height	Altezza (cm)
Weight	Peso (kg)
Position	Posizione del telefono durante la registrazione
Activity	Tipo di attività svolta

Tabella 4.1: Intestazione del file CSV generato da MotionTracker.

Il file risultante viene quindi passato a `WorkManager`, che ne gestisce l'upload asincrono verso **Firebase Storage**, collocandolo in una cartella organizzata per data. Il worker dedicato (`SendToFirebaseWorker`) utilizza un'API sospensiva per attendere il completamento effettivo dell'upload. Nel complesso, le modifiche introdotte hanno reso i dati registrati effettivamente utilizzabili: la pipeline garantisce una frequenza

adeguata alla registrazione di camminate e una corretta sincronizzazione tra sensori, permettendo l'iniezione descritta nella sezione successiva.

4.2 Iniezione dei dati su emulatore

Lo script per l'iniezione dei dati è stato implementato in modo da poter interfacciarsi con l'emulatore Android e simulare i sensori interni. Qui i file registrati con MotionTracker vengono scaricati da *Firebase Storage* e riprodotti in modo da simulare una camminata reale, creando una pipeline semi-automatica per la raccolta dei risultati.

4.2.1 Avvio del sistema

Il prerequisito fondamentale per il corretto funzionamento del sistema è aver installato le tecnologie necessarie che vengono elencate nella Sezione 2.5.2. Il primo passo per l'utilizzo dello script è l'avvio dell'emulatore Android. Per questo progetto, è stato utilizzato l'emulatore fornito con Android Studio. Si rende poi necessario avviare il server *Appium* in modo che resti in ascolto per i comandi da eseguire:

```
appium --allow-insecure "chromedriver_autodownload:
   emulator_console"
```

Il *flag* `-allow-insecure` è necessario per abilitare funzionalità considerate potenzialmente rischiose da Appium. Nello specifico, `emulator_console` permette di accedere alla console dell'emulatore Android per inviare comandi diretti ai sensori virtuali. Senza questo parametro, Appium bloccherebbe l'accesso alla console per motivi di sicurezza, impedendo l'iniezione dei dati sensoristici. Questa configurazione è essenziale per simulare gli eventi dei sensori nell'emulatore durante la riproduzione delle tracce registrate. Al fine di rendere lo script flessibile e configurabile, è possibile modificare alcune variabili d'ambiente. Ciò modifica il comportamento dell'iniezione ma soprattutto si rende necessario per installare sull'emulatore le applicazioni coinvolte. Infatti, è necessario specificare i percorsi ai file APK di queste, che devono dunque essere scaricati preventivamente; l'installazione viene poi gestita automaticamente da Appium. Di seguito sono riportate le tabelle con le variabili d'ambiente disponibili e i loro valori di default.

Variabile	**Descrizione**	**Valore di default**
APPIUM_HOST	Host del server Appium	127.0.0.1
APPIUM_PORT	Porta del server Appium	4723
APPIUM_BASE_PATH	Path base API Appium	/
DEVICE_NAME	Nome dispositivo/emulatore	Android Emulator
AUTOMATION_NAME	Motore di automazione	UiAutomator2
NEW_COMMAND_TIMEOUT	Timeout comandi Appium	600 s
AUTO_GRANT_PERMISSIONS	Permessi auto-grant	true
NO_RESET	Mantiene stato delle app	true

Tabella 4.2: Variabili di configurazione del server Appium e dell'emulatore.

Variabile	**Descrizione**	**Valore di default**
APP_FORLANI_APK	Percorso locale dell'APK StepLab	-
APP_TAYUTAU_APK	Percorso APK Tayutau	-
APP_RUN_APK	Percorso APK Runtastic	-
APP_ACCUPEDO_APK	Percorso APK Accupedo	-
APP_WALKLOGGER_APK	Percorso APK Walklogger	-

Tabella 4.3: Percorsi degli APK installati automaticamente sull'emulatore.

Variabile	**Descrizione**	**Valore di default**
CSV_HAS_HEADER	Il CSV contiene l'intestazione	true
CSV_TIMES_ARE_MS	Timestamp in millisecondi	true
CSV_UNITS	Unità accelerometro (ms2/g)	ms2
CSV_GYRO_UNITS	Unità giroscopio (rad/s o dps)	rad/s
CSV_MAG_UNITS	Unità magnetometro (µT/mGauss)	µT
CSV_LAYOUT	Ordine colonne del CSV	`t,ax,ay,...`

Tabella 4.4: Variabili di configurazione relative al parsing e normalizzazione dei file CSV.

Variabile	**Descrizione**	**Valore di default**
IMMEDIATE_START	Avvio immediato dello streaming	true
INJECT_GYRO	Abilita l'iniezione del giroscopio	true
INJECT_MAG	Abilita l'iniezione del magnetometro	true
LOOP_REPEATS	Ripetizioni del file CSV	1
LOOP_GAP_MS	Pausa tra le ripetizioni	0 ms
AXIS_MAP	Rimappatura assi (es. XYZ)	XYZ
AXIS_SIGN	Segno assi (+/-)	+++
START_AHEAD_MS	Offset iniziale del tempo simulato	0

Tabella 4.5: Parametri che controllano il comportamento dell'injection dei dati sensoristici.

Variabile	**Descrizione**	**Valore di default**
FORLANI_RECORD_GAP_MS	Pausa dopo il salvataggio di un test	1000 ms
FORLANI_RECORD_LOOP	Ripeti registrazione automatica	false
FORLANI_CLICK_WAIT_MS	Attesa dopo ogni interazione UI	1200 ms

Tabella 4.6: Variabili di configurazione specifiche per l'automazione dell'app StepLab.

Lo script è stato dunque reso estremamente parametrico in modo da poter essere adattato a diverse situazioni senza dover modificare il codice sorgente. Una volta configurate quelle necessarie, si può procedere con l'esecuzione dello script. Si rende necessario passare come argomento il nome dell'applicazione da utilizzare, le opzioni sono:

- `forlani` per StepLab utilizzando la modalità *Live Testing*;
- `forlani_register` per StepLab utilizzando la modalità di registrazione dei test;
- `tayutau` per Tayutau;
- `run` per Runtastic;
- `accupedo` per Accupedo;

- `walklogger` per Walklogger.

Nel caso l'applicazione scelta sia StepLab in modalità di *Live Testing*, viene richiesto all'utente di specificare anche la configurazione da utilizzare. L'interazione avviene sempre, in ogni caso di questo script, tramite linea di comando. Le configurazioni disponibili sono quelle considerate mediamente più precise, queste sono:

- Filtro Butterworth + Rilevamento dei picchi;
- Filtro Butterworth + Rilevamento dei picchi + Filtraggio temporale;
- Filtro passa-basso con frequenza di taglio a 10 Hz + Rilevamento dei picchi + Filtraggio temporale;
- Filtro passa-basso con frequenza di taglio a 10 Hz + Rilevamento dei picchi + Intersezione con asse delle ascisse;
- Filtro passa-basso con frequenza di taglio al 2% della frequenza di campionamento + Rilevamento dei picchi.
- Filtro passa-basso con frequenza di taglio al 2% della frequenza di campionamento + Rilevamento dei picchi + Intersezione con asse delle ascisse.

Inoltre è necessario specificare anche che cosa processare. Si può passare un singolo file CSV, specificando il percorso, oppure utilizzare il comando `firebase` per scaricare automaticamente i file. In questo caso, viene richiesto di scegliere una cartella tra quelle disponibili su Firebase Storage, le quali contengono i file registrati tramite *MotionTracker*. Una volta scelta, i file vengono scaricati in locale e salvati in una cartella temporanea che sarà eliminata al termine dell'esecuzione dello script. Dunque, lo script può essere eseguito specificando gli argomenti necessari. Ecco un esempio di comando per eseguire l'iniezione dei dati sensoristici scaricando i file da Firebase.

```
node test_injection.js accupedo firebase
```

4.2.2 Automazione dell'interfaccia utente

In base agli argomenti passati, lo script procede con l'automazione dell'applicazione scelta. L'interazione con l'interfaccia utente avviene attraverso tre elementi principali:

- **WebdriverIO**, che fornisce l'API JavaScript per comunicare con Appium tramite il protocollo WebDriver e inviare comandi all'emulatore;
- **Appium**, che funge da ponte tra lo script e il dispositivo virtuale, traducendo le richieste WebDriver in operazioni concrete sui componenti Android;
- **UiAutomator2**, il framework nativo di Android utilizzato da Appium per eseguire le azioni sull'interfaccia.

Una volta selezionata l'applicazione (e, nel caso di StepLab, anche la configurazione dell'algoritmo), lo script richiama una funzione dedicata che contiene la sequenza di interazioni da simulare. Queste operazioni riproducono i gesti dell'utente, solitamente tocchi su pulsanti o scorrimenti per raggiungere elementi fuori dallo schermo. Queste azioni sono normalmente eseguite prima dell'iniezione dei dati, in modo da portare l'applicazione nello stato appropriato per il test. In alcune applicazioni, *Accupedo* e *Walklogger*, non è necessario alcun intervento manuale perché il conteggio dei passi inizia automaticamente all'avvio. In altre, come *StepLab* in modalità di registrazione dei test, lo script deve eseguire anche azioni dopo l'iniezione, ad esempio premere i pulsanti per completare e salvare la sessione. Se i file CSV da processare sono più di uno, l'intera sequenza di interazioni UI viene ripetuta automaticamente per ogni file. In questo modo, al termine della fase di automazione dell'interfaccia l'applicazione è sempre nello stato corretto per ricevere la traccia sensoristica e procedere con l'iniezione descritta nella sezione successiva.

4.2.3 Iniezione dei dati sensoristici

Dopo aver portato l'applicazione nello stato desiderato, lo script apre il file CSV da processare e lo legge riga per riga. Ogni riga rappresenta un campione sensoristico, contenente i valori dell'accelerometro, del giroscopio e del magnetometro in un preciso istante di tempo. Per ciascun campione, i valori vengono estratti, mappati nei rispettivi vettori e normalizzati. La procedura tiene conto del layout del CSV, di eventuali rimappature degli assi e di inversioni del segno specificate tramite variabili d'ambiente. Se necessario, i valori vengono anche convertiti nelle unità attese dai sensori virtuali dell'emulatore. In questo modo lo script rimane compatibile con file CSV provenienti da diverse sorgenti. L'iniezione non avviene in modo sequenziale: per simulare fedelmente il comportamento reale del dispositivo, lo script calcola l'istante esatto in cui ogni campione deve essere inviato, rispettando gli intervalli temporali originali della registrazione. Se il momento di emissione non

è ancora arrivato, lo script attende; dopodiché invia i valori all'emulatore. L'invio avviene tramite comandi diretti alla console dell'emulatore Android, come mostrato di seguito.

```
if (hasMag) await emuCmd(driver, 'sensor set magnetic-field '
    + mag[0] + ':' + mag[1] + ':' + mag[2]);
if (hasAcc) await emuCmd(driver, 'sensor set acceleration ' +
    acc[0] + ':' + acc[1] + ':' + acc[2]);
if (hasGyr) await emuCmd(driver, 'sensor set gyroscope ' +
   gyr[0] + ':' + gyr[1] + ':' + gyr[2]);
```

Al termine della simulazione, si rende necessaria un'ulteriore interazione con l'utente, come descritto nella sezione successiva.

4.2.4 Raccolta e salvataggio dei risultati

Terminata l'iniezione dei dati sensoristici contenuti in un file CSV, lo script apre un prompt su linea di comando che mostra il nome del file appena processato e le opzioni disponibili per l'utente:

- se viene inserito un numero, questo viene interpretato come il conteggio dei passi rilevati dall'applicazione;
- se viene premuto `r`, lo script ripete l'iniezione del file corrente, permettendo di rivedere i risultati senza riavviare la procedura;
- se viene premuto `n`, il risultato non viene salvato e lo script passa al file successivo (o termina se non vi sono altri file);
- se viene premuto `s`, l'intera procedura viene interrotta immediatamente.

Queste opzioni sono disponibili per tutte le applicazioni, ad eccezione della modalità `forlani_register`. In questo caso, al termine dell'iniezione, lo script completa automaticamente il flusso di salvataggio del test direttamente all'interno di *StepLab*. Se l'utente inserisce comunque un numero di passi, lo script accetta il valore ma segnala che questo non verrà salvato nei file CSV locali come avviene negli altri casi.

Quando l'utente inserisce un numero di passi (nelle modalità diverse da `forlani_register`), lo script crea automaticamente, se non esiste, un file CSV

dedicato all'applicazione e alla configurazione selezionata. In questo file vengono salvati il numero di passi inserito insieme a una serie di metadati estratti dal nome del file CSV originale. Il formato del file prodotto è il seguente:

time-stamp	**csv_file**	**walking type**	**phone position**	**age**	**gen-der**	**device**	**steps counted**

Tabella 4.7: Formato del file CSV per il salvataggio dei risultati.

Il nome del file CSV processato, riportato nella seconda colonna, viene utilizzato dallo script per riconoscere i file già analizzati ed evitare duplicazioni nelle esecuzioni successive. È inoltre presente una diramazione opzionale del flusso principale: se si utilizza *StepLab* con la configurazione che combina filtro passa-basso al 2% della frequenza di campionamento e rilevamento dei picchi, lo script chiede all'utente se attivare la *modalità di verifica*. Se questa viene abilitata, dopo l'inserimento dei passi rilevati mediante iniezione viene richiesto di inserire anche il numero di passi stimato tramite la modalità di comparazione offline di *StepLab* (Capitolo 3). Questa procedura genera un secondo file CSV di verifica, con il seguente formato:

file name	**steps live**	**steps batch**	**error**	**absolute error**

Tabella 4.8: Formato del file CSV di verifica dei risultati.

La modalità di verifica e la registrazione dei test di *StepLab* permettono un confronto diretto fra:

- il conteggio ottenuto tramite iniezione (*simulazione live*),
- il conteggio ottenuto tramite elaborazione offline dello stesso file (*batch*).

Questo confronto consente di misurare l'accuratezza dell'intero processo di iniezione e verificare se, e in quale misura, l'architettura introduce errori nella riproduzione dei dati sensoristici. Nel complesso, la procedura di raccolta dei risultati completa il ciclo di test, fornendo i dati necessari per le analisi illustrate nei Capitoli successivi.

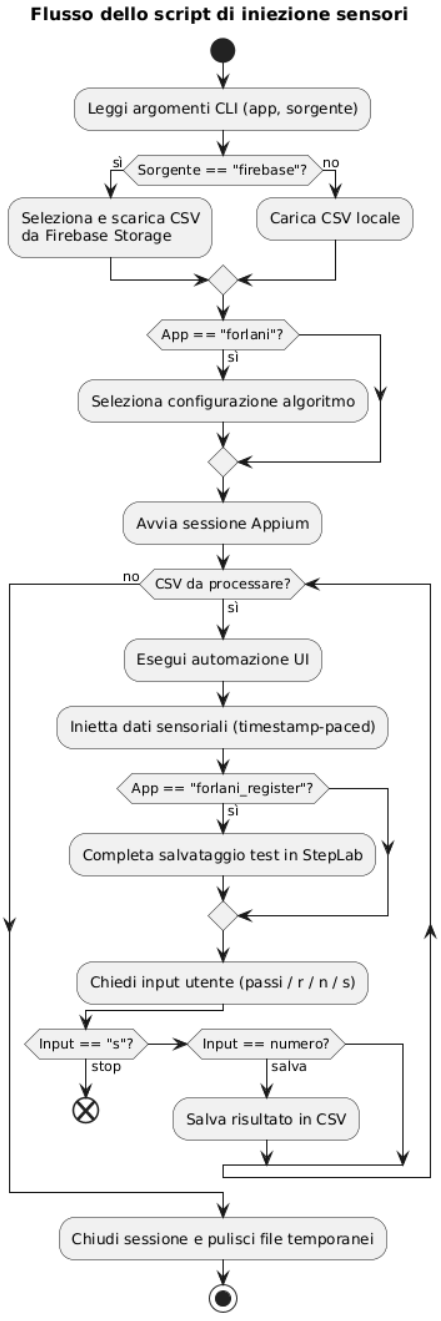

Figura 4.2: Riassunto del flusso dello script di iniezione dei dati sensoristici.

Capitolo 5

Validazione del sistema

L'obiettivo di questo capitolo è validare il sistema di iniezione dei dati sensoristici, ovvero verificare se i dati riprodotti nell'emulatore Android sono fedeli alle registrazioni originali e, in caso contrario, quantificarne lo scostamento. Per effettuare tale validazione è stata utilizzata l'applicazione *StepLab* che, come illustrato nel Capitolo 3, dispone di una funzionalità dedicata alla comparazione ripetibile di registrazioni di camminate. Alla fine del Capitolo 4 è stato inoltre descritto come lo script di iniezione includa specifiche modalità pensate proprio per supportare questa validazione, automatizzando sia la simulazione dei dati sia la registrazione dei risultati.

In questo capitolo vengono presentati la metodologia adottata, i risultati ottenuti e le considerazioni finali.

5.1 Metodologia

Per validare il sistema di iniezione si è deciso di confrontare i conteggi dei passi ottenuti tramite riproduzione dei dati sensoristici con quelli ottenuti elaborando direttamente le registrazioni originali, senza passare dallo strato di iniezione. A questo scopo l'utilizzo di *StepLab* è imprescindibile: l'applicazione mette infatti a disposizione sia la modalità *live*, che consente di contare i passi in tempo reale durante la simulazione, sia la modalità *offline*, che permette di caricare un file CSV e analizzarlo direttamente. Questa duplice modalità rende possibile un confronto rigoroso tra i due processi, isolando l'eventuale errore introdotto dall'iniezione. Per il confronto è stata selezionata una specifica configurazione di *StepLab*, ovvero quella che combina un filtro passa-basso con frequenza di taglio pari al 2% della frequenza di campionamento con l'algoritmo di rilevamento dei picchi. Come descritto nel

Capitolo 4, si tratta dell'unica configurazione che include una modalità di verifica progettata appositamente per registrare i risultati del confronto tra esecuzione *live* e analisi *offline*. Dal punto di vista operativo, il flusso di lavoro seguito per ogni camminata è stato il seguente:

1. esecuzione della simulazione tramite iniezione dei dati registrati con *MotionTracker*;

2. visualizzazione manuale, su un dispositivo reale, del conteggio dei passi ottenuto dalla modalità di comparazione offline di *StepLab*;

3. registrazione dei due conteggi, quello ottenuto tramite iniezione e quello ottenuto tramite elaborazione offline, in un file CSV utilizzando le funzionalità dello script.

Il confronto è stato effettuato sull'intero dataset registrato, per un totale di 360 simulazioni, producendo un file completo di risultati, il cui formato è illustrato nella Tabella 4.8. Nella sezione successiva vengono analizzati nel dettaglio i risultati ottenuti.

5.2 Risultati della validazione

Il risultato ideale sarebbe stato l'assenza totale di errore, ovvero che il conteggio dei passi ottenuto tramite iniezione coincidesse esattamente con quello calcolato direttamente sui file originali. L'analisi dei dati ha però evidenziato un certo scostamento tra i due conteggi. Inoltre, è importante sottolineare che l'errore osservato è relativo alla specifica combinazione di algoritmi utilizzata. Il segno dello scostamento e, seppur in misura non troppo marcata, la sua entità possono variare in base alla configurazione scelta. I risultati sono dunque da intendersi come una misura indicativa dell'accuratezza del sistema e non come una misura assoluta valida per ogni possibile pipeline di rilevamento.

5.2.1 Metriche di valutazione

Per quantificare l'accuratezza del sistema di iniezione sono state utilizzate diverse metriche statistiche, ciascuna con uno specifico obiettivo interpretativo:

- **MAE (Mean Absolute Error)**: rappresenta l'errore assoluto medio, calcolato come

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

dove y_i è il valore reale, ovvero quello che si ottiene senza passare dall'iniezione, e $\hat{y}_i$ è il valore predetto, il conteggio tramite iniezione. Questa metrica fornisce una misura della distanza media tra i valori predetti e quelli reali, indipendentemente dalla direzione dell'errore.

- **Errore medio con segno (Signed Mean Error)**: a differenza del MAE, questa metrica preserva il segno dell'errore:

$$\text{SME} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

Un valore positivo indica una tendenza generale alla sovrastima, mentre un valore negativo indica sottostima. Questa metrica è fondamentale per identificare bias sistematici nel sistema di conteggio.

- **Mediana**: valore centrale della distribuzione degli errori assoluti. Rispetto alla media, la mediana è meno sensibile alla presenza di valori estremi (outliers) e fornisce quindi una misura più robusta della tendenza centrale.

- **Outliers**: valori anomali identificati mediante il metodo IQR (Interquartile Range). Un dato è considerato outlier se cade al di fuori dell'intervallo $[Q_1 - 1.5 \cdot \text{IQR}, Q_3 + 1.5 \cdot \text{IQR}]$, dove Q_1 e Q_3 sono rispettivamente il primo e il terzo quartile, e $\text{IQR} = Q_3 - Q_1$. L'identificazione degli outliers permette di distinguere errori occasionali da quelli sistematici.

- **Deviazione standard**: misura la dispersione degli errori attorno alla loro media. È definita, per un campione di valori $x_1, x_2, \ldots, x_n$, come

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2},$$

dove $\bar{x}$ è la media campionaria. Una deviazione standard elevata indica che gli errori sono molto variabili , alternando sovrastime e sottostime di entità diversa. Al contrario, una deviazione standard ridotta segnala un comportamento più regolare e prevedibile.

5.2.2 Analisi dei risultati

Sul totale delle simulazioni effettuate con la configurazione selezionata di *StepLab*, sono stati riscontrati:

- 193 sovrastime (53,5%);
- 33 sottostime (9,4%);
- 134 conteggi corretti (37,1%).

L'analisi statistica degli errori è stata condotta distinguendo fra statistiche totali, per tipologia di camminata e per posizione del dispositivo. La Tabella 5.1 riporta una sintesi delle metriche principali a livello globale.

Metrica	**Con Outliers**	**Senza Outliers**	**Riduzione (%)**
MAE	2.856	2.395	16.1%
SME	2.418	1.943	19.6%
Mediana MAE	2	2	—
Mediana SME	2	1	50.0%
Dev. Std. MAE	3.74	2.67	28.6%
Dev. Std. SME	4.038	3.017	25.3%

Tabella 5.1: Statistiche generali di MAE e SME.

Nella Tabella 5.2 vengono riportate le metriche di errore suddivise per tipologia di camminata, mentre nella Tabella 5.3 sono mostrate le stesse metriche stratificate per posizione del dispositivo.

Tipo	**MAE**		**SME**	
	c/o	s/o	c/o	s/o
DOWNHILL_WALKING	3.836	3.018	3.443	2.596
UPHILL_WALKING	3.417	2.793	3.283	2.655
BABY_STEPS	3.288	2.895	2.847	2.439
IRREGULAR_STEPS	2.767	1.875	2.2	1.268
PLAIN_WALKING	2.623	2.623	2.098	2.098
RUNNING	1.2	1.2	0.633	0.633

Tabella 5.2: Metriche di errore per tipologia di camminata (c/o = con outliers, s/o = senza outliers).

Posizione	**MAE**		**SME**	
	c/o	s/o	c/o	s/o
POCKET	4.185	3.092	3.782	2.651
HAND	2.478	2.333	1.887	1.737
SHOULDER	1.953	1.849	1.622	1.516

Tabella 5.3: Metriche di errore per posizione del dispositivo (c/o = con outliers, s/o = senza outliers).

Dai risultati emerge che la presenza di outliers incrementa il MAE complessivo, senza di questo la riduzione è di circa il 16%. L'effetto è ancora più marcato per il SME, che diminuisce di quasi il 20%. Le condizioni di camminata in salita e discesa presentano generalmente gli errori più elevati, suggerendo che tali movimenti sono più difficili da riprodurre fedelmente tramite iniezione. In una situazione simile si trovano le metriche riguardanti i passi stretti, leggermente inferiori rispetto a quelle delle camminate in pendenza, ma comunque più alte rispetto agli altri tipi di camminata, eccezion fatta per il MAE senza outliers che è più elevato anche di quello della camminata in salita. Al contrario, la corsa risulta l'attività più stabile, con un MAE di 1.2 passi e un SME inferiore a 1. Per quanto riguarda la posizione del dispositivo, la tasca (*POCKET*) introduce l'errore maggiore, mentre la spalla (*SHOULDER*) risulta la posizione più accurata. Lo SME mostra una tendenza generale alla sovrastima, confermata dalla presenza di un numero significativamente

maggiore di sovrastime rispetto alle sottostime. Nel complesso, l'errore assoluto medio rimane comunque contenuto, suggerendo che il sistema di iniezione riesce a riprodurre in maniera ragionevolmente fedele la dinamica della camminata, pur introducendo un inevitabile margine di imprecisione. Prima di definire lo scostamento riportato nelle tabelle di questa sezione, si è prima cercato di identificare la causa principale degli scostamenti riscontrati e di applicare le correzioni necessarie. Il lavoro svolto è presentato nella sezione successiva.

5.3 Considerazioni sulla metodologia

Prima di interpretare i risultati ottenuti, è stato necessario indagare se gli errori osservati provenissero dall'implementazione di StepLab oppure dal meccanismo stesso di iniezione. Fin dalle prime esecuzioni della validazione è stato osservato un disallineamento tra i conteggi ottenuti tramite iniezione e quelli derivanti dall'elaborazione offline dei file originali. Per individuarne la causa, alcuni test sono stati eseguiti utilizzando *StepLab* in modalità di registrazione dei test, in modo da ottenere nuovamente i tracciati sensoristici dopo la simulazione ed analizzarli direttamente.

Il confronto tra i file registrati tramite iniezione e quelli originali ha confermato che la discrepanza non era legata alle modalità di conteggio di *StepLab*, ma al processo di iniezione stesso: i file post-iniezione risultavano infatti alterati. In particolare, si è osservata la presenza di righe duplicate o campioni mancanti, oltre a differenze nel numero totale di record.

L'analisi ha evidenziato che la causa principale risiede nella temporizzazione dei sensori virtuali dell'emulatore Android. Tali sensori non aggiornano il proprio valore a ogni comando `sensor set`, ma secondo una loro frequenza interna di campionamento, come viene confermato dalla documentazione ufficiale [2]. Se il comando arriva troppo vicino a una lettura già programmata, il valore precedente può essere letto due volte (determinando la duplicazione), oppure il nuovo valore può essere ignorato (determinando la perdita di campioni). Questo effetto è amplificato dalla latenza intrinseca della console dell'emulatore, che non garantisce capacità real-time.

Per verificare se il problema fosse mitigabile, sono stati condotti ulteriori esperimenti modificando le frequenze di registrazione e di iniezione, provando sia a registrare a frequenze più elevate sia a iniettare i dati verso sensori configurati a frequenze inferiori. In nessuno dei due casi si è ottenuto un miglioramento significativo: anzi, l'iniezione ad alta frequenza ha talvolta aumentato la perdita di campioni.

Si è dunque concluso che il problema non è imputabile alla latenza dello script, ma è una limitazione strutturale dell'emulatore Android, che non permette di controllare né l'istante di acquisizione dei sensori virtuali, né la loro frequenza effettiva. Di conseguenza, una perfetta fedeltà tra registrazione originale e tracciato iniettato non è tecnicamente ottenibile.

In ogni caso, gli errori introdotti rimangono limitati e non compromettono la validità della valutazione complessiva. Per questo motivo la validazione è stata condotta come descritto, accettando un inevitabile margine di imprecisione dovuto a tale vincolo tecnico.

Capitolo 6

Raccolta dati e risultati

In questo capitolo viene presentato il dataset ottenuto dalla raccolta dati iniziale e vengono descritti i risultati delle simulazioni condotte utilizzando il sistema di iniezione illustrato nel Capitolo 4.

6.1 Raccolta dati

La raccolta dati, come anticipato nella sezione 2.1, è stata effettuata tramite *MotionTracker* in contesti reali quali parchi pubblici e aree urbane. Ogni sessione è stata supervisionata dallo sperimentatore, che ha fornito indicazioni ai partecipanti sulle attività da svolgere. Ogni volontario ha eseguito sei tipologie di camminata, selezionate per coprire una varietà ampia di condizioni locomotorie:

- Camminata normale
- Corsa
- Camminata a passi stretti
- Camminata a passi irregolari
- Camminata in salita
- Camminata in discesa

Ogni camminata è stata registrata in tre diverse posizioni del dispositivo, ovvero **in mano**, **in tasca** e **in spalla** tramite supporto da braccio. In totale sono stati raccolti 18 tracciati per ciascun partecipante, per un totale di 20 volontari e 360 registrazioni complessive. In alcuni casi è stato necessario riutilizzare lo stesso

dispositivo per partecipanti diversi, qualora non disponessero di uno smartphone Android compatibile con l'applicazione utilizzata.

Età	**Sesso**	**Dispositivo**
21	MALE	Xiaomi M2003J15SC
22	MALE	Xiaomi M2003J15SC
22	MALE	motorola motorola edge 50 fusion
21	MALE	motorola motorola edge 50 fusion
22	MALE	Xiaomi M2003J15SC
22	MALE	samsung SM-A505FN
23	MALE	samsung SM-A505FN
22	MALE	TCL 5030D EEA
18	MALE	samsung SM-A346B
19	MALE	samsung SM-A346B
18	FEMALE	samsung SM-A346B
57	MALE	samsung SM-A326B
23	MALE	motorola motorola edge 50 fusion
22	MALE	motorola motorola edge 50 fusion
55	FEMALE	Xiaomi 2109119DG
53	FEMALE	Xiaomi 2109119DG
22	MALE	samsung SM-G770F
22	MALE	samsung SM-G770F
21	MALE	OPPO CPH2219
23	MALE	OPPO CPH2219

Tabella 6.1: Età, sesso e dispositivo dei partecipanti alle registrazioni.

6.2 Configurazione delle applicazioni

I file registrati sono stati utilizzati per condurre le simulazioni secondo le modalità descritte nel Capitolo 4. Le applicazioni utilizzate per il conteggio dei passi sono:

- Tayutau
- Runtastic
- Accupedo

- Walklogger
- StepLab

Le prime quattro sono applicazioni commerciali che consentono unicamente di regolare la sensibilità del conteggio; per garantire uniformità sperimentale è stata selezionata la sensibilità intermedia disponibile in ciascuna di esse.

StepLab, invece, è interamente configurabile e permette di combinare filtri, algoritmi di rilevamento e modalità di elaborazione differenti. Sono state selezionate sei configurazioni: tre classificate come *Real Time* e tre come *Non Real Time*, seguendo la distinzione adottata in [13] e [19]. Le configurazioni *Real Time* sono state selezionate in base alle performance migliori ottenute nello studio [13]:

- Filtro passa-basso 10 Hz + Rilevamento dei picchi + Intersezione con asse delle ascisse
- Filtro passa-basso al 2% della frequenza di campionamento + Rilevamento dei picchi + Intersezione con asse delle ascisse
- Filtro passa-basso al 2% della frequenza di campionamento + Rilevamento dei picchi

Le configurazioni *Non Real Time* sono state scelte effetuando una valutazione preliminare delle performance e scegliendo le tre più accurate:

- Filtro Butterworth + Rilevamento dei picchi
- Filtro passa-basso 10 Hz + Rilevamento dei picchi + Filtraggio temporale
- Filtro Butterworth + Rilevamento dei picchi + Filtraggio temporale

I risultati ottenuti vengono presentati nella sezione successiva.

6.3 Risultati delle simulazioni

In questa sezione, vengono analizzate le performance delle applicazioni di conteggio dei passi. L'analisi viene divisa in due parti, in base alla tipologia di camminata ed in base alla posizione del telefono durante la registrazione. Viene poi fornita un'analisi complessiva dei risultati. Le metriche valutate vengono illustrate nella sezione 5.2.1, con la differenza che in questo caso il valore reale y_i rappresenta il

conteggio dei passi effettuati, quindi 50, il valore predetto $\hat{y}_i$ è il conteggio ottenuto dall'applicazione o dalla configurazione di *StepLab* in esame.

Per rappresentare i risultati sono stati utilizzati due tipi di grafici:

1. Box plot. I box plot mostrano la distribuzione completa dei passi contati dalle diverse applicazioni o configurazioni. Evidenziano la mediana, i quartili e gli eventuali valori anomali (outliers). Questo tipo di grafico permette di valutare la stabilità di un algoritmo, di identificare la presenza di valori estremi e di osservare quanto le misurazioni siano concentrate attorno al valore atteso.

2. Grafici a barre con intervalli di confidenza al 95%. Per ciascuna applicazione è stato calcolato il valore medio dell'errore assoluto (MAE) e rappresentato tramite un grafico a barre accompagnato da un intervallo di confidenza al 95%. L'intervallo è stato stimato mediante *bootstrap*, una tecnica di campionamento che costruisce una stima dell'incertezza ri-generando numerosi insiemi di dati ottenuti tramite ricampionamento con rimpiazzamento. Per ogni ricampionamento viene ricalcolata la media: la distribuzione delle medie così ottenuta permette di stimare direttamente l'intervallo di confidenza, corrispondente ai percentili 2.5% e 97.5% della distribuzione bootstrap. Le *error bars* rappresentano quindi l'incertezza statistica sulla stima del MAE, indicando quanto la media potrebbe variare se l'esperimento venisse ripetuto più volte nelle stesse condizioni.

Come anticipato nella sezione 5.2, i risultati devono essere considerati indicativi. Per questo motivo, in questa sezione non vengono applicate correzioni basate sugli scostamenti osservati nella fase di validazione: l'incertezza complessiva delle misurazioni viene valutata e comunicata attraverso gli intervalli di confidenza.

6.3.1 Risultati per tipologia di camminata

La tipologia di camminata influenza significativamente le performance dei pedometri. Illustriamo dunque i risultati ottenuti per ciascuna delle sei tipologie considerate.

Camminata normale

La camminata normale rappresenta il caso classico per l'analisi delle prestazioni dei pedometri, poiché caratterizzata da un'andatura regolare e da un segnale accelerometrico generalmente favorevole al rilevamento dei passi. La Tabella 6.2 riassume il MAE e lo SME per ciascuna applicazione e configurazione di *StepLab*, limitatamente alla camminata normale.

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	9.623	-6.246
Tayutau	5.836	-2.689
WalkLogger	9.525	+1.328
Accupedo	11.590	-1.230
StepLab (Peak + Butterworth)	7.820	-1.131
StepLab (Peak + Intersection + Low-Pass 10Hz)	12.623	-11.377
StepLab (Peak + Intersection + Low-Pass 2%)	30.607	-30.607
StepLab (Peak + Low-Pass 2%)	10.607	-9.230
StepLab (Peak + Time Filter + Low-Pass 10Hz)	14.639	+8.443
StepLab (Time Filter + Peak + Butterworth)	6.475	-2.049

Tabella 6.2: Metriche di accuratezza per la camminata normale.

La Figura 6.1 visualizza graficamente il MAE di ciascuna applicazione e configurazione, con gli intervalli di confidenza al 95% che permettono di valutare l'incertezza statistica delle stime.

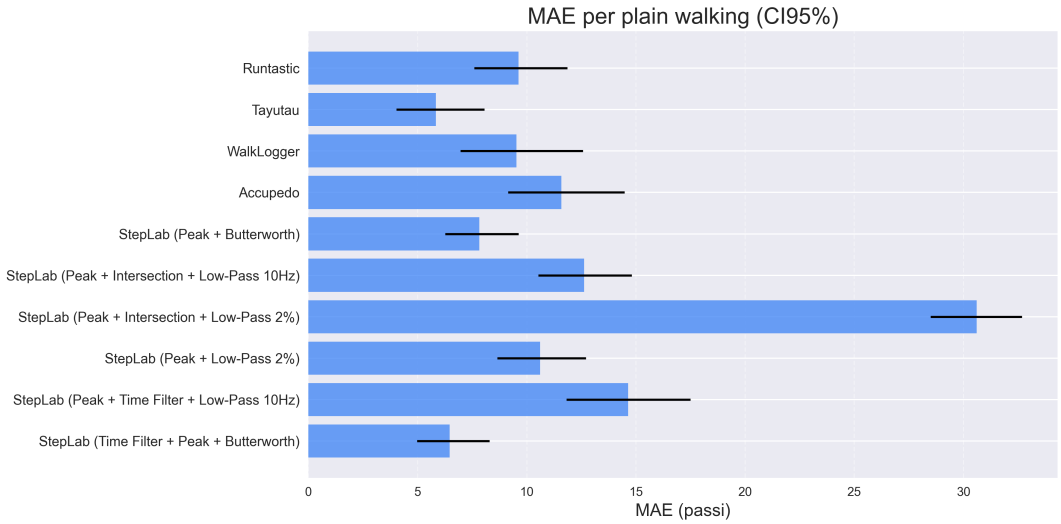

Figura 6.1: MAE con intervalli di confidenza al 95% per camminata normale.

Per le applicazione commerciali, l'analisi dei risultati mostra una forte eterogeneità nelle prestazioni. Il box plot in Figura 6.2 permette di visualizzare la dispersione dei valori e la stabilità dei diversi pedometri. Tra le applicazioni esaminate, **Tayutau** risulta la più accurata nella camminata normale: presenta il MAE più basso (5.836)

e uno SME relativamente contenuto (-2.689), indicando una tendenza moderata alla sottostima ma senza deviazioni estreme. **Runtastic** e **WalkLogger** mostrano invece un comportamento più variabile, con box plot più ampi e la presenza di valori anomali. Runtastic tende chiaramente alla sottostima (SME -6.246), mentre WalkLogger alterna sovrastime e sottostime, riflettendo uno SME vicino allo zero ma con scarsa precisione (MAE 9.525). **Accupedo** è l'app commerciale con le prestazioni peggiori sulla camminata normale, con un MAE di 11.590, nonostante uno SME abbastanza contenuto (-1.230), segno che il problema è da attribuire soprattutto alla dispersione.

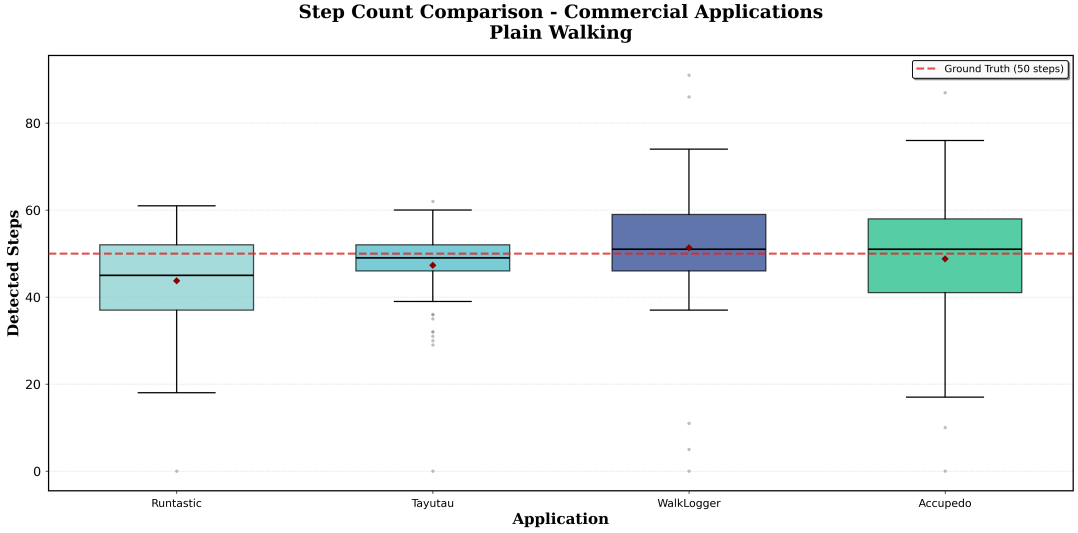

Figura 6.2: Box plot dei risultati per camminata normale delle applicazioni commerciali.

Le configurazioni di *StepLab* presentano una variabilità più marcata. Il box plot in Figura 6.3 evidenzia come alcune combinazioni siano altamente affidabili, mentre altre introducano errori sistematici significativi. Le configurazioni più stabili sono **Time Filter + Peak + Butterworth** (MAE = 6.475) e **Peak + Butterworth** (MAE = 7.820) entrambe caratterizzate da SME moderati e distribuzioni concentrate. All'estremo opposto, filtraggi troppo aggressivi generano prestazioni molto scarse. La configurazione **Peak + Intersection + Low-Pass 2%** è quella con l'accuratezza peggiore di tutto il set, con un MAE di 30.607 e uno SME pari a -30.607, segno evidente di una sistematica e grave sottostima del numero di passi. Anche **Peak + Intersection + Low-Pass 10Hz** presenta un errore significativo (MAE 12.623), confermando che la combinazione di più operazioni di filtraggio può introdurre ritardi o attenuazioni che compromettono la rilevazione dei passi.

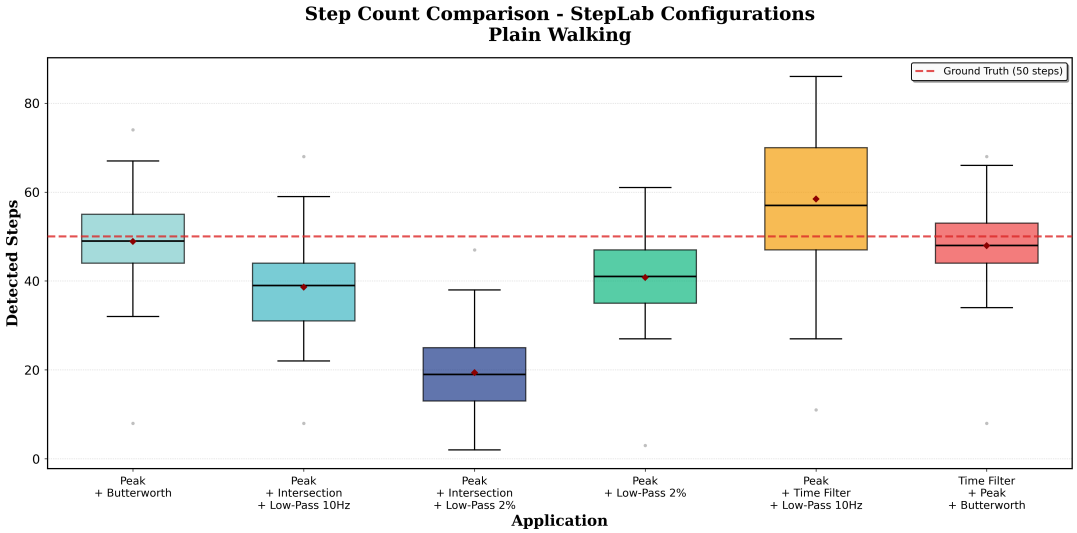

Figura 6.3: Box plot dei risultati per camminata normale di StepLab.

Corsa

La corsa presenta caratteristiche dinamiche diverse rispetto alla camminata: le accelerazioni sono più elevate, la frequenza dei passi aumenta e il segnale tende ad essere più regolare ma più energico. Ciò rende la rilevazione dei passi in alcuni casi più semplice, ma può anche accentuare gli errori sistematici nei filtraggi più aggressivi. La Tabella 6.3 riassume il MAE e lo SME relativi alla corsa per tutte le applicazioni commerciali e per le configurazioni di *StepLab*.

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	4.333	-1.667
Tayutau	6.583	-1.950
WalkLogger	6.117	+2.350
Accupedo	9.217	+3.983
StepLab (Peak + Butterworth)	5.167	+0.933
StepLab (Peak + Intersection + Low-Pass 10Hz)	7.983	-6.683
StepLab (Peak + Intersection + Low-Pass 2%)	29.683	-29.683
StepLab (Peak + Low-Pass 2%)	3.717	-1.183
StepLab (Peak + Time Filter + Low-Pass 10Hz)	6.833	+3.467
StepLab (Time Filter + Peak + Butterworth)	3.950	+0.483

Tabella 6.3: Metriche di accuratezza per la corsa.

La rappresentazione grafica del MAE con intervalli di confidenza è mostrata in Figura 6.4.

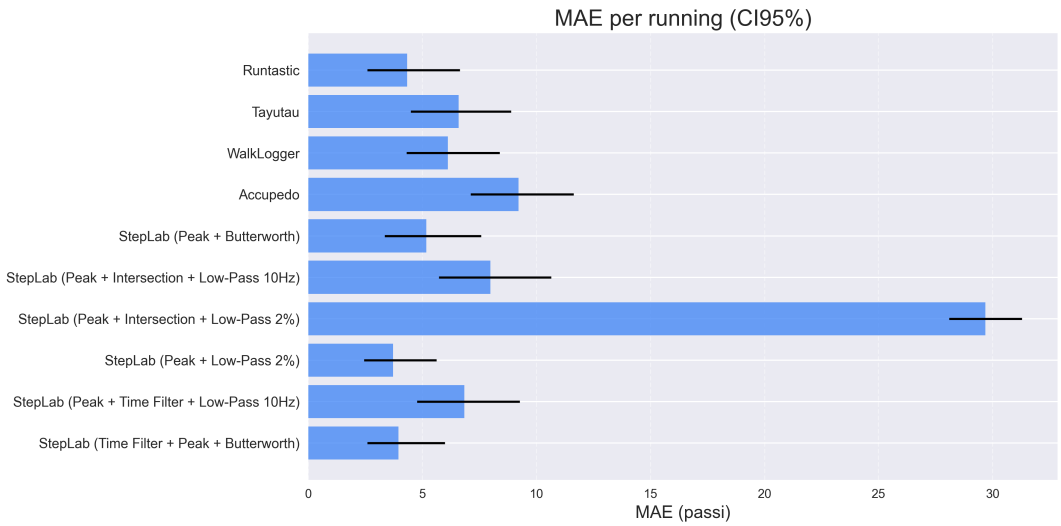

Figura 6.4: MAE con intervalli di confidenza al 95% per la corsa.

Per quanto riguarda le applicazioni commerciali, i risultati mostrano una maggiore stabilità rispetto alla camminata normale. Il box plot in Figura 6.5 evidenzia che tutte le app tendono a raggrupparsi attorno al valore atteso dei 50 passi, con dispersioni più contenute. **Runtastic** è l'app migliore in questo scenario (MAE = 4.333), con una leggera tendenza alla sottostima. Anche **Tayutau** si comporta bene, mentre **Accupedo** mostra la dispersione più ampia e una marcata sovrastima (SME +3.983). **WalkLogger**, pur avendo uno SME vicino allo zero, presenta comunque una certa variabilità.

Le configurazioni di *StepLab* mostrano un comportamento più diversificato. Il box plot in Figura 6.6 mette in evidenza come alcune configurazioni si adattino molto bene alla corsa, mentre altre risultino inadatte. Le configurazioni più accurate sono **Peak + Low-Pass 2%** (MAE = 3.717), la migliore in assoluto, e **Time Filter + Peak + Butterworth** (MAE = 3.950), entrambe caratterizzate da errori bassi e distribuzioni compatte. Al contrario, configurazioni che combinano filtraggio aggressivo e operazioni di intersezione con l'asse delle ascisse mostrano prestazioni pessime: **Peak + Intersection + Low-Pass 2%** produce un MAE estremamente elevato (29.683) con una sottostima sistematica di pari valore.

Nel complesso, la corsa risulta un contesto favorevole per il conteggio dei passi. Tutte le applicazioni registrano SME inferiori rispetto alla camminata normale, e alcune configurazioni di *StepLab* raggiungono prestazioni particolarmente elevate.

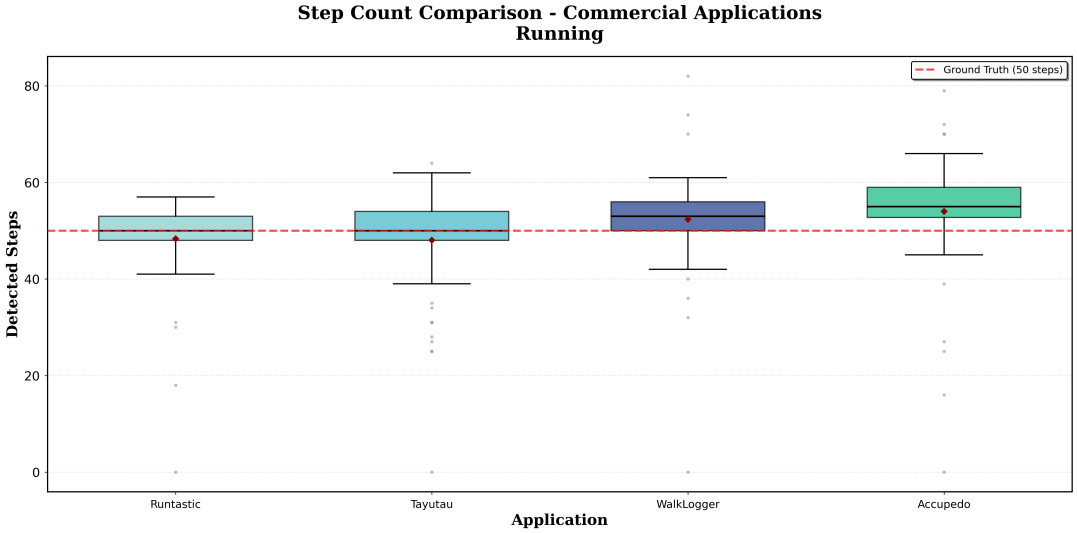

Figura 6.5: Box plot dei risultati per la corsa delle applicazioni commerciali.

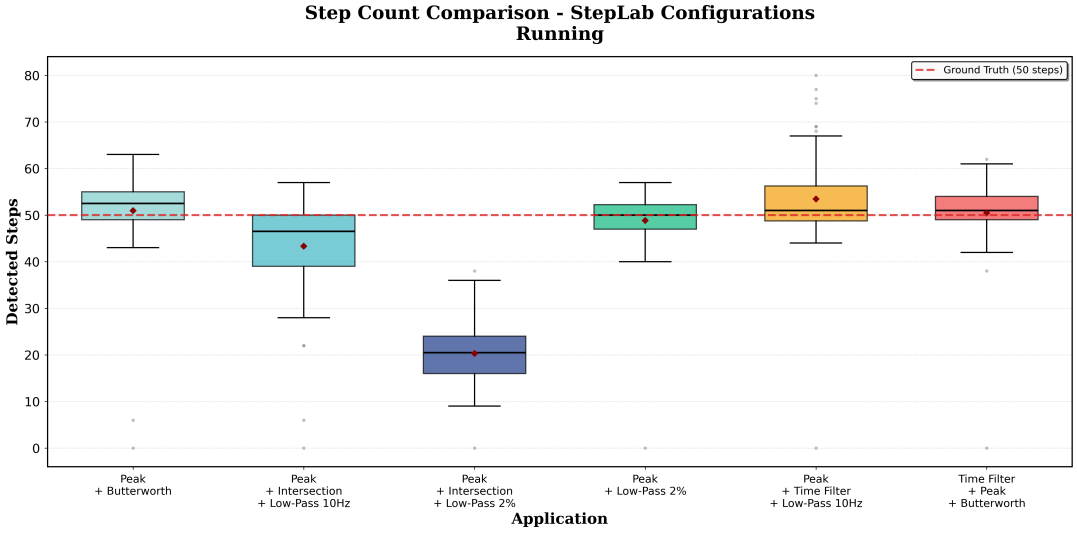

Figura 6.6: Box plot dei risultati per la corsa con le configurazioni di StepLab.

Camminata a passi stretti

La camminata a passi stretti è caratterizzata da un'accelerazione più contenuta rispetto alla camminata normale. Le oscillazioni risultano meno pronunciate e i picchi associati ai passi diventano meno distinti, rendendo questa tipologia una delle più difficili da rilevare correttamente. Nella Tabella 6.4 è possibile visualizzare le metriche relative a questa attività.

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	11.683	-8.250
Tayutau	4.867	-0.100
WalkLogger	11.400	-2.867
Accupedo	14.200	-2.333
StepLab (Peak + Butterworth)	8.000	-3.567
StepLab (Peak + Intersection + Low-Pass 10Hz)	15.017	-14.517
StepLab (Peak + Intersection + Low-Pass 2%)	32.867	-32.867
StepLab (Peak + Low-Pass 2%)	17.966	-17.458
StepLab (Peak + Time Filter + Low-Pass 10Hz)	12.883	+5.850
StepLab (Time Filter + Peak + Butterworth)	8.200	-4.967

Tabella 6.4: Metriche di accuratezza per la camminata a passi stretti.

La Figura 6.7 mostra graficamente i valori di MAE con i rispettivi intervalli di confidenza.

Per quanto riguarda le applicazioni commerciali, le prestazioni peggiorano rispetto alle tipologie precedenti, coerente con la maggiore difficoltà nel rilevare picchi deboli. Dal box plot in Figura 6.8 emerge che **Tayutau** rimane l'app più stabile anche in questo contesto: ottiene il MAE più basso (4.867) e uno SME quasi nullo. **Runtastic** e **WalkLogger** evidenziano un errore sensibilmente maggiore rispetto al caso precedente, mentre **Accupedo** si conferma offrire le prestazioni peggiori a livello di errore assoluto.

Le configurazioni di *StepLab* presentano una variabilità ancora più marcata. Il box plot in Figura 6.9 conferma un comportamento simile a quelli osservati finora. Le configurazioni più accurate sono **Peak + Butterworth** (MAE = 8.000) e **Time Filter + Peak + Butterworth** (MAE = 8.200), mentre la peggiore è nuovamente

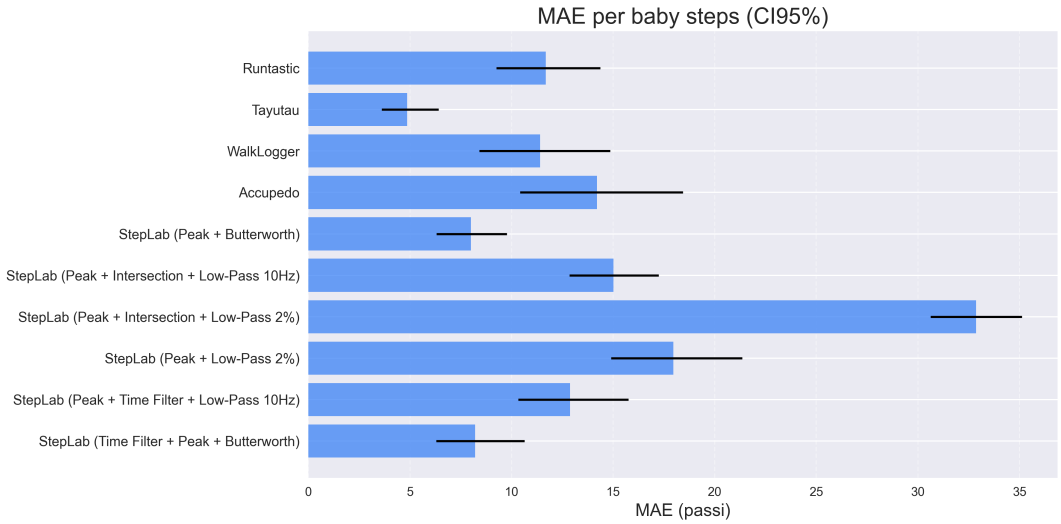

Figura 6.7: MAE con intervalli di confidenza al 95% per la camminata a passi stretti.

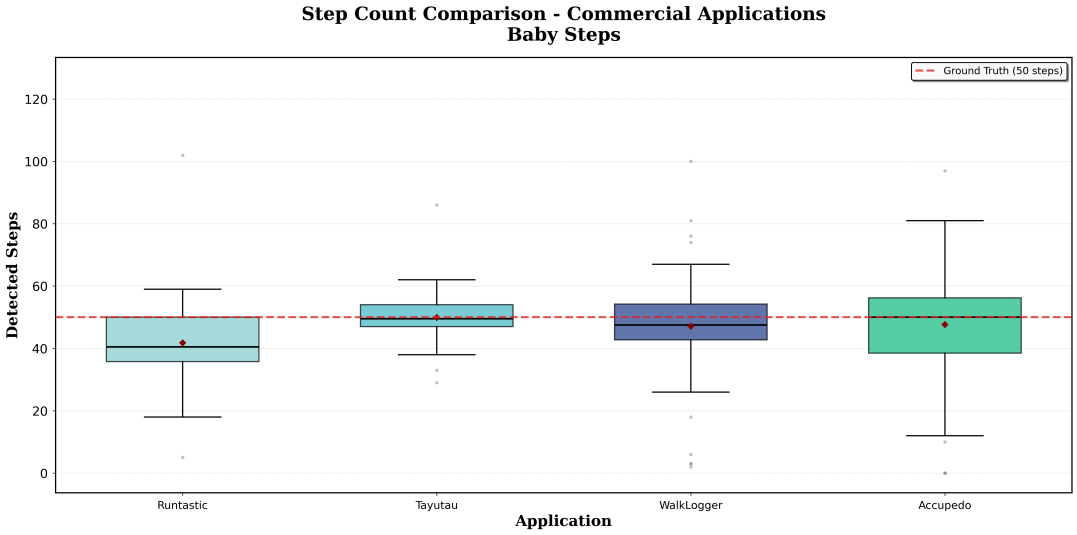

Figura 6.8: Box plot dei risultati per camminata a passi stretti delle applicazioni commerciali.

Peak + Intersection + Low-Pass 2% (MAE = 32.867), con una sottostima sistematica molto marcata.

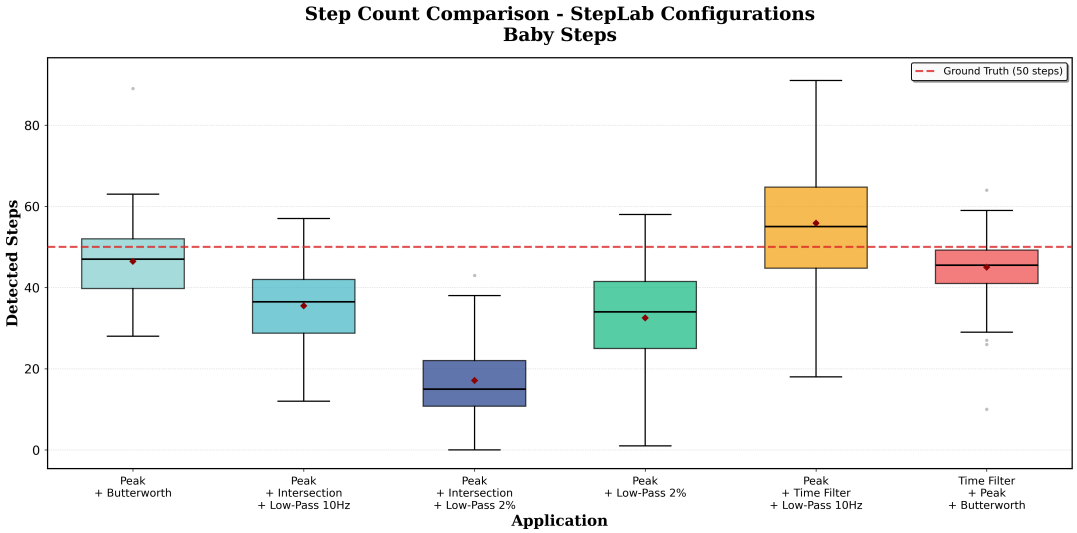

Figura 6.9: Box plot dei risultati per camminata a passi stretti delle configurazioni di StepLab.

Camminata a passi irregolari

Questo tipo di camminata si caratterizza per la variabilità dei passi, i quali possono essere effettuati a ritmo diverso e con intensità differenti. Questa irregolarità rende più complesso per gli algoritmi individuare i passi in modo accurato, poiché il segnale diventa meno prevedibile. I valori delle metriche calcolati per questa tipologia di camminata sono riportati nella Tabella 6.5.

La visualizzazione grafica del MAE con intervalli di confidenza è riportata in Figura 6.10.

Le applicazioni commerciali mostrano una certa variabilità nelle prestazioni. **Tayutau** è ancora una volta l'applicazione più accurata con un MAE di 5.567 e uno SME molto basso (+0.767), indicando un comportamento equilibrato anche in presenza di passi irregolari. **Runtastic** presenta un MAE di 8.067 con una tendenza alla sottostima (SME -3.067). **WalkLogger** mostra invece una marcata tendenza alla sovrastima (SME +6.750) con un MAE di 10.183, mentre **Accupedo** evidenzia uno SME positivo (+3.700) e un MAE di 9.467. Le configurazioni di *StepLab* presentano risultati coerenti con le tipologie di camminata analizzate precedentemente. Le configurazioni più accurate sono **Peak + Butterworth** (MAE = 6.867) e **Time Filter + Peak + Butterworth** (MAE = 7.250). La configurazione **Peak +**

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	8.067	-3.067
Tayutau	5.567	+0.767
WalkLogger	10.183	+6.750
Accupedo	9.467	+3.700
StepLab (Peak + Butterworth)	6.867	-0.633
StepLab (Peak + Intersection + Low-Pass 10Hz)	10.817	-10.117
StepLab (Peak + Intersection + Low-Pass 2%)	27.850	-27.850
StepLab (Peak + Low-Pass 2%)	10.167	-8.067
StepLab (Peak + Time Filter + Low-Pass 10Hz)	12.000	+5.933
StepLab (Time Filter + Peak + Butterworth)	7.250	-2.683

Tabella 6.5: Metriche di accuratezza per la camminata a passi irregolari.

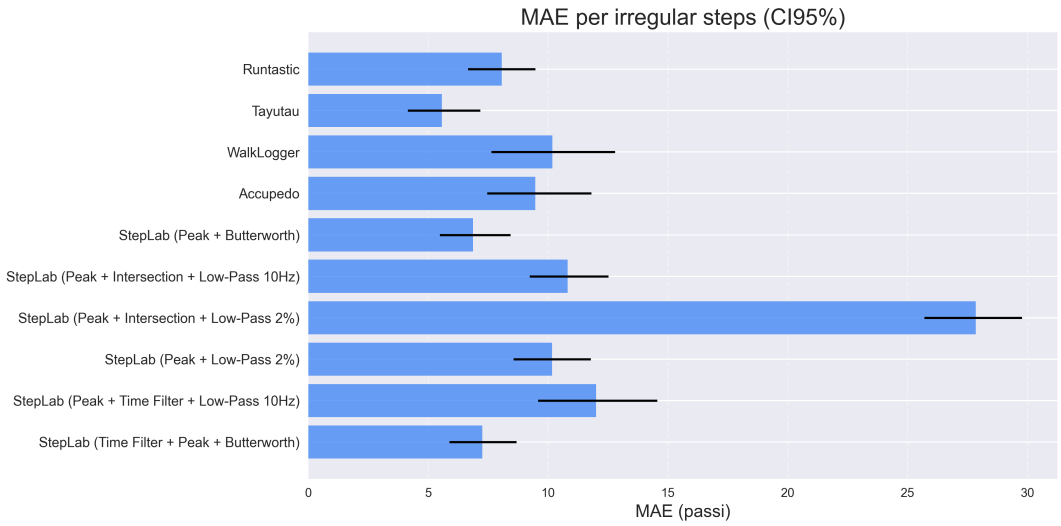

Figura 6.10: MAE con intervalli di confidenza al 95% per la camminata a passi irregolari.

Intersection + Low-Pass 2% si conferma la peggiore, con un MAE di 27.850 e una grave sottostima sistematica (SME -27.850). Anche **Peak + Intersection + Low-Pass 10Hz** mostra prestazioni scarse (MAE = 10.817), con una significativa sottostima dei passi.

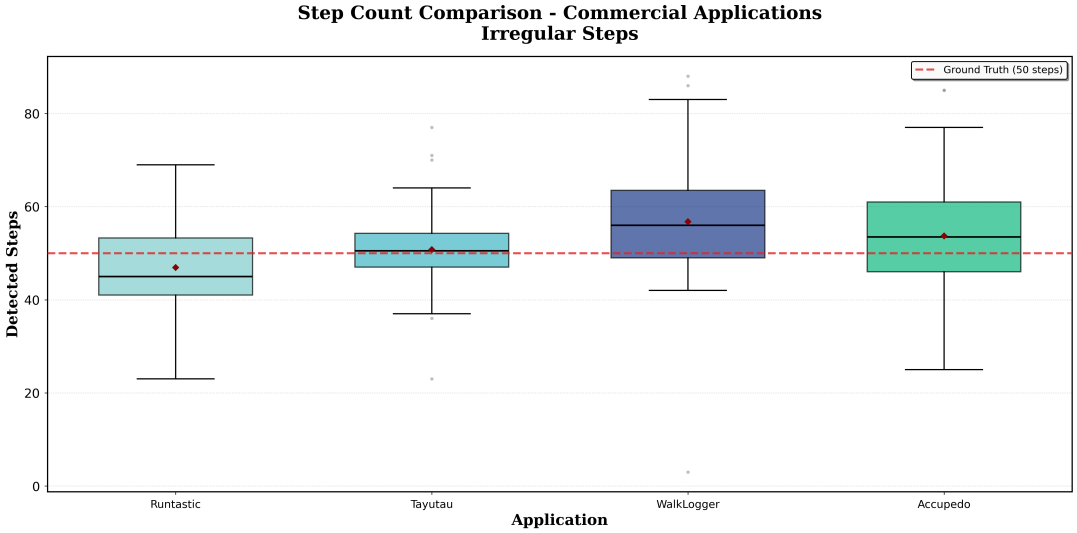

Figura 6.11: Box plot dei risultati per camminata a passi irregolari delle applicazioni commerciali.

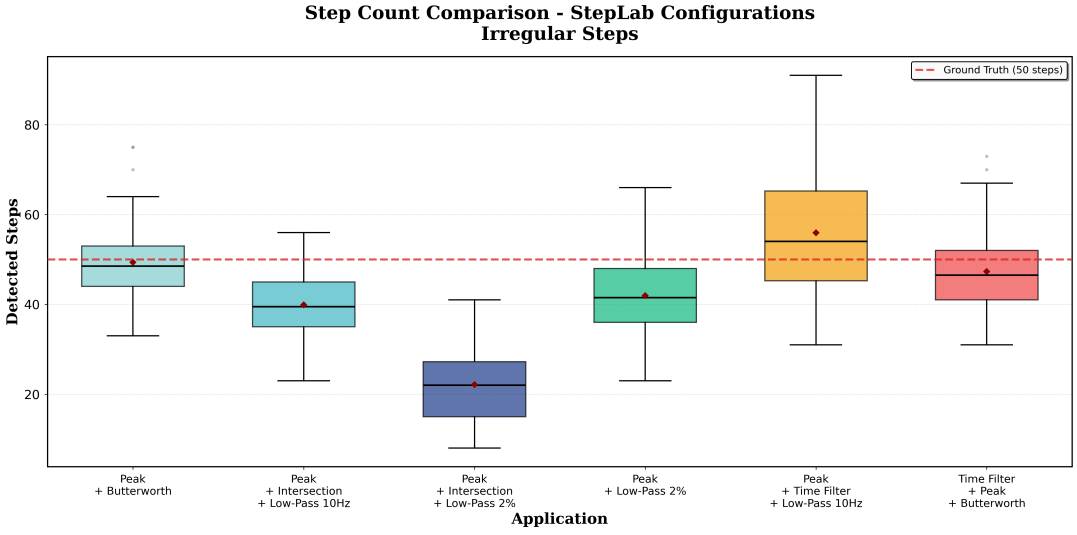

Figura 6.12: Box plot dei risultati per camminata a passi irregolari delle configurazioni di StepLab.

Camminata in salita

Per questo tipo di camminata, i movimenti tendono ad essere più lenti a causa del maggiore sforzo richiesto per salire. Inoltre, l'inclinazione del busto può influenzare il contributo gravitazionale sul segnale accelerometrico. Ciò può rendere più complicata l'individuazione dei passi. Nella Tabella 6.6 sono riportate le metriche riguardanti questa attività.

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	10.217	-7.017
Tayutau	3.483	+0.250
WalkLogger	9.683	-0.017
Accupedo	10.050	+0.250
StepLab (Peak + Butterworth)	6.367	-2.833
StepLab (Peak + Intersection + Low-Pass 10Hz)	14.717	-14.317
StepLab (Peak + Intersection + Low-Pass 2%)	31.650	-31.650
StepLab (Peak + Low-Pass 2%)	13.283	-12.417
StepLab (Peak + Time Filter + Low-Pass 10Hz)	11.350	+4.283
StepLab (Time Filter + Peak + Butterworth)	6.983	-3.383

Tabella 6.6: Metriche di accuratezza per la camminata in salita.

I valori di MAE con i rispettivi intervalli di confidenza al 95% sono rappresentati graficamente in Figura 6.13.

Come nei casi precedenti, **Tayutau** si conferma l'applicazione più accurata con un MAE di 3.483 e uno SME quasi nullo, confermando la sua buona capacità di adattamento a varie situazioni. **Runtastic** presenta un MAE di 10.217 con una marcata tendenza alla sottostima (SME -7.017). **WalkLogger** mostra uno SME praticamente nullo ma un MAE più elevato (9.683), ciò indica una certa dispersione nei risultati. **Accupedo** segue la stessa tendenza, evidenziando un MAE di 10.050 con uno SME minimo.

Le configurazioni di *StepLab* mantengono il pattern osservato nelle tipologie precedenti. Le configurazioni più accurate sono **Peak + Butterworth** (MAE = 6.367) e **Time Filter + Peak + Butterworth** (MAE = 6.983), entrambe con SME moderati. La configurazione **Peak + Intersection + Low-Pass 2%** si conferma

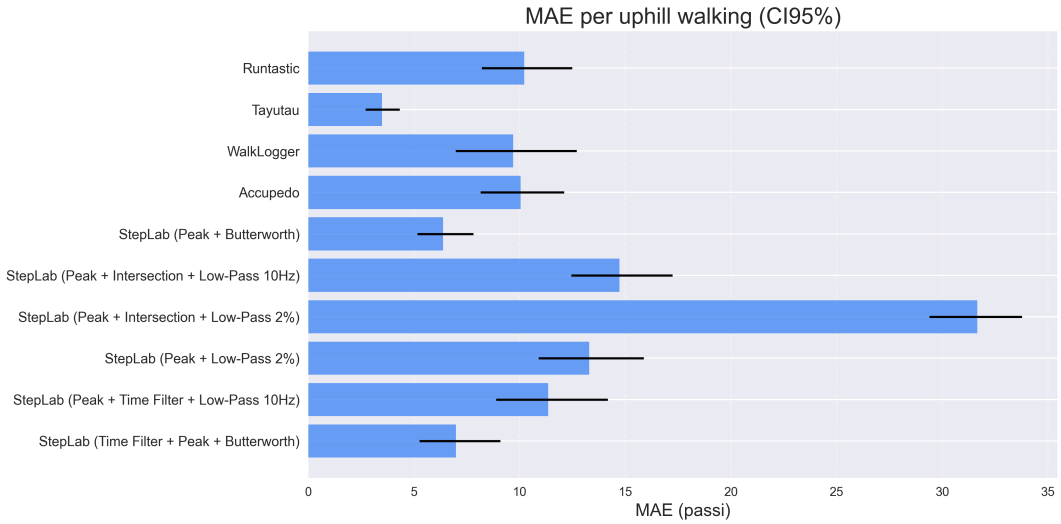

Figura 6.13: MAE con intervalli di confidenza al 95% per la camminata in salita.

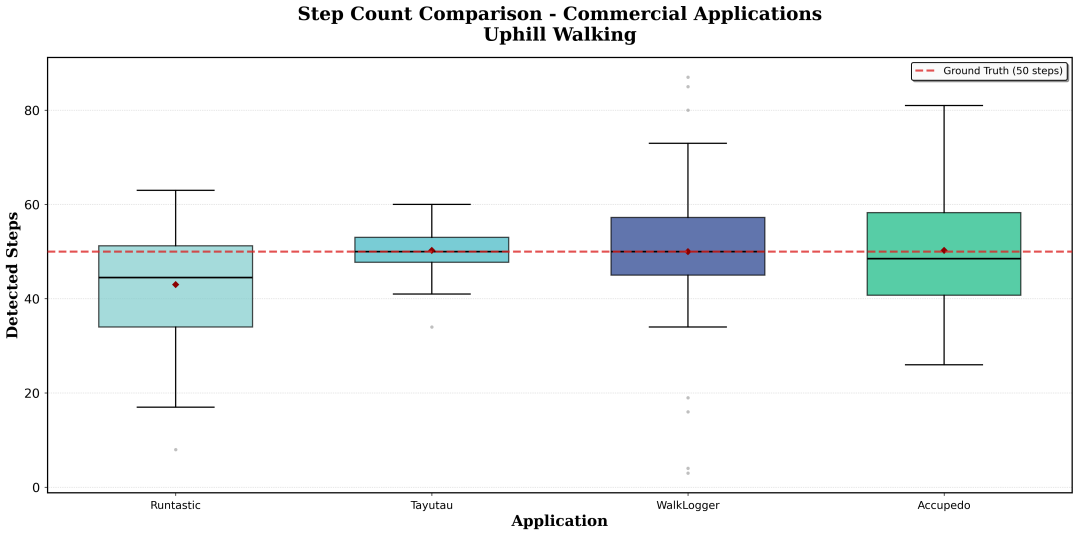

Figura 6.14: Box plot dei risultati per camminata in salita delle applicazioni commerciali.

ancora una volta la peggiore, con un MAE di 31.650 e una sottostima sistematica molto grave.

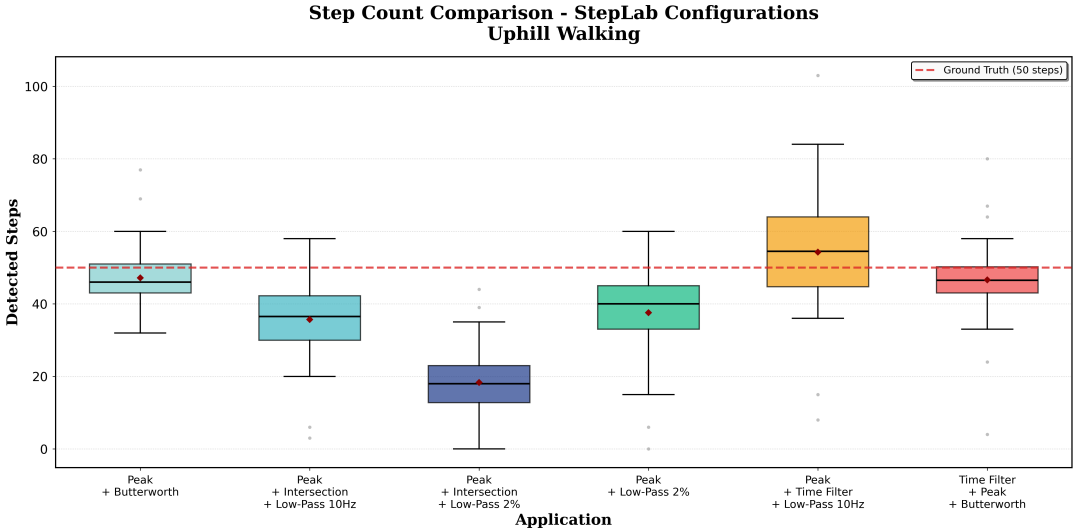

Figura 6.15: Box plot dei risultati per camminata in salita delle configurazioni di StepLab.

Camminata in discesa

Per questo tipo di camminata, i movimenti tendono ad essere più rapidi poiché il corpo tende ad accelerare verso il basso. Inoltre, il busto si inclina all'indietro, modificando il contributo gravitazionale e le piccole frenate che si effettuano per mantenere l'equilibrio contribuiscono a rendere il segnale meno omogeneo e dunque più difficile da analizzare. I valori delle metriche calcolati per questa tipologia di camminata sono riportati nella Tabella 6.7.

La Figura 6.16 illustra graficamente l'errore assoluto medio con gli intervalli di confidenza.

Tra le applicazione commerciali, **Tayutau** continua a distinguersi come la più accurata, con un MAE di 4.148 e uno SME molto basso. **WalkLogger** e **Accupedo** hanno sorprendentemente ottenuto lo stesso MAE di 8.951, ma il primo presenta uno SME quasi nullo, mentre il secondo tende leggermente alla sovrastima. **Runtastic** con questo tipo di camminata è invece l'applicazione con le prestazioni peggiori. Le configurazioni di *StepLab* mostrano ancora una volta un comportamento coerente con le tipologie precedenti. Le configurazioni più accurate sono **Peak + Butterworth** (MAE = 6.443) e **Time Filter + Peak + Butterworth** (MAE = 6.934). La configurazione **Peak + Intersection + Low-Pass 2%** si conferma la peggiore,

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	9.951	-7.820
Tayutau	4.148	-0.574
WalkLogger	8.951	-0.066
Accupedo	8.951	+1.639
StepLab (Peak + Butterworth)	6.443	-3.885
StepLab (Peak + Intersection + Low-Pass 10Hz)	14.033	-13.934
StepLab (Peak + Intersection + Low-Pass 2%)	31.049	-31.049
StepLab (Peak + Low-Pass 2%)	12.623	-11.967
StepLab (Peak + Time Filter + Low-Pass 10Hz)	9.590	+2.836
StepLab (Time Filter + Peak + Butterworth)	6.934	-4.639

Tabella 6.7: Metriche di accuratezza per la camminata in discesa.

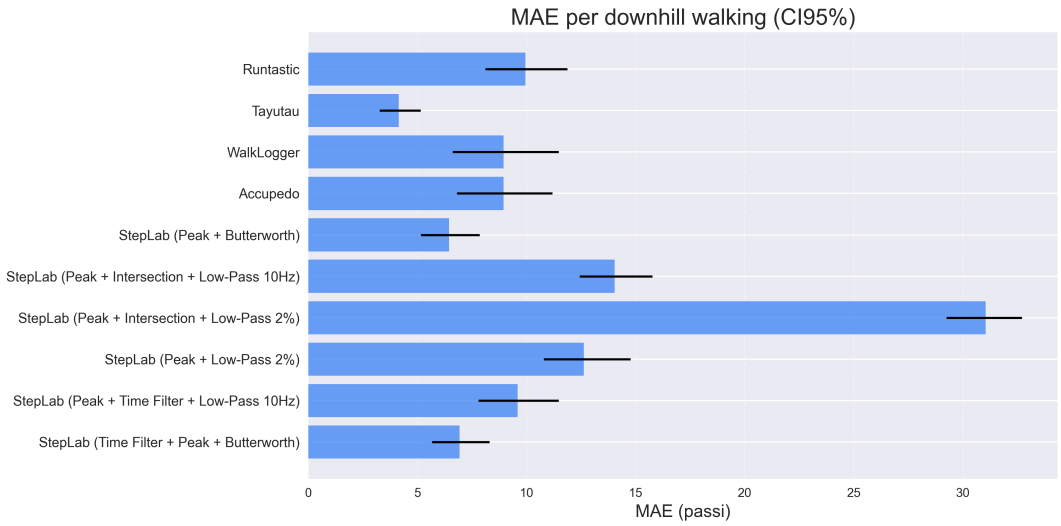

Figura 6.16: MAE con intervalli di confidenza al 95% per la camminata in discesa.

mentre le altre presentano un errore poco superiore rispetto alle performance ottenute con le applicazioni commerciali.

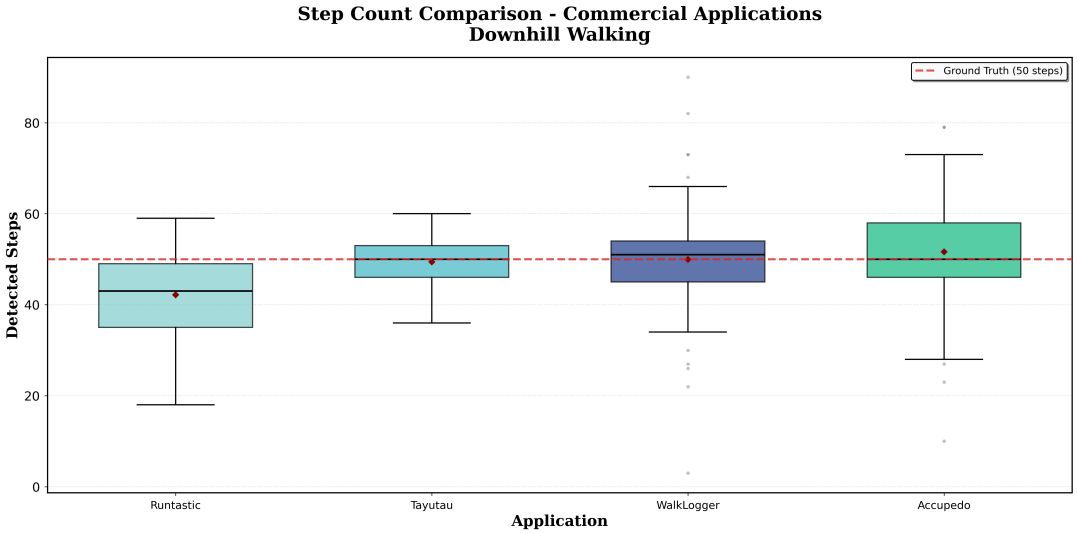

Figura 6.17: Box plot dei risultati per camminata in discesa delle applicazioni commerciali.

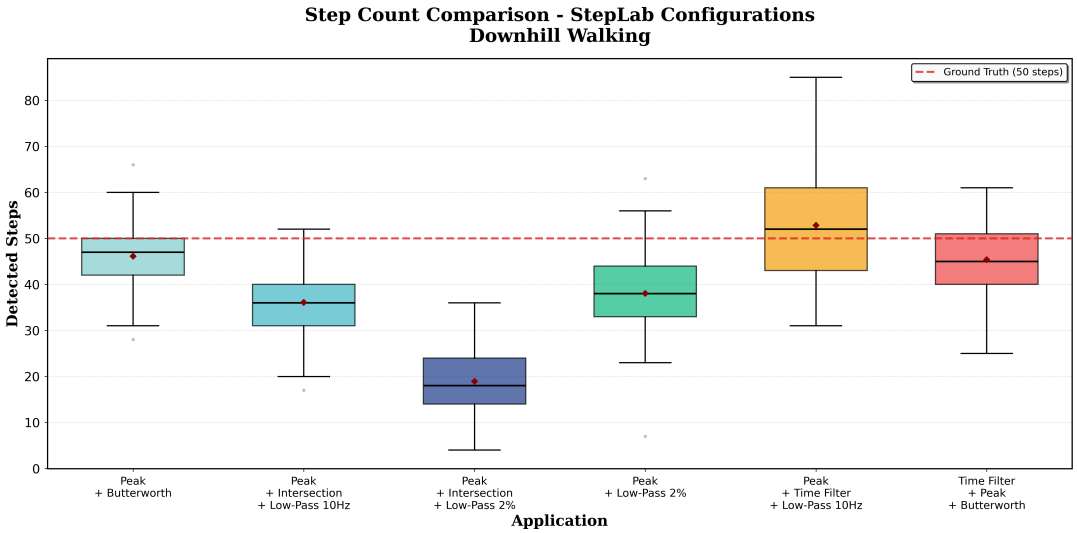

Figura 6.18: Box plot dei risultati per camminata in discesa delle configurazioni di StepLab.

6.3.2 Risultati per posizione del dispositivo

La posizione del dispositivo influenza significativamente il conteggio dei passi, poiché una posizione stabile tende a produrre meno rumore nel segnale rispetto a una posizione più mobile. Analiziamo dunque i risultati distinguendo per le tre posizioni considerate.

Risultati con dispositivo in mano

Una registrazione effettuata con il dispositivo in mano può essere soggetta a rumore aggiuntivo dovuto ai movimenti del braccio. Tuttavia, può anche offrire un segnale più definito relativo ai movimenti del corpo. Ciò varia molto in base alla persona e al modo in cui tiene il dispositivo. Ad esempio, durante la raccolta dati, si è notato che alcuni partecipanti tendevano a muovere il braccio in modo più marcato, mentre altri lo tenevano più fermo. Dunque, tra le posizioni considerate, questa è forse quella con maggiore variabilità. Nella Tabella 6.8 sono riportate le metriche calcolate per questa posizione.

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	10.139	-9.774
Tayutau	5.930	-3.217
WalkLogger	6.374	+0.809
Accupedo	9.670	-2.783
StepLab (Peak + Butterworth)	6.704	-4.930
StepLab (Peak + Intersection + Low-Pass 10Hz)	14.513	-14.235
StepLab (Peak + Intersection + Low-Pass 2%)	28.800	-28.800
StepLab (Peak + Low-Pass 2%)	13.600	-13.409
StepLab (Peak + Time Filter + Low-Pass 10Hz)	7.200	-3.687
StepLab (Time Filter + Peak + Butterworth)	7.287	-6.261

Tabella 6.8: Metriche di accuratezza per dispositivo in mano.

La rappresentazione grafica dei risultati è mostrata in Figura 6.19.

Tra le applicazioni commerciali, **Tayutau** si conferma l'applicazione più accurata con un MAE di 5.930. Tuttavia, anche **WalkLogger** mostra buone prestazioni in questa posizione, con un MAE di 6.374 e uno SME compreso tra 0 e 1, inferiore a quello di Tayutau. **Runtastic** ha le prestazioni peggiori mentre **Accupedo** si posiziona al terzo posto con performance di poco migliori rispetto all'app precedente.

Le configurazioni di *StepLab* mostrano un comportamento simile a quello osservato in precedenza. La più accurata è **Peak + Butterworth** (MAE = 6.704). Anche **Peak + Time Filter + Low-Pass 10Hz** (MAE = 7.200) e **Time Filter + Peak + Butterworth** (MAE = 7.287) ottengono buoni risultati. La penultima ottiene tra l'altro lo SME più basso tra tutte le configurazioni. Ancora una volta,

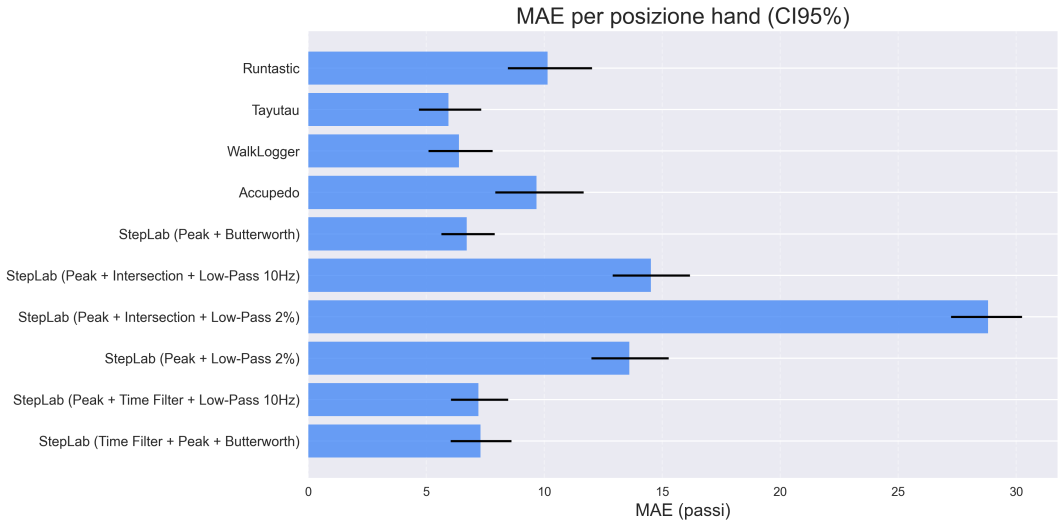

Figura 6.19: MAE con intervalli di confidenza al 95% per dispositivo in mano.

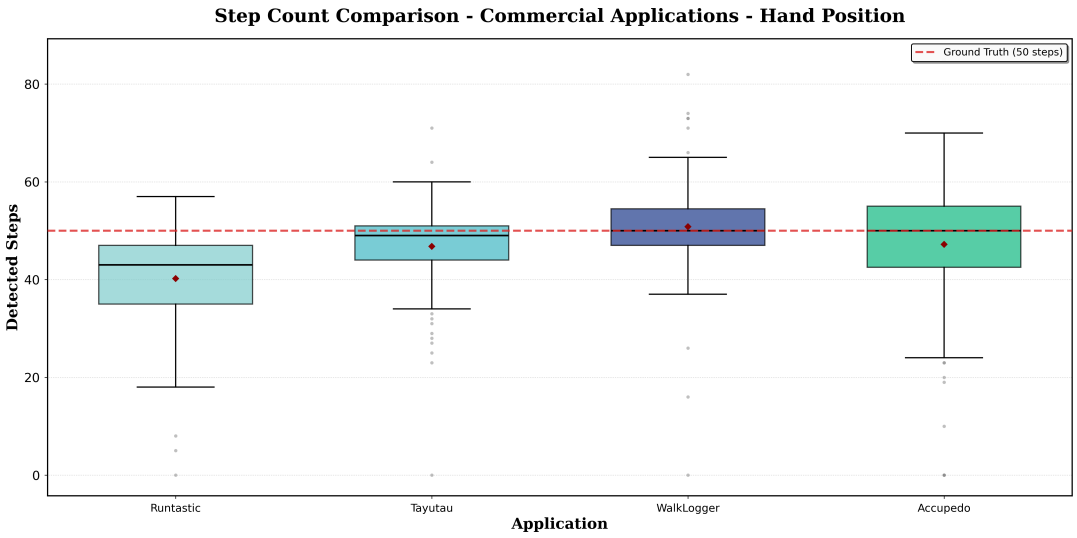

Figura 6.20: Box plot dei risultati con dispositivo in mano delle applicazioni commerciali.

la peggiore è **Peak + Intersection + Low-Pass 2%**, mentre le altre ottengono prestazioni fra loro non troppo distanti in termini di errore assoluto e risultati significativamente peggiori rispetto alle applicazioni commerciali.

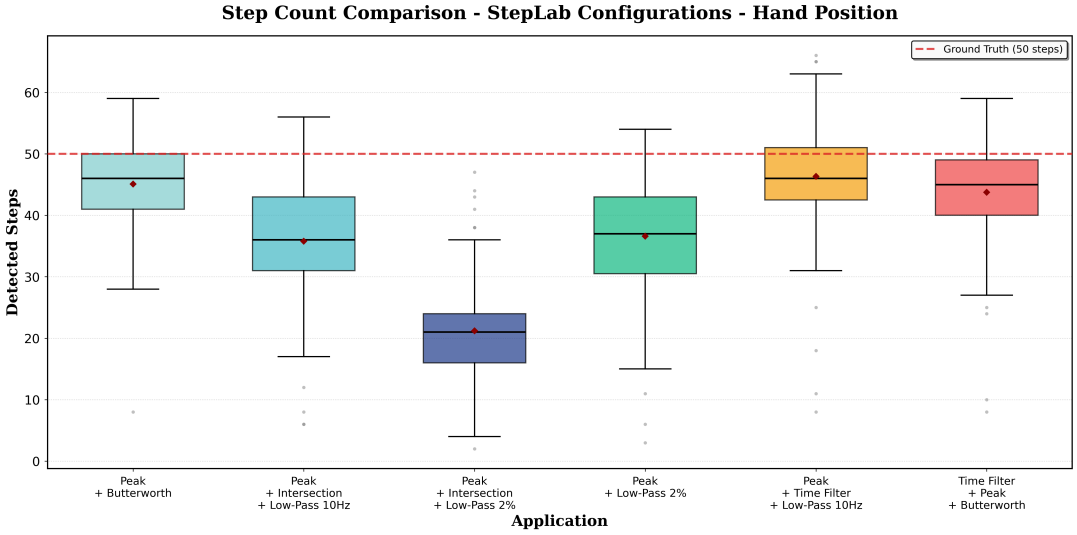

Figura 6.21: Box plot dei risultati con dispositivo in mano delle configurazioni di StepLab.

Risultati con dispositivo in tasca

Quando il dispositivo è in tasca, spesso il segnale risulta più disturbato rispetto alle altre posizioni considerate. Ciò succede perché non essendo in una posizione salda, il telefono può muoversi abbastanza liberamente. Questo varia in base al tipo di abbigliamento indossato, una tasca più stretta tenderà a limitare i movimenti del dispositivo, mentre una tasca più ampia permetterà al telefono di oscillare maggiormente. Nella Tabella 6.9 sono riportate le metriche calcolate per questa posizione.

Tra le applicazioni commerciali, **Tayutau** si conferma l'applicazione più accurata con un MAE di 5.442, seguita da **Runtastic** con 6.075. Più distanti risultano essere **WalkLogger** e **Accupedo**, entrambe con un margine di errore significativo. In tutti i casi si nota una tendenza alla sovrastima del numero di passi, frutto proprio del rumore aggiuntivo introdotto dalla posizione in tasca.

Per *StepLab*, invece, la tendenza è sempre quella dei risultati precedenti: le tre configurazioni più accurate sono **Time Filter + Peak + Butterworth** (MAE = 6.275), **Peak + Butterworth** (MAE = 7.458) e **Peak + Low-Pass 2%** (MAE = 8.420). La peggiore è ancora una volta **Peak + Intersection + Low-Pass**

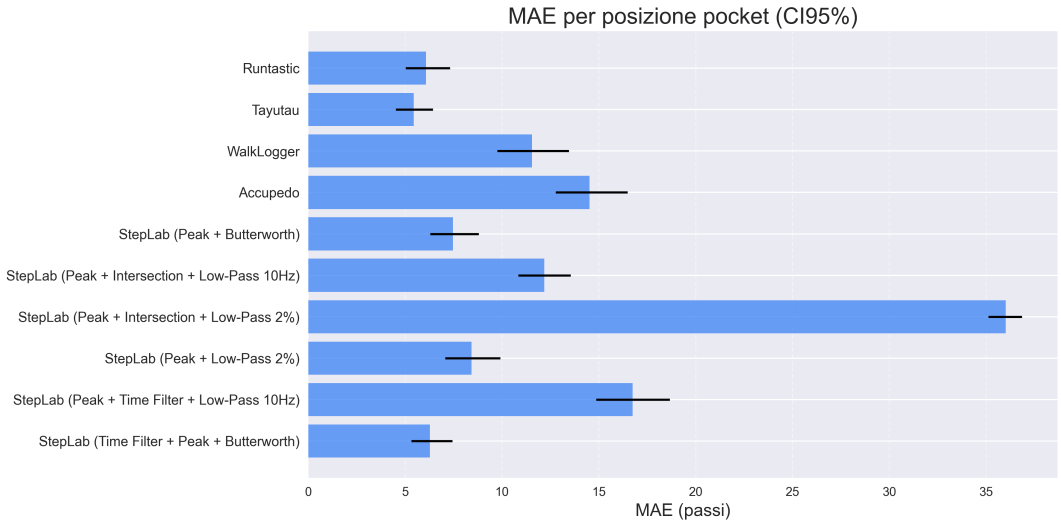

Figura 6.22: MAE con intervalli di confidenza al 95% per dispositivo in tasca.

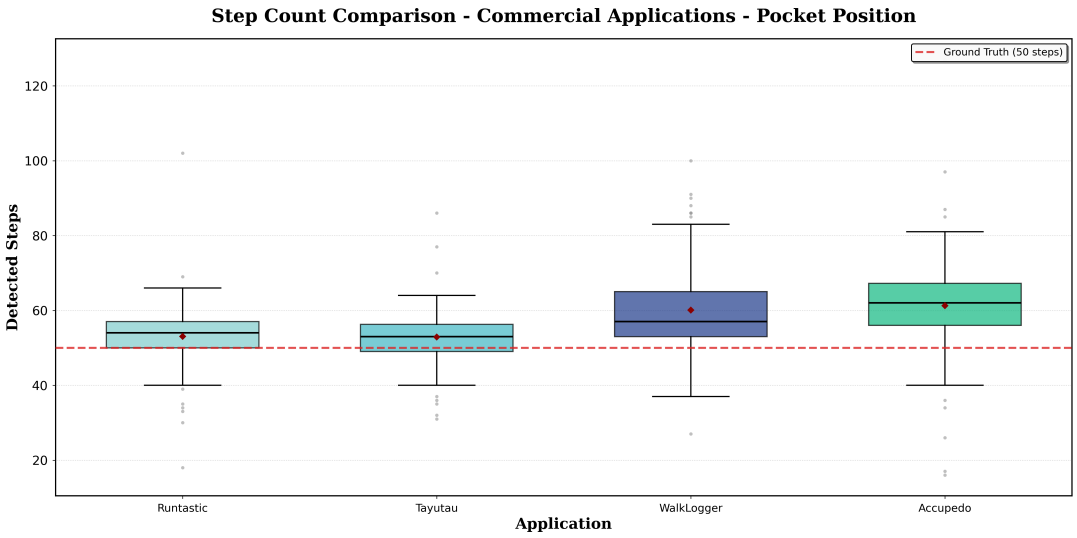

Figura 6.23: Box plot dei risultati con dispositivo in tasca delle applicazioni commerciali.

Applicazione / Configurazione	MAE	SME
Runtastic	6.075	+3.042
Tayutau	5.442	+2.875
WalkLogger	11.550	+10.083
Accupedo	14.508	+11.242
StepLab (Peak + Butterworth)	7.458	+2.475
StepLab (Peak + Intersection + Low-Pass 10Hz)	12.183	-11.050
StepLab (Peak + Intersection + Low-Pass 2%)	36.008	-36.008
StepLab (Peak + Low-Pass 2%)	8.420	-4.958
StepLab (Peak + Time Filter + Low-Pass 10Hz)	16.742	+15.025
StepLab (Time Filter + Peak + Butterworth)	6.275	+1.908

Tabella 6.9: Metriche di accuratezza per dispositivo in tasca.

2%, mentre le due restanti configurazioni ottengono risultati piuttosto distanti dalle migliori tre.

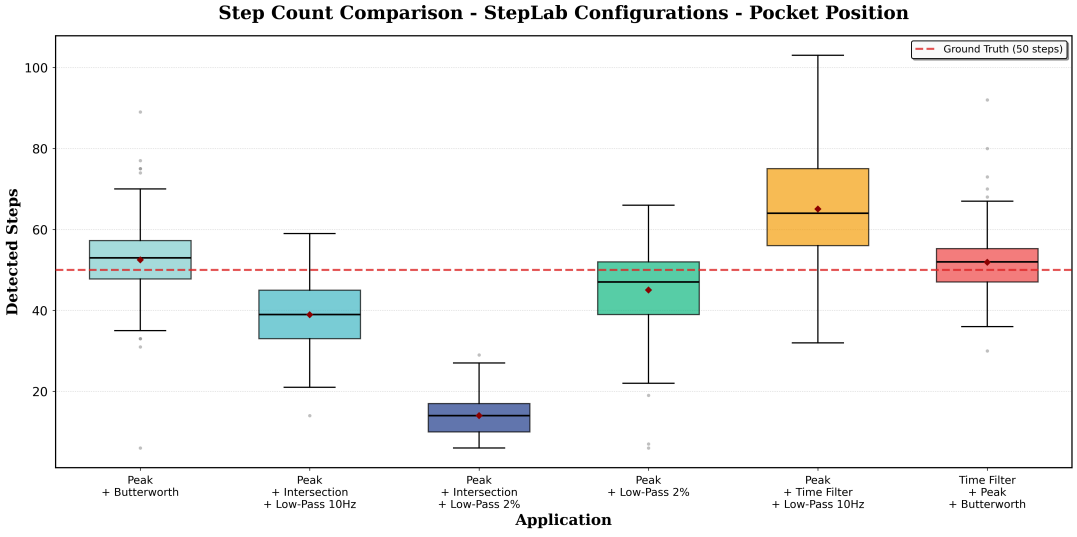

Figura 6.24: Box plot dei risultati con dispositivo in tasca delle configurazioni di StepLab.

Risultati con dispositivo in spalla

Quando il dispositivo è fissato in spalla tramite una fascia elastica dedicata, il segnale tende a essere più stabile rispetto alle altre posizioni considerate. Ciò

è dovuto al fatto che il dispositivo è saldamente ancorato al corpo, riducendo i movimenti indesiderati. Tuttavia, questa posizione può anche far tendere il segnale a essere meno pronunciato, ciò varia in base allo stile di camminata della persona. Nella Tabella 6.10 sono riportate le metriche calcolate per questa posizione. La Figura 6.25 riporta la visualizzazione grafica del MAE con intervalli di confidenza.

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	10.685	-10.228
Tayutau	3.969	-1.858
WalkLogger	9.850	-6.717
Accupedo	7.685	-5.260
StepLab (Peak + Butterworth)	6.205	-3.165
StepLab (Peak + Intersection + Low-Pass 10Hz)	11.079	-10.386
StepLab (Peak + Intersection + Low-Pass 2%)	27.173	-27.173
StepLab (Peak + Low-Pass 2%)	12.134	-11.740
StepLab (Peak + Time Filter + Low-Pass 10Hz)	9.646	+3.787
StepLab (Time Filter + Peak + Butterworth)	6.378	-4.331

Tabella 6.10: Metriche di accuratezza per dispositivo in spalla.

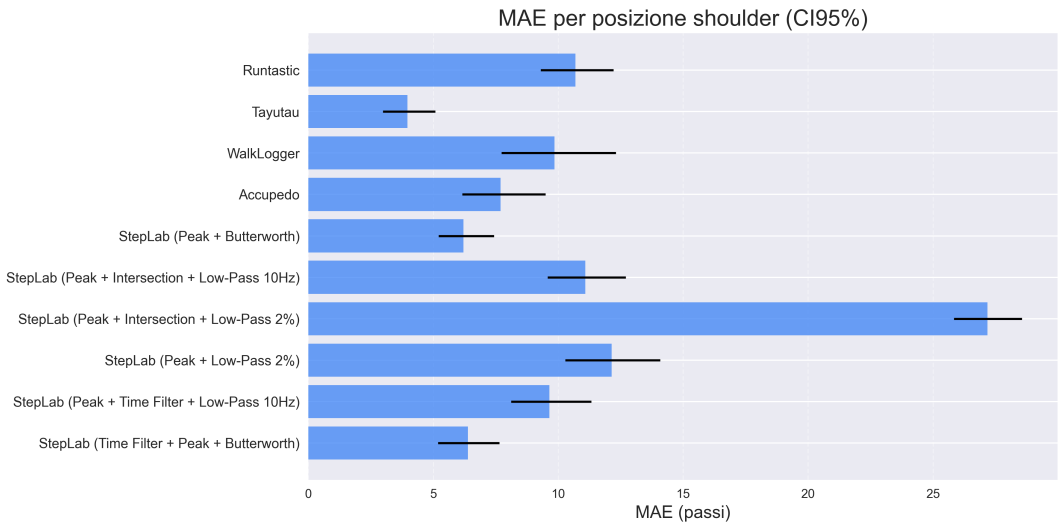

Figura 6.25: MAE con intervalli di confidenza al 95% per dispositivo in spalla.

Tayutau si conferma l'applicazione più accurata con un MAE di 3.969 e uno SME di -1.858. Le prestazioni delle altre applicazioni commerciali presentano in

questo caso un distacco più marcato, fornendo prestazioni significativamente peggiori rispetto alla migliore.

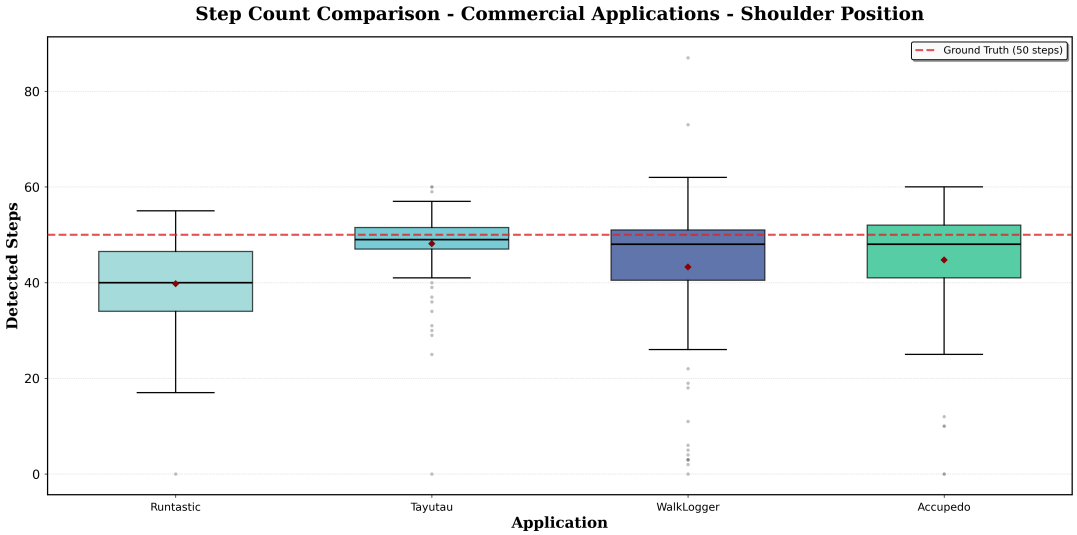

Figura 6.26: Box plot dei risultati con dispositivo in spalla delle applicazioni commerciali.

Per *StepLab*, le configurazioni più accurate rimangono **Peak + Butterworth** (MAE = 6.205) e **Time Filter + Peak + Butterworth** (MAE = 6.378). La peggiore è ancora una volta **Peak + Intersection + Low-Pass 2%**, mentre le altre configurazioni ottengono risultati piuttosto distanti dalle migliori due.

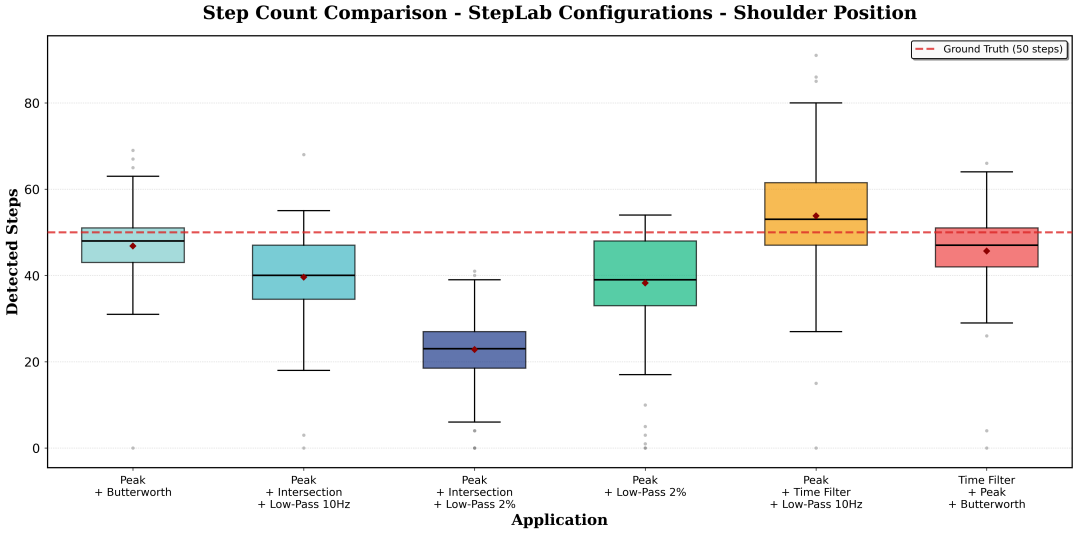

Figura 6.27: Box plot dei risultati con dispositivo in spalla delle configurazioni di StepLab.

In conclusione, la maggiore stabilità offerta da questa posizione trova riscontro nella media dell'errore assoluto, che risulta essere la più bassa tra le tre posizioni considerate, seguita da quella in mano ed infine quella in tasca.

6.3.3 Analisi complessiva dei risultati

Per avere una visione d'insieme delle prestazioni delle varie applicazioni e configurazioni considerate, la Tabella 6.11 riporta il MAE e lo SME calcolati sull'intero dataset, senza distinzione di tipologia di camminata o posizione del dispositivo.

Applicazione / Configurazione	**MAE**	**SME**
Runtastic	8.983	-5.685
Tayutau	5.080	-0.721
WalkLogger	9.309	+1.243
Accupedo	10.577	+0.997
StepLab (Peak + Butterworth)	6.779	-1.856
StepLab (Peak + Intersection + Low-Pass 10Hz)	12.536	-11.829
StepLab (Peak + Intersection + Low-Pass 2%)	30.619	-30.619
StepLab (Peak + Low-Pass 2%)	11.377	-10.036
StepLab (Peak + Time Filter + Low-Pass 10Hz)	11.221	+5.138
StepLab (Time Filter + Peak + Butterworth)	6.633	-2.876

Tabella 6.11: Metriche di accuratezza complessive.

Dall'analisi emerge che, tra le applicazioni commerciali, **Tayutau** è quella che si comporta meglio, mostrando uno SME inferiore a 1 e un MAE nettamente più basso rispetto a quello delle altre app. Runtastic e WalkLogger ottengono prestazioni intermedie, mentre **Accupedo** risulta la meno accurata del gruppo.

Per quanto riguarda *StepLab*, le configurazioni più affidabili sono quelle che utilizzano il filtro Butterworth. In particolare, **Time Filter + Peak + Butterworth** e **Peak + Butterworth** ottengono un MAE inferiore a 7, posizionandosi immediatamente dopo Tayutau e superando le altre app commerciali. Le configurazioni con filtraggio molto aggressivo e intersezione con l'asse delle ascisse, invece, mostrano i risultati peggiori, con una sottostima sistematica del numero di passi.

Sulla base dei risultati globali, la classifica finale ordinata per errore assoluto medio è la seguente:

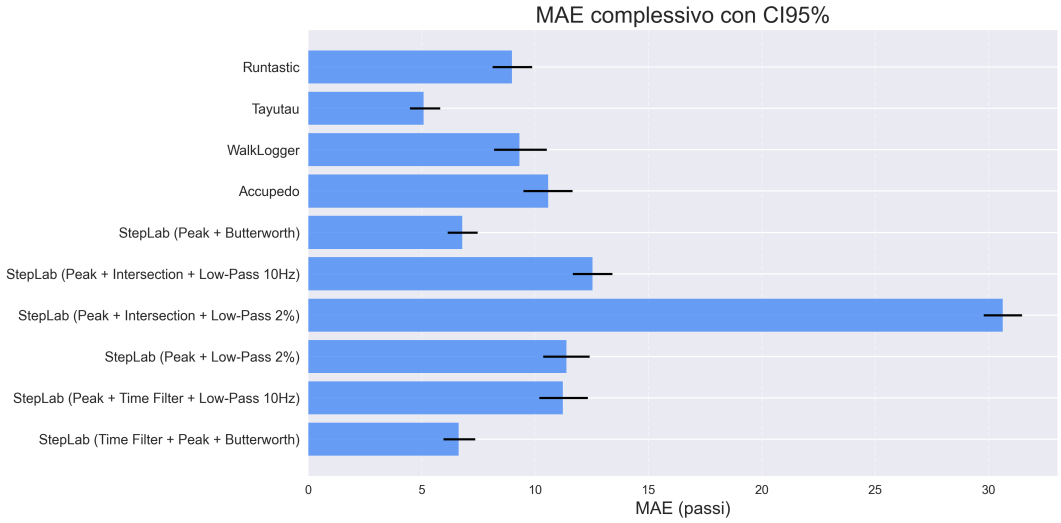

Figura 6.28: MAE con intervalli di confidenza al 95% per i risultati complessivi.

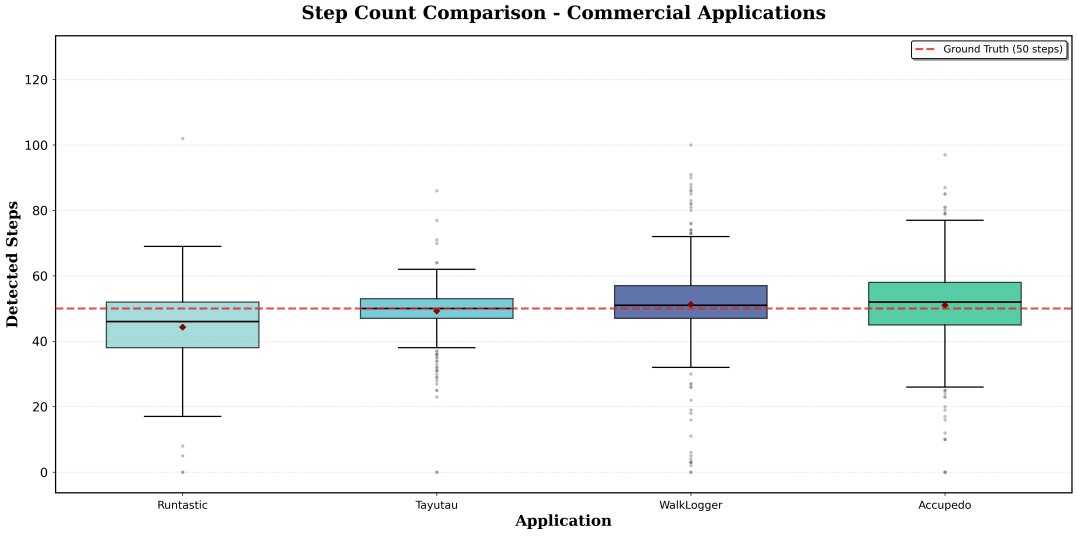

Figura 6.29: Box plot dei risultati complessivi delle applicazioni commerciali.

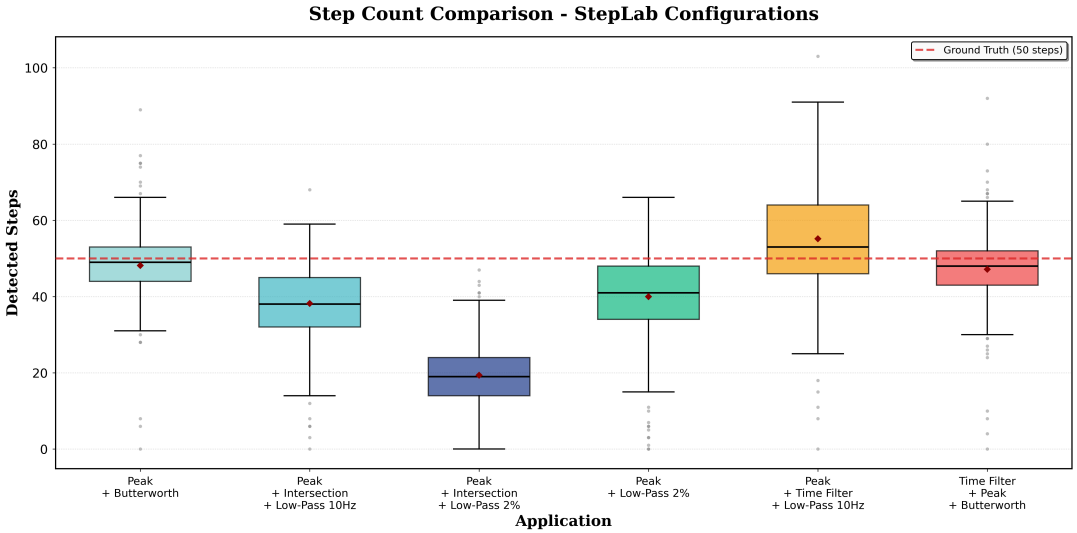

Figura 6.30: Box plot dei risultati complessivi delle configurazioni di StepLab.

1. Tayutau (MAE = 5.080)
2. StepLab (Time Filter + Peak + Butterworth) (MAE = 6.633)
3. StepLab (Peak + Butterworth) (MAE = 6.779)
4. Runtastic (MAE = 8.983)
5. WalkLogger (MAE = 9.309)
6. Accupedo (MAE = 10.577)
7. StepLab (Peak + Low-Pass 2%) (MAE = 11.377)
8. StepLab (Peak + Time Filter + Low-Pass 10Hz) (MAE = 11.221)
9. StepLab (Peak + Intersection + Low-Pass 10Hz) (MAE = 12.536)
10. StepLab (Peak + Intersection + Low-Pass 2%) (MAE = 30.619)

Nel complesso, i risultati confermano quanto osservato nelle analisi per singola tipologia di camminata e per posizione del dispositivo. Le applicazioni commerciali mostrano generalmente prestazioni più affidabili, ma alcune configurazioni di *StepLab* riescono ad avvicinarsi (e in alcuni casi superare) i risultati offerti da diverse app proprietarie.

Conclusioni

In questo lavoro di tesi è stato progettato e valutato un sistema completo per il confronto delle prestazioni di applicazioni commerciali e algoritmi open-source per il conteggio dei passi. La prima fase ha riguardato la raccolta di un dataset sperimentale, coinvolgendo 20 partecipanti. Successivamente è stata sviluppata l'applicazione *StepLab* per Android, che permette di configurare diverse tipologie di algoritmi per il rilevamento. Infine, è stato realizzato un sistema di valutazione basato sull'iniezione controllata dei dati, capace di riprodurre in emulatore le stesse condizioni delle registrazioni reali e dunque di valutare in modo riproducibile le prestazioni degli algoritmi. Questo sistema è stato validato per verificarne l'accuratezza. Nonostante alcune limitazioni dovute al meccanismo di campionamento dell'emulatore, i risultati hanno mostrato un livello di precisione complessivamente soddisfacente, che ha permesso di condurre un'analisi comparativa affidabile. Le metriche adottate hanno evidenziato che alcune configurazioni di *StepLab* sono in grado di competere con le applicazioni commerciali considerate, raggiungendo in diversi scenari prestazioni paragonabili o addirittura superiori. L'analisi statistica condotta ha messo in luce le differenze di accuratezza tra le varie soluzioni, consentendo di identificare gli algoritmi più stabili e le condizioni che influenzano maggiormente l'errore. Questo lavoro apre la strada a ulteriori sviluppi. In particolare, il sistema di simulazione descritto in 4 si è dimostrato non completamente accurato nel replicare le condizioni reali. I ritardi introdotti dal sistema di iniezione rappresentano oggi la principale fonte di disallineamento. Un miglioramento possibile sarebbe intervenire sul funzionamento dei sensori virtuali dell'emulatore Android, affinché campionino i dati non a intervalli fissi, ma al momento dell'arrivo di un nuovo campione. Ciò permetterebbe di eliminare gran parte delle problematiche di sincronizzazione e migliorerebbe sensibilmente la fedeltà delle simulazioni, fermo restando che gli algoritmi basati su soglie temporali avrebbero comunque limitazioni per la simulazione, questo perché la soluzione proposta non assicurerebbe la consegna con intervalli di tempo esattamente identici a quelli reali. In conclusione, il sistema sviluppato ha dimostrato

l'efficacia dell'approccio basato sull'iniezione dei dati e rappresenta una base solida per futuri lavori sull'analisi e l'ottimizzazione degli algoritmi di conteggio dei passi.

Bibliografia

[1] Tom Mikael Ahola. Pedometer for running activity using accelerometer sensors on the wrist. *Medical Equipment Insights*, 3:MEI–S3748, 2010.

[2] Android Developers. Sensors overview. `https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview`. Accesso: 23 novembre 2025.

[3] Yunhoon Cho, Hyuntae Cho, and Chong-Min Kyung. Design and implementation of practical step detection algorithm for wrist-worn devices. *IEEE Sensors Journal*, 16(21):7720–7730, 2016.

[4] Frederic Ehrler, Chloé Weber, and Christian Lovis. Influence of pedometer position on pedometer accuracy at various walking speeds: a comparative study. *Journal of medical Internet research*, 18(10):e268, 2016.

[5] ff225. Motiontracker. `https://github.com/ff225/MotionTracker`, 2025. Accessed: 2025-11-04.

[6] Francesco Forlani. Steplab. `https://github.com/Forla03/StepLab`, 2025. Accessed: 2025-11-27.

[7] Jonatan Fridolfsson, Mats Börjesson, Christoph Buck, Örjan Ekblom, Elin Ekblom-Bak, Monica Hunsberger, Lauren Lissner, and Daniel Arvidsson. Effects of frequency filtering on intensity and noise in accelerometer-based physical activity measurements. *Sensors*, 19(9):2186, 2019.

[8] Ngoc-Huynh Ho, Phuc Huu Truong, and Gu-Min Jeong. Step-detection and adaptive step-length estimation for pedestrian dead-reckoning at various walking speeds using a smartphone. *Sensors*, 16(9):1423, 2016.

[9] Sampath Jayalath and Nimsiri Abhayasinghe. A gyroscopic data based pedometer algorithm. In *2013 8th International Conference on Computer Science & Education*, pages 551–555, 2013.

[10] Dinesh John, Alvin Morton, Diego Arguello, Kate Lyden, and David Bassett. "what is a step?" differences in how a step is detected among three popular activity monitors that have impacted physical activity research. *Sensors*, 18(4):1206, 2018.

[11] Maan Khedr and Nasser El-Sheimy. A smartphone step counter using imu and magnetometer for navigation and health monitoring applications. *Sensors*, 17(11):2573, 2017.

[12] Jia Yan Leong and Jyh Eiin Wong. Accuracy of three android-based pedometer applications in laboratory and free-living settings. *Journal of sports sciences*, 35(1):14–21, 2017.

[13] Giacomo Neri. Pedometri per smartphone: analisi, implementazione e confronto dei modelli proposti in letteratura, 2022.

[14] Giacomo Neri, Federico Montori, Lorenzo Gigli, Luca Bedogni, Marco Di Felice, and Luciano Bononi. Pedometers for smartphones: Analysis and comparison of real-time algorithms. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*, pages 1–6, 2022.

[15] Meng-Shiuan Pan and Hsueh-Wei Lin. A step counting algorithm for smartphone users: Design and implementation. *IEEE Sensors Journal*, 15(4):2296–2305, 2015.

[16] PhilJay. Mpandroidchart. `https://github.com/PhilJay/MPAndroidChart`, 2024. Accessed: 2025-11-18.

[17] Bernd Porr. iirj - infinite impulse response filters for java. `https://github.com/berndporr/iirj`, 2024. Accessed: 2025-11-19.

[18] Valérie Renaudin and Christophe Combettes. Magnetic, acceleration fields and gyroscope quaternion (magyq)-based attitude estimation with smartphone sensors for indoor pedestrian navigation. *Sensors*, 14(12):22864–22890, 2014.

[19] Alessandro Rossi. Implementazione di algoritmi di conteggio passi, 2021.

[20] Dario Salvi, Carmelo Velardo, Jamieson Brynes, and Lionel Tarassenko. An optimised algorithm for accurate steps counting from smart-phone accelerometry. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4423–4427, 2018.

[21] João Santos, António Costa, and Maria João Nicolau. Autocorrelation analysis of accelerometer signal to detect and count steps of smartphone users. In *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–7. IEEE, 2019.

[22] Arun Kumar Siddanahalli Ninge Gowda, Swarna Ravindra Babu, and Dhineshkumar Chandra Sekaran. Umoisp: Usage mode and orientation invariant smartphone pedometer. *IEEE Sensors Journal*, 17(3):869–881, 2017.

[23] Anabela G Silva, Patrícia Simões, Alexandra Queirós, Mário Rodrigues, and Nelson P Rocha. Mobile apps to quantify aspects of physical activity: a systematic review on its reliability and validity. *Journal of medical systems*, 44(2):51, 2020.

[24] Fabio A Storm, Ben W Heller, and Claudia Mazzà. Step detection and activity recognition accuracy of seven physical activity monitors. *PloS one*, 10(3):e0118723, 2015.

[25] Young Soo Suh, Ebrahim Nemati, and Majid Sarrafzadeh. Kalman-filter-based walking distance estimation for a smart-watch. In *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pages 150–156, 2016.

[26] Abhijit Suprem, Vishal Deep, and Tarek Elarabi. Orientation and displacement detection for smartphone device based imus. *IEEE Access*, 5:987–997, 2016.

[27] Alessio Terzi. Architettura per testing di applicazioni android, 2020.

[28] Alessio Terzi, Federico Montori, Lorenzo Gigli, Luca Bedogni, Marco Di Felice, and Luciano Bononi. Comparison of commercial pedometer applications: A rigorous approach. In *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 272–277, 2024.

[29] Xiaokun Yang and Baoqi Huang. An accurate step detection algorithm using unconstrained smartphones. In *The 27th Chinese Control and Decision Conference (2015 CCDC)*, pages 5682–5687, 2015.

Ringraziamenti

Vorrei ringraziare tutte le persone che hanno contribuito alla realizzazione di questo lavoro di tesi. In particolare, un sentito ringraziamento va a tutti i partecipanti che hanno dedicato gratuitamente il loro tempo alla raccolta dei dati, senza i quali questo studio non sarebbe stato possibile.